SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 1, January 1987

NEW DIRECTIONS

With this issue, the *SIAM Journal on Scientific and Statistical Computing* (SISSC) begins bimonthly publication. Founded seven years ago by SIAM Past-president Gene Golub, SISSC has come to occupy a major place in the scientific community.

These past years have brought us from birth to maturity: we receive a steady flow of excellent and exciting papers, we have taken our place among the outstanding journals in our field as a widely cited reference, and we have a good base of subscribers that will enable us to grow over the next decade. Now that we are established and thriving, the question that confronts us is how SISSC can best serve the advancement of scientific computing.

Clearly, the editorial purpose of SISSC has been and is to assist in the development of new and/or better methods of computation. To accomplish this end, the editorial focus of SISSC must be as dynamic as scientific computing itself—but, if we are truly to serve our community well, the directions we take must be considered, deliberate, and based on sound scientific judgement that will stand the test of time. Because SISSC addresses a spectrum of computational issues I consider our appeal and mission to be directed principally to mathematicians and numerical analysts who use computational techniques to solve or clarify problems in other disciplines. If we are to have an impact on scientific computing, given our audience, it will be because we publish material that, first, demonstrates the solution of "real" computational problems that will be useful to mathematicians and others working in the field and, second, uses rigorous and provocative techniques that will lead to the development of new methods or models or be widely applicable.

Because our field is so dynamic, SISSC editors must be constantly striving for prescience: to determine where the trends in scientific computing will lead us five years from now, to judge which methods now in their infancy will be broadly influential over the next decade. All of this is certainly not easy, but, if we can accomplish it, it will enable us to influence the way scientific computing is done in an enduring and substantive way.

Preceding this editorial is a new statement of editorial policy that is derived from reflections by the Editors on the directions SISSC should take over the next few years. We strongly invite your reactions and suggestions.

Finally, I would like to draw your attention to the inclusion in this issue of selected papers from a conference on parallel processing held in Norfolk, Virginia in November 1985. We are particularly pleased to publish these papers, as we wish to expand our coverage of parallel processing in those areas that are appropriate to our scope and hope to continue to devote sections of future issues to the subject.

The Editors also wish to emphasize that SISSC welcomes the opportunity to publish collections of papers from seminal conferences in the field of scientific computing as an excellent means of accomplishing our editorial goals. Such papers would be reviewed using standard procedures but would be expedited to reduce publication delay.

I hope you will join with me in support of SISSC as we enter a new era of growth and change.

—C. W. Gear, *Managing Editor*

© 1987 Society for Industrial and Applied Mathematics

A NOTE FROM THE EDITORS

The future of scientific computing will certainly include parallel processing. The editors of the *SIAM Journal on Scientific and Statistical Computing* (SISSC) are pleased to publish in the January and March issues of this volume selected papers from the Second Conference on Parallel Processing for Scientific Computing held in Norfolk, Virginia on November 18-21, 1985.

This conference covered a broad spectrum of issues related to parallel computing, including algorithm development and analysis, computer architecture, and programming languages in such areas as automatic detection of parallelism, performance prediction, system simulation, communication and synchronization complexity, and load balancing.

The conference program consisted of 16 invited talks that surveyed some aspect of parallel computation, 24 research papers chosen on the basis of extended abstracts, and a large number of contributed papers. (Because many of the topics addressed were outside the scope of SISSC, a number of quite good papers will not appear in this journal. We expect that additional papers from the conference will be published in journals that cover other areas of parallel computation.)

Of the papers selected for publication in SISSC, two general themes are worth noting. The first is that parallel computers are indeed a reality—results on the performance of particular algorithms on particular systems are now being reported. This experimental activity is clearly quite important to the development of scientific computing, although it is true that a strong and unifying mathematical foundation to this work is as yet lacking. Second is that new algorithms are appearing that have been motivated by parallel computing. In at least one case (the eigenvalue algorithm discussed in the Dongarra-Sorensen paper, to appear in the next issue) the algorithm appears to be superior to the best traditional methods for serial computers. This is an exciting development that vividly demonstrates the value of a new perspective.

The editors of SISSC wish to thank Jack Dongarra and Garry H. Rodrigue, who joined the regular editors A. Björck, D. Gannon, J. A. George, F. O'Sullivan, A. Sameh, D. Sorensen, P. Swarztrauber, and G. Wahba in overseeing the review of the nearly 40 papers that were submitted. Standard SIAM policy for selecting referees and making recommendations for publication were followed.

We also wish to thank the organizing committee of the conference, which consisted of Robert G. Voigt, Chairman, Institute for Computer Applications in Science and Engineering; Billy L. Buzbee, Los Alamos National Laboratory; Dennis B. Gannon, Indiana University; and Garry H. Rodrigue, Lawrence Livermore National Laboratory.

The Second Conference in Parallel Processing was sponsored by the SIAM Activity Group on Supercomputing with partial support from the Air Force Office of Scientific Research, the Army Research Office, and the National Science Foundation.

C. W. Gear
R. G. Voigt

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 1, January 1987
© 1987 Society for Industrial and Applied Mathematics
008

THE WY REPRESENTATION FOR PRODUCTS OF HOUSEHOLDER MATRICES*

CHRISTIAN BISCHOF† AND CHARLES VAN LOAN†

Abstract. A new way to represent products of Householder matrices is given that makes a typical Householder matrix algorithm rich in matrix-matrix multiplication. This is very desirable in that matrix-matrix multiplication is the operation of choice for an increasing number of important high performance computers. We tested the new representation by using it to compute the QR factorization on the FPS-164/MAX. Preliminary results indicate that it is a very efficient way to organize Householder computations.

Key words. Householder matrices, QR factorization, vector parallelism

AMS(MOS) subject classification. 65

1. Introduction. During the past five years a great deal has been written about matrix computations in high performance "supercomputing" environments. For example, techniques have emerged for designing linear algebra codes that "squeeze" the most out of various vector pipeline architectures. The reader is referred to the excellent survey article by Dongarra, Gustavson and Karp (1984). One theme in their work is how to take an algorithm such as Gaussian elimination and organize it in such a way that the resulting code is rich in matrix-vector multiplication. The regularity of that operation allows for maximum pipelining and minimum traffic to and from memory. The papers by Dongarra and Eisenstat (1984) and Dongarra, Kaufman and Hammarling (1985) further dramatize this point.

It is now the case with some new architectures that high performance can best be achieved with algorithms that are rich in *matrix-matrix* multiplication. Such a requirement leads naturally to block algorithms. For methods such as Gaussian elimination and the Cholesky factorization the course of action is relatively straightforward. Consider the LU factorization:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix}.$$

If we assume that the first block row of U and the first block column of L are known, then the blocks L_{22}, U_{22}, L_{32}, and U_{23} can be resolved from the equations

$$L_{22}U_{22} = A_{22} - L_{21}U_{12},$$

$$L_{22}U_{23} = A_{23} - L_{21}U_{13},$$

$$L_{32}U_{22} = A_{32} - L_{31}U_{12}.$$

Note that the right-hand sides are rich in matrix-matrix multiplication. If block dimensions are chosen properly then the overall method will be sufficiently rich in

* Received by the editors December 12, 1985; accepted for publication (in revised form) April 9, 1986. Computations associated with this research were performed on the Production Supercomputer Facility at Cornell University which is supported in part by the National Science Foundation and the IBM Corporation. Additional funding was provided by Office of Naval Research contract N00014-83-K-0640. This paper appeared in the Proceedings of the 1986 ARRAY Conference, Portland, Oregon, April 6-10, 1986.

† Department of Computer Science, Cornell University, Ithaca, New York 14853.

that operation to ensure high performance. This has been demonstrated by Steve Oslon of Floating Point Systems who has implemented the above scheme (with pivoting) on the FPS-164/MAX.

The FPS-164/MAX has an architecture that "likes" matrix-matrix multiplication and it is for this machine that we have developed a new algorithm for the QR factorization that performs extremely well. Although our technique has been tailored to the FPS-164/MAX, we suspect that our work will be of interest to users of other high performance computers such as the Cray-2 and the IBM 3090.

It should be noted that the block procedures for matrix decompositions involving orthogonal transformations are not so straightforward. Consider the QR factorization:

$$A=[A_1, A_2, A_3]=QR=[Q_1, Q_2, Q_3]\begin{bmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \\ 0 & 0 & R_{33} \end{bmatrix}.$$

If the first block column of Q and the first block row of R are known, then Q_2 and R_{22} can be obtained by computing the QR factorization

$$Q_2R_{22}=A_2-Q_1R_{12}$$

whereupon $R_{23}=Q_2^TA_3$. These are essentially the key steps of a block version of modified Gram-Schmidt (MGS). Block MGS is certainly rich in matrix multiplication but the "Q" matrix may not be sufficiently orthonormal for certain applications. (See Golub and Van Loan (1983, pp. 151-152) for a description of MGS and its properties.) It is an open question how to organize block MGS and what its numerical properties are. We shall not treat the matter further in this paper.

Our intention is to examine how the perfectly stable Householder QR factorization procedure can be written in a form that is rich in matrix multiplications. Householder transformations underpin the eigenvalue and least squares solvers in EISPACK and LINPACK. A discussion of the algorithms in these packages may be found in Golub and Van Loan (1983).

The key to obtaining block Householder algorithms that are rich in matrix multiplication turns out to be what we call the "WY representation." It amounts to writing products of Householder matrices like

$$Q_k=P_1\cdots P_k, \qquad P_i\in R^{m\times m}$$

in the form

$$Q_k=I+WY^T, \qquad W, Y\in R^{m\times k}. \tag{1.1}$$

Several authors have looked at similar ways to group and generalize Householder matrices and we review this work in § 2. The basic properties of the WY representation are detailed in § 3. In § 4 we show how to compute the QR factorization using the WY representation. An implementation of our algorithm on the FPS-164/MAX is then described in § 5. The last section suggests how the WY representation can be used to compute other decompositions that involve orthogonal transformations.

2. Aggregating Householder transformations. The idea of "aggregating" Householder transformations for performance is not new. Dongarra, Kaufman and Hammarling (1985) suggest pairing Householder matrices in order to reduce the number of vector memory references. For example, they recommend that the update

$$A:=(I-\alpha ww^T)(I-\beta uu^T)A$$

be computed as follows:

$$v^T := w^T A,$$
$$x^T := u^T A,$$
$$y^T := v^T - (\beta w^T u) x^T,$$
$$A := A - \beta u x^T - \alpha w y^T.$$

As the authors point out, this maneuver actually increases the number of required flops. But the resulting algorithm is faster in pipelined environments because there is less memory traffic. The WY representation (1.1) is a generalization of this pairwise grouping strategy.

Note that an orthogonal matrix of the form $I + WY^T$ can be thought of as a generalized Householder matrix, an idea that was examined by Dietrich (1976). Given an m-by-p matrix partitioned as follows

$$X = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \begin{matrix} p \\ m-p \end{matrix},$$

Dietrich shows how to compute a full rank m-by-p matrix

$$W = \begin{bmatrix} W_1 \\ W_2 \end{bmatrix} \begin{matrix} p \\ m-p \end{matrix},$$

so that if

$$P = I - 2W(W^T W)^{-1} W^T,$$

then

$$P^T X = \begin{bmatrix} T \\ 0 \end{bmatrix} \begin{matrix} p \\ m-p \end{matrix}. \tag{2.1}$$

The matrix P above is clearly a block generalization of a Householder matrix. A stable method for computing W using some recent algorithmic developments is as follows:

Step 1. Compute the QR factorization:

$$X = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R.$$

Step 2. Compute the CS decomposition:

$$Q_1 = U_1 C V^T, \qquad C = \operatorname{diag}(c_1, \cdots, c_p), \qquad 1 \geqq c_1 \geqq \cdots \geqq c_p \geqq 0,$$
$$Q_2 = U_2 S V^T, \qquad S = \operatorname{diag}(s_1, \cdots, s_p), \qquad 0 \leqq s_1 \leqq \cdots \leqq s_p \leqq 1.$$

(Here, U_1, U_2 and V are orthogonal.)

Step 3. Set

$$W = \begin{bmatrix} U_1(C+I) \\ U_2 S \end{bmatrix}.$$

The trouble with this method of aggregating Householder transformations is that some delicate SVD computations are involved in Step 2. See Stewart (1983) or Van Loan (1985) for details concerning the computation of the CS decomposition. In addition to these difficulties, the matrix T in (2.1) will not generally be upper triangular

($T = U_1 V^T R$). Thus, if we incorporated Dietrich's idea in a QR factorization algorithm the resulting R would be block upper triangular. True upper triangular form could easily be achieved but it is an inconvenient post-computation.

3. The WY representation. Recall that an m-by-m Householder matrix is an orthogonal matrix of the form

$$P = I + uv^T$$

where $v \in R^m$ has unit 2-norm length and $u = -2v$. Householder matrices can be used to compute all kinds of important matrix decompositions and the large number of subroutines in LINPACK and EISPACK that rely on these transformations underscores their importance.

In a typical application a sequence of Householder transformations $P_i = I + u_i v_i^T$ is applied to a given matrix. Consider the following algorithm for computing the QR factorization of

$$A = [a_1, \cdots, a_n], \qquad a_j \in R^m.$$

ALGORITHM 3.1.
For $k = 1 : n$
  For $j = 1 : k-1$
    $a_k := a_k + (u_j^T a_k) v_j, \qquad (a_k := P_{k-1} \cdots P_1 a_k)$
  end j
  Compute unit 2-norm v_k so that if $u_k = -2v_k$ and $P_k = I + u_k v_k^T$ then $P_k a_k$ is zero in components $k+1$ to m.
end k

This computation is rich in inner products and SAXPY's but not in matrix multiplication. (A "SAXPY" is the operation $y := \alpha x + y$ where x and y are vectors and α is a scalar.) The SAXPY's in the j loop cannot be performed in parallel because the a_k are different for each j.

Reversing the order of the loops in Algorithm 3.1 essentially gives the LINPACK QR factorization procedure. In this version the key computations are updates of the form $A := (I + u_j v_j^T)^T A$. The inner products associated with $u_j^T A$ can be computed in parallel as can the SAXPY's associated with $A := A + v_j (u_j^T A)$. What prevents "super-vector performance" is an unfavorable ratio of vector loads and unloads to actual computation. See Dongarra, Kaufman and Hammarling (1985).

We propose a new way to represent products of Householder matrices that rectifies this problem. In particular, we exploit the fact that a product of Householders

$$Q_k = P_1 \cdots P_k$$

can be written in the form

$$Q_k = I + W_k Y_k^T$$

where W_k and Y_k are m-by-k matrices. Note that the mission of the j loop in Algorithm 3.1 is to compute $Q_{k-1}^T a_k$.

To anticipate the value of the WY representation, observe that if B is a matrix then the computation $B := Q_k^T B$ is rich in matrix multiplication:

$$B := B + Y_k (W_k^T B).$$

However, before we can pursue the algorithmic benefits of the WY representation we must establish its existence and show how it can be updated.

The WY form certainly exists for the $k=1$ case. Indeed, since $Q_1 = P_1 = I + u_1 v_1^T$ we just set $W = u_1$ and $Y = v_1$. To establish the result for general k, we show how to obtain (W_k, Y_k) from (W_{k-1}, Y_{k-1}) upon receipt of the kth Householder matrix $P_k = I + u_k v_k^T$. From the manipulations

$$\begin{aligned} Q_k = Q_{k-1}P_k &= (I + W_{k-1}Y_{k-1}^T)(I + u_k v_k^T) \\ &= I + [W_{k-1},\ Q_{k-1}u_k][Y_{k-1},\ v_k]^T \\ &= I + [W_{k-1},\ u_k][P_k Y_{k-1},\ v_k]^T, \end{aligned}$$

we see that there are actually two ways to accomplish this:

Method 1.

$W_k = [W_{k-1},\ Q_{k-1}u_k]$,
$Y_k = [Y_{k-1},\ v_k]$.

Method 2.

$W_k = [W_{k-1},\ u_k]$,
$Y_k = [P_k Y_{k-1},\ v_k]$.

We have used Method 1 in our implementations feeling that the matrix-vector multiplication $Q_{k-1}u_k$ is easier to optimize than the rank one update $P_k Y_{k-1}$. However, this is not a clear choice. Indeed, it may be wiser to update WY factors using Method 2 for the following reason. In many applications such as in Algorithm 3.1 the first $k-1$ components of the Householder vector v_k are zero. Thus, if Method 2 is used then a simple check reveals that *both* Y_k and W_k are lower trapezoidal. However, if Method 1 is used than *only* Y_k is lower trapezoidal.

We point out that manipulation of Householder matrices through the WY form is stable. The same favorable roundoff properties that attend "conventional" Householder algorithms apply if the computations are arranged in WY form. This claim is justified in a brief appendix to be found at the end of the paper.

Finally, we mention that the above updating formulae generalize. In particular, if $U = I + W_U Y_U^T$ and $V = I + W_V Y_V^T$ are orthogonal matrices in WY form then $Q = UV$ has its WY factors given by $W_Q = [W_U,\ UW_V]$ and $Y_Q = [Y_U,\ Y_V]$.

4. Computing the QR factorization using the WY representation. To illustrate the attractiveness of the WY representation, we show how it can be used to compute the QR factorization of a rectangular matrix $A = [a_1, \cdots, a_n] \in R^{m\times n}$. A naive (but instructive) WY implementation of Algorithm 3.1 is as follows:

ALGORITHM 4.1.
For $k = 1:n$
  $a_k := (I + WY^T)^T a_k$, $\quad (a_k := Q_{k-1}^T a_k$, skip if $k=1)$
  Compute unit v such that if $u = -2v$ then $(I + uv^T)a_k$ is zero in components $k+1$ to m.
  $W := [W, (I + WY^T)u]$, $\quad (W = [u]$ if $k=1)$
  $Y := [Y, v]$, $\quad (Y = [v]$ if $k=1)$
end k

There are three problems with this algorithm. (1) It requires $2mn^2 - n^3/2$ flops, approximately twice what is required by the usual Householder QR factorization scheme, e.g., Algorithm 3.1. (2) It requires an m-by-n workspace for W. (3) It is not rich in matrix-matrix multiplication.

The same criticisms would essentially apply if Method 2 updating was used although one could get by with $n^2/2$ fewer storage locations and $p^3/2$ fewer flops.

To obtain a successful QR factorization scheme using the WY representation it is necessary to partition A into block columns:

$$A = [A_1, \cdots, A_N]$$

where each A has p columns. (If p does not divide n then A_N may have fewer than p columns, an unimportant detail that we hereafter suppress.) The appropriate column width p depends on the underlying architecture. (For the FPS-164/MAX it will turn out to be equal to the maximum number of parallel dot products that can be performed.)

The basic idea of our procedure is as follows. At the beginning of Step k ($1 \leqq k \leqq N$) the matrix A has been overwritten with

$$P_{(k-1)p} \cdots P_1 A = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ 0 & \bar{A}_k & B \end{bmatrix} \begin{matrix} (k-1)p \\ r \end{matrix}$$
$$\qquad\quad (k-1)p \quad p \quad q$$

where $r = m-(k-1)p$ and $q = n-kp$. The QR factorization of $\bar{A}_k$ is then computed and the orthogonal matrix (in WY form) is applied to B.

In our formal description of this procedure we use the notation $A(i_1 : i_2, j_1 : j_2)$ to designate the submatrix of A defined by rows i_1 through i_2 and columns j_1 through j_2.

ALGORITHM 4.2.
For $k = 1 : N$
  $s := (k-1)p+1$
  Compute W and Y such that $I + WY^T$ is orthogonal and $(I + WY^T)A(s : m, s : s+p-1)$ is upper triangular.
  $A(s : m, s : n) := (I + WY^T)^T A(s : m, s : n)$
end k

Upon emergence from this algorithm the matrix A is overwritten by R. The Householder vectors v_i that are generated can be stored below the diagonal in the usual manner. (See Golub and Van Loan (1983, p. 148).) It is not necessary to save the W matrices from step to step and so we just need an m-by-p workspace for W. (This is in contrast to the m-by-n workspace required by Algorithm 4.1.) Algorithm 4.2 is clearly rich in matrix-matrix multiplication. Indeed much more work is spent applying the WY factors than generating them which is good in that it implies richness in matrix multiplication. To be specific, $(2mn^2 - n^3)/N$ flops are required to generate the W's and Y's while $(mn^2 - n^3/3)$ flops are needed to apply them. Since the LINPACK QR algorithm requires $mn^2 - n^3/3$ flops we see that Algorithm 4.2 is more expensive by a factor of $(1+2/N)$ from the standpoint of flops. For modest (and realistic) values of N this factor is for all practical purposes equal to one. Moreover, in high performance environments just counting flops is an inadequate measure of efficiency. The number of references to memory usually has a much greater bearing on the efficiency of an algorithm. This is borne out by our experience implementing Algorithm 4.2 on the FPS-164/MAX.

5. Performance on the FPS-164/MAX system. Up to now a "flop" has meant the amount of work that is roughly associated with an operation of the form $a_{ij} := a_{ij} + a_{ik}a_{kj}$. However, manufacturers of "high performance" computers quantify speed using "high performance flops." A "high performance flop" equals two "regular" flops. Thus, n-by-n matrix multiplication requires $2n^3$ flops. In this section we will quantify speed

in terms of millions of high performance flops per second as we will be reporting on the FPS-164/MAX. (Independent adder and multiplier units are what makes high performance flop rates relevant.)

The FPS-164 is an 11 Megaflop (Mflop), 64-bit general purpose scientific processor built by Floating Point Systems Inc. It is attached to the I/O system of a general purpose host computer. In a typical situation the program executing on the 164 is a subroutine called from a host FORTRAN program.

The FPS-164 can be enhanced with additional "MAX boards" that can perform selected linear algebra calculations very fast. Each board buys 22 Mflops and up to fifteen can be installed in a single 164. A fully configured FPS-164/MAX thus has a theoretical peak performance level of 341 Mflops. A nice overview of the architecture is given in Madura, Broussard and Strickland (1985). We cover just enough of its features to build an appreciation for the type of algorithm that performs efficiently in the MAX environment.

Each MAX module operates synchronously and in parallel with the 164's CPU. The modules appear as memory to the 164's CPU in that they are accessed via reads and writes to reserved addresses. The essential components of a MAX module are two fully pipelined adder/multiplier pairs and eight vector registers (length = 2047). The MAX modules have a limited repertoire of operations but by design they figure heavily in matrix computations. If $n_{\max}$ denotes the number of installed MAX boards then the system is able to perform

$$n_d = 4 + 8 \cdot n_{\max}$$

parallel dot products or

$$n_v = 1 + 4 \cdot n_{\max}$$

parallel SAXPY's as follows:

Parallel Dot Product.

If $V = [v_1, \cdots, v_d] \in R^{m \times d}$ $(d \leqq n_d)$ resides in the MAX vector registers and $w \in R^m$ is stored in main memory, then it is possible to compute the d inner products $v_i^T w$ concurrently. If $d = n_d$ then this means that $V^T w$ can be formed at an approximate rate of $11 + 22 \cdot n_{\max}$ Mflops.

Parallel SAXPY.

If $V = [v_1, \cdots, v_d] \in R^{m \times d}$ $(d \leqq n)$ resides in the MAX vector registers and the vectors $z \in R^d$ and $u \in R^m$ are in main memory, then it is possible to compute the s SAXPY's $v_i := v_i + z_i u$ concurrently. If $d = n_v$ then the computation $V := V + uz^T$ can be formed at the approximate rate of $5 + 11 \cdot n_{\max}$ Mflops.

FPS has extended its ANSI FORTRAN 77 to accommodate a set of subroutines that perform the above computations. This includes a set of routines for executing the required loads and unloads that are responsible for moving data between the MAX modules and main memory. (To those familiar with the FPS-164, table memory essentially acts like a half MAX board. So when we refer to "MAX boards" we really mean "MAX boards plus table memory.")

Because the overhead associated with the loading and unloading of the MAX modules can easily dominate a floating point computation, it is desirable that the vectors loaded into a MAX module be reasonably long and (more critically) that they be "re-used" as much as possible before they are returned to main memory. That way, the cost of a load (or unload) is spread over a significant amount of high speed computation.

With this review of the FPS-164/MAX we can present the MAX version of Algorithm 4.2. In the kth step of the algorithm attention is focussed on the submatrix

$$A(s:m, s:n) = \underset{p}{[\bar{A}_k,} \underset{r}{B}] \quad q$$

where $q = m-(k-1)p$, $r = n-kp$, and $s = 1+(k-1)p$. The block width p is taken to be the maximum number of parallel dot products that can be performed, i.e., $p = n_d$. There are three steps to consider.

Step 1. Compute the QR Factorization of $\bar{A}_k$.

Algorithm 4.1 is used for this purpose. The MAX boards are *not* used here as the algorithm is not rich in matrix multiplication.

Step 2. Compute $Z^T = W^T B$.

Let $B = [b_1, \cdots, b_r]$ be a column partitioning of B. Noting that W entirely fits into the MAX vector registers, $Z \in R^{r\times p}$ is computed as follows:

Load W into the MAX boards.
For $i = 1:r$
  Compute $z(i:i, 1:p) := b_i^T W$ (Parallel Dot Product)
end i

Note that W is reused $r = n-kp$ times. Except at the very end of the algorithm this portion of the computation can proceed at close to peak parallel dot product speed.

Step 3. Overwrite B with $B + YZ^T$.

This is a parallel SAXPY operation. For clarity, assume that s divides r and that $t = r/s$. Partition B, Y, and Z as follows:

$$B = [B_1, \cdots, B_t], \qquad B_i \in R^{q\times s},$$

$$Y = [y_1, \cdots, y_p], \qquad y_i \in R^q,$$

$$Z = \begin{bmatrix} z_{11} & \cdots & z_{1p} \\ \vdots & & \vdots \\ z_{t1} & \cdots & z_{tp} \end{bmatrix}, \qquad z_{ij} \in R^s.$$

With this partitioning the update $B := B + YZ^T$ can be computed as follows:

For $i = 1:t$
  Load B_i into the MAX vector registers.
  For $j = 1:p$
    $B_i := B_i + y_j z_{ji}^T$ (Parallel SAXPY)
  end j
  Unload B_i into main memory.
end i

The re-use factor for this portion of the algorithm is p.

The MAX algorithm just described has been tested on Cornell's FPS-164/MAX system. Currently, this is a one MAX Board installation implying 33 Mflop peak dot product performance and 15 Mflop peak SAXPY performance.

We point out that the fraction of the overall computation that is performed on the MAX boards is $N/(N+2)$ where $N = n/n_d = n/12$. For the various values of n

represented in the table, this fraction ranges from .71 to .97. Of all the MAX computations in our implementation half are parallel dot products and half are parallel SAXPY's. Table 1 indicates the megaflop rates that our code has achieved for problems of varying dimension. An optimized "LINPACK QR" code running on the 164 without MAX boards would perform in the vicinity of 6 Mflops. We feel that these benchmarks confirm that the WY representation is a viable way of organizing Householder computations on the FPS-164/MAX.

TABLE 1
Performance in Megaflops (Mflops).

m	$n=.25m$	$n=.50m$	$n=.75m$	$n=m$
250	7	9	11	11
500	10	13	14	15
750	13	14	16	17
1000	14	15	17	18

6. Other factorizations. Householder matrices are traditionally used in the computation of the following decompositions:

$$Q^H AQ = T \qquad (A \text{ symmetric, } T \text{ tridiagonal}),$$
$$Q^H AQ = H \qquad (H \text{ Hessenberg}),$$
$$U^T AV = B \qquad (B \text{ bidiagonal}).$$

When one contemplates WY versions of these algorithms some new difficulties arise that are not present in the QR factorization. These problems stem from the fact that the above decompositions involve both left and right transformations. This makes "delayed" application of the Householder matrices problematical. In the QR scheme we can reduce a subset of columns without touching the "rest" of the matrix. This is because the Householders are applied from just one side. However, in Householder tridiagonalization (for example) this is not possible. The second Householder P_2 is a function of *all* the matrix elements because it is based on the second column of

$$P_1^T AP_1 = \begin{bmatrix} x & x & 0 & 0 & 0 \\ x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{bmatrix}.$$

In particular, n^2 flops are required to compute this column. This is *not* an order of magnitude less work than performing the entire update $A := P_1^T AP_1$ which costs $2n^2$ flops. Thus, unlike in the QR factorization method, nothing is gained by aggregating Householders and *then* applying them in WY form.

One way out of this jam is to strive for block tridiagonalization. Partition A as follows:

$$A = \begin{bmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} \begin{matrix} p \\ n-p \end{matrix}.$$
$$\qquad\quad p \qquad n-p$$

If we compute the QR factorization

$$A_{21} = QR, \qquad Q = I + WY^T,$$

then update

$$B = Q^T A_{22} Q = (I + WY^T)^T A_{22}(I + WY^T)$$

we obtain

$$A := \text{diag}\,(I_p, Q)^T A \,\text{diag}\,(I_p, Q) = \begin{bmatrix} A_{11} & R^T \\ R & B \end{bmatrix}.$$

This illustrates the basic step of the algorithm. Although it is rich in matrix multiples it leaves us with a bandwidth p eigenproblem. One possibility would be to reduce the resulting matrix to tridiagonal form using the method of Schwartz (1968). The overall success of this procedure in the MAX environment is the next item on our agenda and will be reported elsewhere along with some related procedures for bidiagonalization and Hessenberg reduction.

Appendix. Roundoff properties of the WY representation. In this Appendix we have opted for an "$O(\mathbf{u})$" analysis feeling that complete rigor would add only tedium to an otherwise simple argument. For further comments on the "philosophy" of roundoff analysis, see Golub and Van Loan (1983, p. 32 ff).

Suppose $\mathbf{u}$ is the unit roundoff and A and B are floating point matrices. If $\text{fl}(AB)$ denotes the computed product of A and B then

$$\text{fl}(AB) = AB + E$$

where

$$\|E\|_2 = O(\mathbf{u})\|A\|_2\|B\|_2.$$

Likewise, if A and B are compatible for addition, then

$$\text{fl}(A+B) = A + B + E$$

with

$$\|E\|_2 = O(\mathbf{u})(\|A\|_2 + \|B\|_2).$$

Repeated use will be made of these well-known facts.

We begin with a definition. Suppose W and Y are m-by-k matrices with floating point entries. We say that

$$Q = I + WY^T$$

$\mathbf{u}$-orthogonal if the following three properties hold:

(A1) $$\|W\|_2 = O(1),$$

(A2) $$\|Y\|_2 = O(1),$$

(A3) $$\|Q^TQ - I\|_2 = O(\mathbf{u}).$$

If we apply Q to a floating point matrix A then

$$\text{fl}(QA) = QA + F$$

where

$$\|F\|_2 = O(\mathbf{u})\|A\|_2.$$

Since (A3) implies that the singular values σ_i of Q are all of the form $\sigma_i = 1 + O(\mathbf{u})$, it follows that

(A4) $$\mathrm{fl}(QA) = Q(A + Q^{-1}F) = Q(A+E)$$

where

(A5) $$\|E\|_2 = \|Q^{-1}F\|_2 = O(\mathbf{u})\|A\|_2.$$

Thus, if (A1), (A2), and (A3) hold then manipulations with Q in WY form are stable because it allows for a favorable inverse error analysis.

A corollary of this result is that

$$P = I + uv^T$$

is $\mathbf{u}$-orthogonal provided the floating point vectors v and u satisfy

(A6) $$\|v\|_2 = 1 + O(\mathbf{u})$$

and

(A7) $$u = -2v.$$

This can be used to confirm the stability of conventional Householder matrix manipulations.

What we must show is that when the WY representation is updated (say by Method 1) then properties (A1)-(A3) remain in force. To this end, suppose

$$W_+ = [\,W, \mathrm{fl}(Qu)],$$
$$Y_+ = [\,Y, v],$$
$$Q_+ = I + W_+ Y_+^T$$

where u and v satisfy (A6) and (A7) and (W_+, Y_+) is the computed WY representation of the approximate orthogonal matrix QP. Setting $A = u$ in (A4) and (A5) gives

(A8) $$\mathrm{fl}(Qu) = Qu + e$$

with

(A9) $$\|e\|_2 = O(\mathbf{u})\|u\|_2 = O(\mathbf{u}).$$

Here we used the fact that $\|u\|_2 = 2 + O(\mathbf{u}) = O(1)$. It follows from (A1), (A3), (A8), (A9), and the definition of W_+ that

$$\|W_+\|_2 \leqq \|W\|_2 + \|\mathrm{fl}(Qu)\|_2 = O(1).$$

Likewise, (A2), (A6), and the definition of Y_+ imply that

$$\|Y_+\|_2 \leqq \|Y\|_2 + \|v\|_2 = O(1).$$

All that remains for us to show is that Q_+ satisfies (A3). Using (A8) and the definition of Q_+ we have

$$\begin{aligned} Q_+ &= I + W_+ Y_+^T = I + WY^T + \mathrm{fl}(Qu)v^T \\ &= Q + (Qu + e)v^T = Q(I + uv^T) + ev^T = Q(P + Q^{-1}ev^T). \end{aligned}$$

Using (A9) it is easy to show that if

$$\bar{P} = P + Q^{-1} e v^T$$

then

$$\| \bar{P}^T \bar{P} - I \|_2 = O(\mathbf{u}).$$

This coupled with (A3) ensures that $\| Q_+^T Q_+ - I \|_2 = O(\mathbf{u})$.

We mention that if W and Y are updated using Method 2, then a similar error analysis goes through.

Acknowledgments. We are indebted to Steve Oslon of Floating Point Systems who kindly shared his MAX expertise with us as we developed our code. We are also obliged to Gene Golub of Stanford University for calling our attention to the reference Dietrich (1976).

REFERENCES

G. DIETRICH (1976), *A new formulation of the hypermatrix Householder-QR decomposition*, Comput. Meth. Appl. Mech. Engrg., 9, pp. 273-280.

J. DONGARRA, J. D. BUNCH, C. B. MOLER AND G. W. STEWART (1979), *Linpack User's Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA.

J. DONGARRA AND S. EISENSTAT (1984), *Squeezing the most out of an algorithm in Cray Fortran*, ACM Trans. Math. Software, 10, pp. 216-230.

J. DONGARRA, F. G. GUSTAVSON AND A. KARP (1984), *Implementing linear algebra algorithms on dense matrices on a vector pipeline machine*, SIAM Rev., 26, pp. 91-112.

J. DONGARRA, L. KAUFMAN AND S. HAMMARLING (1985), *Squeezing the most out of eigenvalue solvers on high-performance computers*, Technical Memorandum 46, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439.

G. H. GOLUB AND C. VAN LOAN (1983), *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD.

D. MADURA, R. BROUSSARD AND D. STRICKLAND (1985), FPS-164 MAX: *parallel multiprocessing for linear algebra operation*, Proc. 1985 Array Processing Conference, New Orleans, pp. 33-50.

H. R. SCHWARTZ (1968), *Tridiagonalization of a symmetric band matrix*, Numer. Math., 12, pp. 231-241.

B. T. SMITH, J. M. BOYLE, Y. IKEBE, V. C. KLEMA AND C. B. MOLER (1970), *Matrix Eigensystem Routines*: EISPACK *Guide*, 2nd ed., Springer-Verlag, New York.

G. W. STEWART (1983) *An algorithm for computing the* CS *decomposition of a partitioned orthonormal matrix*, Numer. Math., 40, pp. 297-306.

C. VAN LOAN (1985), *Computing the* CS *and the generalized singular value decomposition*, Numer. Math., 46, 479-491.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 1, January 1987
© 1987 Society for Industrial and Applied Mathematics

A DOMAIN-DECOMPOSED FAST POISSON SOLVER ON A RECTANGLE*

TONY F. CHAN† AND DIANA C. RESASCO†

Abstract. We present a new domain decomposed fast Poisson solver on a rectangle divided into parallel strips or boxes. The method first performs uncoupled fast solves on each subdomain, and then the interface variables are computed exactly by Fast Fourier Transform, without computing or inverting the capacitance matrix explicitly. Finally, the solution on the interior of the subdomains can be computed by one more fast solve on each subdomain. This method is especially suited for parallel implementation, since the independent problems in the subdomains can be solved in parallel, and the communication involves the interface variables only.

Key words. domain decomposition, substructuring, capacitance matrix, elliptic partial differential equations, preconditioning, fast Poisson solvers, parallel computation

AMS(MOS) subject classification. 65

1. Introduction. We consider the solution of the Poisson equation in a rectangular region partitioned into strips. By the method of Domain Decomposition, the solution of the algebraic equations resulting from the discretization on a regular grid is reduced to the solution of problems in the subdomains and a linear system for the interface unknowns, given by the *capacitance matrix.* This is an important tool in the solution of elliptic partial differential equations. There are several reasons why these techniques might be attractive:

- The method is suited for the solution of very large problems on machines with limited storage.
- Special solution techniques might exist to solve the problems on the subdomains that cannot be applied efficiently to the entire domain. This is often the case, for example, when the domain has irregular geometry, but it can be broken up into regular subdomains, like rectangles.
- The equations in the different subdomains might have different parameters or even be of different nature, in which case the idea of substructuring comes very naturally.
- The idea is attractive for parallel processing, since the problem can be decoupled in independent subproblems and the communication needed will be only for the interface values.

The capacitance matrix is expensive to compute or invert explicitly, while it is relatively simple to compute the product of such a matrix with an arbitrary vector. For that reason the interface system is usually solved by conjugate gradient methods instead of a direct method. In order to improve the convergence of the method, several preconditioning techniques have been given in the literature [7], [8], [2], [3].

For the problem considered in this paper, we present a fast direct method for the solution of the capacitance matrix that does not require the computation of the elements of the matrix. This method is related to the k-reduction method, a technique proposed by Bank and Rose [1] in order to improve the stability of the marching algorithm. In

*Received by the editors August 26, 1985; accepted for publication (in revised form) March 7, 1986. This work was supported in part by the Department of Energy under contract DE-AC02-81ER10996 and by a BID-CONICET fellowship from Argentina.

† Department of Computer Science, Research Center for Scientific Computing, Yale University, New Haven, Connecticut 06520.

fact, the k-reduction method can be interpreted as a particular implementation of the domain decomposition algorithm, where the number of strips is chosen to fulfill the stability constraints. In the case of uniform sized strips, the eigenvectors and eigenvalues of the capacitance matrix can be derived explicitly. This gives not only a direct method to solve the interface system, but also a framework to analyse other preconditioners.

In § 2, we apply the method of Domain Decomposition to the solution of the Poisson equation in a rectangular domain subdivided into strips, and derive a system for the interface unknowns and the capacitance matrix. In § 3, we analyze the capacitance matrix, giving its eigenvectors and eigenvalues. Based on this, we derive a fast direct method for the solution of the interface system. The eigen-decomposition of the capacitance matrix also provides a clear way of analyzing some other preconditioners. In § 4, we analyze the preconditioners given by Golub and Mayers, Dryja and Proskurowski, and Bjorstad and Widlund. Finally, in §§ 5 and 6, we discuss extensions of the method to rectangular domains subdivided into boxes, and more general variable coefficient operators and summarize the results, giving some concluding remarks.

2. Domain decomposition. We consider the Poisson equation

$$\Delta u = f \quad \text{on } \Omega \tag{2.1}$$

with boundary conditions

$$u = u_b \quad \text{on } \partial\Omega$$

where the domain Ω is as illustrated in Fig. 1.

We partition Ω into $k+1$ strips Ω_i, $i=1, \cdots, k$ of sizes 1 by l_i and denote the interfaces between $\mathcal{N}_i$ and Ω_{i+1}, by Γ_i, $i=1, \cdots, k$. We use a uniform mesh with grid

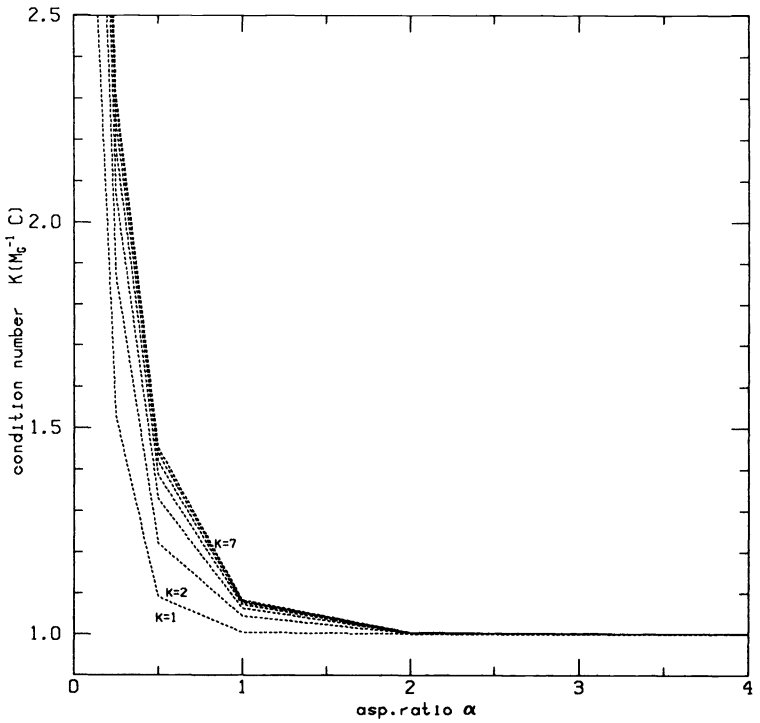


FIG. 1. *The domain* Ω *and its partition.*

size h on Ω with n internal grid points in the x-direction, i.e.

$$h = \frac{1}{n+1}$$

and m_i internal grid points in Ω_i in the y-direction, i.e. for $i = 1, \cdots, k+1$,

$$l_i = (m_i + 1)h.$$

Let us consider a standard 5-point centered difference approximation to (2.1). If we order the unknowns for the internal points of the subdomains first and then those in the interfaces Γ_i, then the discrete solution vector $u = (u_\Omega, u_\Gamma)$ satisfies a linear system that can be expressed in block form as

$$\begin{pmatrix} Q_\Omega & P \\ P^T & Q_\Gamma \end{pmatrix} \begin{pmatrix} u_\Omega \\ u_\Gamma \end{pmatrix} = \begin{pmatrix} f_\Omega \\ f_\Gamma \end{pmatrix}, \tag{2.2}$$

where the right-hand side depends on f and u_b and

$$Q_\Omega = \begin{pmatrix} L_1 & & & \\ & L_2 & & \\ & & \ddots & \\ & & & L_{k+1} \end{pmatrix}, \tag{2.3}$$

$$Q_\Gamma = \begin{pmatrix} L_{k+2} & & & \\ & & \ddots & \\ & & & L_{2k+1} \end{pmatrix} \tag{2.4}$$

where L_i corresponds to the discrete Laplacian on Ω_i for $i \leqq k+1$, and L_i for $i > k+1$ is the tridiagonal matrix tridiag$(1, -4, 1)$. The matrix P has the following block bi-diagonal form:

$$P = \begin{pmatrix} P_{1,1} & & & & \\ P_{2,1} & P_{2,2} & & & \\ & & \ddots & & \\ & & P_{k,k-1} & P_{k,k} & \\ & & & P_{k+1,k} & \end{pmatrix}, \tag{2.5}$$

where P_{ij} corresponds to the coupling between the unknowns in the subdomain Ω_i with those in the interface Γ_j.

By applying block Gaussian elimination to (2.2), we obtain the following system for the interface unknowns:

$$Cu_\Gamma = g \tag{2.6}$$

where

$$g \equiv f_\Gamma - P^T Q_\Omega^{-1} f_\Omega, \tag{2.7}$$

and

$$C \equiv Q_\Gamma - P^T Q_\Omega^{-1} P. \tag{2.8}$$

C is sometimes called the *capacitance matrix*. Since Q_Ω is block diagonal, the right-hand side of (2.6) given by (2.7) can be evaluated by solving a problem like (2.1) on each subdomain Ω_i, with zero Dirichlet boundary conditions on the interfaces Γ_{i-1} and Γ_i,

$i=1,\cdots,k+1$. Once (2.6) is solved, the problem is decoupled and the solution u_{Ω_i} at the subdomains can be computed by solving $k+1$ independent subproblems:

$$L_i u_{\Omega i} = f_{\Omega i} - P_{i,i-1} u_{\Gamma i-1} - P_{i,i} u_{\Gamma i}$$

for $i=1,\cdots,k+1$. This is nothing more than solving for $u_{\Omega i}$ on each subdomain Ω_i with the computed $u_{\Gamma i-1}$ and $u_{\Gamma i}$ as boundary conditions.

This technique of reducing the problem on Ω to the solution of decoupled problems on the subdomains and a smaller system for the interface is usually called *domain decomposition* or *substructuring.*

3. The capacitance system. By substituting (2.3), (2.4) and (2.5) in (2.8), we can see that the matrix C has the following block tridiagonal form:

$$C = \begin{pmatrix} C_1 & B_2 & & \\ B_2 & C_2 & \ddots & \\ & \ddots & \ddots & B_k \\ & & B_k & C_k \end{pmatrix}, \tag{3.1}$$

where C_i is the capacitance matrix corresponding to the interface Γ_i, i.e.

$$C_i = L_{k+1+i} - P_{i,i}^T L_i^{-1} P_{i,i} - P_{i+1,i}^T L_{i+1}^{-1} P_{i+1,i}, \tag{3.2}$$

and

$$B_i = -P_{i,i-1}^T L_i^{-1} P_{i,i}. \tag{3.3}$$

The matrix C is expensive to compute explicitly, since the computation of the blocks C_i and B_i requires the solution of $2n$ subproblems on each Ω_i. Also, the resulting blocks are dense n by n matrices, and therefore, the solution to (2.6) by Gaussian elimination would require $\mathcal{O}(n^3)$ operations, which may overcome the $\mathcal{O}(n^2 \log(n))$ complexity of the fast solvers for the subdomains, when $m \approx n$.

Instead of using a direct method to solve the system (2.6), preconditioned conjugate gradient (PCG) methods can be applied, where only matrix-vector products of the form Cw for a given vector w are needed. From (3.2), we can see that the evaluation of $C_i w$ for each i requires the solution of two subdomain problems. For example, the product $-P_{2,2}^T L_2^{-1} P_{2,2} w$ can be computed by solving the discretized version of (2.1) on Ω_2 with homogeneous right-hand side (i.e., the Laplace equation) and boundary conditions $u = w$ on Γ_2, $u = 0$ on the rest of the boundary, and then taking the solution on the first row of grid points above Γ_2. Similarly, $B_2 w$ corresponds to taking the same solution on the first row below Γ_1.

Each iteration of the PCG method requires the solution of at least one problem at each subdomain. Therefore, it is important to keep the number of iterations low. A number of preconditioners for the matrix C have been given in the literature in order to improve the convergence [7], [8], [2], [3].

It turns out that for the problem we are considering, an exact decomposition for the capacitance matrix is possible. This fact was first pointed out by Dryja [7], who noticed that, in the uniform strip case, the same matrix C appears in the context of the Generalized Marching Algorithm in [1], where a decomposition of C is given in terms of Chebyshev polynomials. We give a different derivation of the eigenvalues and eigenvectors of the blocks C_i and B_i in (3.1) based on an extension of the two strips case given in [5]. The system (2.6) can then be solved by Fast Fourier Transforms or cyclic reduction [4], [9].

We will show that the matrices C_i and B_i in (3.1) have the same eigenvectors. Let us define the orthogonal matrix W as the matrix whose columns are

$$w_j = \sqrt{\frac{2}{n+1}}\,(\sin j\pi h, \sin 2j\pi h, \cdots, \sin nj\pi h)^T \tag{3.4}$$

for $j = 1, \cdots, n$,

$$\sigma_j = 4 \sin \frac{2j\pi h^2}{2}, \tag{3.5}$$

and

$$\gamma_j = \left(1 + \frac{\sigma_j}{2} + \sqrt{\sigma_j + \frac{\sigma_j^2}{4}}\right)^2. \tag{3.6}$$

Then, we have the following

LEMMA 3.1. *The matrices C_i and B_i have the same eigenvectors* (3.4). *For $i = 1, \cdots, k$, we have*

$$W^T C_i W = \Lambda_i = \operatorname{diag}(\lambda_{i1}, \cdots, \lambda_{in}), \tag{3.7}$$

and for $i = 2, \cdots, k$, we have

$$W^T B_i W = D_i = \operatorname{diag}(\delta_{i1}, \cdots, \delta_{in}) \tag{3.8}$$

where

$$\lambda_{ij} = -\left(\frac{1+\gamma_j^{m_i+1}}{1-\gamma_j^{m_i+1}} + \frac{1+\gamma_j^{m_{i+1}+1}}{1-\gamma_j^{m_{i+1}+1}}\right)\sqrt{\sigma_j + \frac{\sigma_j^2}{4}}, \tag{3.9}$$

and

$$\delta_{ij} = \sqrt{\gamma_j^{m_i}}\left(\frac{1-\gamma_j}{1-\gamma_j^{m_i+1}}\right). \tag{3.10}$$

Proof. We will compute the three terms of $C_i w_j$ in (3.2) separately. As we stated before, the product $-P_{i,i}^T L_i^{-1} P_{i,i} w_j$ can be computed by solving the problem

$$\begin{aligned} &\Delta_h v = 0 && \text{in } \Omega_i, \\ &v = 0 && \text{on } \partial\Omega_i \cap \partial\Omega, \\ &v = 0 && \text{on } \Gamma_{i-1}, \\ &v = w_j && \text{on } \Gamma_i, \end{aligned} \tag{3.11}$$

and taking the values of the solution at the first row of grid points above Γ_i. By using separation of variables, it can be shown [5] that the solution to (3.11) at each grid point (s, t) is given by:

$$v(sh, th) = (c_1 r_+^t + c_2 r_-^t) \sin sj\pi h \tag{3.12}$$

where r_+ and r_- are the roots of the characteristic polynomial corresponding to substituting (3.12) in (3.11), namely,

$$\begin{aligned} r_+ &= 1 + \frac{\sigma_j}{2} + \sqrt{\sigma_j + \frac{\sigma_j^2}{4}}, \\ r_- &= 1 + \frac{\sigma_j}{2} - \sqrt{\sigma_j + \frac{\sigma_j^2}{4}}, \end{aligned} \tag{3.13}$$

with σ_j given by (3.5). The constants c_1 and c_2 are determined by the boundary conditions and they are given by

$$c_1 = -\frac{r_-^{m_1+1}}{r_+^{m_1+1} - r_-^{m_1+1}}, \qquad c_2 = \frac{r_+^{m_1+1}}{r_+^{m_1+1} - r_-^{m_1+1}}.$$

Therefore, we can compute the product $-P_{i,i}^T L_i^{-1} P_{i,i} w_j$ by taking $t = 1$ in (3.12), i.e.

$$-P_{i,i}^T L_i^{-1} P_{i,i} w_j = \left(\frac{r_- - r_+ \gamma^{m_i+1}}{1 - \gamma^{m_i+1}} \right) w_j \tag{3.14}$$

where

$$\gamma_j = \frac{r_-}{r_+}. \tag{3.15}$$

Since $r_+ r_- = 1$, $\gamma_j = r_-^2$ and, therefore, (3.6) and (3.15) are equivalent. By a similar computation we can prove that

$$-P_{i+1,i}^T L_{i+1}^{-1} P_{i+1,i} w_j = \left(\frac{r_- - r_+ \gamma_j^{m_{i+1}+1}}{1 - \gamma_j^{m_{i+1}+1}} \right) w_j. \tag{3.16}$$

Finally, it can also be verified that

$$L_{k+1+i} w_j = (-2 - \sigma_j) w_j. \tag{3.17}$$

Combining (3.14), (3.16) and (3.17), we have

$$C_i w_j = \lambda_{ij} w_j$$

where

$$\lambda_{ij} = -2 - \sigma_j + \left(\frac{r_- - r + \gamma_j^{m_1+1}}{1 - \gamma_j^{m_1+1}} \right) + \left(\frac{r_- - r_+ \gamma_j^{m_2+1}}{1 - \gamma_j^{m_2-1}} \right),$$

which, after some simplifications, leads to the expression (3.9).

By similar arguments, $B_i w_j$ requires the computation of the solution to (3.11) at the m_ith row away from Γ_i and we can prove that:

$$B_i w_j = v(\cdot, m_i h) = -\delta_{ij} w_j$$

where

$$\begin{aligned} \delta_{ij} &= c_1 r_+^{m_i} + c_2 r_-^{m_i} \\ &= \sqrt{\gamma_j^{m_i}} \left(\frac{1 - \gamma_j}{1 - \gamma_j^{m_i+1}} \right). \end{aligned}$$

□

Let us partition the vectors u_Γ and g in (2.6) as:

$$u_\Gamma = \begin{pmatrix} u_{\Gamma_1} \\ u_{\Gamma_2} \\ \vdots \\ u_{\Gamma_k} \end{pmatrix}, \qquad g = \begin{pmatrix} g_{\Gamma_1} \\ g_{\Gamma_2} \\ \vdots \\ g_{\Gamma_k} \end{pmatrix}$$

where, for $i = 1, \cdots, k$,

$$u_{\Gamma_i} = \begin{pmatrix} u_{1i} \\ u_{2i} \\ \vdots \\ u_{ni} \end{pmatrix}, \qquad g_{\Gamma_i} = \begin{pmatrix} g_{1i} \\ g_{2i} \\ \vdots \\ g_{ni} \end{pmatrix}.$$

The system (2.6) may be written

$$
\begin{aligned}
&C_1 u_{\Gamma_1} + B_2 u_{\Gamma_2} = g_{\Gamma_1}, \\
&B_i u_{\Gamma_{i-1}} + C_i u_{\Gamma_i} + B_{i+1} u_{\Gamma_{i+1}} = g_{\Gamma_i}, \qquad i = 2, \cdots, k-1, \\
&B_k u_{\Gamma_{k-1}} + C_k u_{\Gamma_k} = g_{\Gamma_k}.
\end{aligned} \tag{3.18}
$$

Using (3.7) and (3.8), the system (3.18) becomes

$$
\begin{pmatrix}
\Lambda_1 & D_2 & & \\
D_2 & \ddots & \ddots & \\
 & \ddots & \ddots & D_k \\
 & & D_k & \Lambda_k
\end{pmatrix}
\begin{pmatrix}
\hat{u}_{\Gamma_1} \\ \hat{u}_{\Gamma_2} \\ \vdots \\ \hat{u}_{\Gamma_k}
\end{pmatrix}
=
\begin{pmatrix}
\hat{g}_{\Gamma_1} \\ \hat{g}_{\Gamma_2} \\ \vdots \\ \hat{g}_{\Gamma_k}
\end{pmatrix} \tag{3.19}
$$

where, for $i \leqq k$,

$$
\hat{g}_{\Gamma_i} = W^T g_{\Gamma_i}, \tag{3.20}
$$

and

$$
u_{\Gamma_i} = W \hat{u}_{\Gamma_i}. \tag{3.21}
$$

By reordering the equations in (3.19), the system can be reduced to the solution of n tridiagonal systems of dimension k and u_Γ can then be computed by the expression (3.21). Note that Fast Fourier Transforms can be used in computing $\hat{g}$ and u_Γ. An outline of the algorithm follows:

ALGORITHM DDFPS (Domain-Decomposed Fast Poisson Solver)

Step 1: Compute right-hand side g by (2.7). This requires the solution of $k+1$ decoupled problems of the form (2.1) on Ω_i, $i = 1, \cdots, k+1$, namely

$$
\begin{aligned}
&\Delta_h v_i = f \quad \text{in } \Omega_i, \\
&v_i = u_{b_i} \quad \text{on } \partial\Omega_i \cap \partial\Omega, \\
&v_i = 0 \quad \text{on } \Gamma_{i-1}, \\
&v_i = 0 \quad \text{on } \Gamma_i,
\end{aligned}
$$

and then compute

$$
g_{\Gamma_i} = f_{\Gamma_i} - v_i(\cdot, m_i) - v_{i+1}(\cdot, 1).
$$

Step 2: Obtain new right-hand side $\hat{g}$ by (3.20) using Fast Fourier Transforms.

Step 3: For $j = 1, \cdots, n$, solve

$$
\begin{pmatrix}
\lambda_{1j} & \delta_{2j} & & \\
\delta_{2j} & \ddots & \ddots & \\
 & \ddots & \ddots & \delta_{kj} \\
 & & \delta_{kj} & \lambda_{kj}
\end{pmatrix}
\begin{pmatrix}
\hat{u}_{j1} \\ \hat{u}_{j2} \\ \vdots \\ \hat{u}_{jk}
\end{pmatrix}
=
\begin{pmatrix}
\hat{g}_{j1} \\ \hat{g}_{j2} \\ \vdots \\ \hat{g}_{jk}
\end{pmatrix}. \tag{3.22}
$$

Step 4: Fast Fourier Transform the resulting $\hat{u}_{\Gamma_i}$'s to obtain u_{Γ_i} by (3.21).

Step 5: Solve the following problem in each subdomain

$$
\Delta_h u_i = f \quad \text{on } \Omega_i
$$

with boundary conditions

$$u_i = u_{b_i} \quad \text{on } \partial\Omega_i \cap \partial\Omega,$$
$$u_i = u_{\Gamma_{i-1}} \quad \text{on } \Gamma_{i-1},$$
$$u_i = u_{\Gamma_i} \quad \text{on } \Gamma_i.$$

3.1. Uniform size strips. When $m_i = m$ for all i, i.e. all strips have identical dimensions, all the blocks C_i in (3.1) are identical, as well as the blocks B_i. In this case, the eigenvalues and eigenvectors of C can be explicitly computed.

LEMMA 3.2. *If $m_i \equiv m$, $1 \leqq i \leqq k+1$, then the capacitance matrix has the form*

$$C = \begin{pmatrix} C_1 & B & & \\ B & C_1 & \ddots & \\ & \ddots & \ddots & B \\ & & B & C_1 \end{pmatrix},$$

and the eigenvalues of C are given by

$$\alpha_{ij} = \lambda_j + \delta_j \cos\frac{i\pi}{k+1}$$

for $i = 1, \cdots k$ and $j = 1, \cdots n$, where λ_j are the eigenvalues of C_1 and δ_j are the eigenvalues of B. The eigenvectors v_{ij} are the direct product of the vectors w_j defined in (3.4) *and*

$$z_i = \left(\sin\frac{\pi i}{k+1}, \sin 2\frac{\pi i}{k+1}, \cdots, \sin k\frac{\pi i}{k+1}\right)^T,$$

i.e.,

$$v_{ij} = z_i \otimes w_j = \begin{pmatrix} \left(\sin\dfrac{\pi i}{k+1}\right) w_j \\ \left(\sin\dfrac{2\pi i}{k+1}\right) w_j \\ \vdots \\ \left(\sin\dfrac{k\pi i}{k+1}\right) w_j \end{pmatrix}.$$

Proof. It can be verified that

$$Cv_{ij} = \alpha_{ij} v_{ij}$$

by simple substitution and using the fact that z_i and α_{ij} are the eigenvectors and eigenvalues of the tridiagonal matrix:

$$T_j = \begin{pmatrix} \lambda_j & \delta_j & & \\ \delta_k & \ddots & \ddots & \\ & \ddots & \ddots & \delta_j \\ & & \delta_j & \lambda_j \end{pmatrix}.$$

□

As an improvement over the plain Fourier Analysis method, which has complexity $\mathcal{O}(kn \log_2 n)$, block-cyclic reduction techniques can be combined with Fast Fourier Transforms to get an algorithm for solving the interface system for the regular sized strips case, which has asymptotic complexity $\mathcal{O}(kn \log_2 \log_2 n)$ [4], [10].

4. Analysis of some preconditioners. For more general operators and domains, the system (3.1) is solved by the preconditioned conjugate gradient method. Several preconditioners have been proposed in order to improve the convergence of the method. For the case where there is only one interface ($k=1$), Dryja [7] proposed as preconditioner for C the square root of the one-dimensional discrete Laplacian. This matrix, which will be denoted by M_D, can be inverted by Fast Fourier Transform, since it has the following decomposition:

$$M_D = W \operatorname{diag}(\lambda_1^D, \lambda_2^D, \cdots, \lambda_n^D) W^T \tag{4.1}$$

where the columns of W are given by (3.4) and

$$\lambda_j^D = -2\sqrt{\sigma_j} \tag{4.2}$$

with σ_j given by (3.5). Golub and Mayers [8] improve Dryja's results with a preconditioner given by

$$M_G = W \operatorname{diag}(\lambda_1^G, \lambda_2^G, \cdots, \lambda_n^G) W^T \tag{4.3}$$

where

$$\lambda_j^G \equiv -2\sqrt{\sigma_j + \frac{\sigma_j^2}{4}}. \tag{4.4}$$

For the two strips case, Bjorstad and Widlund [2] give the following approximation to C as a preconditioner:

$$M_B = L_3 - 2P_{1,1}^T L_1^{-1} P_{1,1}.$$

When the two strips have the same dimensions, this preconditioner is exact, although their method involves an extra solve in the subdomain Ω_1 in order to invert M_B, as opposed to solving by only one Fast Fourier Transform on the interface.

In [6], Dryja and Proskurowski propose an algorithm for the case of multiple strips. By ignoring the off-diagonal blocks B_i in (3.1) and using the preconditioners for the two-strips case on the diagonal, they propose using either the block-diagonal matrix diag (M_D) or diag (M_G).

Using the results of the previous section, we can analyze these preconditioners. It is a well-known fact that the rate of convergence of the preconditioned conjugate gradient method depends on the condition number of the matrix $M^{-1}C$, where M is the preconditioner, being close to one. We will use the 2-norm condition number in this paper, and for simplicity, we will only consider the case of regular sized strips, with $m_i = m$. In [5], Chan shows for the two-strips case that $K(M_G^{-1}C)$ and $K(M_G^{-1}C)$ depend on the aspect ratio

$$\alpha = \frac{m+1}{n+1} \tag{4.5}$$

and in particular,

$$\lim_{h\to 0} K(M_G^{-1}C) = \frac{1+e^{-2\pi\alpha}}{1-e^{-2\pi\alpha}}. \tag{4.6}$$

The asymptotic expression (4.6) is plotted in Fig. 2. As we can see, the condition number will be large for small α.

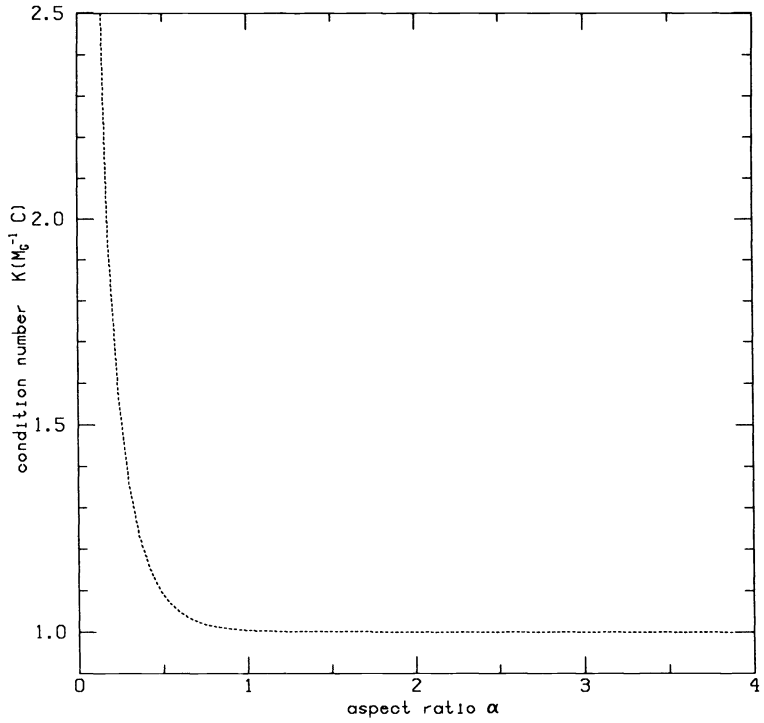


FIG. 2. *Condition number of the preconditioned capacitance matrix with Golub and Mayers' preconditioner (asymptote for $h \to 0$).*

For the case of multiple strips, we have to analyze the effect of dropping the blocks B_i's. Note that when the eigenvalues of B_i, namely δ_j, are small compared to λ_j, the system (3.22) becomes diagonal and therefore, it can be trivially solved. We will see that the values of δ_j are small compared to λ_j when the aspect ratio (4.5) is large. This can be derived from Lemma 3.2, where we give an asymptotic expression for the ratio δ_j/λ_j in terms of the aspect ratio α when $h \to 0$.

By computing (3.10) and (3.9) for the regular strips case, we can see that

$$\frac{\delta_j}{\lambda_j} = \frac{\gamma_j^{(m+1)/2}}{1+\gamma_j^{m+1}}, \tag{4.7}$$

which is a decreasing function of j. Therefore,

$$\frac{\delta_1}{\lambda_1} = \max_{1 \le j \le n} \frac{\delta_j}{\lambda_j}.$$

Moreover, we have:

LEMMA 4.1.

$$\lim_{h \to 0} \frac{\delta_1}{\lambda_1} = \frac{1}{e^{\alpha\pi} + e^{-\alpha\pi}}.$$

Proof. Since $\sigma_1 \to 0$ when $h \to 0$, we have

$$\lim_{h \to 0} (\log \gamma_1^{m+1}) = \alpha \lim_{h \to 0} \frac{1}{\gamma_1} \frac{d\gamma_1}{dh} = -2\alpha\pi.$$

Therefore,

$$\lim_{h\to 0} \gamma_1^{m+1} = e^{-2\alpha\pi}. \tag{4.8}$$

By (4.7) and (4.8), we have

$$\lim_{h\to 0} \frac{\delta_1}{\lambda_1} = \frac{e^{-\alpha\pi}}{1+e^{-2\alpha\pi}} = \frac{1}{e^{\alpha\pi}+e^{-\alpha\pi}}. \qquad \square$$

When α is large enough, the δ_j's can be ignored, so that the system (3.1) becomes block-diagonal and moreover, for α large, the eigenvalues λ_{ij} approach (4.4) and in that case, (4.3) is a good approximation to C. For α small, the condition number of the preconditioned capacitance matrix becomes large. Figure 3 shows the dependence of the condition number on the aspect ratio α for different numbers of strips.

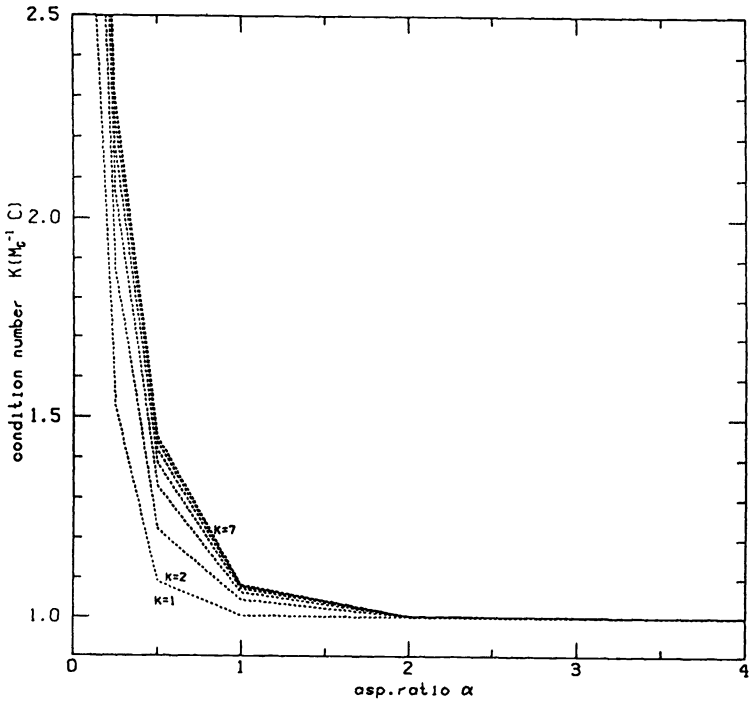


FIG. 3. *Condition number of the preconditioned capacitance matrix with Golub and Mayers' preconditioner for two, three, and up to eight strips (k is the number of interfaces).*

5. Dividing the domain into boxes. In the parallel implementation of domain decomposition, the ratio between the amount of communication and the amount of computation required grows as the strips become narrower. For this reason, it seems natural to divide the domain into boxes.

Boxes introduce a new difficulty, the cross points. Cross points are more difficult to handle, since they represent a strong coupling between interfaces. We refer the interested reader to [3].

One way around cross points is to use algorithm DDFPS recursively. We can, for example, apply algorithm DDFPS to the domain Ω divided into horizontal strips Ω_i, and then solve the subproblems in Steps 1 and 5 by subdividing each strip Ω_i in vertical strips Ω_{ij}. Another approach is given by nested dissection, which can be described

recursively as follows: the domain Ω is divided in two strips Ω_1 and Ω_2. We will call this partition P_1, and given a partition P_i, P_{i+1} is obtained by subdividing each subdomain of P_i into two (vertical or horizontal) strips. An example of the partition procedure is given in Fig. 4. Algorithm DDFPS with $k=1$ is applied to P_1, and it is applied recursively in Steps 1 and 5 to solve for the problems in the subdomains. Conventional fast solvers are used for the subproblems in the finest partition.

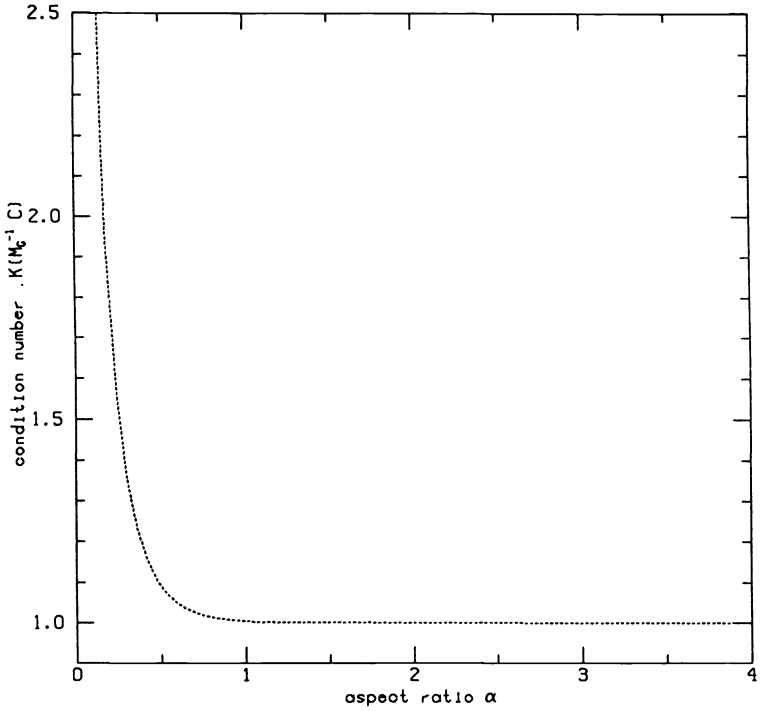


FIG. 4. *An example of nested dissection.*

6. Concluding remarks. We present a fast direct Poisson solver on a rectangle divided into parallel strips or boxes. The method is especially suited for parallel implementation, since the independent problems in the subdomains can be solved in parallel, and the communication can be reduced to k vectors of length n, where n is the number of points in the interface.

Since the method is based on the fact that C can be solved by FFT's, it can be generalized to other separable operators with coefficients which are constant in the x or y direction, and other boundary conditions. Similar arguments can be used to derive a fast solver for the Poisson equation in three dimensions. The expressions obtained for the corresponding eigenvalues λ's and δ's show that the preconditioners $\sqrt{K}$ and $\sqrt{K^2/4+K}$ have natural extensions to the three-dimensional case, where K is now the two-dimensional Laplacian operator. These results will be presented in a forthcoming report.

It appears that domain decomposition requires more computation than a conventional fast solver on the whole domain, since the algorithm requires two solves on each subdomain, namely Steps 1 and 5. Many operations can be saved, however, since only one row of the solution is needed in Step 1 for the computation of the right-hand side (2.7). Also, some of the computations in Step 1 can be saved for Step 5, since the right-hand side for the subproblems in Steps 1 and 5 differ only at one or two rows

of the grid points. With these savings, algorithm DDFPS can be made computationally equivalent to the fast Poisson solver on a rectangle using FFT's in the x-direction and solving n tridiagonal systems in the y-direction.

Acknowledgment. The authors would like to thank Professor M. Dryja for pointing out the relationship between the generalized marching algorithms of Bank and Rose and the fast domain decomposed fast Poisson solver presented in this paper.

REFERENCES

[1] R. BANK AND D. ROSE, *Marching algorithms for elliptic boundary value problems.* I: *The constant coefficient case*, SIAM J. Numer. Anal., 14 (1977), pp. 792-828.

[2] P. BJORSTAD AND O. WIDLUND, *On some trends in elliptic problem solvers*, in Elliptic Problem Solvers, G. Birkhoff and A. Schoenstadt, eds., Academic Press, New York, 1984.

[3] J. H. BRAMBLE, J. E. PASCIAK AND A. H. SCHATZ, *Preconditioners for interface problems on mesh domains*, Math. Comp., to appear.

[4] 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.

[5] T. F. CHAN, *Analysis of preconditioners for domain decomposition*, SIAM J. Numer. Anal. (1986), to appear.

[6] M. DRYJA AND W. PROSKUROWSKI, *Capacitance matrix method using strips with alternating Neumann and Dirichlet boundary conditions*, in Advances in Computer Methods for Partial Differential Equations - V, R. Vichnevetsky and R. S. Stepleman, eds., IMACS, 1984, pp. 360-368.

[7] M. DRYJA, *A capacitance matrix method for Dirichlet problem on polygonal region*, Numer. Math., 39 (1982), pp. 51-64.

[8] G. H. GOLUB AND D. MAYERS, *The use of pre-conditioning over irregular regions*, 1983. Lecture at Sixth International Conference on Computing Methods in Applied Sciences and Engineering, Versailles, France, December 1983.

[9] P. N. SWARZTRAUBER, *A direct method for the discrete solution of separable elliptic equations*, SIAM J. Numer. Anal., 11 (1974), pp. 1136-1150.

[10] ———, *A direct method for the discrete solution of separable elliptic equations*, SIAM Rev., 19 (1976), pp. 490-501.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 1, January 1987

MULTIPROCESSOR FFT METHODS*

WILLIAM L. BRIGGS†, LESLIE B. HART†, ROLAND A. SWEET† AND ABBIE O'GALLAGHER‡

Abstract. Proceeding first by experiment and then by analysis, the problem of implementing FFT algorithms on a shared memory multiprocessor is investigated. Several algorithms for performing a single FFT and multiple FFTs are implemented and compared on the Denelcor HEP computer. These algorithms are then analyzed using performance models which reproduce the experimental timing curves. For both the single and multiple FFT a clear choice of superior algorithms can be made. These algorithms are expected to be useful and to preserve their performance characteristics on other shared memory multiprocessors.

Key words. fast Fourier transform multiple FFTs, multiprocessor

AMS(MOS) subject classifications. 65N30, 68BO5

1. Introduction. The fast Fourier transform (FFT) is certainly one of the most widely used and thoroughly studied numerical algorithms in existence. Among its many uses in science, engineering, and applied mathematics are signal and image processing, time series and spectral analysis, and the solution of partial differential equations such as Poisson's equation by fast direct methods. Sequential FFT algorithms (those for conventional, serial computers) have been highly refined during the twenty years since the FFT was first introduced [4]. The early FFT papers [1], [2], [3], [6] offer good reading from an historical point of view and reflect both the variety of FFT algorithms and the different ways in which they may be developed.

As computers have evolved, there has been a corresponding adaptation of FFT algorithms to fit the new architectures. For example, the appearance of vector computers led to new implementations of the FFT [10], [13]-[16], [19]. The FFT proved to be highly amenable to vector or SIMD (single instruction stream-multiple data stream) computation and significant improvements in performance have been realized for this class of computers. Now, as multiprocessing and distributed computing systems become available, there is a continued interest in the FFT and its implementation on this class of MIMD (multiple instruction stream-multiple data stream) computers. This paper is a part of this latter effort, namely the study of FFT algorithms on computers which possess many processors which are capable of working independently on different sets of data. As will be seen, the inherent parallelism of the FFT can be exploited on MIMD computers, but often in very different ways than it is on vector computers.

All computations described in this study were carried out on the Denelcor HEP-1 (Heterogeneous Element Processor) computer. Briefly, the HEP is a shared memory MIMD computer in which both the arithmetic units and the instruction stream are pipelined. The one PEM (Process Execution Module) system which was used for these computations has a maximum execution rate of 10 million instructions per second and can support up to 50 independent user-created processes [11]. All programs were written in a parallel version of FORTRAN which allows for inter-process communication through the use of semaphored (or asynchronous) variables. The HEP has a

* Received by the editors August 19, 1985; accepted for publication (in revised form) March 12, 1986. This work was supported by National Bureau of Standards Cooperative Agreement 70NANB4H0008 and National Science Foundation grant DMS-8504350.

† Mathematics Department, University of Colorado, Denver, Colorado 80202.

‡ Scientific Computing Division, National Bureau of Standards, Boulder, Colorado 80202. (Contribution of the National Bureau of Standards. Not subject to copyright in the U.S.A.)

relatively low degree of parallelism, particularly when compared to vector computers and many array processors. For this reason, its parallelism is best used on a rather coarse level. As opposed to many vector algorithms, parallelism on the HEP (which has no vector capabilities) is generally introduced on the outermost levels of an algorithm. This principle will be applied throughout this study.

The demise of Denelcor as a commercial venture does not diminish the many novel features of the HEP. It is a highly versatile, general purpose computer and has served well as a laboratory for parallel algorithm development. While it is only one expression of shared memory MIMD architecture, it is viewed as a representative of many shared memory or tightly coupled systems. Although the algorithms of this paper were developed for the HEP, the ideas underlying these algorithms and much of their structure should be directly transferable to other multiprocessor environments. It is our intent to minimize HEP specificity as much as possible and to draw some general conclusions about FFTs on shared memory MIMD systems in general.

This study proceeds first in a rather experimental way to develop parallel algorithms for performing a single FFT (§ 2) and multiple FFTs (§ 3). Performance curves for these algorithms are presented which suggest the superior algorithms. In § 4, which unavoidably deals with the features of HEP architecture, the performance of the single and multiple FFT algorithms is modeled using the ideas of Hockney [7], [9]. This allows for a further comparison of the algorithms both within and outside the parameter ranges of the experiments. The paper concludes with a summary and some remarks on the applicability of these results to other multiprocessors.

2. Parallel algorithms for a single FFT. There are many different FFT algorithms and there are several different ways to derive and present them. In the interest of brevity and clarity, we will use the splitting algorithm to present one of the Cooley–Tukey versions of the FFT. In all that follows we will confine ourselves to the problem of transforming complex sequences of length N where $N=2^m$. The problem of transforming sequences of fairly arbitrary length (the mixed radix case) can be treated by sequential algorithms and the necessary generalizations can also be applied to the parallel algorithms which will be discussed here.

Given a sequence $\{x_n\}$ of length $N=2^m$, the discrete Fourier transform (DFT) is a sequence $\{X_k\}$, also of length N, given by

$$X_k = \sum_{n=0}^{N-1} x_n w_N^{nk}, \qquad 0 \leqq k \leqq N-1, \tag{2.1}$$

where $w_N = e^{i2\pi/N}$. The inverse transform takes a very similar form. Given the transform sequence $\{X_k\}$, the original sequence may be recovered by

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k w_N^{-nk}, \qquad 0 \leqq n \leqq N-1. \tag{2.2}$$

Notice that both (2.1) and (2.2) amount to a complex matrix-vector multiplication. Hence, the computation of the DFT or its inverse requires roughly N^2 complex additions and multiplications. Successive applications of the splitting algorithm result in a progressive improvement in this operation count.

Splitting $\{x_n\}$ into its odd and even parts and letting

$$y_n = x_{2n} \quad \text{and} \quad z_n = x_{2n+1} \quad \text{for } 0 \leqq n \leqq \frac{N}{2}-1,$$

the DFT may be written

$$X_k = \sum_{n=0}^{(N/2)-1} y_n w_N^{2nk} + z_n w_N^{(2n+1)k}.$$

The crucial symmetry of the complex exponential now enters. Noting that $w_N^2 = w_{N/2}$, the transform may now be written as

$$X_k = \sum_{n=0}^{(N/2)-1} y_n w_{N/2}^{nk} + w_N^k \sum_{n=0}^{(N/2)-1} z_n w_{N/2}^{nk}.$$

However, the two sums on the right side are simply the $N/2$-point transforms of the sequences $\{y_n\}$ and $\{z_n\}$ which we call $\{Y_k\}$ and $\{Z_k\}$, respectively. Therefore, for $0 \leqq k \leqq (N/2)-1$,

$$X_k = Y_k + w_N^k Z_k.$$

For indices between $N/2$ and $N-1$, note that the sequences $\{Y_k\}$ and $\{Z_k\}$ have period $N/2$. Therefore,

$$X_{k+(N/2)} = Y_k + w_N^{k+(N/2)} Z_k.$$

Using the fact that $w_N^{N/2} = -1$, we have the two fundamental relations

$$\begin{aligned} X_k &= Y_k + w_N^k Z_k, \\ X_{k+(N/2)} &= Y_k - w_N^k Z_k, \end{aligned} \qquad 0 \leqq k \leqq \frac{N}{2} - 1, \tag{2.3}$$

which give the N-point transform $\{X_k\}$ in terms of the two associated $N/2$-point transforms $\{Y_k\}$ and $\{Z_k\}$. It is useful to note the savings that can be realized by a single splitting. If $\{Y_k\}$ and $\{Z_k\}$ are computed by matrix multiplication, the cost is $2(N/2)^2$ complex additions and multiplications. The combine step (2.3) costs N complex additions and $N/2$ complex multiplications which is negligible. Therefore, one application of the splitting algorithm reduces the operation count of the N-point transform from roughly N^2 to $N^2/2$.

The full FFT results by noting that the splitting algorithm may also be applied to the $N/2$-point transforms $\{Y_k\}$ and $\{Z_k\}$ and again to the resulting $N/4$-point transforms and so on. After $m = \log_2 N$ halvings, the problem reduces to computing N 1-point transforms which is trivial. The operation count is straightforward. At each of $\log_2 N$ stages, $N/2$ pairs must be computed using (2.3). Each pair requires one complex multiplication (assuming that the powers of w are available) and two complex additions. This gives a total of $N/2 \log_2 N$ complex multiplications and $N \log_2 N$ complex additions.

One consequence of the splitting procedure is that at each stage the input sequence is divided into its even and odd components. After $\log_2 N$ such splittings, the original sequence $\{x_n\}$ no longer appears in its natural order, but rather in a scrambled or bit-reversed order. The nuisance of reordering the input sequence before doing the combine phase is offset by the fact that the algorithm may be carried out "in-place" with no additional storage requirements. Furthermore, in many applications the forward transform is ultimately followed by an inverse transform. By an appropriate choice of FFT algorithms, the recordering of the data may be effectively avoided.

It should be mentioned that there are versions of the FFT (the Stockham version and its variants) which avoid the reordering of either the input or output sequence. The price which is paid for this convenience is an extra array of storage, a price which could become prohibitive in higher dimensional problems. A more recent development

is the class of prime factor FFTs [17], [18]. These algorithms avoid all reordering of data and are also in-place. They are most effective on sequences whose length is highly composite and, in this sense, they complement the classical FFT algorithms. Neither the self-sorting nor prime factor FFTs will be considered in this paper.

The splitting procedure just described gives one version of the original Cooley-Tukey algorithm. The combine phase consists of $m = \log_2 N$ stages. Each stage involves $N/2$ pairwise combinations of the form (2.3). At stage k, $1 \leqq k \leqq m$, the pairs reside in 2^{m-k} blocks each of length 2^k. Each pairwise combination requires a particular power of w and, at stage k, 2^{k-1} different powers of w are required. These observations begin to suggest strategies for a parallel implementation of the FFT. The fundamental unit of work in the FFT is the pairwise combination. Within each stage each pairwise combination can be done in place and independently of all others. It is on this level that the algorithm will be parallelized initially. There is no parallelism on a coarser level because the $\log_2 N$ stages themselves must be done sequentially.

With these considerations in mind, two strategies for the evaluation of a single FFT were implemented in HEP-FORTRAN and compared. The first strategy will be called *scheduling-on-pairs* since each process is assigned independent pairwise combinations to perform. At the top of the algorithm, p independent processes are created and each process is given a unique integer tag j, where $1 \leqq j \leqq p$. At each stage of the algorithm, the jth process executes a loop of the form DO 10 $i = j$, $N/2$, p which distributes the $N/2$ pairwise combinations over the p processes. Before doing a pairwise combination, the appropriate power of w must be computed, which involves some modular arithmetic and a complex exponentiation to an integer power. At the end of each of the $\log_2 N$ stages, each process is suspended in a synchronizing barrier until all processes have finished the stage. Thus, at each stage, each of the p processes does roughly $N/2p$ complex exponentiations, $N/2p$ complex multiplications, N/p complex additions, and $N/2p$ index calculations which involve some modular arithmetic.

At this point, a remark should be made concerning the ground rules for these numerical experiments. It has been decided, somewhat arbitrarily, that all algorithms will compute powers of w while in progress. This is not as efficient as pre-computing powers of w and storing them, but it does give a uniform set of conditions under which the algorithms may be compared. In § 4, an attempt will be made to estimate the savings which would result from replacing complex exponentiations by fetches to data memory.

Improvements on the scheduling-on-pairs algorithm can be realized only if the overall computational load is reduced. All good sequential FFT codes recognize the fact that once a power of w is computed it should be used immediately throughout the stage rather than recomputing it for each pair. Furthermore, complex exponentiations can be minimized if powers of w are updated by multiplications. While this idea is easily written into a sequential code, there are genuine subtleties in incorporating it into a parallel code, especially for arbitrary p. The algorithm which results will be called *scheduling-on-w*. The difficulty in implementing scheduling-on-w lies in the fact that the number of distinct powers of w which are needed changes at each stage of the algorithm. At stage k, there are 2^{k-1} distinct powers of w which must be computed. Thus in the early stages of the computation, when $2^{k-1} < p$, several processes may share the same power of w and the pairwise combinations associated with that power of w. However, during the later stages when $2^{k-1} > p$, each process may be assigned to more than one power of w and the pairwise combinations associated with each power of w. Clearly the scheduling and the structure of the algorithm change at the point when $2^{k-1} > p$. Some additional logic is required to accomplish this change in scheduling.

The question is whether this additional overhead pays for the savings in complex exponentiations. Figure 1 shows an example of scheduling-on-w for the case $N = 32$, $p = 5$.

Stage $k = 1$																
Powers of w_{32}:	0															
	p_1	p_2	p_3	p_4	p_5											
Stage $k = 2$																
Powers of w_{32}:	0				8											
	p_1		p_3		p_5	p_2		p_4								
Stage $k = 3$																
Powers of w_{32}:	0		4		8		12									
	p_1		p_5	p_2		p_3		p_4								
Stage $k = 4$																
Powers of w_{32}:	0	2	4	6	8	10	12	14								
	p_1	p_2	p_3	p_4	p_5	p_1	p_2	p_3								
Stage $k = 5$																
Powers of w_{32}:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	p_1	p_2	p_3	p_4	p_5	p_1	p_2	p_3	p_4	p_5	p_1	p_2	p_3	p_4	p_5	p_1

FIG. 1. *Assignment of five processors, p_1, p_2, p_3, p_4, and p_5, in the scheduling-on-w algorithm for an $N = 32$ point transform.*

The scheduling-on-pairs and the scheduling-on-w algorithms were both implemented in HEP FORTRAN and applied to a wide range of problem sizes with $1 \leqq p \leqq 20$ processes. Figure 2 shows a sample of these experiments for the cases $\log_2 N = 6$, 10, 12, 16. The most significant result is that economizing the powers of w computation by scheduling-on-w has a dramatic effect on computation times. The scheduling-on-w algorithm ran faster than the scheduling-by-pairs algorithm by 25% (for small N) to 80% (for large N). Since the cost of doing pairwise combinations is the same for both algorithms, this difference is due primarily to the cost of computing powers of w. These differences will be quantified in § 4.

The implementation of the scheduling-on-w algorithm relies on no special features of the HEP. Furthermore, its success could also be expected to transfer to other shared memory systems in which complex exponentiation requires a significant number of floating point operations. (On the HEP, the cost is 10–50 flops, depending on the arguments.)

The timing curves also show that the maximum speed-up and the number of processes which gives the maximum speed-up both increase with the problem size. This is due, in part, to the increase in the fraction of fully parallel code which can be executed between process synchronizations. Said differently, as N increases, the amount of time spent in synchronizing barriers becomes insignificant. Because of features of the HEP hardware (discussed in § 4), speed-ups level off with increasing N at about 10 and are generally achieved with 15–18 processes. When more than the optimal number of processes is used, the execution times show a slow rise due to contention among processes during the synchronization phases. These qualitative features are not unique to the HEP. Any tightly coupled multiprocessor executing a program which contains some fraction of sequential code will have an optimal configuration in which speed-up and interprocessor contention are balanced.

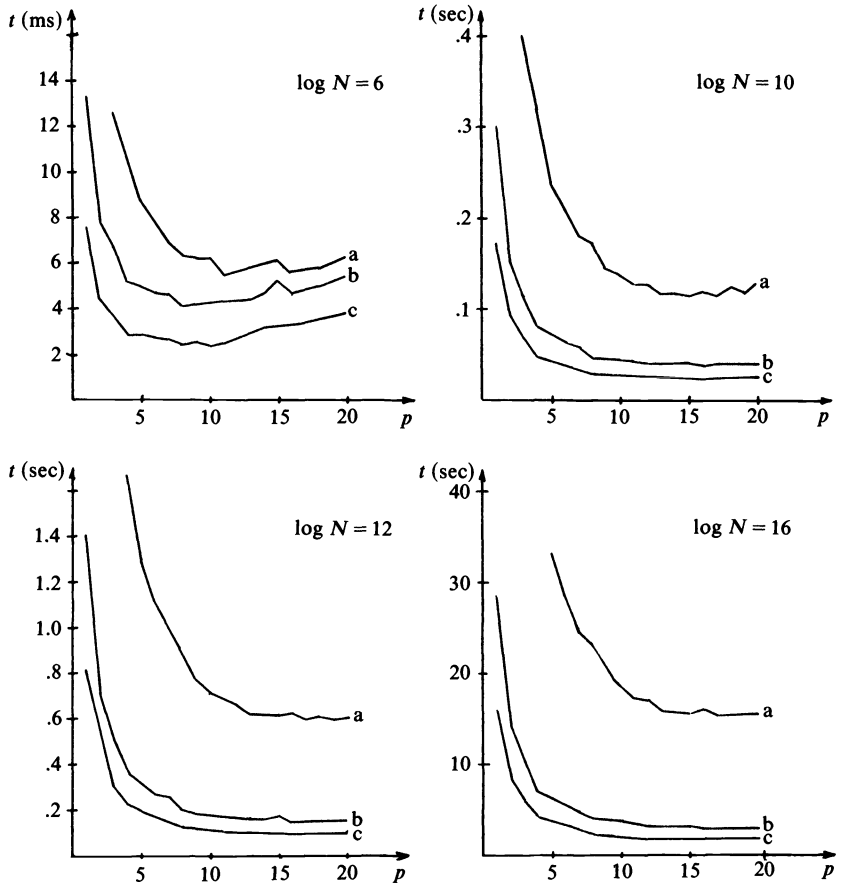


FIG. 2. *Execution times for* (a) *scheduling-on-pairs,* (b) *scheduling-on-w and* (c) *Edson's algorithm for a single* FFT *of length* $N = 256$; 1,024; 4,096; 65,536.

In a large percentage of applications, the input sequence consists of real numbers. In this case, it is not difficult to show that the transform $\{X_k\}$ of a real sequence has the property of conjugate symmetry; i.e., $X_k = \bar{X}_{N-k}$. Furthermore, the partial transforms of length 2^k which are computed at stage $1 \leqq k \leqq \log_2 N$ also have conjugate symmetry. This fact may be used to do the entire transform in real arithmetic which reduces both the operation count and the storage requirement by a factor of two. The resulting method is generally referred to as Edson's algorithm [1]. The scheduling issues discussed for the complex FFT also pertain to the real FFT. Given the superiority of the scheduling-on-*w* approach, Edson's algorithm was implemented using this strategy. As a point of reference, the execution times for this version of Edson's algorithm are also given in Fig. 2. These times are consistently 60% of the execution times for the corresponding complex FFT. This is due to the fact that while the overhead for the scheduling-on-*w* remains the same, the pairwise combinations cost half as much for Edson's algorithm as they do for the complex FFT. Additional symmetries of the input sequence (e.g., even, odd, quarter wave even and odd) could also be exploited to give even further savings.

Having determined a clearly preferable method for performing a single FFT, we turn to the problem of doing multiple FFTs. As will be seen, the benefits of the scheduling-on-*w* strategy can also be applied to this problem to give significant savings.

3. Parallel algorithms for multiple FFTs. Many applications require not a single FFT, but rather multiple FFTs. For example, the Fourier analysis of a two-dimensional field of data may be computed through one-dimensional FFTs along the rows of the grid followed by a sweep of FFTs along the columns of the grid. Similarly, the solution of a two-dimensional boundary value problem can be carried out by transforming (i.e., doing FFTs) along grid lines in one coordinate direction and then possibly (depending upon the algorithm) transforming in the orthogonal direction. These forward transforms are eventually followed by inverse transforms in both directions. Therefore, it is important to have parallel algorithms for performing multiple FFTs across either the rows or columns of a two-dimensional grid.

Consider the problem of computing M FFTs of length N, where again N is a power of two. Three strategies have been considered, implemented in HEP FORTRAN, and compared.

(a) *Parallel calls to sequential* FFTs. This approach elevates the level of parallelism one degree by imbedding a sequential N-point FFT algorithm within each process. The M sequences are then distributed over the processes. This method, which will be called the *parallel calls algorithm*, is probably the most obvious way to do a multiple FFT in parallel, but it is not the only way.

(b) *Parallel* FFT *with inner loop.* This version uses the best algorithm from the previous section (scheduling-on-w) for doing a single FFT. However, each component of the input sequence $\{x_n\}$ is now considered to be an M-vector. Each pairwise combination is done within a loop which runs over all M components. Although the HEP cannot vectorize this new, innermost loop, the method can still be expected to be competitive. This method will be called the *inner loop algorithm.*

(c) *Truncated parallel* FFT. As a serial method, this idea was first presented by Korn and Lambiotte [13] and is discussed further by Swarztrauber [14] and Temperton [16]. Assume that the M input sequences are individually bit-reversed and then concatenated to form a sequence of length MN. If an algorithm for a single FFT is applied to this long sequence and terminated after $m = \log_2 N$ stages, then the M transform sequences are produced in the locations originally held by the respective input sequences. As with the single FFT, the bit reversals may be avoided if inverse transforms are eventually done. The parallel form of the truncated algorithm simply uses scheduling-on-w to perform the MN-point transform of the concatenated sequence. For these experiments, M is also assumed to be a power of two. Most of the classical FFT algorithms have a truncated version; however the position of the transformed sequences may vary from one method to another.

Hockney and Jesshope [10] give an extensive presentation and analysis of multiple FFT methods for vector and array processors. Of the five methods discussed there, only one has a clear analogue in this study. The $(A+B)_{\text{par}}$ and $(\text{PARAFT})_{\text{par}}$ methods require a machine parallelism which far exceeds that of the HEP or most current MIMD computers. The $(A+B)_{\text{seq}}$ and $(\text{PARAFT})_{\text{seq}}$ methods, in which the best parallel algorithm is applied successively to each input sequence, really have no analogue. Hockney and Jesshope's MULTFT algorithm is reflected in the parallel calls algorithm in which a serial FFT is applied in parallel across the M input vectors.

Figure 3 shows the results of applying these three algorithms to M FFTs of length N where $M = N$ and $\log_2 N = 5, 6, 7, 8$. The performances of the truncated FFT and the inner loop algorithm are indistinguishable over all problem sizes and levels of parallelism. This fact might have been anticipated since both algorithms have identical computational loads and synchronization overheads. The only difference in the two algorithms is in the DO LOOP structure and the use of a doubly-subscripted array in

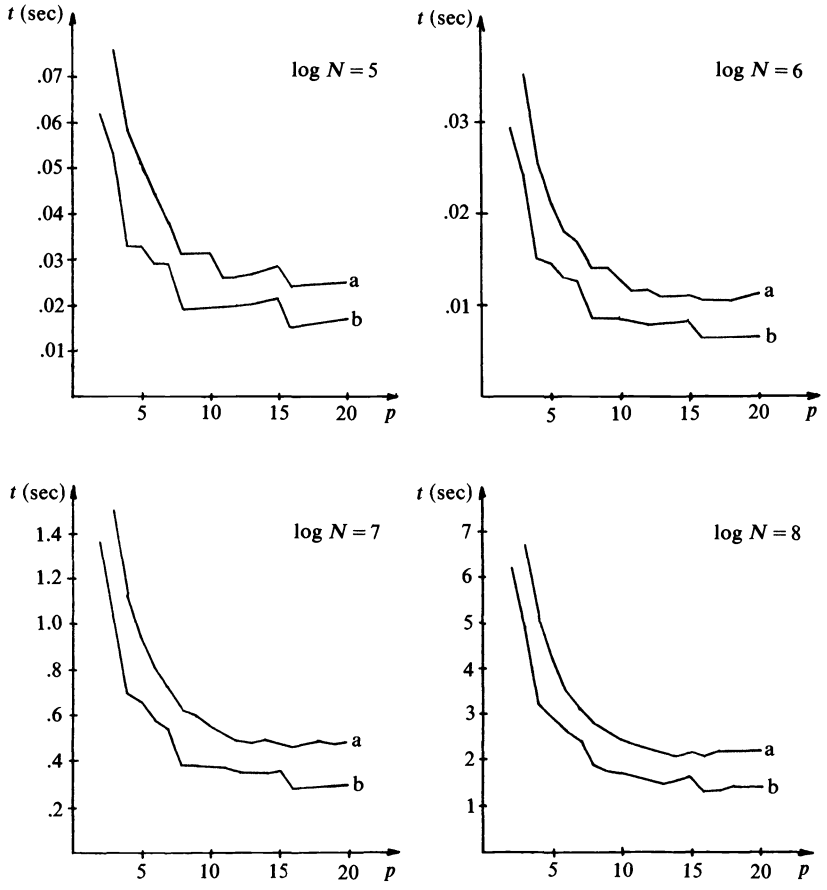


FIG. 3. *Execution times for* (a) *parallel calls and* (b) *inner loops and truncated* FFT *algorithms for M complex sequences of length N with* $M = N = 32, 64, 128, 256$ *with* 1 *to* 20 *processes.*

the inner loop algorithm as opposed to a singly subscripted array in the truncated FFT. These differences become negligible with large problems. However, there are applications in which the inner loop algorithm could have some advantages. For problems in which transforms along rows are followed by transforms along columns, the inner loop algorithm can be easily modified to avoid transposition of the data. A problem which requires the transform of every qth row of an array ($q > 1$) is difficult to handle with the truncated FFT, while the inner loop algorithm requires only a minor modification.

The parallel calls algorithm runs 55–60% slower than the inner loop and truncated FFT algorithms over all problem sizes. This difference will be quantified in the following section. It can be explained by the fact that in the parallel calls algorithm all of the overhead for a single FFT (computation of sines and cosines, DO LOOP set-up) must be repeated for each of the M/p transforms which each process performs. The other two algorithms minimize the computation of powers of w through scheduling-on-w. In addition, the overhead which *is* necessary in the inner loop and truncated FFT algorithms is used on roughly M times as many pairwise combinations which makes much more efficient use of the overhead. The parallel calls algorithm does have the attribute that it requires no synchronizing barriers. With these experimental results

and qualitative observations, we now turn to a more analytical investigation of these algorithms.

4. Models of performance. In this section, a quantitative analysis of the algorithms of the previous two sections is given. Models of performance will be developed which can be used to reproduce the experimental timing curves and to predict algorithm performance outside of the experimental parameter ranges. The models make extensive use of the ideas proposed by Hockney for vector computers [7] and for the HEP [9]. This analysis does require some specific discussion of HEP architecture which has been minimized until this point.

Of particular importance to the following analysis is the fact that the HEP achieves its parallelism through an instruction pipeline that has eight segments. When this pipeline is filled, an instruction can be executed every clock cycle (100 ns). One can imagine that each of the p active processes successively inserts an instruction into the pipeline. Therefore, a given process will wait 8 clock cycles between instructions if $p \leqq 8$ or p clock cycles between instructions if $p > 8$. However, the HEP rarely operates in this idealized mode. While most instructions can be executed in 8 clock cycles, delays typically occur. One source of delay is the storing and fetching of data in the main memory. If data are not available or if there is contention for asynchronous variables, the pending instruction is temporarily suspended. Delays also occur when an instruction requires the result of a floating point divide. Because the divide unit requires 1700 ns per result, subsequent instructions are "waved off" until the results are available. The effect of these delays is that the effective length of the instruction pipeline is not 8, but somewhere between 12 and 16 segments, varying with the instruction mix of the program being run.

In Hockney's benchmark experiments [9] which involve a single floating point operation (flop) per process and a single type of synchronization, the effective pipeline length, L, can be identified as the number of processes which gives the minimum execution time. If we let P denote the average time between the instructions of a single process, it follows that if $p \leqq L$, then $P = L$ cycles, whereas if $p > L$, then $P = p$ cycles.

As will be shown below, when a program involves a variety of different operations and more than one type of synchronization, it is no longer possible to identify an effective pipeline length L which is constant for all values of p. This makes it difficult to determine, a priori, values for the parameter P. The timing models which will be developed depend critically upon having good estimates of P for various values of p. These estimates of P will be determined experimentally which leads to some modifications of Hockney's model. Figure 4 shows schematically how P varies with p in Hockney's benchmark model and in a model of a more general algorithm.

With these remarks, we now turn to the modeling of the various FFT algorithms. Figure 5 shows schematically the form of the algorithms which have been developed in this study. They all consist of a fan-in stage in which processes are created, followed by the execution stage in which all processes are active, terminated by a fan-out phase in which the processes die. The execution stage itself generally consists of several (usually $\log_2 N$) substages which are separated by synchronizing barriers. After each of these barriers there is also a fan-in phase in which processes are successively released from the barrier.

Let process 1 initially be the main calling program. Process 1 creates $p-1$ processes and then itself becomes the pth independent process. Therefore the total execution time for the algorithm is simply the total time that process 1 is working. It should be mentioned that HEP programs have a nondeterministic character and at some point

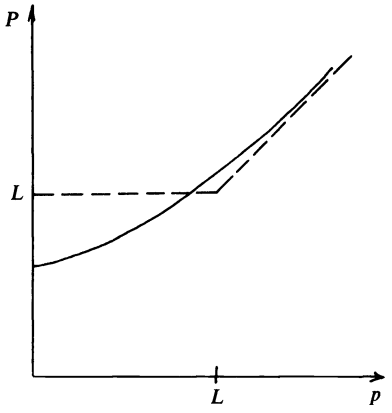


FIG. 4. *The variation of P, the average time in clock cycles between the instructions of an individual process, with p, the number of active processes. Dashed line shows an idealized model with a fixed effective pipeline length L. Solid line shows a more typical situation for a program with a mix of instructions.*

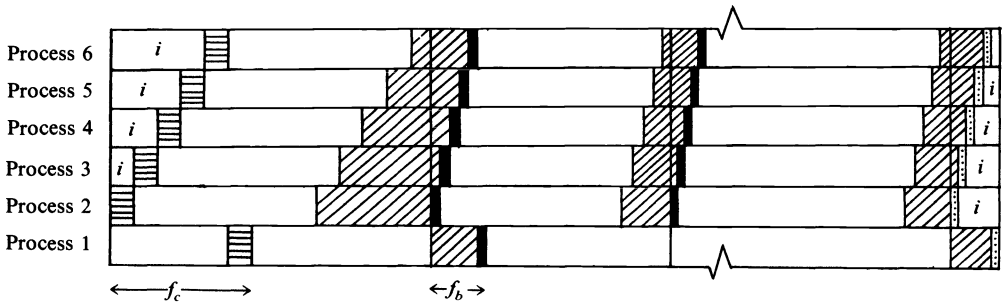


FIG. 5. *Schematic time line of a multistage algorithm showing the scheduling of 6 processes through the phases of creation, computation and barrier synchronization. The times f_c and f_b (see text) are indicated. The execution time for the algorithm is the total time that process 1 is active.* ▤: *creation of process,* □: *computation,* ▨: *barrier,* ■: *fan-out from barrier,* ⊡: *final fan-out, i: idle time.*

the identity of specific processes cannot be followed. This fact does not affect the following analysis. Another simplification which will be made concerns the divisibility of work. As suggested by Fig. 1, Q independent units of work cannot always be spread evenly over p processes. At least one process must do $[Q/p]$ units of work, which gives the irregular peaks in the timing curves. No attempt will be made to account for these effects which do become insignificant for large problems. Jordan [12] discusses divisibility effects and is able to model them quite accurately.

The following quantities will be used in the performance models. The parameter values were determined from separate timing experiments.

i_0 = number of instructions to create a process = 29.
i_1 = number of instructions to enter a synchronizing barrier = 9.
i_2 = number of instructions to exit from a synchronizing barrier = 2.
i_3 = average number of instructions/flop.
i_4 = number of instructions to set up a DO LOOP = 33.
i_5 = number of flops per INT or MOD operation = 4.
i_6 = number of flops per complex exponential = 11.
i_7 = number of flops per integer divide = 4.
i_8 = number of flops per complex multiply = 6.

i_9 = number of flops per floating point divide = 2.
i_{10} = number of flops per SIN or COS = 20.
B = number of flops for a pairwise combination (one complex multiply and two complex adds) = 10.
p = number of active processes created.
P = average time between the instructions of an individual process.
$m = \log_2 N$.
$l = \lceil \log_2 p \rceil$.
τ = clock cycle = 100 ns.

The values of i_6 and i_{10} vary depending on the arguments of the function. The given values are averages which give accurate estimates for the flop count. The parameter i_3 also varies with the instruction mix, but values between 5.4 and 6.5 fit these algorithms very closely. Finally, the pairwise combinations of the first stage are simpler (since $w^0 = 1$) and require only 4 flops. While these savings are incorporated into the programs, it will be neglected to simplify the analysis. With these preliminaries, the execution times of the various FFT algorithms may be described.

Each algorithm involves an initial fan-in stage for the creation of p processes. The time required for this phase is $f_c = pPi_0$ (a departure from Hockney's expression since P is defined differently). At the end of this fan-in stage, all p processes are active and will remain active either doing work or waiting in a barrier until the computation is finished. After each synchronizing barrier, there is a fan-in stage as processes are successively released from the barrier. This phase requires a time of $f_b = pPi_2$. At the end of the computation, there is a final fan-out phase as the processes retire. This fan-out requires a time of f_b in cases in which the processes were last synchronized in a barrier.

The execution times for each of the FFT algorithms may now be described. In each case, the total computation time is determined as the amount of time that process 1 is active (see Fig. 5).

(a) *Single transform by scheduling-on-pairs.* In this algorithm all $m = \log_2 N$ stages require essentially the same amount of work. Therefore,

$$\frac{T_0}{\tau} = f_c + m\left[i_3 P\left(\frac{N}{2p}(2i_5 + i_6 + i_7 + B) + i_9 + 2i_{10}\right) + P(i_1 + i_4) + f_b \right]. \tag{4.1}$$

The quantity multiplied by i_3 represents the cost of floating point operations. Of that cost, the quantity multiplied by $N/2p$ reflects each processes' share of the pairwise combinations. The overhead for DO LOOPS and barrier synchronization must be paid m times.

(b) *Multiple* FFTs *by parallel calls.* In this algorithm there are no intermediate barrier synchronizations. The M input sequences are simply distributed over p processes. The time required is

$$\frac{T_1}{\tau} = f_c + P\left[i_3 \frac{M}{p}\left(m\left(\frac{NB}{2} + i_9 + 2i_{10}\right) + Ni_8 \right) + i_4 \frac{M}{p}(N + m) \right]. \tag{4.2}$$

Each process bears the full expense of M/p sequential FFTs, each of which has a floating point cost (the term multiplied by i_3) and an overhead for DO LOOPs. On the other hand, inter-process synchronization is negligible.

(c) *Truncated* FFT *for M sequences.* This analysis with $M = 1$ will give the model for scheduling-on-w for a single FFT. Recall that the structure of this algorithm changes at stage k, where $k > l = \lceil \log_2 p \rceil$. With this in mind, the time required for this algorithm

is given by

$$\begin{aligned}\frac{T_{2,M}}{\tau} &= f_c + P\sum_{k=1}^{l}\left[i_3\left(\frac{MNB}{2p}+2i_5+i_6+3i_7\right)+i_1+i_4\right] \\ &\quad + P\sum_{k=l+1}^{m}\left[i_3\left(\frac{MN}{2p}B+2i_6+i_7+\left(\frac{2^{k-1}}{p}\right)i_8\right)+i_1+i_4(1+2^{k-1})\right]+mf_b \\ &\simeq f_c + P\left[i_3\left(\frac{mMNB}{2p}+2li_5+(2m-l)i_6+(m+2l)i_7+\frac{N}{p}i_8\right)\right. \\ &\quad \left.+\left(\frac{N}{p}+m\right)i_4+mi_1\right]+mf_b.\end{aligned} \tag{4.3}$$

(d) *Parallel* FFT *on vector components.* As discussed in the previous section, this algorithm performs almost identically to the truncated FFT. The differences cannot be captured in these timing models. Therefore, expression (4.3) will be taken as the model for this algorithm also.

Each of the timing models (4.1), (4.2), and (4.3) can be expressed in terms of Hockney's generic formula

$$T = r_\infty^{-1}(s + s_{1/2}),$$

where r_∞ is the maximum flop rate of the computer with p active processes, s is the *total* flop count for all processes, and $s_{1/2}$ is the overhead of fan-ins, fan-outs, DO LOOPs and barriers, measured in flops. In this case, $r_\infty = (p/P)1/i_3\tau$ and therefore as p approaches P, the computer nears its maximum possible execution rate, $(10/i_3)M$flops. The difference between p and P is a measure of the inefficiency or lack of utilization in the instruction pipeline.

The timing models (4.1), (4.2), and (4.3) each lead to generic expressions of the form

$$T = r_\infty^{-1}(\underbrace{Q+pR}_{s} + \underbrace{U+Vp+Wp^2}_{s_{1/2}}).$$

The term Q represents flops which are distributed over the processes. This work is reduced per process as p increases. The term including R represents redundant computations which all processes must do independent of p. This term does not appear in the parallel calls algorithm (4.2). The term U represents overhead which is reduced as processes are added and appears in (4.2) and (4.3) as DO LOOP set-ups. The term involving V represents fixed overhead in DO LOOP set-ups and synchronization barriers which affect all processes independent of p. Finally, the term Wp^2 represents the cost of fan-in and fan-out phases which appear in all three algorithms. Clearly these algorithms include some additional costs which do not appear in Hockney's benchmark model.

A comparison of model times may now be made. Equations (4.1) and (4.3) for a single FFT show that the cost of doing pairwise combinations, $mNB/2p$, is identical. The primary difference arises in the auxiliary arithmetic terms i_5, i_6, i_7, i_8, i_9, and i_{10}. For fixed p, these terms are weighted by mN in (4.1) and by m and N in (4.3), which accounts for much of the difference in computation times. Equating T_0 and $T_{2,1}$ indicates that there is no equal performance curve for the two algorithms that lies in a feasible region of the (N, p) parameter plane. As suggested by the experiments of § 2, the scheduling-on-w algorithm is superior to the scheduling-on-pairs algorithm over a wide range of problem sizes and levels of parallelism.

A comparison of timing expressions (4.2) and (4.3) for multiple FFTs indicates that the work involved in pairwise combinations is identical. Once again the difference in the performance curves arises in the cost of doing auxiliary arithmetic. In the parallel calls algorithm, the expense of computing sines, cosines, and complex multiplications must be repeated for each of M sequences each of length N. In the truncated FFT algorithm, these auxiliary computations are minimized to begin with and once they are done, they may be used over M sequences. Equating (4.2) and (4.3) shows that the only possible equal performance curve occurs in the limit of large p and small M, a regime which fits neither useful applications nor current MIMD architectures. Therefore, the performance models support the conclusion of § 3 that the truncated FFT and the inner loops algorithms are superior to the parallel calls algorithm.

In Fig. 6, the timing models (4.2) and (4.3) are used to fit the experimental timing curves for the case of 256 256-point transforms. Table 1 gives the values of P which were used for each value of p as well as the flop rate which was achieved for each value of p. Both the values of P and the flop rates were determined from a system timing routine which resides on the HEP. The timing curves fit the data very closely. In particular, the point of minimum execution time is resolved very well, indicating that the models capture the effects of process contention fairly accurately. Both algorithms reach almost 100% utilization of the instruction pipeline (10 million instructions/second) with 16 processes. However, the truncated FFT algorithm attains a higher

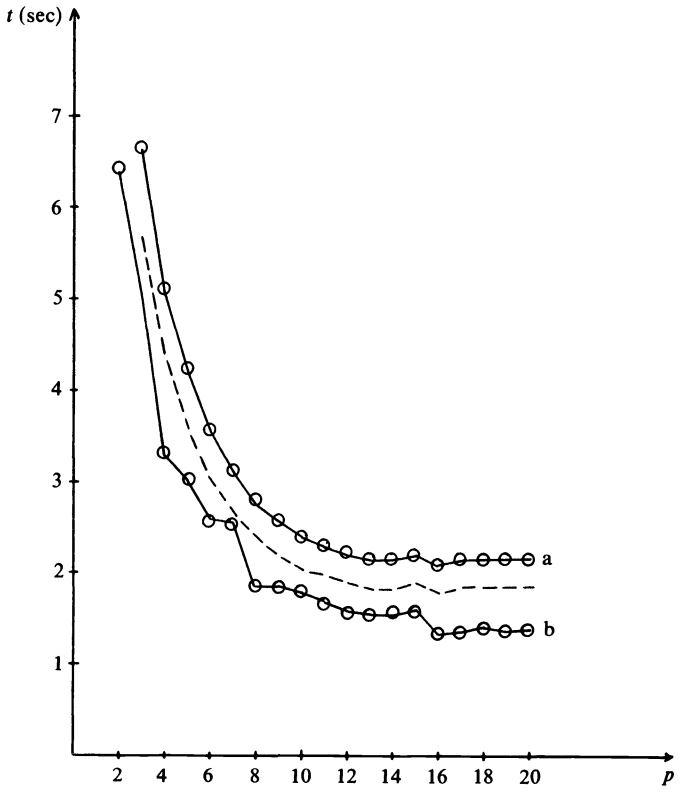


FIG. 6. *Execution times for the* (a) *parallel calls and* (b) *inner loops and truncated* FFT *algorithms for* 256 *complex sequences of length* 256. *Open circles give predicted times from the performance models. The dashed line gives the projected performance curve of the parallel calls algorithm when computation of powers of w is replaced by data fetches.*

TABLE 1
$M = 256$ FFTs *of length* $N = 256$.

	Parallel calls ($i_3 = 6.5$)		Truncated FFT ($i_3 = 5.4$)	
p	P (cycles)	r_∞ (Mflop)	P (cycles)	r_∞ (Mflop)
1	9.4	0.16	9.7	0.20
2	9.5	0.32	9.8	0.38
3	9.6	0.47	11.4	0.49
4	9.8	0.62	9.9	0.74
5	10.1	0.74	11.5	0.80
6	10.2	0.88	11.9	0.93
7	10.5	1.00	13.5	0.95
8	10.7	1.12	11.2	1.31
9	11.1	1.20	12.7	1.30
10	11.5	1.30	13.5	1.36
11	12.1	1.35	14.0	1.45
12	12.7	1.41	14.4	1.53
13	13.3	1.46	15.3	1.57
14	14.4	1.46	16.6	1.55
15	15.9	1.41	18.4	1.50
16	16.0	1.50	16.0	1.84
17	17.5	1.46	17.2	1.82
18	18.4	1.46	19.6	1.43
19	19.4	1.46	19.6	1.79
20	20.1	1.49	20.7	1.77

P is the average time (in clock cycles) between instructions of an individual process. r_∞ is the flop rate attained with p active processes.

flop rate, because of the mix of instructions. For the truncated FFT, $i_3 = 5.4$ instructions/flop, while for the parallel calls algorithm, $i_3 = 6.5$ instructions/flop.

It has become clear that the computation of powers of w is an important factor in determining the relative performance of the various FFT algorithms. The models of this section may be used to estimate the effect of this expense. If powers of w are pre-computed and stored in the main data memory, then complex exponentiations/multiplications and sine/cosine evaluations may be replaced by data fetches. Making this modification in the truncated FFT model results in a 1% reduction in computation time, fairly uniformly across all problem sizes and levels of parallelism. A similar modification in the parallel calls model results in a uniform 15% reduction in computation times (see Fig. 6). The truncated FFT algorithm experiences little improvement since auxiliary arithmetic in computing powers of w has been minimized to a great degree. While the parallel calls algorithm shows a noticeable improvement, it still runs 35% slower than the truncated FFT on a 256×256 problem (compared to 55% slower when computing powers of w). It is difficult to predict the further improvement if powers of w could be stored in a high speed cache memory (which does not exist on the HEP) or partially in registers. However, within the bounds of these models the truncated FFT (and the inner loop algorithm) appear to have a comfortable margin of superiority over the parallel calls algorithm.

5. Summary and conclusions. Proceeding first in an experimental manner and then analytically, the problem of implementing single and multiple FFT algorithms on an MIMD computer has been investigated. For computing a single transform, the strategy

of scheduling-on-w minimizes the expense of computing powers of w and leads to significant savings over the alternate scheduling-on-pairs algorithm. The scheduling-on-w approach was also applied to Edson's algorithm for transforming a real sequence and could also be applied to other symmetric transforms.

Of the three algorithms studied for transforming multiple sequences, two algorithms, the truncated FFT and the inner loops algorithm, performed almost identically and significantly better than the parallel calls algorithm. The two superior algorithms are both based on the scheduling-on-w idea. The inner loops algorithm is preferable for applications in which alternate rows or columns of an array must be transformed.

The performance of all of these algorithms was modeled using the ideas of Hockney. The model equations support the conclusions of the experiments and can be used to fit the experimental timing curves very closely. The models also suggest that the algorithms which emerge superior in the experiments remain superior over all reasonable ranges of problem size and levels of parallelism. The models were also used to estimate the improvements which would result if powers of w were precomputed and stored rather than computed in progress. This modification would lead to a 15% improvement in the parallel calls algorithm for multiple FFTs, but still does not make it competitive with the inner loops and truncated FFT algorithms.

The HEP is admittedly just one representative of the class of MIMD computers. Nevertheless, it would be useful to extract some conclusions which might apply to other multiprocessors. Like the HEP, most shared memory systems which are available today have a fairly low degree of parallelism. Computers like the CRAY-XMP, CRAY-2, the ETA-10, and the Alliant have ten or fewer processors which work in an MIMD fashion. It is felt that the algorithms which were developed in this study can be implemented on such tightly coupled, shared memory systems and that the performance characteristics of the algorithms would remain qualitatively unchanged. The algorithms would require modification to account for system-specific synchronization mechanisms, the possible existence of high speed cache memory and the availability of vector capabilities. In this latter regard, the inner loops algorithm for multiple FFTs holds yet another advantage in that it possesses a high degree of inner parallelism which could be exploited by vector processors.

Within the domain of loosely coupled or distributed MIMD systems, it would be difficult to apply the results of this study. In such systems, the cost of communication between processors and their local memories becomes significant and cannot be neglected in comparison to computation costs. It is possible that the implementation and performance of the algorithms of this study could be significantly different for such systems. Models which explicitly include communication costs [5] are necessary to make any comparisons.

The algorithms which have been described will be implemented on other shared memory systems to investigate their generality and applicability. Furthermore, in a project just completed on the HEP, these FFT methods have been incorporated into the FACR (ℓ) algorithm [8] for the direct solution of Poisson's equation. The results of this study are forthcoming.

Acknowledgments. We are grateful to Harry Jordan and Paul Swarztrauber for several valuable discussions during the course of this work. The referees of this paper offered many valuable suggestions which have been incorporated with much appreciation. Denelcor and Los Alamos National Laboratory provided computer time on their HEPs, without which this research would have been impossible. In addition, the staff

of both facilities gave generously of their time and expertise. The students of the Spring 1984 and Fall 1984 CU-Denver Math Clinic made many contributions to the early stages of this work.

REFERENCES

[1] G. D. BERGLAND, *A fast Fourier transform for real-valued series*, Comm. ACM, 11 (1968), pp. 703-713.
[2] E. O. BRIGHAM, *The Fast Fourier Transform*, Prentice-Hall, Englewood Cliffs, NJ, 1974.
[3] W. T. COCHRAN ET AL., *What is the fast Fourier transform*? IEEE Trans. Audio. Electroacoustics, An-15 (1967), pp. 45-55.
[4] J. W. COOLEY AND J. W. TUKEY, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp., 19 (1965), pp. 297-301.
[5] D. B. GANNON AND J. VAN ROSENDALE, *On the impact of communication complexity on the design of parallel numerical algorithms*, IEEE Trans. Comput. (1984), pp. 1180-1194.
[6] W. M. GENTLEMAN AND G. SANDE, *Fast Fourier transforms for fun and profit*, 1966 Fall Joint Computer Conference, AFIPS Proc., 29 (1966), pp. 563-578.
[7] R. W. HOCKNEY, *Characterizing computers and optimizing the* FACR (ℓ) *Poisson-solver on parallel unicomputers*, IEEE Trans. Comput. (1983), pp. 933-941.
[8] ———, *A fast direct solution of Poisson's equation using Fourier analysis*, J. Assoc. Comput. Mach., 12 (1965), pp. 95-113.
[9] ———, *Performance characterization of the* HEP, in Parallel MIMD Computation: HEP Supercomputer and Its Applications, J. S. Kowalik, ed., MIT Press, Cambridge, MA, 1985, pp. 59-90.
[10] R. W. HOCKNEY AND C. R. JESSHOPE, *Parallel Computers—Architecture, Programming and Algorithms*, Adam Hilger, Bristol, 1981.
[11] H. F. JORDAN, *Experience with pipelined multiple instruction streams*, Report CSDG 83-4, Dept. Electrical Comput. Enging., Univ. Colorado, Boulder, CO, 1983.
[12] ———, *Interpreting Parallel Processor Performance Measurements*, Report CSDG 85-1, Dept. Electrical Comput. Enging., Univ. Colorado, Boulder, CO, 1985.
[13] D. G. KORN AND J. J. LAMBIOTTE, *Computing the fast Fourier transform on a vector computer*, Math. Comput., 33 (1979), pp. 977-992.
[14] P. N. SWARZTRAUBER, FFT *algorithms for vector computers*, Parallel Computing, 1 (1984), pp. 45-63.
[15] ———, *Vectorizing the* FFT's, in Parallel Computations, G. Rodrigue, ed., Academic Press, New York, 1982.
[16] C. TEMPERTON, *Fast mixed-radix real Fourier transforms*, J. Comput. Phys., 52 (1983), pp. 340-350.
[17] ———, *Implementation of a self-sorting in-place prime factor* FFT *algorithm*, J. Comput. Phys., 58 (1985), pp. 283-299.
[18] ———, *A note on the prime factor* FFT *algorithm*, J. Comput. Phys., 52 (1983), pp. 198-204.
[19] ———, *Self-sorting mixed-radix fast Fourier transforms*, J. Comput. Phys., 52 (1983), pp. 1-23.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 1, January 1987

© 1987 Society for Industrial and Applied Mathematics

ASYNCHRONOUS RELAXATIONS FOR THE NUMERICAL SOLUTION OF DIFFERENTIAL EQUATIONS BY PARALLEL PROCESSORS*

DEBASIS MITRA†

Abstract. We consider asynchronous relaxations for the numerical solution by parallel processors of initial value problems involving a class of ordinary differential equations. The differential equations may be nonlinear, nonautonomous and nonhomogeneous. The proposed algorithms have several advantages in the parallel processing context, and these apply irrespective of the number of processors and the architecture of the computing system. The form of the given equations is as a partitioned system which closely corresponds to the composition of physical systems from subsystems. The algorithms iterate asynchronously on functions. We prove that the computed functions uniformly converge at a geometric rate to the unique solution of the given equations. The assumption on the asynchronous aspects of the algorithm are that delays are uniformly bounded and that a nonstarvation condition applies. In practice these are not burdensome. The assumptions on the differential equations are dominance conditions. For linear autonomous equations the assumption is that a certain matrix is an M-matrix.

Key words. parallel processing, asynchronous relaxations, waveform relaxations, differential equations

AMS(MOS) subject classifications. 65L05, 65W05

1. Introduction.

We propose asynchronous relaxations for obtaining, by parallel processing, the solutions of initial value problems on ordinary differential equations. The differential equations may be nonlinear, nonautonomous and nonhomogeneous; the results of this paper apply when various restrictions are placed on the equations. Relaxation algorithms can be devised with various degrees of asynchronism, and an asynchronous computational model is proposed in this paper for their unified specification and analysis. The generality of the asynchronous computational model is derived from the presence of its various parameters. A particular limiting case, which is obtained by appropriate selections of the parameters, is synchronous relaxations. At the other extreme, we obtain a purely asynchronous algorithm of particular interest. This algorithm is characterized by the absence of essentially all synchronization or coordination on the processors. Other algorithms with intermediate degrees of asynchronism are also described by the model. The asynchronous computational model is precisely described in § 2.2.

Asynchronous algorithms have been previously proposed [1]–[5] and analyzed for use in parallel processors for the solution of various problems other than the solution of differential equations. Certain advantages are obvious: first, the reduction in synchronization between processors yields reduced idle times for the processors; contentions over communication recources and memory accesses are reduced; task management and programming are simplified. On the crucial question of algorithmic efficiency there is evidence that, in certain interesting cases, asynchronous algorithms outperform their synchronous counterparts on account of the different nature of information flow during computations. For instance, [5] gives experimental data which shows over an interesting range of parameters the improvement in performance gained from asynchronism. An important goal, then, is to be able to recognize problems which can be solved by asynchronous relaxations. The analysis in this paper is devoted to that end.

* Received by the editors November 26, 1985; accepted for publication (in revised form) March 19, 1986.
† AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

We prove convergence results for the asynchronous computational model. Let $\|\mathbf{x}\|$ denote a norm of $\mathbf{x} \in \mathbb{R}^N$ and X the set of all N-vector functions which are continuous in the interval $[0, t']$. Define the norm of any function $\mathbf{x}$ in X to be

$$\|\mathbf{x}\| = \sup_{0 \leqq t \leqq t'} \|\mathbf{x}(t)\|.$$

We prove that $\|\mathbf{x}^{(i)} - \mathbf{x}\| \leqq cR^i$, $0 \leqq R < 1$, $c < \infty$, where $\mathbf{x}^{(i)}$ is the vector function available after the ith update in the asynchronous relaxations, and $\mathbf{x}$ is the unique vector function solution of the given initial value problem. Thus uniform convergence at a geometric rate is proven. See § 6.2 for a discussion and summary.

There are two sets of assumptions made to arrive at the above results. The first specifically concerns the asynchronous aspects and the second concerns the differential equations. Roughly, the first set (see § 2.3) requires that the delays in the implementation are bounded by a finite constant and that every type of task is guaranteed at least one completion in all sequences of task completions of an arbitrary, but fixed, length. These requirements are generally easy to satisfy. The assumptions on the differential equations (see §§ 4.1 and 5.1 for the linear and nonlinear equations, respectively) are all related to dominance conditions. In the special case where the homogeneous part of the equation is autonomous and linear, the only assumption is that a certain matrix, $\mathbf{M} - \mathbf{N}$, is an M-matrix [8]. In this case the rate of convergence is given in part by the spectral radius of $\mathbf{M}^{-1}\mathbf{N}$.

The iteration in the algorithm is on functions. From the point of view of this paper, the primitive tasks consist primarily of the computation of functions which are solutions of nonhomogeneous, initial value problems. The method of numerical integration used to solve the problem is not discussed in this paper.

The purely asynchronous algorithm, which we have mentioned before as a limiting case of the asynchronous computational model, is characterized by each processor successively executing the following cycle until termination, independently of the state of the other processors: read the functions of the task allocated to it, solve the initial-value problem associated with the task, and finally disseminate the solution functions by either writing into shared memory or by sending messages to the appropriate processors. Note that this form of asynchronous computations is compatible with the use of a scheduler to order the tasks. Typically such a scheduler is required to parallelize Gauss–Seidel schemes in which a single processor performs the tasks in some specific order [12].

The asynchronous properties of the relaxations are derived primarily from two sources. First, the times taken to perform tasks are allowed to vary unpredictably within large bounds, and, second, great latitude is allowed in the sequencing of tasks. The convergence result is for a computational model which accommodates major differences in the number and speed of processors, memory, communications and synchronization primitives. These differences are reflected in the values of parameters of the computational model, i.e. the delays and the compositions of the update sets.

The synchronous version of the algorithm, see § 2.2, has sometimes been called "waveform relaxations" [6], and these have been successfully applied to the solution of VLSI circuit-equations by uniprocessors. It is shown below that the synchronous algorithm is a special case of the asynchronous computational model described in § 2.2.

The paper is organized as follows. In § 2.1 we describe the equations, and in §§ 2.2 and 2.3 we give the computational model and the accompanying assumptions. Section 3 is mainly concerned with differential inequalities, which are used throughout the paper. Sections 4 and 5 analyze, in turn, linear and nonlinear equations. We feel that

such a separation benefits the exposition. Section 6 picks up the qualitatively similar results obtained at the ends of §§ 4 and 5, to prove uniform convergence for both types of equations. Concluding remarks are contained in § 7.

2. Model of asynchronous relaxations. We give, first, a preliminary description of a class of differential equations and, second, a model of the asynchronous relaxations for numerically solving the equations.

2.1. A class of differential equations. The most general equations that are considered in this paper are nonlinear, nonautonomous and nonhomogeneous. The initial value is given and a unique solution is known to exist for the interval of interest, say $0 \leqq t < t'$. The value of t' is ignored here since our analysis applies for all t'; however, it is recognized that t' may be an important parameter in the implementation since storage requirements alone may cause the interval of interest to be partitioned into several smaller intervals [12].

The equations are of the form

$$\begin{aligned} &\frac{d}{dt}\mathbf{x}(t)+\mathbf{F}(\mathbf{x}, t)=\mathbf{G}(\mathbf{x}, t)+\mathbf{u}(t), \qquad t \geqq 0, \\ &\mathbf{x}(0)=\mathbf{x}_0, \end{aligned} \tag{2.1}$$

in which $\mathbf{x}$, $\mathbf{u}$, $\mathbf{F}$, and $\mathbf{G}$ are elements of $\mathbb{R}^N$ for each value of t. In addition, it is assumed that $\mathbf{F}$ has a special structure:

$$\mathbf{F}(\mathbf{x}, t)=\{\mathbf{F}_1(\mathbf{x}_1, t), \mathbf{F}_2(\mathbf{x}_2, t), \cdots, \mathbf{F}_n(\mathbf{x}_n, t)\}'. \tag{2.2}$$

$\mathbf{G}$ may not have this special structure. Thus (2.1) allows the following partitioning into n subsystems which are indexed by j.

$$\begin{aligned} &\frac{d}{dt}\mathbf{x}_j(t)+\mathbf{F}_j(\mathbf{x}_j, t)=\mathbf{G}_j(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n, t)+\mathbf{u}_j(t), \qquad t \geqq 0, \quad 1 \leqq j \leqq n, \\ &\mathbf{x}_j(0)=\mathbf{x}_{j,0}. \end{aligned} \tag{2.3}$$

In (2.2) and (2.3); $\mathbf{x}_j$, $\mathbf{u}_j$, $\mathbf{F}_j$ and $\mathbf{G}_j$ are elements of $\mathbb{R}^{N_j}$ for each value of t, and $\sum_{j=1}^{n} N_j = N$.

The assumed structure is naturally associated with physical systems composed of subsystems, in which case the internal dynamical relations of the jth subsystem are incorporated in the function $\mathbf{F}_j$, while the cross-couplings between variables of the jth subsystem and the variables in the other subsystems are reflected in the function $\mathbf{G}_j$. The function $\mathbf{G}_j$ is allowed to have $\mathbf{x}_j$ as an argument because in certain nonlinear physical systems a complete separation of variables may not be possible without defining very large subsystems.

The solution of the nonlinear equations (2.3) is considered in § 5, while in § 4 the simpler, linear counterpart of the equation and its solution are considered and the necessary techniques developed.

2.2. The model for asynchronous computations. In order to describe the computational model we need to first introduce a set of primitive tasks, $\{T_j\}$, of various types. The subscript gives the type, of which there are as many as there are subsystems in the system of differential equations, namely, n. As this suggests, there is a close connection between tasks T_j and the jth subsystem in (2.3).

For a task T_j to be completely defined, its arguments $(\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_n)$ must be specified. These arguments are continuous vector functions of t, $t \geqq 0$, and $\mathbf{z}_j \in \mathbb{R}^{N_j}$,

$1 \leqq j \leqq n$, for each value of t. We say that $\mathbf{y} = T_j(\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_n)$ if $\mathbf{y}(t)$, $t \geqq 0$, is the unique solution of the following initial value problem:

$$\begin{aligned} &\frac{d}{dt}\mathbf{y} + \mathbf{F}_j(\mathbf{y}, t) = \mathbf{G}_j(\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_n, t) + \mathbf{u}_j(t), \qquad t \geqq 0, \\ &\mathbf{y}(0) = \mathbf{x}_{j,0}. \end{aligned} \tag{2.4}$$

Thus apart from its arguments, the task T_j is tied to the jth subsystem in (2.3). For this reason we sometimes refer to the completion of a task T_j as an update of the jth subsystem.

In the following description of the model for asynchronous computations, the symbol i is reserved for indexing the updates, i.e., task completions. Typically, but not always, the completion of only one task comprises an update. To allow the more general case, $U(i)$, a subset of $\{1, 2, \cdots, n\}$, denotes the types of tasks (or subsystem indices) whose completion comprise the ith update. If the ith update is due, perhaps in part only, to a task T_j being completed, i.e., $j \in U(i)$, then we denote the result of this completed task by $\mathbf{x}_j^{(i+1)}$.

The computational model is

$$\mathbf{x}_j^{(i+1)} = \begin{cases} T_j(\mathbf{x}_1^{(i-d(i,j,1))}, \mathbf{x}_2^{(i-d(i,j,2))}, \cdots, \mathbf{x}_n^{(i-d(i,j,n))}) & \text{if } j \in U(i), \\ \mathbf{x}_j^{(i)} & \text{if } j \notin U(i) \end{cases} \tag{2.5}$$

for $i = 0, 1, 2, \cdots$.

The terms $\{d(i, j, k)\}$ are referred to as delays. The delays are not assumed to be known in advance, and in fact, they will have many components, such as numerical integration time, communication delays, memory access delays and software delays. The substantial extent to which the implementation is allowed to be asynchronous in the model is due to the allowed dependence of the delays on their three arguments, and the minimal nature of the restriction placed on delay values. The only restriction that will be placed on the delay values, see A2 in § 2.3, is one of uniform boundedness.

Synchronous relaxations are obtained from (2.5) in the special case where $d(i, j, k) \equiv 0$ and $U(i) = \{1, 2, \cdots, n\}$ for each i. The synchronous relaxations have also been called "waveform relaxations" in the literature [6] on VLSI circuit-equations solution techniques. Also, the relaxations in which $U(i) = \{(i \bmod n) + 1\}$, $i \geqq 0$, and $d(i, j, k) \equiv 0$, may reasonably be called Gauss-Seidel; these relaxations may be implemented by only 1 processor.

To start the relaxations, we require the continuous vector functions $\mathbf{x}_j^{(0)}(t)$, $t \geqq 0$, $1 \leqq j \leqq n$, which are arbitrary except that their initial values match the given initial values for the subsystems, i.e. $\mathbf{x}_j^{(0)}(0) = \mathbf{x}_{j,0}$. We assume that

$$\mathbf{x}_j^{(0)}(t) = \mathbf{x}_j^{(i)}(t), \quad t \geqq 0, \quad 1 \leqq j \leqq n \quad \text{for } i < 0. \tag{2.6}$$

This convenient assumption places no additional restrictions on the implementation.

The tasks as defined here have not previously been considered in the literature on asynchronous algorithms. The model for asynchronous computations given in (2.5), augmented by appropriate definitions of the tasks, gives models closely related to those in [1]–[5].

2.3. Assumptions on the asynchronous relaxations. While additional assumptions will be made, in the course of the analysis on the differential equations given in § 2.1, there are only two assumptions on the model in (2.5) for asynchronous computations.

ASSUMPTION A1. There exists a finite d which uniformly bounds the delays, i.e.,

$$0 \leqq d(i,j,k) \leqq d < \infty, \quad i \geqq 0, \quad 1 \leqq j \leqq n, \quad 1 \leqq k \leqq n. \tag{2.7}$$

ASSUMPTION A2. There exists an integer s, $s < \infty$, such that every subsystem is updated at least once in every s consecutive updates.

In connection with Assumption A1 note that $d(i,j,k)$ is not greater than the total number of updates that occur between the ith update and the prior update which marks the beginning of the integration which results in the ith update. Therefore the bound d in A1 exists in the typical case where a given number of processors, working at possibly different positive speeds, implement in some arbitrary order the integrations of the subsystems, the associated workloads being possibly unequal. A naive value for d is obtained from the lower and upper bounds of the speeds and workloads.

Assumption A2 is a nonstarvation condition. Since each update concerns at least one subsystem, violations of A2 can only occur if the ratio of updates for some pair of subsystems is allowed to become unbounded.

Different versions of A1 and A2 have been assumed in previous work on asynchronous numeric algorithms [1]-[5].

3. Preliminaries.

3.1. Differential inequalities. The subsequent analysis makes considerable use of differential inequalities. For this reason we collect here certain well-known results related to differential inequalities; for additional information the reader is referred to the excellent text by Coppel [7].

Observe that where $x(t)$ is continuously differentiable, the derivative of $|x(t)|$ may not exist at points where $x(t) = 0$. However, the right derivative, $(d^+/dt)|x(t)|$, which is defined below, does exist,

$$\frac{d^+}{dt}|x(t)| \triangleq \lim_{h \to +0} \frac{|x(t+h)| - |x(t)|}{h}. \tag{3.1}$$

To calculate $(d^+/dt)|x(t)|$ it is helpful to have the function sgn (x) where

$$\operatorname{sgn}(x) = 1 \quad \text{if } x > 0; \quad = -1 \text{ if } x < 0; \quad = 0 \text{ if } x = 0. \tag{3.2}$$

It then follows that

$$\frac{d^+}{dt}|x(t)| = \operatorname{sgn}(x(t)) \frac{d}{dt}x(t) \quad \text{if } x(t) \neq 0, \tag{3.3i}$$

$$\frac{d^+}{dt}|x(t)| = \left|\frac{d}{dt}x(t)\right| \quad \text{if } x(t) = 0. \tag{3.3ii}$$

The main result on differential inequalities that we will need is a result from Coppel [7]. This theorem applies to differential inequalities in which the right-hand side is restricted to be of type K (after Kamke). A vector function $\mathbf{f} = (f_1, f_2, \cdots)$ of a vector variable $\mathbf{x} = (x_1, x_2, \cdots)$ is of type K in a set S if for any two vectors in S, $\mathbf{a}$ and $\mathbf{b}$,

$$\left.\begin{aligned} a_p &= b_p \\ a_q &\leqq b_q, \ q \neq p \end{aligned}\right\} \quad \text{implies } f_p(\mathbf{a}) \leqq f_p(\mathbf{b}). \tag{3.4}$$

For the example

$$\mathbf{f}(\mathbf{x}, t) = \mathbf{M}(t)\mathbf{x} + \mathbf{v}(t) \tag{3.5}$$

where $\mathbf{M}$ is a continuous, matrix function of t and $\mathbf{v}(t)$ is a continuous vector function of t, $\mathbf{f}(\mathbf{x}, t)$ is of type K for each fixed value of t if and only if all off-diagonal elements of $\mathbf{M}$ are nonnegative. In the analysis to follow we shall be concerned with differential inequalities involving vector functions of the above kind.

Coppel [7] proves, in particular, that the following holds. Let $\mathbf{z}(t)$ be the unique solution on an interval $[a, b]$ of the equation

$$\frac{d}{dt}\mathbf{z}(t) = \mathbf{f}(\mathbf{z}, t) \tag{3.6}$$

where $\mathbf{f}(\mathbf{z}, t)$ is continuous in an open set, satisfies the Lipschitz condition [7], and is of type K for each fixed value of t. If $\mathbf{x}(t)$ is continuous on $[a, b]$, satisfies the differential inequality

$$\frac{d^+}{dt}\mathbf{x}(t) \leqq \mathbf{f}(\mathbf{x}, t) \tag{3.7}$$

on (a, b) and $\mathbf{x}(a) \leqq \mathbf{z}(a)$, then $\mathbf{x}(t) \leqq \mathbf{z}(t)$ for $a \leqq t \leqq b$.

3.2. Notation. All vectors are assumed to be column vectors. Vector transposition is denoted by the superscript $'$. For $\mathbf{x}$ a vector with a typical element denoted by x_p, let $|\mathbf{x}|$ denote the vector in which $|\mathbf{x}|_p = |x_p|$ for each p. In particular

$$\frac{d^+}{dt}|\mathbf{x}(t)| = \left\{\frac{d^+}{dt}|x_1(t)|, \frac{d^+}{dt}|x_2(t)|, \cdots\right\}'.$$

We employ the inequality convention between vectors in which $\mathbf{x} \leqq \mathbf{y}$ is equivalent to $x_p \leqq y_p$ for each p.

The above conventions are extended to matrices in the natural way. Thus $|\mathbf{M}|$ denotes the matrix obtained from the matrix $\mathbf{M}$ by replacing elements by their absolute values. Also, we let $\mathbf{I}$ denote the identity matrix and $\mathbf{e}_q$ the qth column of $\mathbf{I}$. The vector $\mathbf{1}$ is defined to have unity for all its elements.

4. Linear differential equations. The problem that we will consider here is the numerical computation of the unique solution $\mathbf{x}(t)$, $t \geqq 0$, of a linear counterpart of the equations described in § 2.1. If we let $\mathbf{F}_j(\mathbf{x}_j, t) = \mathbf{D}(j; t)\mathbf{x}_j$, and $\mathbf{G}_j(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n, t) = \sum_{k=1}^{n} \mathbf{B}(j, k; t)\mathbf{x}_k(t)$ for $1 \leqq j \leqq n$, then we obtain from (2.3) the following system of partitioned differential equations:

$$\begin{aligned} &\frac{d}{dt}\mathbf{x}_j(t) + \mathbf{D}(j; t)\mathbf{x}_j(t) = \sum_{k=1}^{n} \mathbf{B}(j, k; t)\mathbf{x}_k(t) + \mathbf{u}_j(t), \qquad t \geqq 0, \quad 1 \leqq j \leqq n, \\ &\mathbf{x}_j(0) = \mathbf{x}_{j,0}. \end{aligned} \tag{4.1}$$

The constituents of the given quantities, $\mathbf{D}$, $\mathbf{B}$, and $\mathbf{u}$, are continuous in t, $t \geqq 0$, and bounded. The matrices $\mathbf{D}(j; t)$ and $\mathbf{B}(j, k; t)$ are, respectively, N_j by N_j and N_j by N_k.

We let $D_{pq}(j; t)$, $1 \leqq p \leqq N_j$, $1 \leqq q \leqq N_j$ denote the components of $\mathbf{D}(j; t)$. Similarly, $B_{pq}(j, k; t)$, $1 \leqq p \leqq N_j$, $1 \leqq q \leqq N_k$ denote the components $\mathbf{B}(j, k; t)$.

4.1. Assumption. We let $\mathbf{M}(j; t)$ denote the N_j by N_j matrix derived from $\mathbf{D}(j; t)$ in the following manner:

$$M_{pq}(j; t) \triangleq \begin{cases} D_{pq}(j; t), & p = q, \\ -|D_{pq}(j; t)|, & p \neq q. \end{cases} \tag{4.2}$$

The N by N matrix $\mathbf{M}(t)$ is then defined in the natural way:

$$\mathbf{M}(t) = \mathbf{M}(1; t) \oplus \mathbf{M}(2; t) \oplus \cdots \oplus \mathbf{M}(n; t). \tag{4.3}$$

Thus $\mathbf{M}(t)$ is block-diagonal with nonpositive off-diagonal elements for each t.

The N_j by N_k matrix $\mathbf{N}(j, k; t)$ is derived from $\mathbf{B}(j, k; t)$: for $1 \leqq j \leqq n$, $1 \leqq k \leqq n$, $1 \leqq p \leqq N_j$, $1 \leqq q \leqq N_k$,

$$N_{pq}(j, k; t) = |B_{pq}(j, k; t)|. \tag{4.4}$$

Finally, the N by N matrix $\mathbf{N}(t)$ is defined to have $\mathbf{N}(j, k; t)$ as its (j, k)th block constituent.

In the analysis of linear differential equations we will need the following.

ASSUMPTION L1. There exists a constant N-vector $\boldsymbol{\pi}$ with all components finite and positive, and a positive constant ε such that

$$[\mathbf{M}(t) - \mathbf{N}(t)]\boldsymbol{\pi} \geqq \varepsilon \mathbf{1}, \qquad t \geqq 0. \tag{4.5}$$

COROLLARY 1. *There exists a constant* r, $0 \leqq r < 1$, *such that*

$$\mathbf{N}(t)\boldsymbol{\pi} \leqq r\mathbf{M}(t)\boldsymbol{\pi}, \qquad t \geqq 0. \tag{4.6}$$

Proof. From the prior assumption of boundedness of the constituent functions of the matrices appearing in (4.1), it follows that there exists some b, $b < \infty$, such that

$$\mathbf{M}(t)\boldsymbol{\pi} \leqq b\mathbf{1}, \qquad t \geqq 0. \tag{4.7}$$

Let $r = 1 - \varepsilon/b$, so that $0 \leqq r < 1$. With this identification it is straightforward to verify (4.6). □

COROLLARY 2. *Suppose that in* (4.1), $\{\mathbf{D}(j; t)\}$ *and* $\{\mathbf{B}(j, k; t)\}$ *are constant. Then Assumption* L1 *is equivalent to the condition*

$$(\mathbf{M} - \mathbf{N}) \text{ is an } M\text{-matrix.} \tag{4.8}$$

Proof. $(\mathbf{M} - \mathbf{N})$ is a matrix with all off-diagonal elements nonpositive. The assertion is a well-known property of such matrices [8]. (The reader will find in [9] a variety of other equivalent properties.) □

When (4.1) is autonomous, it is possible to obtain a sharp estimate of the important constant r which appears in (4.6).

COROLLARY 3. *If* $\{\mathbf{D}(j; t)\}$ *and* $\{\mathbf{B}(j, k; t)\}$ *in* (4.1) *are independent of* t, *then*

(4.9i) (i) $\rho < 1$, *where* ρ *is the spectral radius of* $\mathbf{M}^{-1}\mathbf{N}$.

(ii) *For any* $r > \rho$, *there exists a positive vector* $\boldsymbol{\pi}$ *such that*

$$\mathbf{N}\boldsymbol{\pi} < r\mathbf{M}\boldsymbol{\pi}. \tag{4.9ii}$$

Proof. The proof of (i) is in [8]. The proof of (ii) follows from well-known properties of M-matrices: $\{r\mathbf{I} - \mathbf{M}^{-1}\mathbf{N}\}$ and $\mathbf{M}$ are M-matrices and consequently their inverses exist and are nonnegative; hence the inverse of $(r\mathbf{M} - \mathbf{N})$ exists and is nonnegative. Let $\boldsymbol{\pi} = (r\mathbf{M} - \mathbf{N})^{-1}\boldsymbol{\mu}$, where $\boldsymbol{\mu}$ is any positive vector. Clearly $\boldsymbol{\pi}$ is itself positive and also (4.9ii) holds. □

Note that in (ii) of the above corollary, r may be taken to be arbitrarily close to ρ.

4.2. Asynchronous relaxations. Recall that the symbol i is used to index updates, $U(i)$ denotes the set of indices of subsystems which are updated in the ith update, and $\{\mathbf{x}_j^{(i+1)}(t),\ t \geqq 0,\ 1 \leqq j \leqq n\}$ are the computed function values on completion of the

ith update. From (2.4), (2.5) and (4.1) we see that these functions are the solutions of the equations

(4.10i)
$$\frac{d}{dt}\mathbf{x}_j^{(i+1)}(t)+\mathbf{D}(j;\,t)\mathbf{x}_j^{(i+1)}(t)=\sum_{k=1}^{n}\mathbf{B}(j,\,k;\,t)\mathbf{x}_j^{(i-d(i,j,k))}(t)+\mathbf{u}_j(t),\quad t\geqq 0\quad \text{if } j\in U(i),$$
$$\mathbf{x}_j^{(i+1)}(0)=\mathbf{x}_{j,0},$$

(4.10ii)
$$\mathbf{x}_j^{(i+1)}(t)\equiv\mathbf{x}_j^{(i)}(t),\quad t\geqq 0\quad \text{if } j\notin U(i).$$

It is easy to see that a unique solution exists to the initial value problem in (4.10i). The N_j-vector function $\mathbf{x}_j^{(i+1)}(t)$, $t\geqq 0$, is the estimate of $\mathbf{x}_j(t)$, as given by (4.1), after the ith update.

4.3. Analysis.

4.3.1. Comparison system. Define the N_j-vector $\mathbf{y}_j^{(i)}(t)$ thus:

(4.11)
$$\mathbf{y}_j^{(i)}(t)\triangleq\mathbf{x}_j^{(i)}(t)-\mathbf{x}_j(t),\qquad t\geqq 0,\quad 1\leqq j\leqq n,\quad i=0,1,\cdots.$$

From (4.1) and (4.10) it is clear that

(4.12)
$$\mathbf{y}_j^{(i)}(0)=\mathbf{0},\qquad 1\leqq j\leqq n,\quad i=0,1,\cdots.$$

Also, from (4.1) and (4.10),

(4.13i)
$$\frac{d}{dt}\mathbf{y}_j^{(i+1)}(t)+\mathbf{D}(j;\,t)\mathbf{y}_j^{(i+1)}(t)=\sum_{k=1}^{n}\mathbf{B}(j,\,k;\,t)\mathbf{y}_k^{(i-d(i,j,k))}(t)\quad \text{if } j\in U(i),$$

(4.13ii)
$$\mathbf{y}_j^{(i+1)}(t)\equiv\mathbf{y}_j^{(i)}(t),\quad t\geqq 0\quad \text{if } j\notin U(i).$$

4.3.2. Differential inequalities. Let $y_{j,p}^{(i+1)}(t)$ denote the pth constituent of the N_j-vector $\mathbf{y}_j^{(i+1)}(t)$. From (4.3i), for $j\in U(i)$,

(4.14)
$$\begin{aligned}&\{\operatorname{sgn} y_{j,p}^{(i+1)}(t)\}\frac{d}{dt}y_{j,p}^{(i+1)}(t)+D_{pp}(j;\,t)|y_{j,p}^{(i+1)}(t)|\\&\qquad+\sum_{q:q\neq p}D_{pq}(j;\,t)[\{\operatorname{sgn} y_{j,p}^{(i+1)}(t)\}y_{j,q}^{(i+1)}(t)]\\&=\sum_{k=1}^{n}\sum_{q=1}^{N_q}B_{pq}(j,\,k;\,t)[\{\operatorname{sgn} y_{j,p}^{(i+1)}(t)\}y_{k,q}^{(i-d(i,j,k))}(t)],\qquad t\geqq 0.\end{aligned}$$

It follows that

(4.15i)
$$\begin{aligned}&\{\operatorname{sgn} y_{j,p}^{(i+1)}(t)\}\frac{d}{dt}y_{j,p}^{(i+1)}(t)+D_{pp}(j;\,t)|y_{j,p}^{(i+1)}(t)|-\sum_{q:q\neq p}|D_{pq}(j;\,t)||y_{j,q}^{(i+1)}(t)|\\&\leqq\sum_{k=1}^{n}\sum_{q=1}^{N_q}|B_{pq}(j,\,k;\,t)||y_{k,q}^{(i-d(i,j,k))}(t)|,\qquad t\geqq 0.\end{aligned}$$

Also, when $y_{j,p}^{(i+1)}(t)=0$ it follows directly from (4.13i) that

(4.15ii)
$$\left|\frac{d}{dt}y_{j,p}^{(i+1)}(t)\right|-\sum_{q:q\neq p}|D_{pq}(j;\,t)||y_{j,q}^{(i+1)}(t)|\leqq\sum_{k=1}^{n}\sum_{q=1}^{N_q}|B_{pq}(j,\,k;\,t)||y_{k,q}^{(i-d(i,j,k))}(t)|.$$

Hence, from the definition of the right derivative, see (3.3) in § 3.1, it follows from (4.2), (4.3) and (4.4) that

(4.16)
$$\frac{d^+}{dt}|\mathbf{y}_j^{(i+1)}(t)|+\mathbf{M}(j;\,t)|\mathbf{y}_j^{(i+1)}(t)|\leqq\sum_{k=1}^{n}\mathbf{N}(j,\,k;\,t)|\mathbf{y}_k^{(i-d(i,j,k))}(t)|,\qquad t\geqq 0.$$

In summary, if $j \in U(i)$ then (4.16) augmented by the initial condition $|\mathbf{y}_j^{(i+1)}(0)| = \mathbf{0}$ holds, and if $j \notin U(i)$ then

$$|\mathbf{y}_j^{(i+1)}(t)| = |\mathbf{y}_j^{(i)}(t)|, \qquad t \geqq 0. \tag{4.17}$$

4.3.3. Bounds. Making use of (4.6) in Corollary 1 to Assumption L1, the right-hand side of (4.16) may be upper bounded thus:

$$\sum_{k=1}^{n} \mathbf{N}(j, k; t)|\mathbf{y}_k^{(i-d(i,j,k))}(t)| \leqq [r\{\max_{1 \leqq k \leqq n} w_k^{(i-d(i,j,k))}\}]\mathbf{M}(j; t)\boldsymbol{\pi}_j, \tag{4.18}$$

for $t \geqq 0$, where

$$w_j^{(i)} \triangleq \sup_{t \in [0,\infty)} \max_{1 \leqq p \leqq N_j} \{|y_{j,p}^{(i)}(t)|/\pi_{j,p}\}, \qquad 1 \leqq j \leqq n, \quad i \geqq 0. \tag{4.19}$$

From (4.16) and (4.18) we then have, for $j \in U(i)$, $t \geqq 0$,

$$\begin{aligned} &\frac{d^+}{dt}\{|\mathbf{y}_j^{(i+1)}(t)| - s_j^{(i+1)}\boldsymbol{\pi}_j\} + \mathbf{M}(j; t)\{|\mathbf{y}_j^{(i+1)}(t)| - s_j^{(i+1)}\boldsymbol{\pi}_j\} \leqq 0, \qquad t \geqq 0, \\ &\{|\mathbf{y}_j^{(i+1)}(0)| - s_j^{(i+1)}\boldsymbol{\pi}_j\} = -s_j^{(i+1)}\boldsymbol{\pi}_j \end{aligned} \tag{4.20}$$

where

$$s_j^{(i+1)} \triangleq r\{\max_{1 \leqq k \leqq n} w_k^{(i-d(i,j,k))}\}. \tag{4.21}$$

Now, if the differential inequality in (4.20) is stated in the form

$$\frac{d^+}{dt}\mathbf{z}(t) \leqq \mathbf{f}(t, \mathbf{z}), \qquad t \geqq 0 \tag{4.22}$$

then $\mathbf{f}$ is of type K for each t, see § 3.1. The prime reason for this is that the off-diagonal terms of $\mathbf{M}(j; t)$ are nonpositive. An application of the main result on differential inequalities stated in § 3.1 then yields

$$|\mathbf{y}_j^{(i+1)}(t)| - s_j^{(i+1)}\boldsymbol{\pi}_j \leqq \mathbf{z}(t), \qquad t \geqq 0 \tag{4.23}$$

where $\mathbf{z}(t)$ is the unique solution of the following initial value problem:

$$\begin{aligned} &\frac{d}{dt}\mathbf{z}(t) + \mathbf{M}(j; t)\mathbf{z}(t) = 0, \qquad t \geqq 0, \\ &\mathbf{z}(0) = -s_j^{(i+1)}\boldsymbol{\pi}_j. \end{aligned} \tag{4.24}$$

Since $\mathbf{z}(0) \leqq \mathbf{0}$, it is easy to see that

$$\mathbf{z}(t) \leqq \mathbf{0}, \qquad t \geqq 0. \tag{4.25}$$

Hence, from (4.23) and (4.25),

$$|\mathbf{y}_j^{(i+1)}(t)| \leqq s_j^{(i+1)}\boldsymbol{\pi}_j, \qquad t \geqq 0.$$

From the definitions of $w_j^{(i+1)}$ and $s_j^{(i+1)}$ in (4.19) and (4.21), we obtain the result which is summarized now.

PROPOSITION 1. *For the linear differential equations in* (4.1) *subject to Assumption* L1, *the asynchronous relaxations in* (4.10) *satisfy the following: for* $i = 0, 1, 2, \cdots$

$$w_j^{(i+1)} \leqq r\{\max_{1 \leqq k \leqq n} w_k^{(i-d(i,j,k))}\} \quad \textit{if } j \in U(i), \tag{4.26i}$$

$$w_j^{(i+1)} = w_j^{(i)} \quad \textit{if } j \notin U(i) \tag{4.26ii}$$

where (*see* (4.6)) $0 \leqq r < 1$, *and*

$$w_j^{(i)} = \sup_{t\in[0,\infty]} \max_{1\leqq p\leqq N_j} \{|y_{j,p}^{(i)}(t)|/\pi_{j,p}\},$$

and

$$y_{j,p}^{(i)}(t) = x_{j,p}^{(i)}(t) - x_{j,p}(t). \qquad \square$$

We will return to this result in § 6.

5. Nonlinear differential equations. We consider the equations given in (2.3), namely

$$\begin{aligned} &\frac{d}{dt}\mathbf{x}_j + \mathbf{F}_j(\mathbf{x}_j, t) = \mathbf{G}_j(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n, t) + \mathbf{u}_j(t), \qquad t \geqq 0, \quad 1 \leqq j \leqq n, \\ &\mathbf{x}_j(0) = \mathbf{x}_{j,0}, \end{aligned} \tag{5.1}$$

and analyze the convergence properties of the asynchronous relaxations described in § 2.2. In § 5.2 we give some examples of systems of equations satisfying the assumptions made on them.

5.1. Assumptions. In the notation we employ, $\mathbf{F}_j = (F_{j,1}, F_{j,2}, \cdots, F_{j,N})'$. The constituents functions in $\mathbf{G}_j$ are similarly represented.

ASSUMPTION N1. The function $\mathbf{F}(\mathbf{x}, t)$ is continuous for $t \geqq 0$ and for $\|\mathbf{x}\| < \infty$ ($\|\mathbf{x}\|$ any norm in $\mathbb{R}^N$). Furthermore, for $1 \leqq j \leqq n$, all $\mathbf{u} \in \mathbb{R}^{N_j}$ and all $\mathbf{v} \in \mathbb{R}^{N_j}$, there exist nonnegative, uniformly bounded functions $D_{pq}(j; t)$, $1 \leqq p \leqq N_j$, $1 \leqq q \leqq N_j$, continuous in t, $t \geqq 0$, such that, for $t \geqq 0$

$$\{\operatorname{sgn}(u_p - v_p)\}\{F_{j,p}(\mathbf{u}, t) - F_{j,p}(\mathbf{v}, t)\} \geqq D_{pp}(j; t)|u_p - v_p| - \sum_{q:q\neq p} D_{pq}(j; t)|u_q - v_q|, \tag{5.2i}$$

and, if $u_p = v_p$, then

$$|F_{j,p}(\mathbf{u}, t) - F_{j,p}(\mathbf{v}, t)| \leqq \sum_{q:q\neq p} D_{pq}(j; t)|u_q - v_q|. \tag{5.2ii}$$

Note that (5.2ii) is expected from (5.2i) if $(u_p - v_p)$ is perturbed about a small neighborhood of 0.

ASSUMPTION N2. The function $\mathbf{G}(\mathbf{x}, t)$ is continuous for $t \geqq 0$ and for $\|\mathbf{x}\| < \infty$. Furthermore, there exist nonnegative, continuous functions $\phi_{j,p}(t)$ such that for all $\mathbf{u} \in \mathbb{R}^N$ and all $\mathbf{v} \in \mathbb{R}^N$,

$$|G_{j,p}(\mathbf{u}, t) - G_{j,p}(\mathbf{v}, t)| \leqq \phi_{j,p}(t)\|\mathbf{u} - \mathbf{v}\|, \qquad t \geqq 0. \tag{5.3}$$

In (5.3), the norm is the weighted L_∞-norm in $\mathbb{R}^N$, i.e., for $\mathbf{x} \in \mathbb{R}^N$,

$$\|\mathbf{x}\| = \max_{1\leqq j\leqq n} \max_{1\leqq p\leqq N_j} |x_{j,p}|/\pi_{j,p} \tag{5.4}$$

where $\boldsymbol{\pi}$ is a fixed, positive, weight vector.

ASSUMPTION N3. There exists a constant ε, $\varepsilon > 0$, such that for $t \geqq 0$

$$\mathbf{M}(t)\boldsymbol{\pi} - \boldsymbol{\phi}(t) \geqq \varepsilon \mathbf{1}, \tag{5.5}$$

where the matrix function $\mathbf{M}(t)$ is constituted from the functions $D_{pq}(j, t)$ as specified before in (4.2) and (4.3), § 4.1.

This completes the assumptions that we will need in connection with the analysis of the asynchronous relaxations for nonlinear, nonautonomous differential equations.

Since $\mathbf{u}(t)$ is continuous in t, the continuity and the Lipschitz condition are satisfied in the initial-value problem in (5.1), and thus a unique solution exists [7].

Note that if we identify $\boldsymbol{\phi}(t)$ in (5.5) with $\mathbf{N}(t)\boldsymbol{\pi}$ in (4.5), the two inequalities are identical. We may therefore conclude the following, as in the first corollary to Assumption L1 in § 4.1.

COROLLARY. *There exists a constant* r, $0 \leqq r < 1$, *such that*

$$\boldsymbol{\phi}(t) \leqq r\mathbf{M}(t)\boldsymbol{\pi}, \qquad t \geqq 0. \tag{5.6}$$

□

5.2. Examples. We briefly consider some simple and typical examples of the N-vector function $\mathbf{F}(\mathbf{x}, t)$, and investigate the requirements for Assumption N1 to be satisfied and the specification of the functions $\{D_{pq}(j; t)\}$ which appear in the statement of the assumption.

5.2.1. A simple example is

$$\mathbf{F}(\mathbf{x}, t) = \mathbf{A}\mathbf{x} \tag{5.7}$$

where $\mathbf{A}$ is a constant matrix. In this case (5.2) requires that

$$A_{pp}(j) \geqq 0 \tag{5.8}$$

in which case we may satisfy (5.2) by taking

$$D_{pp}(j; t) = A_{pp}(j) \quad \text{and} \quad D_{pq}(j; t) = |A_{pq}(j)|, \quad p \neq q. \tag{5.9}$$

5.2.2. Another example is

$$\mathbf{F}_j(\mathbf{x}_j, t) = [f_{j,1}(x_{j,1}), f_{j,2}(x_{j,2}), \cdots, f_{j,N_j}(x_{j,N_j})]' \tag{5.10}$$

for $1 \leqq j \leqq n$, in which for each p, $f_{j,p}(x_{j,p})$ is continuously differentiable. In this case, in (5.2) we may take $D_{pq}(j; t) \equiv 0$ for $p \neq q$, and the requirement in (5.2) then translates to

$$\{\operatorname{sgn}(u - v)\}\{f_{j,p}(u) - f_{j,p}(v)\} \geqq D_{pp}(j; t)|u - v|, \tag{5.11}$$

for all real u and v, where $D_{pp}(j; t) \geqq 0$. Thus $f_{j,p}(u)$ must be monotonic, nondecreasing with increasing x, but note that it is not necessary that $f_{j,p}(0) = 0$. When the monotonicity condition holds, (5.2) is satisfied if we take

$$D_{pp}(j; t) = \inf_x f'_p(x) \geqq 0. \tag{5.12}$$

5.2.3. In the final example, for $1 \leqq j \leqq n$,

$$\mathbf{F}_j(\mathbf{x}_j, t) = \mathbf{A}(j)[f_{j,1}(x_{j,1}), f_{j,2}(x_{j,2}), \cdots, f_{j,N_j}(x_{j,N_j})]', \tag{5.13}$$

in which $f_{j,p}$ for each p is continuously differentiable and satisfies the Lipschitz condition, and $\mathbf{A}(j)$ is a constant matrix. It is easy to see that (5.2) is satisfied if

$$A_{pp}(j)\{\operatorname{sgn}(u - v)\}\{f_{j,p}(u) - f_{j,p}(v)\} \geqq D_{pp}(j; t)|u - v| \tag{5.14i}$$

and, for $p \neq q$,

$$|A_{pq}(j)||f_q(u) - f_q(v)| \leqq D_{pq}(j; t)|u - v| \tag{5.14ii}$$

for all real u and v.

In considering (5.14i) first, we note that $A_{pp}(j)$ may have either sign. However, necessarily, (a) if $A_{pp}(j) \geqq 0$ then $f_{j,p}(x)$ must be monotonic, nondecreasing with increasing x; (b) if $A_{pp}(j) \leqq 0$ then $f_{j,p}(x)$ must be monotonic, nonincreasing with

increasing x. That is,

$$A_{pp}(j)f'_{j,p}(x) \geqq 0 \quad \text{for all } x \text{ and } p. \tag{5.15}$$

If this condition is satisfied then, it is quite easy to see that (5.14i) and (5.14ii), and therefore (5.2), are satisfied with the following choices:

$$D_{pp}(j;\, t) = |A_{pp}(j)|S_{j,p} \quad \text{where } S_{j,p} \triangleq \inf_x |f'_{j,p}(x)|,$$

$$\text{and, for } p \neq q, \tag{5.16}$$

$$D_{pq}(j;\, t) = |A_{pq}(j)|L_{j,q} \quad \text{where } L_{j,q} \triangleq \sup_x |f'_{j,q}(x)|.$$

5.3. Asynchronous relaxations. From (2.3), (2.4) and (2.5) we have, for $i = 0, 1, \cdots$ and $1 \leqq j \leqq n$,

$$\frac{d}{dt}\mathbf{x}_j^{(i+1)}(t) + \mathbf{F}_j(\mathbf{x}_j^{(i+1)}, t) = \mathbf{G}_j(\mathbf{x}_1^{(i-d(i,j,1))}, \cdots, \mathbf{x}_n^{(i-d(i,j,n))}, t) + \boldsymbol{u}_j(t), \quad t \geqq 0, \tag{5.17i}$$

$$\mathbf{x}_j^{(i+1)}(0) = \mathbf{x}_{j,0} \quad \text{if } j \in U(i),$$

$$\mathbf{x}_j^{(i+1)}(t) = \mathbf{x}_j^{(i)}(t), \quad t \geqq 0 \quad \text{if } j \notin U(i). \tag{5.17ii}$$

Note that by our earlier assumptions, continuity and the Lipschitz condition are satisfied in the initial-value problem (5.17i) and thus a unique solution exists.

5.4. Analysis.

5.4.1. Comparison system. Define, as in (4.11)

$$\mathbf{y}_j^{(i)}(t) \triangleq \mathbf{x}_j^{(i)}(t) - \mathbf{x}_j(t), \qquad t \geqq 0, \quad 1 \leqq j \leqq n, \quad i = 1, 2, \cdots. \tag{5.18}$$

Clearly

$$\mathbf{y}_j^{(i)}(0) = \mathbf{0}.$$

From (5.1), (5.17) and (5.18),

$$\frac{d}{dt}y_{j,p}^{(i+1)}(t) + [F_{j,p}(x_j^{(i+1)}, t) - F_{j,p}(x_j, t)]$$

$$= [G_{j,p}(\mathbf{x}_1^{(i-d(i,j,1))}, \cdots, \mathbf{x}_n^{(i-d(i,j,n))}, t) - G_{j,p}(\mathbf{x}_1, \cdots, \mathbf{x}_n, t)], \tag{5.19i}$$

$$t \geqq 0, \quad y_{j,p}^{(i+1)}(0) = 0, \quad 1 \leqq p \leqq N_j \quad \text{if } j \in U(i)$$

$$\mathbf{y}_j^{(i+1)}(t) = \mathbf{y}_j^{(i)}(t), \qquad t \geqq 0 \quad \text{if } j \notin U(i). \tag{5.19ii}$$

We proceed to examine the case of $j \in U(i)$ and to obtain, with the aid of Assumptions N1 and N2, a differential inequality for $|\mathbf{y}_j^{(i+1)}(t)|$. The right-hand side of (5.19i) may be bounded by using Assumption N2:

$$|G_{j,p}(\mathbf{x}_1^{(i-d(i,j,1))}, \cdots, \mathbf{x}_n^{(i-d(i,j,n))}, t) - G_{j,p}(\mathbf{x}_1, \cdots, \mathbf{x}_n, t)|$$

$$\leqq \phi_{j,p}(t)[\max_{1 \leqq k \leqq n} w_k^{(i-d(i,j,k))}], \qquad t \geqq 0, \tag{5.20}$$

where, as in (4.19),

$$w_j^{(i)} \triangleq \sup_{t \in [0,\infty]} \max_{1 \leqq p \leqq N_j} |y_{j,p}^{(i)}(t)|/\pi_{j,p}, \qquad 1 \leqq j \leqq n, \quad i \geqq 0. \tag{5.21}$$

By multiplying (5.19i) by $\{\operatorname{sgn} y_{j,p}^{(i+1)}(t)\}$ and taking note of Assumption N1, and also

by evaluating $|(d/dt)y_{j,p}^{(i+1)}(t)|$ when $y_{j,p}^{(i+1)}(t)=0$, we obtain, analogously to (4.15),

$$\frac{d^+}{dt}|y_{j,p}^{(i+1)}(t)|+D_{pp}(j;t)|y_{j,p}^{(i+1)}(t)|-\sum_{q:q\neq p}D_{pq}(j;t)|y_{j,q}^{(i+1)}(t)|$$
$$\leqq \phi_{j,p}(t)[\max_{1\leqq k\leqq n} w_k^{(i-d(i,j,k))}], \qquad t\geqq 0. \tag{5.22}$$

To recapitulate, if $j\in U(i)$ then we have (5.22), and if $j\notin U(i)$ then

$$|\mathbf{y}_j^{(i+1)}(t)|=|y_j^{(i)}(t)|, \qquad t\geqq 0.$$

5.4.2. Bounds. We may now make use of (5.6), a corollary to Assumption N3, which bounds $\boldsymbol{\phi}(t)$, to state the right-hand side of (5.22) in a more convenient form. Thus,

$$\phi_{j,p}(t)\left[\max_{1\leqq k\leqq n} w_k^{(i-d(i,j,k))}\right]\leqq s_j^{(i+1)}\left[D_{pp}(j;t)\pi_{j,p}-\sum_{q:q\neq p}D_{pq}(j;t)\pi_{j,q}\right] \tag{5.23}$$

where

$$s_j^{(i+1)}\triangleq r\{\max_{1\leqq k\leqq n} w_k^{(i-d(i,j,k))}\}. \tag{5.24}$$

From (5.22) and (5.23),

$$\frac{d^+}{dt}\{|y_{j,p}^{(i+1)}(t)|-s_j^{(i+1)}\pi_{j,p}\}+D_{pp}(j;t)\{|y_{j,p}^{(i+1)}(t)|-s_j^{(i+1)}\pi_{j,p}\}$$
$$-\sum_{q:q\neq p}D_{pq}(j;t)\{|y_{j,q}^{(i+1)}(t)|-s_j^{(i+1)}\pi_{j,q}\}\leqq 0, \qquad t\geqq 0,$$

which in vector form is

$$\frac{d^+}{dt}\{|\mathbf{y}_j^{(i+1)}(t)|-s_j^{(i+1)}\boldsymbol{\pi}_j\}+\mathbf{M}(j;t)\{|y_j^{(i+1)}(t)|-s_j^{(i+1)}\boldsymbol{\pi}_j\}\leqq 0, \qquad t\geqq 0. \tag{5.25}$$

This differential inequality is identical to (4.20) and we may therefore conclude that

$$|\mathbf{y}_j^{(i+1)}(t)|\leqq s^{(i+1)}\boldsymbol{\pi}_j, \qquad t\geqq 0$$

and, thereby,

$$w_j^{(i+1)}\leqq r\{\max_{1\leqq k\leqq n} w_k^{(i-d(i,j,k))}\},$$

where $w_j^{(i+1)}$ is as defined in (5.21). In summary we have

PROPOSITION 2. *For the nonlinear differential equations in* (5.1) *subject to Assumptions* N1, N2 *and* N3, *the asynchronous relaxations in* (5.17) *satisfy the following: for* $i=0,1,2,\cdots$

$$w_j^{(i+1)}\leqq r\{\max_{1\leqq k\leqq n} w_k^{(i-d(i,j,k))}\} \quad \text{if } j\in U(i),$$

$$w_j^{(i+1)}=w_j^{(i)} \quad \text{if } j\notin U(i),$$

where $0\leqq r<1$ (*see* (5.6)), *and*

$$w_j^{(i)}=\sup_{t\in[0,\infty]}\max_{1\leqq p\leqq N_j}\{|y_{j,p}^{(i)}(t)|/\pi_{j,p}\}$$

and,

$$y_{j,p}^{(i)}(t)=x_{j,p}^{(i)}(t)-x_{j,p}(t). \qquad \square$$

6. Convergence of the asynchronous relaxations. The end results of the two preceding sections on, respectively, linear and nonlinear differential equations differ only in the values that may be attached to the parameter r. These results rely only on assumptions on the respective differential equations and not at all on those aspects of the asynchronous process which have to do with the relative frequencies of updates and the delays in the updates. Here we introduce these aspects and, by making use of Assumptions A1 and A2 given in § 2, give a unified proof of uniform convergence.

6.1. Convergence of the discrete, asynchronous process. From Propositions 1 and 2,

$$w_j^{(i+1)} \leqq r\{\max_{1\leqq k\leqq n} w_k^{(i-d(i,j,k))}\} \quad \text{if } j\in U(i),$$

$$w_j^{(i+1)} = w_j^{(i)} \quad \text{if } j\notin U(i).$$

Hence from Assumption A1,

$$w_j^{(i+1)} \leqq r[\max_{0\leqq\delta\leqq d}\max_{1\leqq k\leqq n} w_k^{(i-\delta)}] \quad \text{if } j\in U(i). \tag{6.1}$$

By recalling that $r<1$ we have, in particular,

$$\max_{0\leqq\delta\leqq d}\max_{1\leqq j\leqq n} w_j^{(i'-\delta)} \leqq \max_{0\leqq\delta\leqq d}\max_{1\leqq j\leqq n} w_j^{(i-\delta)}, \qquad 0\leqq i\leqq i'-1. \tag{6.2}$$

We also have

PROPOSITION 3.

$$\max_{1\leqq j\leqq n} w_j^{(i')} \leqq r[\max_{0\leqq\delta\leqq d}\max_{1\leqq j\leqq n} w_j^{(i-\delta)}], \qquad 0\leqq i\leqq i'-s. \tag{6.3}$$

Proof. From Assumption A2, subsystem j must be updated at least once in the series of consecutive updates with indices in $[i'-s, i'-1]$. Say that the last update of subsystem j in this series is indexed τ.

$$w_j^{(i')} = w_j^{(\tau+1)} \leqq r[\max_{0\leqq\delta\leqq d}\max_{1\leqq k\leqq n} w_k^{(\tau-\delta)}], \qquad \text{from (6.1)},$$

$$w_j^{(i')} = w_j^{(\tau+1)} \leqq r[\max_{0\leqq\delta\leqq d}\max_{1\leqq k\leqq n} w_k^{(i-\delta)}], \quad i\leqq\tau\leqq i'-s, \quad \text{from (6.2)}. \qquad \square$$

The above proposition has the following corollary in the form of a recursive inequality:

$$[\max_{0\leqq\delta\leqq d}\max_{1\leqq j\leqq n} w_j^{(i'-\delta)}] \leqq r[\max_{0\leqq\delta\leqq d}\max_{1\leqq j\leqq n} w_j^{(i-\delta)}], \qquad 0\leqq i\leqq i'-(d+s). \tag{6.4}$$

We now state the main result, in which the norm $\|\ \|$ in $\mathbb{R}^N$ is

$$\|\mathbf{x}\| = \max_{1\leqq j\leqq n}\max_{1\leqq p\leqq N_j} |x_{j,p}|/\pi_{j,p}.$$

PROPOSITION 4. *For* $i=0, 1, 2, \cdots$

$$\sup_{t\in[0,\infty]} \|\mathbf{x}^{(i)}(t)-\mathbf{x}(t)\| \leqq Cr^l, \tag{6.5}$$

where $0\leqq r<1$, $l=\lfloor i/(d+s)\rfloor$ *and* $C=\max_{1\leqq j\leqq n} w_j^{(0)}$.

Proof. Equation (6.4) yields

$$[\max_{0\leqq\delta\leqq d}\max_{1\leqq j\leqq n} w_j^{(i-\delta)}] \leqq Cr^l \tag{6.6}$$

where

$$C = \max_{0 \leq \delta \leq d} \max_{1 \leq j \leq n} w_j^{(-\delta)} = \max_{1 \leq j \leq n} w_j^{(0)}.$$

The statement in the proposition is implied by (6.6). □

6.2. Discussion. We may conclude from the last proposition that the long-term average rate of geometric convergence (per update) is not less than $r^{1/(d+s)}$.

We have proven uniform convergence for the proposed asynchronous relaxations under different conditions for three variants of the differential equations in (2.1):

$$\frac{d}{dt}\mathbf{x}(t) + \mathbf{D}\mathbf{x}(t) = \mathbf{B}\mathbf{x}(t) + \mathbf{u}(t), \tag{6.7i}$$

$$\frac{d}{dt}\mathbf{x}(t) + \mathbf{D}(t)\mathbf{x}(t) = \mathbf{B}(t)\mathbf{x}(t) + \mathbf{u}(t), \tag{6.7ii}$$

$$\frac{d}{dt}\mathbf{x}(t) + \mathbf{F}(\mathbf{x}, t) = \mathbf{G}(\mathbf{x}, t) + \mathbf{u}(t). \tag{6.7iii}$$

We have assumed A1 and A2, see § 2.3, in all cases. In case (i) uniform convergence at a geometric rate occurs when $\mathbf{M}^{-1}\mathbf{N}$ is an M-matrix; in this case the parameter r in the bound on the rate of convergence may be chosen to be arbitrarily close to ρ, the spectral radius or $\mathbf{M}^{-1}\mathbf{N}$, where $\mathbf{M}$ and $\mathbf{N}$ are obtained from $\mathbf{D}$ and $\mathbf{B}$ as in (4.2) and (4.4). In particular, if $\mathbf{D}$ has nonpositive off-diagonal elements and $\mathbf{B}$ is nonnegative, the $\mathbf{M} = \mathbf{D}$ and $\mathbf{N} = \mathbf{B}$.

The uniform convergence for case (ii) requires Assumption L1, see § 4.1, and case (iii) requires Assumptions N1, N2 and N3, see § 5.1. The result for case (iii) subsumes the result for case (ii).

7. Concluding remarks. We have proven convergence results for an asynchronous computational model. This model has several parameters and, by appropriate selections of these parameters, it can be made to describe algorithms with various degrees of asynchronism. Similarly, architectural differences giving rise to varying rates of execution of the primitive tasks are also tolerated in the model.

Algorithmic efficiency, synchronization penalty, the software overhead for coordination and communication congestion are major factors which will contribute towards greater use of asynchronism in algorithms. These factors become more pronounced as the number of processors is increased. The Ultracomputer [10] and the CHOPP [11] are examples of designs in which ultimately thousands of processors are proposed to be used. The asynchronous computational model and results for it should therefore prove useful in the future.

We mention that [12] describes implementations of asynchronous relaxations on a parallel processor[1] for systems with up to 64,000 tightly coupled, linear differential equations which describe the evolution with time of the state probabilities of Markov processes. The Markov processes model manufacturing lines with random service times and finite buffers. The procedure employed in [12] consists of first obtaining a good Gauss–Seidel-like relaxation for uniprocessors and then obtaining parallel asynchronous versions of it. Also described are competing implementations and related experimental results on the scheduling of tasks, the use of "windows" related to finite integration intervals, distributed convergence tests and termination, and the distributed

[1] The BALANCE™ (trademark of Sequent Computer Systems, Inc.) 8000.

computation of statistics of the Markov process. Finally, a result is given which bounds the effect of independent error functions which are introduced in the execution of each of the primitive tasks.

REFERENCES

[1] D. CHAZAN AND W. MIRANKER, *Chaotic Relaxation*, Linear Algebra Appl., 2 (1969), pp. 199-222.

[2] G. M. BAUDET, *Asynchronous iterative methods for multiprocessors*, J. Assoc. Comput. Mach., 25 (1978), pp. 226-244.

[3] D. P. BERTSEKAS, *Distributed asynchronous computation of fixed points*, Math. Programming, 27 (1983), pp. 107-120.

[4] ———, *Distributed dynamic programming*, IEEE Trans. Automat. Control, AC-27 (June 1982), pp. 610-616.

[5] B. LUBACHEVSKY AND D. MITRA, *A chaotic, asynchronous algorithm for computing the fixed point of a nonnegative matrix of unit spectral radius*, J. Assoc. Comput. Mach., 33 (1986), pp. 130-150.

[6] 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 of Integrated Circuits and Systems, CAD-1 (July 1982), pp. 131-145.

[7] W. A. COPPEL, *Stability and asymptotic behavior of differential equations*, Heath Mathematical Monographs, Boston, MA, 1965.

[8] R. S. VARGA, *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1962.

[9] M. FIEDLER AND V. PTAK, *On matrices with nonpositive off-diagonal elements and positive principal minors*, Czechoslovak. Math. J., 12 (1962), pp. 382-450.

[10] A. GOTTLIEB, R. GRISHMAN, C. P. KRUSKAL, K. P. MCAULIFFE, L. RUDOLPH AND M. SNIR, *The NYU ultracomputer—designing an MIMD shared memory parallel computer*, IEEE Trans. Comput., C-32 (Feb. 1983), pp. 175-189.

[11] L. A. COHN, *A conceptual approach to general purpose parallel computer architecture*, Ph.D. dissertation, Columbia University, New York, 1983.

[12] D. MITRA, *Implementations of asynchronous relaxations for the solution of linear differential equations on a parallel processor and related theoretical results*, in preparation.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 1, January 1987

© 1987 Society for Industrial and Applied Mathematics
012

SOLVING EQUATIONS OF MOTION ON A VIRTUAL TREE MACHINE*

W. W. ARMSTRONG†, T. A. MARSLAND†, M. OLAFSSON† AND J. SCHAEFFER†§

Abstract. The dynamic equations of motion for rigid links connected at hinges to form a tree are amenable to a parallel solution. Such a solution can potentially provide a real-time simulation of the dynamic behavior of a broad class of objects used in animation and robotics even when the number of links is large. However, the design of a parallel algorithm poses problems not encountered in the sequential, one-processor case, such as assignment of links to processes, allocation of processes to physical processors, synchronization of processes, and the reduction of communication losses. The computation time using a parallel algorithm and an unlimited number of processors is shown to grow with the height of the tree rather than with its number of links. Increased communication costs degrade performance by a factor at most equal to the fanout. The facility used to investigate these problems is a Virtual Tree Machine (VTM) multi-computer implemented on a network of autonomous VAX-11/780 computers and SUN-2 workstations, each running a UNIX operating system.‡ Although the VTM is physically connected via a local area network, its performance in this study reflects what would occur in a point-to-point interconnection of processors in a tree configuration which would be much more expensive to build and test.

Key words. dynamics of open chains, equations of motion, animation, robotics, parallel computation, ordinary differential equations

AMS(MOS) subject classifications. 65L99 70-04 70-08 70-09

1. Introduction.

Recent papers have discussed the use of dynamics for purposes of graphical simulation or animation. The thesis of Wilhelms gives a general overview of the area [1]. It has been noted that the computation time required to solve the equations of motion of a collection of rigid links, connected at hinges, by techniques such as that of Gibbs-Appel can grow as the cube or fourth power of the number of links in the system [2], making such techniques inappropriate when the number of links is large. Recently, the equations of motion of such tree linkages have been formulated in hinge-centered coordinates, and solved by a method requiring time growing linearly with the number of links [3]. In addition, this new algorithm allows a significant amount of parallelism. Indeed, the exploitation of parallel execution on several processors has the potential to reduce the growth of the computation time to being linear in the height of the tree, rather than in its number of links.

The improved performance due to parallel processing should make possible applications of dynamics to graphical simulation and animation which have heretofore been beyond the capacity of existing general-purpose computers. For example, the use of dynamics to generate frame-to-frame motions of animated characters, under real-time control of an animator, can possibly add realism and reduce costs of animation, as well as allowing some exploration of alternative possibilities by the animator. The addition of numerous "sensor" links to the dynamic system, no longer an impediment to real-time operation, has been suggested as a way to control the shape of deformable surfaces. Again, this could remove considerable burden from the animator. There are applications of the parallel simulation to computer-aided manufacturing. For example, the graphical simulation of a robot manipulator in real-time can be useful during its design as well as for the training of operators. Recently, methods of using

* Received by the editors November 25, 1985; accepted for publication (in revised form) May 14, 1986.
† Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, T6G 2H1.
§ Funding for this work was provided by the Natural Sciences and Engineering Research Council of Canada.
‡ VAX is a trademark of Digital Equipment Corporation, SUN is a trademark of Sun Microsystems, Inc., and UNIX is a trademark of AT&T Bell Laboratories.

parallel processing for manipulator control have been proposed which depend on solving a scheduling problem [4], however that work takes the desired motion as input and produces joint forces and torques to realize the motion as output. In animation, the forces and torques (or control information from which they can be derived) are the inputs, and the motion is the output. In this way, built-in limitations on forces and torques constrain the animator to create a motion which could realistically be produced by muscular action.

The main thrust of our paper is to describe a parallel solution for determining the motion of tree linkages. In § 2 the equations are given and a solution is outlined which imposes an order on the computations reflecting the interconnection of links. In § 3 we show how these computations can be mapped to a tree of processes. Details of these computations are deferred to the Appendix. The Virtual Tree Machine multicomputer which supports the parallel implementation is described in § 4. Experimental and analytical results follow.

2. The equations of motion and their solution. The linkage trees considered consist of rigid links connected at hinges. The hinges allow three rotational degrees of freedom. This corresponds to a ball and socket joint. The reason for using this type of joint is the greater ease of formulating the equations of motion without the constraints which reduce the rotational degrees of freedom. Prismatic joints, which allow translations, are not considered. A typical linkage is shown in Figure 1.

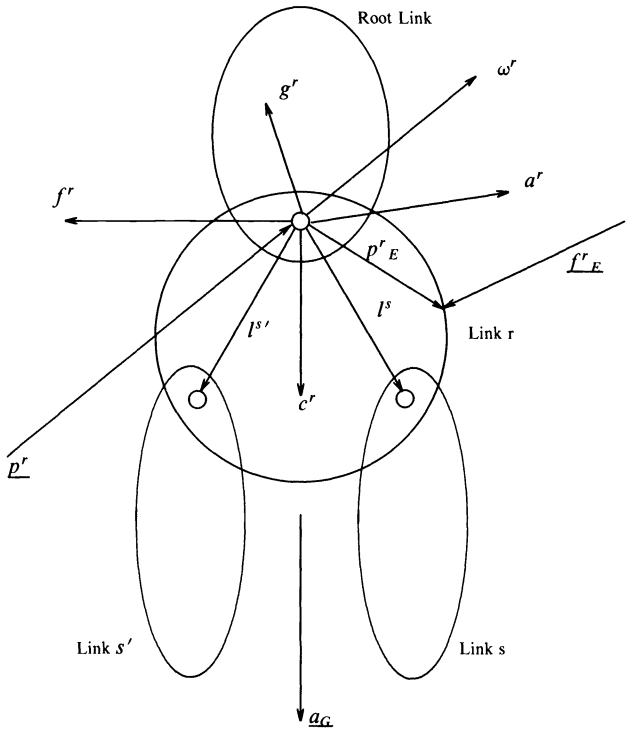


FIG. 1. *Quantities associated with link r.*

The following representations are used for physical quantities, whereby vectors, unless otherwise noted, are here represented in a frame attached to link r and moving with it: m_r is the mass of link r; a_G is the acceleration of gravity (in an inertial frame); f_E^r is an external force (inertial frame) acting on link r at the point p_E^r; g_E^r is an external torque (inertial frame) acting on link r; a^r is the acceleration of the proximal (parent) hinge of link r; ω^r is the angular velocity of link r; c^r is the vector from the proximal hinge to the center of mass of link r; f^r and g^r are the force and torque which link r exerts on its parent at the proximal hinge; l^s is the vector from the proximal hinge of link r to the proximal hinge of child s of link r. R^r converts vector representations in the frame of link r to the representations in the frame of the parent link; R_I^r converts to representations in the inertial frame; J^r is the 3×3 moment of inertia matrix of link r about its proximal hinge; S_r is the set of all links having link r as parent.

The first equation of motion, giving the rate of change of ω^r as a result of various torques, is:

$$J^r \dot{\omega}^r = g_\Sigma^r - m_r c^r \times a^r + \sum_{s \in S_r} l^s \times R^s f^s \tag{1}$$

where

$$g_\Sigma^r = -\omega^r \times (J^r \omega^r) - g^r + \sum_{s \in S_r} R^s g^s + R_I^{r\,T} g_E^r + m_r c^r \times R_I^{r\,T} a_G + p_E^r \times R_I^{r\,T} f_E^r. \tag{2}$$

The next equation of motion gives the constraint force which link r exerts on its parent at the proximal hinge:

$$f^r = f_\Sigma^r - m_r a^r + m_r c^r \times \dot{\omega}^r + \sum_{s \in S_r} R^s f^s \tag{3}$$

where

$$f_\Sigma^r = -m_r \omega^r \times (\omega^r \times c^r) + R_I^{r\,T}(f_E^r + m_r a_G). \tag{4}$$

The last equation comes from the hinge constraints:

$$R^s a^s = a_c^s + a^r - l^s \times \dot{\omega}^r \tag{5}$$

where

$$a_c^s = \omega^r \times (\omega^r \times l^s). \tag{6}$$

The hypothesis which aids in solving the equations is that there exist linear relationships

$$\dot{\omega}^r = K^r a^r + d^r \tag{7}$$

$$f^r = M^r a^r + f'^r. \tag{8}$$

In solving the equations of motion at each time step, we set the external forces f_E^r, the external torques g_E^r and the hinge torques g^r according to the control parameters, then we solve for K^r, d^r, M^r, and f'^r, starting at the leaves of the tree of links and proceeding towards the root. Intuitively, the solution method can be understood as follows. Suppose some agent accelerates a certain point on a rigid body r by an amount a^r. Then there will be a certain change of the angular velocity vector

(represented in the frame of link r) given by the derivative $\dot{\omega}^r$. At the same time, there will be a force f^r (represented in frame r) acting upon the agent. Here the agent is the parent link of link r. In the absence of any child links, the equations of motion will suffice to determine $\dot{\omega}^r$ and f^r as a function of a^r. The function will be linear as expressed in equations (7) and (8) through the coefficients K^r, d^r, M^r, and f'^r. If there are child links, then link r will act as an agent to accelerate them. Assuming that the coefficients of (7) and (8) for the child links are known, and without knowing any of the accelerations, the coefficients for link r can be computed [3]. In this manner the coefficients for all links can be computed in an *inbound* pass towards the root link. Then, since the root is not subject to forces and torques from any parent body in the tree, we can solve the equations to get the linear and angular accelerations of all the links in an *outbound* pass toward the leaves.

3. Parallel implementation. The design of a parallel solution to these equations poses problems not encountered in the sequential case. Difficulties such as synchronizing processes, process to processor allocation, and communication overhead must be addressed [5], [6]. Synchronization overhead occurs any time a processor becomes idle, waiting for others to complete their tasks. Processor allocation deals with the problem of minimizing the possibility of two or more processes trying to use the same processor at the same time. Communication overhead is the additional burden incurred by sending messages between processes.

For example, in real-time animation of the human body, the problem can be represented by a tree-like process structure with each process representing a link. In Figure 2, processes are assigned in a one-to-one fashion to each link in the stick-figure. If two processes which are neighbors in the tree are assigned to different processors (*single-link-per-process*), there is overhead involved in their exchange of data, but only slightly more than if they are assigned to the same processor. Alternatively, this overhead can be minimized by assigning several interconnected links to the same process (*multiple-links-per-process*). Both of these approaches are examined, but for simplicity the single-link-per-process model is used to explain the distributed implementation, unless otherwise stated.

No matter whether a single-link-per-process or multiple-links-per-process model is used, each cycle (time step) of the computation proceeds in three phases. In a first outbound phase, a process waits for control information from its parent, processes it, passes it on to all its descendants, and then waits. In the following inbound phase, a process must receive data from all its descendants before it can perform its

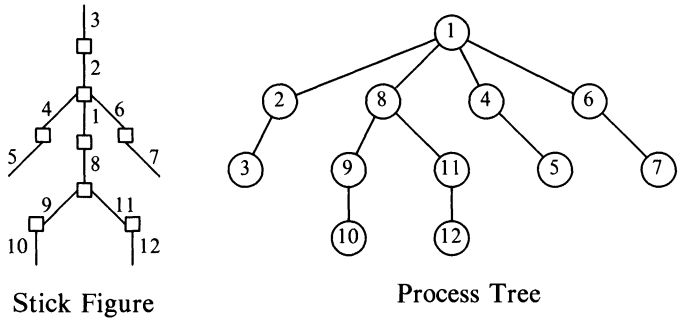


FIG. 2. *Stick figure and the corresponding process tree.*

computations and send the results back to the parent. In particular, the computation of the quantities K^r, d^r, M^r, and f'^r is done in this phase. The second outbound phase calculates the linear and angular accelerations of the links and performs part of their integration. Since not all nodes can be computing at the same time, the processes should be allocated to processors so that two processes that may be active at the same time are not on the same processor.

During initialization, the data from a non-distributed version of the dynamics program are read in and converted into the corresponding data for the distributed version. Following initialization, the dynamic equations are solved using a time step δt, whereby certain quantities are updated only every ν time steps. The functions which update the latter quantities belong to what we shall call the "slowband" computations, and the rest will be said to belong to the "fastband." The state of the simulation is graphically displayed every μ cycles.

It is worth noting at this point that the difference of execution times during fast- and slowband cycles means that an assignment of links to processes and processes to processors which is optimal for execution of the fastband may cause synchronization problems for the slowband, and vice versa. This would considerably complicate an attempt to formulate and solve the mathematical performance optimization problem.

A complete simulation cycle during the step δt, including the slowband functions and a display call, which are really only done periodically, consists of the following sequence of calls by each link process:

Outbound control pass	Inbound pass	Outbound pass
slowband_control_out	receive_data_in	receive_data_out
fastband_control_out	slowband_integration_in	fastband_out
send_control_out	slowband_in	fastband_integration_out
	fastband_in	send_data_out
	send_data_in	

An optional pass can be done during initial testing runs to verify the correctness of the solution of the equations by using the fastband_check procedure.

The control functions transmit data from a control process through the root to the various links. Again some "control" quantities may be updated in the slowband: *slowband_control_out*. For example there can be certain external forces and torques on links which vary slowly (like wind pressure). Other "control" quantities vary quickly (relative to δt), such as the forces of contact with the earth or other objects, and the torques generated by muscular action. These are updated in *fastband_control_out*. We can model contacts with the earth or with fixed or moving objects using springs and dampers. In this way the contact does not need to be considered as a constraint, which would require reformulation of the equations of motion, and perhaps loss of the efficient solution technique. In order to compute contact forces and to display the links, the root must have the orientations of all links as determined by the rotation matrices. The latter can be used to compute accurate positions and orientations of all links with respect to the root link. Passing such information inward is done by *receive_data_in*.

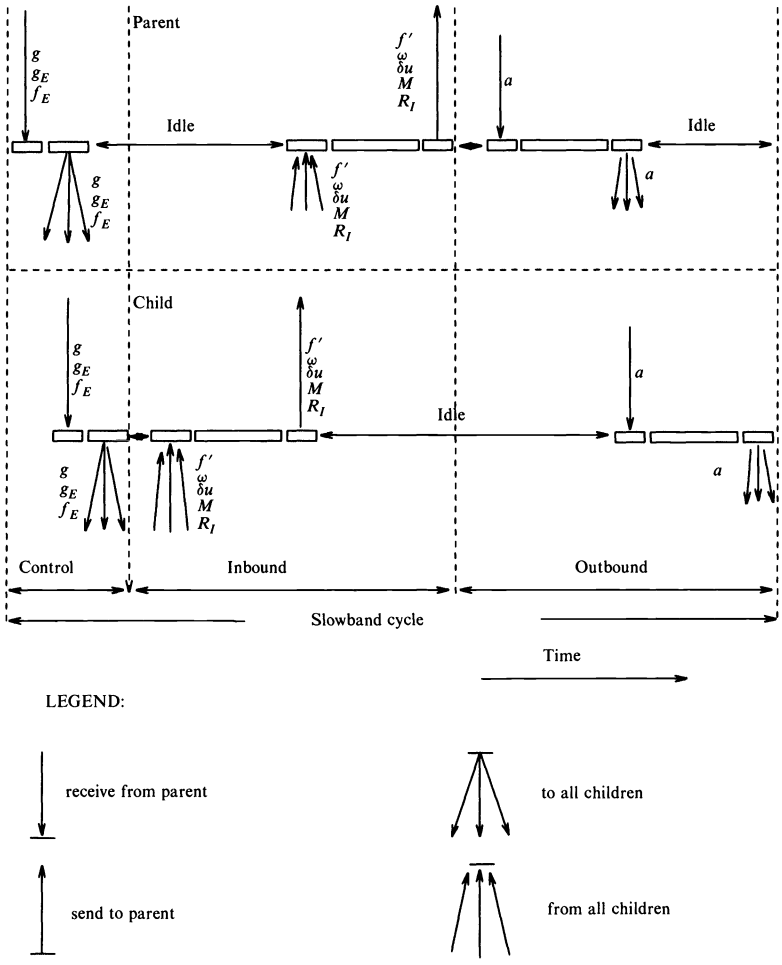


FIG. 3. *Quantities communicated between links.*

The three functions *slowband_in*, *fastband_in* and *fastband_out* perform the algebraic solution of the equations of motion. Given control forces and torques applied externally and the torques at the hinges generated by the muscles or motors, the linear accelerations of the hinges and the rates of change of the components of the angular velocity vectors of the links are determined. The *fastband_check* function determines whether the solution performed by those three functions is correct or not. The current simple integration technique in *fastband_integration_out* assumes that the previously mentioned linear and angular accelerations will be constant over the next time step. The accelerations are multiplied by δt and added to the current angular and linear velocities to get the new ones, and then the velocities are integrated again to get infinitesimal rotation vectors and positions. The *slowband_integration* call converts the infinitesimal rotation vectors into rotation matrices. To prevent accumulation of

errors, the positions of the links are all derived from the root position using the rotation matrices. However, if positions are needed in the fastband for control purposes, this can be done using the results of *fastband_integration_out*. Figure 3 is a schematic view of the communications between an arbitrary interior node in the process tree, and one of its children.

4. The virtual tree machine. The facility used to solve the equations of motion is a *multi-computer* called the Virtual Tree Machine [7]. It is implemented on a network of autonomous VAX-11/780's and SUN-2 processors each running the 4.2BSD UNIX operating system [8]. The Virtual Tree Machine consists of processes under operating system control and communication paths implemented as virtual connections between processes over a local area network. The experimenter views the machine as a collection of processors (interconnected to form a tree) - each with its own local memory and peripherals. In reality, a VTM is a collection of procedures, callable from application programs, and a collection of servers, responsible for the creation of the nodes in the tree-machine according to the description provided by the user. From this description the environment is created automatically with no intervention by the user. No restriction is placed on the mapping between virtual processing elements and physical processors. During development, the whole machine might reside on one physical processor before being distributed over the selected physical machines for productive use.

The Virtual Tree Machine makes it possible to investigate issues such as synchronization, allocation and communication overhead and thus can provide valuable insight into the behavior of unavailable real machines. For example, one such machine has tree structure and high parallelism and may prove suitable for the dynamics computations [9]. Algorithms for execution on these architectures can be developed, tested and debugged using this facility, and their synchronization and communication delay properties can be studied.

5. Experimental results. Several 10-second simulations of the man-like stick figure depicted in Figure 2 were carried out. Both the *single-link-per-process* and the *multiple-link-per-process* implementations were tried. The single-link-per-process model assigns each link to its own process and then assigns processes on a path to a leaf to the same processor P_i. One such assignment is shown in Figure 4. The advantage of this method is that it ensures that information necessary for calculations within each link is made available as soon as possible. Thus, the flow of data up and down the longest path from root to leaf of the process tree determines the length of the computation cycle. The disadvantage is that the cost of communicating all the needed information between processes is high.

The solid line of Figure 5 depicts the time needed for simulation of the 12 link figure. The figure's "torso" is selected as the root of the process tree to minimize the depth of the tree and execution starts with the root reading the control information and passing it on down the tree. The children are idle until this information arrives. When the control information has propagated down to the leaves (the head and lower arms and legs of the figure) the inbound pass is started. Calculations for the inbound pass are made first in the leaf nodes and the results passed up to their parents which begin their inbound calculations only after receiving the inbound data from all their children. Once the root link has received all the inbound data from its subtrees and performed its own inbound calculations the outbound pass is started by calculating the outbound data and sending it to all the child links. Refer to Figure 3 for a graphical

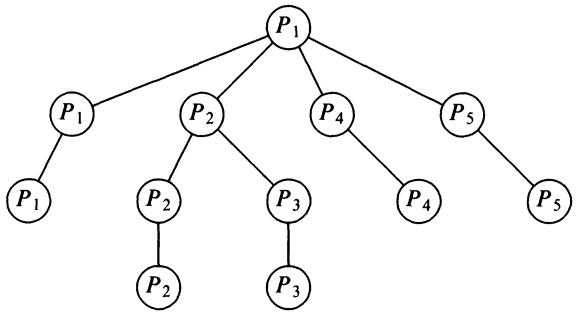


FIG. 4. *Assignment of links to twelve processes on five processors.*

representation of the communication and synchronization characteristics of this implementation.

In our simulation the time-step is set to 0.01 second. The execution time is plotted against various degrees of parallelism, from all processes assigned to one physical machine to the highest degree of useful parallelism (5 processors). For the optimal configuration of 5 processors, the execution time is reduced to 148 seconds from 227 seconds in the sequential case. It is not clear from these results whether it is the synchronization overhead or the communication overhead that plays the major role in the total overhead experienced. (The total overhead is over 100 seconds, comparing the sequential case to the single-link-per-process case when all processes run on a single machine.)

The multiple-links-per-process model eliminates the need for communication between links on the same branch by allowing each process to handle more than one link. Several links are assigned to each process, and each process is assigned to a different processor. Figure 6 shows one such assignment. Links are grouped into five processes and each process is assigned to a separate processor P_1 through P_5. Thus, only the solid lines in Figure 6 represent actual communication links. Here,

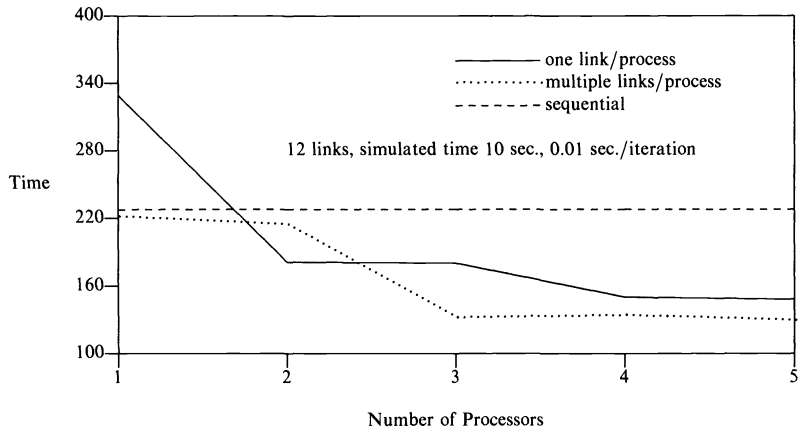


FIG. 5. *Execution time vs. number of physical processors.*

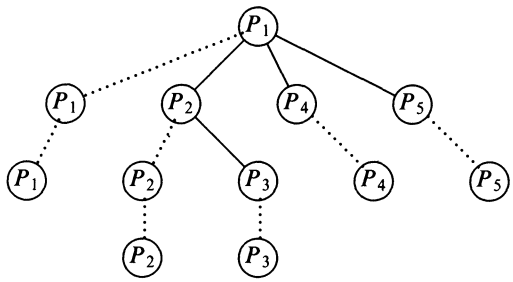


FIG. 6. *Assignment of links to five processes on five processors.*

calculations are done for all links assigned to a process before any information is sent on to its child processes or its parent. Although this reduces the communication overhead, the synchronization overhead increases as each link must wait slightly longer for the data from its parent or its children (if assigned to a different processor). The results from the multiple-links-per-process implementation are shown as the dotted line in Figure 5. The reason for the small drop in speed going from one process to two, is the fact that calculations for all links that share a process with the root link are done before any results are sent to the process handling the remaining links, resulting in almost sequential execution. For the 3 process case, the root link is given to a separate process and the remaining links are divided between the other two processes, resulting in close to parallel execution for all but one link.

Comparison of the results from these two implementations shows that the synchronization overhead rather than communication overhead is the principal cause for the lack of performance of the parallel implementations. (Eliminating much of the communication does not result in significantly better performance.) We expect that simulating several similar stick figures at once, as would be required for certain animated sequences, would reduce the effect of the synchronization overhead and permit better speedup with nearly full processor utilization. In the next section we will analytically derive formulas for estimating the maximum speedup. This will lead to a better understanding of how to use parallelism.

6. A simple model for estimating maximum speedup. To obtain an upper bound on the possible relative improvement of the VTM implementation over the sequential implementation (the "speedup"), given arbitrarily many processors, we shall consider a simple case: balanced trees of height h and fanout f. We suppose that the first control pass is nonexistent, and that the inbound pass mirrors the outbound pass as far as time is concerned, so that we can limit consideration to just the outbound pass. The speedups for the two passes are different in our case, but dealing with the synchronization of communications in both passes at once is more complex, and so we shall use the more expensive of the two passes for our estimate. We assume further that the execution time for one link is equal for all links and is t_E. Similarly all communications from a link to a child take time $t_C \leq t_E$.

Computation in the outbound pass of a tree of height $h + 1$ starts at the root link, and takes time t_E. Then the root successively communicates to $f - 1$ of its f children the information they require, taking time $(f - 1)\, t_C$ to do it. The same processor goes on to compute one of the subtrees of height h, while the other subtrees of height h either have started, or are just starting. If an upper bound on the total

time to complete the computation for a tree of height h and fanout f is denoted by $T(h, f)$, then we have

$$T(h + 1, f) = t_E + (f - 1)\, t_C + T(h, f).$$

Since $T(1, f) = t_E$, we get the general formula

$$T(h, f) = t_E + (h - 1)(t_E + (f - 1)\, t_C).$$

The number of nodes in the tree is $(f^h - 1) / (f - 1)$, and in a sequential version the computation time would be that number times t_E. Thus the speedup resulting

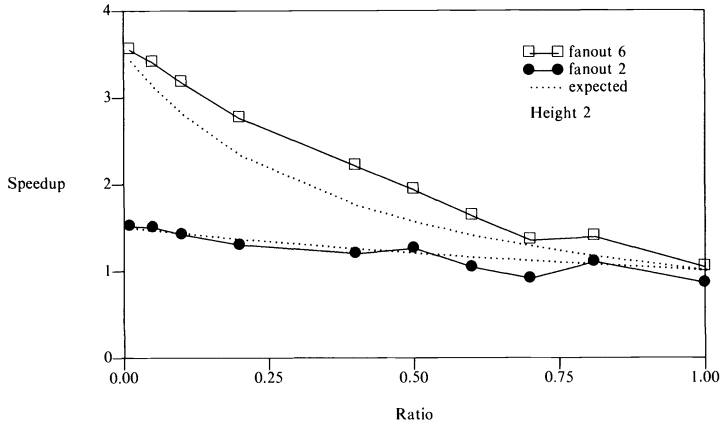


FIG. 7. *Observed speedup vs. communication to execution time ratio* (t_C/t_E).

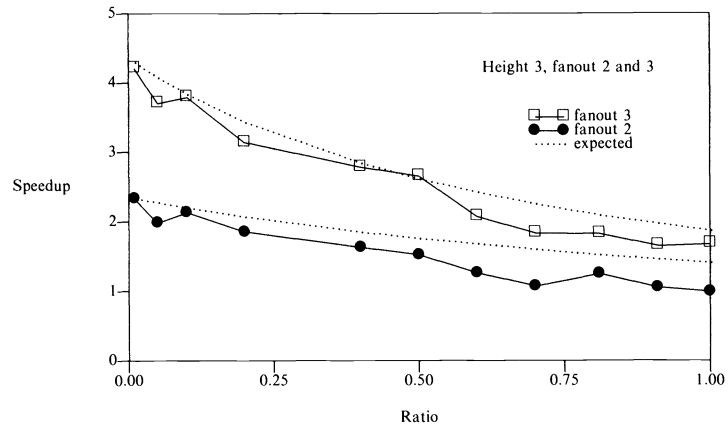


FIG. 8. *Observed speedup vs. communication to execution time ratio* (t_C/t_E).

from parallel computation is given by

$$\frac{f^h - 1}{(f - 1)\,[\,1 + (h - 1)(1 + (f - 1)\,t_C / t_E\,)]}.$$

For zero communication time, this becomes

$$\frac{(f^h - 1)}{(f - 1)\,h}.$$

When $t_C = t_E$, we get the maximum speedup

$$\frac{(f^h - 1)}{(f - 1)[\,1 + (h - 1)f]}.$$

If $h = 2$, there is no speedup at all in the latter case.

The computation time using the parallel algorithm and an unlimited supply of processors is thus theoretically shown to grow with the height of a balanced linkage tree rather than with its number of links. Considering communication times between links even as large as the execution times for the links worsens this improvement by a factor at most equal to the fanout.

Experiments were done on balanced trees using the VTM. The execution time was varied by adjusting a waiting time t_E, while t_C was measured. The observed speedups shown in Figures 7 and 8 are close to the above theoretical predictions. If we examine the speedup which could be obtained with the tree representing our stick-figure (Figure 2) by using parallelism, we get $12t_E$ compared to $4(t_E + t_C)$, for a speedup of $3/(1 + t_C/t_E)$. Averaged over both the in- and the outbound pass for the stick figure, the ratio t_C/t_E is about 0.7 which, according to the above formula, gives maximum speedup of 1.76 with five processors. This is also in close conformity to our observations.

For animation purposes, we may want to simulate several stick-figures at once, say four of them. Then t_C/t_E would be reduced by a factor of four, assuming that the cost of communication were unchanged. Then the speedup, by the above, would be 2.55, much closer to the theoretical limit of three. The synchronization overheads causing the nonlinear speedup (of three using five processors) remain, and could be dealt with by running several stick figures out of phase, so that the idle times of one stick-figure are used by others. For example, for two-stick figures on six processors, the synchronization overhead disappears if the link execution times are equal.

7. Conclusions. The parallel solution of the equations of motion of a tree-linkage using the VTM has been successful in providing a significant speedup. The speedup is a function of the tree size and configuration. The computation time using the parallel algorithm was shown to grow with the height of a balanced linkage tree rather than with its number of links. Considering communication times between links even as large as the execution times for the links worsens this exponential speedup by a factor at most equal to the fanout.

Two steps were taken which subsequently proved to be beneficial in reducing the effects of communication overheads: 1) several links were handled by a single Unix process, avoiding interprocess as well as interprocessor communications, and 2) information was not sent as soon as it became available within a process but was accumulated and sent in a single communication at the end of an iteration cycle (even though this runs counter to a dataflow philosophy).

The example of the 12-link human stick-figure which has low fanout, a height of four, and communication times which are comparable to the execution times, illustrates a case of poor speedup due to both communication and synchronization overheads. The simultaneous, in-phase simulation of several stick-figures at once would give a speedup near the maximum of three using five processors, since this reduces the ratio of communication time to execution time. To reduce synchronization overhead and approach linear speedup, several figures could be simulated out of phase so the idle times for one are used by the others.

The VTM implementation was, of course, much easier to implement and test than a physical network of processors would have been; yet the kinds of communication and synchronization problems closely reflected what could be expected in a "real" tree machine. Because there is negligible contention in the local area network while solving this problem, the VTM emulates a multiprocessor machine with point-to-point connections except for the time to form packets in the processes and ship them off serially. With more voluminous computations in the processes, and communications which better utilize the packet size, the approximation to point-to-point communications would be much closer than has been observed in the case of the stick-figure.

Many computing facilities currently use a local area network similar to the one on which the VTM is based, and hence, with the aid of VTM software, or its equivalent, it becomes possible to study parallel algorithms on networks of processors without resorting to actual construction of hardware.

Appendix: Details of the dynamics computations. The details of the dynamics computations for the VTM implementation are outlined here. No attention is paid to the control pass, since that could vary considerably depending on how control is to be achieved. In the following, the unit 3-by-3 matrix will be denoted by **I**. The tilde operation on a 3-vector v, with components $v1$, $v2$, $v3$, produces a 3-by-3 matrix V such that for any 3-vector w, the vector $v \times w$ is equal to the product of V and the column-vector w.

$$\tilde{v} = V = \begin{bmatrix} 0 & -v3 & v2 \\ v3 & 0 & -v1 \\ -v2 & v1 & 0 \end{bmatrix}.$$

The following variables are also introduced in the solution since they appear as common subexpressions or are computed in the slowband and used in the fastband: W^r, T^r, Q^r, a_c^r, f'''^r.

The *receive_data_in* function gets f'^s, δu^s (which is an infinitesimal rotation vector), and ω^s from all children s of link r. In addition, during a slowband pass, *receive_data_in* gets M^s, and R_j^s. The force f'^s is the part of the force exerted by the child s on the proximal hinge which is independent of the child's acceleration. The difference between δu^s and δu^r (converted to frame s) can be used in determining a torque due to rapid changes in the angles at hinge s deriving from hinge stiffness. The difference between the omegas (the parent one converted) is similarly used for computing a frictional torque at that hinge. These quantities are treated as fast-varying, although ω, whose rate of change depends on a physical inertia, should vary slowly and could probably be placed in the slowband. (N. B. This points out a need for careful study of the rates of change of various quantities which could lead to improved simulations involving more than two bands. In such simulations, one might

also determine, either beforehand or adaptively, that certain terms in our solution can be entirely neglected.)

The *slowband_integration_in* computation takes the infinitesimal rotation vector δu^r and uses it to update the rotation matrix R_I^r, resulting in a new link orientation. The result must be orthonormalized to prevent accumulation of errors.

The *slowband_in* computation for link r proceeds as follows:

Compute $a_c^s = \omega^r \times (\omega^r \times l^s)$ for $s \in S_r$. (9)

Compute $Q^s = R^s M^s R^{s\,T}$ for $s \in S_r$. (10)

Compute $W^s = \tilde{l}^s Q^s$ for $s \in S_r$. (11)

Compute $T^r = (J^r + \sum_{s \in S_r} W^s \tilde{l}^s)^{-1}$. (12)

Compute $K^r = T^r (\sum_{s \in S_r} W^s - m_r \tilde{c}^r)$. (13)

Compute $M^r = (m_r \tilde{c}^r) K^r - m_r \mathbf{I} + \sum_{s \in S_r} Q^s (\mathbf{I} - \tilde{l}^s K^r)$. (14)

Compute $g_\Sigma^{1r} = - \omega^r \times (J^r \omega^r) + R_I^{r\,T} g_E^r$
$+ p_E^r \times R_I^{r\,T} f_E^r + (m_r c^r) \times R_I^{r\,T} a_G$ (15)

(assuming f_E^r and g_E^r are slowly-varying).

Compute $f_\Sigma^r = - \omega^r \times (\omega^r \times (m_r c^r)) + R_I^{r\,T} (f_E^r + m_r a_G)$ (16)

(assuming f_E^r is slowly-varying).

The *fastband_in* computation proceeds as follows:

Compute $f''^s = R^s f'^s$ for $s \in S_r$. (17)

Compute $g_\Sigma^r = g_\Sigma^{1r} - g^r + \sum_{s \in S_r} R^s g^s$. (18)

Compute $d^r = T^r (g_\Sigma^r + \sum_{s \in S_r} l^s \times (f''^s + Q^s a_c^s))$. (19)

Compute $f''^r = f_\Sigma^r + (m_r c^r) \times d^r$
$+ \sum_{s \in S_r} (f''^s + Q^s (a_c^s - l^s \times d^r))$. (20)

The *send_data_in* function, which occurs last in the inbound pass, provides the data to the parent for its *receive_data_in* function. In the multiple-links-per-process version, this data is, of course, not sent or received, but merely written to and accessed from the shared memory possible when only one process is involved.

Now we turn our attention to the outbound pass. The first function there is *receive_data_out*. All it does for link r is get a^r from the parent. If r is the root, this quantity must instead be computed, which is done in the following computation (equation 21). The *fastband_out* computation for link r proceeds as follows:

If r is the root then compute $a^r = - (M^r)^{-1} f''^r$. (21)

Compute $\dot{\omega}^r = K^r a^r + d^r$. (22)

Compute $a^s = R^{s\,T} (a_c^s + a^r - l^s \times \dot{\omega}^r)$ (23)

for all children of link r.

In *fastband_integration_out*, $\dot{\omega}^r$ is multiplied by δt and the result added to ω. Then δu^r is obtained from a further integration step. The velocity and position of the root link are similarly determined by integration starting from a^1. Recall that other links have their positions determined by the root's position and link orientations. This integration technique is not stable, and development of some efficient "implicit" technique would be very useful. For animated figures, accuracy is not the goal, only realism, and the present method should be satisfactory. In an interactive system, the animator will correct for inaccuracies. Applications in robotics or biomechanics require better accuracy and stability. To the best of our knowledge, how to achieve this is an open question.

The *send_data_out* procedure merely sends a^s to all children of r. The *fastband_check* procedure goes as follows:

Receive f^s from all children.

Compute $f^r = M^r a^r + f'^r$. (24)

Compute $\dot{\omega}^r$ from equations 1 and 2, and check that it agrees with the solution value.

Compute f^r from equations 3 and 4, check that it agrees with the value computed in (24), and send it to the parent link.

REFERENCES

[1] J. WILHELMS, *Graphical simulation of the motion of articulated bodies such as humans and robots, with particular emphasis on the use of dynamic analysis*, Ph.D. Thesis, Computer Science Division, Department of Electrical Engineering and Computer Sciences, Univ. California, Berkeley, 1985.

[2] J. WILHELMS and B. BARSKY, *Using dynamic analysis to animate articulated bodies such as humans and robots*, Proc. Graphics Interface '85, Montreal (May 1985), pp. 97-104.

[3] W.W. ARMSTRONG and M.W. GREEN, *The Dynamics of articulated rigid bodies for purposes of animation*, The Visual Computer, 1 (1985), pp. 231-240 .

[4] H. KASAHARA and S. NARITA, *Parallel processing of robot-arm control computation on a multimicroprocessor system*, IEEE Journal of Robotics and Automation, RA-1 (1985), pp. 104-113 .

[5] J. MOHAN, *A Study of parallel computations - the traveling salesman problem*, Tech. Rep. CMU-CS-82-136, Dept. Comput. Sci. Carnegie Mellon Univ. (1982).

[6] T.A. MARSLAND and F. POPOWICH, *Parallel game-tree search*, IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-7 (1985), pp. 442-452 .

[7] M. OLAFSSON and T.A. MARSLAND, *A* UNIX *based virtual tree machine*, CIPS Congress '85, Montreal (June 1985), pp. 176-181.

[8] D. M. RITCHIE and K. THOMPSON, *The* UNIX *time-sharing system*, Bell Sys. Tech. J., 57 (1978), pp. 1905-1929 .

[9] *Myrias* 4000 *System Description*, Myrias Research Corporation, Edmonton, February 1986.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 1, January 1987
© 1987 Society for Industrial and Applied Mathematics
013

DESIGN AND ANALYSIS OF PARALLEL MONTE CARLO ALGORITHMS*

V. C. BHAVSAR† AND J. R. ISAAC‡

Abstract. This paper demonstrates that the potential of intrinsic parallelism in Monte Carlo methods, which has remained essentially untapped so far, can be exploited to implement these methods efficiently on SIMD and MIMD computers. Two basic static and dynamic computation assignment schemes are proposed for assigning the primary estimate computations (PECs) to processors in a parallel computer. These schemes can be used to design parallel Monte Carlo algorithms for many applications. The time complexity analyses of static computation assignment (SCA) schemes are carried out using some results from order statistics, whereas those of dynamic computation assignment (DCA) schemes are carried out using results from order statistics, renewal and queueing theories. It is shown that for smaller number of processors, linear speedup can be achieved with the SCA schemes and the speedup almost equal to the number of processors can be achieved with the DCA schemes. Some computational results for Monte Carlo solutions of Laplace's equation are given to illustrate the performance of the various SCA and DCA schemes.

Key words. Monte Carlo methods, parallel algorithms, parallel computers, analysis of algorithms, order statistics, renewal theory, pseudorandom number generators, partial differential equations

AMS(MOS) subject classifications. 35J05, 60J15, 60K05, 60K10, 62E25, 62G30, 65C05, 65W05, 68Q25

1. Introduction.

Monte Carlo methods are widely used in many scientific and engineering applications [11], [12], [18], [19], [36], [37]. In these methods, the solution of a problem is represented as a parameter of a hypothetical population. Then, a random sequence of numbers is used to construct a sample of that population from which statistical estimates of the parameter are obtained [18]. The solution estimates converge slowly (expected error is proportional to $(1/\sqrt{K})$, where K is the sample size) to the exact solution and therefore a large sample size (K equal to 10^3 to 10^5 is quite common) is required to obtain a reasonable accuracy. The estimate computations often involve random walk processes, with some random walks consisting of a large number of steps. As a consequence of these two characteristics, these methods require large computing time when implemented on sequential computers and they have not been very attractive. In this paper, we present some techniques of implementing these methods on parallel computers with a motivation to reduce their execution time and we hope that this will make these techniques more attractive in the future. Recently, Monte Carlo simulation has been identified as one of the most challenging areas for the application of parallel computers [20].

Parallel algorithms can be developed by exploiting the intrinsic parallelism in sequential algorithms. Intrinsic parallelism in Monte Carlo methods at the level of primary estimate computations (PECs) was first indicated, surprisingly, as early as in 1949 by Metropolis and Ulam in their paper defininig the Monte Carlo method [30]. Parallelism at this level, as also within the computations of primary estimates, was exploited only to some extent in [33], [34] to solve partial differential equations (PDEs) on special purpose parallel computers. Thus, the potential of intrinsic parallelism in Monte Carlo methods had remained essentially untapped in their implementation on parallel computers, until our recent work [2]–[10]. Recently, the application of ICL

* Received by the editors November 20, 1985; accepted for publication (in revised form) March 28, 1986.
† School of Computer Science, University of New Brunswick, Fredericton, New Brunswick, Canada E3B 5A3.
‡ Department of Computer Science and Engineering, Indian Institute of Technology, Bombay, 400076, India.

DAP in solving some problems in solid-state and elementary particle physics has been reported [12].

In [2], we suggested the use of an SIMD computer, based on bit-slice microprocessors, to implement Monte Carlo methods for solving PDEs. In [3] simulation results about the effectiveness of SIMD and MIMD computer architectures for solving PDEs by Monte Carlo methods were presented. We have given some parallel algorithms for Monte Carlo solutions of linear equations in [4]. The Monte Carlo methods for solving PDEs were reconsidered in [5] and the simulation results of a parallel algorithm based on an idea by Rosin [33] were presented. This preliminary work culminated in the development of a unified treatment for the design and analysis of parallel Monte Carlo algorithms given in [6]. Recently, we have also considered the problem of implementing Monte Carlo algorithms with VLSI technology [9], [10]. In this paper, we detail some of the ideas presented in earlier papers and present the unified treatment given in [6].

This paper is organized as follows. In § 2, we briefly discuss the computational characteristics of Monte Carlo methods and show that the primary estimate computation (PEC) times are independent and identically distributed (i.i.d.) random variables. This crucial result is shown to have significant influence on the design and analysis of parallel Monte Carlo algorithms in § 3. We propose two basic schemes for assigning PECs to processors in a parallel computer: static computation assignment (SCA) and dynamic computation assignment (DCA) schemes. These schemes can be used to design parallel algorithms for any Monte Carlo method. In § 4, we describe the various SCA schemes. The details of the parallel algorithms are not included for the sake of brevity and they are given in [6]. The time complexity analyses of parallel Monte Carlo algorithms based on these schemes are carried out using some results from order statistics. Two DCA schemes are presented in § 5. The time complexity analyses of these schemes are carried out using results from order statistics, renewal and queueing theories. In § 6, we consider the standardized exponentially distributed PEC times and derive the time complexity results for the SCA and DCA schemes. We consider the problem of solving Laplace's equation by a Monte Carlo method in § 7. Some computational results illustrating the performance of the various schemes are presented. Concluding remarks are given in § 8.

2. Sequential Monte Carlo algorithms.

2.1. The Monte Carlo method. The Monte Carlo method is generally defined as representing the solution of a problem as a parameter of a hypothetical population, and using a random sequence of numbers to construct a sample of the population, from which statistical estimates of the parameter can be obtained [18]. In practice, usually, the required solution consists of a real number, which is expressed as the expected value of some random variable X with finite variance on a probability space. A set of points $\zeta_1, \zeta_2, \cdots, \zeta_K$ are sampled *independently* from the probability space, commonly using pseudorandom or quasirandom number generators. In our discussion, we will assume that "ideal" random number generators are used. A set of primary estimates, $X(\zeta_1)$, $X(\zeta_2), \cdots, X(\zeta_K)$ are obtained. The secondary estimate of the solution is obtained by the arithmetic mean

$$Y = Y(\zeta_1, \zeta_2, \cdots, \zeta_K) = \frac{1}{K} \sum_{i=1}^{K} X(\zeta_i), \tag{1}$$

which converges to the required solution in quadratic mean, in probability, and with probability one, as $K \to \infty$. The sampling of each point ζ_i from the population of the

probability space often involves lengthy computations, such as tracing a random walk in an intricate and highly inhomogeneous space. In these cases, determination of a primary estimate usually consists of a run through either an absorbing or an ergodic Markov chain. For example, such computations are involved in Monte Carlo solutions of linear algebraic equations [18], partial differential equations [34], and many other applications [11], [12], [37].

2.2. Time complexity of sequential Monte Carlo algorithms. In general, the primary estimate computations (PECs) have the following characteristics.

C1. They are essentially *independent* of each other because $\zeta_1, \zeta_2, \cdots, \zeta_K$ are sampled independently from the probability space using ideal random number generators.

C2. They consist of the execution of *identical* steps.

C3. Their execution times are, in general, random variables because: (i) they may consist of a run through an absorbing Markov chain (i.e., a random walk) and the number of state transitions determines their execution times, and (ii) the number of comparisons required to make decisions based on a sample from random number generators is variable.

LEMMA 1. *The* PEC *execution times are independent and identically distributed* (*i.i.d.*) *random variables.*

Proof. The PEC execution times are random variables due to C3. The property of independency follows from C1 and identicalness follows from C2. □

This is a very crucial result in the design and analysis of parallel Monte Carlo algorithms, as discussed in § 3.

Now, let μ and σ^2 denote the mean and variance of the i.i.d. PEC times. Also, let $T(1, K)$ denote the time complexity of a sequential Monte Carlo algorithm in computing K primary estimates. Since in many applications of the Monte Carlo method the secondary estimate computation time turns out to be much smaller than the PEC times, in the ensuing analysis we will not take into account this time; if this time is not negligible in some cases, the time complexity results given in this paper can be trivially modified, as shown in [6].

THEOREM 1. *The time complexity of a sequential Monte Carlo algorithm carrying out* K PECs, $T(1, K)$, *has the expectation* $E[T(1, K)] = K.\mu$, *and the variance* Var $[T(1, K)] = K.\sigma^2$.

Proof. Follows directly from Lemma 1. □

Thus, it is seen that a sequential Monte Carlo algorithm has $O(K)$ expected time complexity and also $O(K)$ time complexity variance. In practice, it is required to choose a large value of K (10^3 to 10^5 is not uncommon) in order to attain a reasonable accuracy and hence by the Central Limit theorem it follows that $T(1, K)$ will tend to follow a normal distribution with mean $(K.\mu)$ and variance $(K.\sigma^2)$. The cumulative distribution function of a sequential Monte Carlo algorithm can be easily found out by using Theorem 1 and the techniques discussed in [6], [32].

3. Design and analysis of parallel Monte Carlo algorithms—approach and assumptions. In this section, we discuss our approach for the development of parallel Monte Carlo algorithms. First, we detail the intrinsic parallelism in Monte Carlo methods, identified in our earlier papers [2]-[5] and discuss its implications in the design of the parallel algorithms and formulate our approach. We introduce two basic schemes for the development of parallel Monte Carlo algorithms. Finally, we state the assumptions which are used in the time complexity analysis of parallel algorithms.

3.1. Intrinsic parallelism. We identify the intrinsic parallelism in Monte Carlo methods at four levels as follows.

(1) *Parallelism between computations of variables.* In Monte Carlo methods for problems involving many variables, the estimates of each of the variables can usually be obtained *independently* of the remaining variables and consequently their computations can be executed in parallel. For example, the estimates of each of the unknowns in a set of linear algebraic equations can be obtained independently of the other unknowns [18], [4]. The complete independence of the computations of variables implies that the computational modules' granularities will be large and therefore these computations are very well suited for implementation on MIMD computer architectures.

Further, the computational algorithms of each of these variables are usually *identical*, but use different data. As a result, these computations are also very well suited for implementation on SIMD computer architectures.

We have for the first time exploited the intrinsic parallelism at this level and used it in solving a set of linear algebraic equations [4].

(2) *Parallelism in secondary estimate computation.* As defined in § 2, the secondary estimate of the desired solution is usually obtained by averaging the values of K primary estimates. In an SIMD computer, this summation can obviously be carried out in $\lceil \log_2 K \rceil$ time steps, if required interprocessor communication network is available. In an MIMD computer, some part of the summation time may be overlapped with other computations, as shown in ensuing sections. We have exploited the parallelism at this level in solving linear algebraic equations [4] and partial differential equations [5].

(3) *Parallelism between different PECs.* The required K PECs to be carried out for a variable are completely independent of each other (the characteristic C1). Consequently, these PECs can be executed in parallel. Since each of the PECs often involves a lengthy computation, the computational module granularities are large enough to make these computations suitable for implementation on MIMD computers. Moreover, the PECs are also very well suited for implementation on SIMD computers due to the characteristic C2.

The parallelism at this level was hinted at as early as in 1949, by Metropolis and Ulam [30], in their paper defining the Monte Carlo method. However, its use was first suggested in [33] and subsequently it was also exploited in [34] for solving PDEs with special purpose computers. In [33] the use of a drum array processor was suggested, whereas in [34] a macro-modular computer was used. Recently, we have exploited the parallelism at this level in solving linear algebraic equations and PDEs on SIMD and MIMD computers.

(4) *Parallelism in a* PEC. In general, a PEC involves: (i) multidimensional random number generation, (ii) updating the value of a primary estimate based on a sample from random number generators, and (iii) determination of the value of a termination condition for the computation. The intrinsic parallelism exists in all these steps.

The generation of each of the components in multidimensional random number generation can be carried out in parallel. In updating the value of a primary estimate, the intervals in which a random number sample falls has to be determined and this involves many decision operations, which can be executed in parallel. Finally, the step (iii) also involves many comparison operations, which can be carried out in parallel.

In [34] pipelining has been used to reduce the execution time, whereas in [2] we have suggested the use of hardware random number generators.

3.2. Design approach. From the above discussion, it is now evident that Monte Carlo methods have a large potential for intrinsic parallelism, which has remained almost untapped so far. Further, this intrinsic parallelism can be exploited equally well for implementation on SIMD as well as MIMD computers.

In our design and analysis of parallel Monte Carlo algorithms, we consider the intrinsic parallelism only between the computations of different PECs for a variable, because these computations primarily determine the order of the time complexities of these algorithms. The results given in this paper can be trivially extended for problems involving the estimation of more than one variables, because such computations are independent of each other. If the intrinsic parallelism within each of the PECs is exploited, this usually changes the time complexity only by some constant, its order remaining unchanged.

We view the problem of designing parallel Monte Carlo algorithms exploiting the intrinsic parallelism between the PECs, as that of the assignment of PECs to processors in a parallel computer, the PEC times being i.i.d. random variables. Alternatively, each PEC can be considered as a task with random execution time. Then, we will have tasks with i.i.d. execution times and the problem can be considered as basically the scheduling of independent tasks onto multiple processors with task execution times as i.i.d. random variables.

Although the problem of scheduling a set of tasks on multiple processors has been discussed in literature, much of it assumes constant execution times (see e.g. [13], [16], [25]), and only the treatment in [1], [26], [32] is with the task execution times as random variables. Motivated by the problem decomposition strategies proposed in [32], we propose two basic schemes, static and dynamic, for the assignment of PECs to processors in a parallel computer. In a static computation assignment (SCA) scheme, a *fixed* number of PECs are assigned to each of the processors *before* any PECs are initiated. In contrast, in a dynamic computation assignment (DCA) scheme, the number of PECs carried out by each processor is determined *during* the computations, based on some global criteria.

In the SCA schemes, very small (if at all) time overheads are incurred to make the computation assignment decisions during the computations. The parallel algorithms based on these schemes cannot adapt to the variations in the PEC times (which are random variables), in the sense that the computation time is minimized based on the run-time behavior. In DCA schemes, since the assignment of PECs to processors is done during execution, the execution time of a parallel algorithm can be minimized based on its run-time behavior. Obviously, a larger time overhead is incurred in making the computation assignment decisions. It should be noted that these schemes can be used for developing parallel algorithms for almost any Monte Carlo method and hence are fundamental in nature. The various SCA and DCA schemes for parallel Monte Carlo algorithms are discussed in §§ 4 and 5.

Since usually a large number of primary estimates are used, parallel Monte Carlo algorithms may involve simultaneous operation of a large number of pseudorandom number (PRN) generators. Obviously, PRN sequences of smaller lengths will be used on each of the processors, compared to the corresponding sequential algorithms. Thus, the PRN generators should be such that the PRN sequences generated exhibit desired statistical properties for smaller lengths. In addition, the PRN sequences generated in various processors, pooled together, should satisfy the desirable properties, e.g., small cross-correlation. These and other issues related to the pseudo-random number generation are discussed in [28], [29].

3.3. Analysis of parallel Monte Carlo algorithms. In many Monte Carlo applications, PEC time distributions cannot be determined a priori. In some cases, even if it may be possible to determine the distributions, the computations involved may be too time consuming. For example, the determination of the expectation of PEC times in solving an elliptic PDE itself involves the solution of another elliptic PDE, i.e., the problem of determining the expectation has the same order of time complexity as that of the problem being solved with the Monte Carlo method [6]. Therefore in the analysis of parallel Monte Carlo algorithms we obtain the time complexity results which are independent of the probability distributions of the PEC times (i.e. distribution-free) and in closed forms.

As in sequential Monte Carlo algorithms, we assume that the secondary estimate computation time is negligible. Further, we assume, as usual in literature (see e.g. [21], [26], [32]): (i) the use of identical processors, (ii) the register load/store and communication costs as negligible, (iii) no memory or data alignment penalties, and (iv) any one of the four arithmetic operations being performed in one time unit. Further in the analysis of MIMD algorithms, we assume, as in [32]: (i) for any runnable process a processor is always assigned, and (ii) initially all runnable processes start execution at the same time. Finally, to enable easier comparison of the performance of various algorithms we assume that, in a parallel algorithm the PEC time characteristics (e.g. their mean, variance, etc.) remain exactly same as in the corresponding sequential algorithm. This implies that, for example, the possible increase in the PEC times due to conditional branches in a SIMD algorithm (see [27]) is considered as negligible. Also, the consequences of multiprocessor system characteristics, as discussed in [1], [32], are neglected. However, such changes in the PEC times can be *implicitly* taken into account by suitably modifying the statistical characteristics of the PEC times. As a consequence of the stated assumptions, the computational time behavior and the time complexities of the SIMD and MIMD algorithms corresponding to a sequential algorithm, will turn out to be the same.

Since the PEC times are random variables, in the analysis of parallel Monte Carlo algorithms we have to employ the techniques of order statistics, queueing theory, and renewal theory, as discussed in §§ 4 and 5.

4. Static computation assignment (SCA) schemes. In SCA schemes for parallel Monte Carlo algorithms, we assign a fixed number of PECs to the processors before any PECs are initiated, as discussed in § 3. We propose four SCA schemes for the development of SIMD and MIMD algorithms, dependent upon the number of processors (P) used:

(i) SCA Scheme 1: $P \ll K$,
(ii) SCA Scheme 2: $K/P \sim 10$,
(iii) SCA Scheme 3: $P = K$,
(iv) SCA Scheme 4: $P > K$.

In the Schemes 1 and 2, we assign K PECs among P processors in such a way that their sum is equal to K and each processor is assigned the same number of PECs, as far as possible; obviously, if P divides K exactly then we will assign (K/P) PECs to each of the processors. In Scheme 3, we assign only one PEC to each of the processors. In Scheme 4 also we assign one PEC to each of the processors, but terminate the computations in all active processors once K PECs are completed. The first three schemes can be considered to be as the mere special cases of a single scheme. However, we have chosen to treat them separately because the SIMD and MIMD algorithms based on them differ (see [6] for details) and moreover their analyses have to be carried out differently.

4.1. SCA Scheme 1. Assume that $M=(K/P)$, where M is an integer. Then, in this scheme we would obtain K primary estimates when all the processors complete M PECs assigned to each of them. The expected time complexity of SIMD and MIMD algorithms based on this scheme is given in the following.

THEOREM 2. *The expected time complexity of parallel Monte Carlo algorithms based on* SCA *Scheme* 1 *is given as*

$$\frac{K\mu}{P} \leqq E[T(P, K)] \leqq \frac{K\mu}{P} + \frac{\sqrt{K}.\sigma}{2}. \tag{2}$$

Proof. The execution time of the ith processor in carrying out M PECs, say t_i, is given by the sum of the execution times of each of the PECs. By Lemma 1, the PEC times are i.i.d. random variables and therefore the computation times $t_1, t_2, \cdots, t_P$ will also be i.i.d. random variables. Since $P \ll K$, M will be a large number and then by the Central Limit theorem it follows that t_i's will follow a normal distribution with mean $(M.\mu)$ and variance $(M.\sigma^2)$.

The execution time of the parallel algorithms, $T(P, K)$, is given by the random variable

$$T(P, K) = \max [t_1, t_2, \cdots, t_P].$$

Thus $T(P, K)$ represents the extreme order statistic of the i.i.d. normal random variables, $t_1, t_2, \cdots, t_P$. Since a closed form expression is not available for such an order statistic, we use the upper bound on the extreme order statistic for any symmetrically distributed i.i.d. random variables given in [17, p. 62] and obtain the upper bound given in the theorem.

The lower bound on the time complexity follows from the fact that $E[\max (t_1, t_2, \cdots, t_P)] \geqq \max [E(t_1), E(t_2), \cdots, E(t_P)] = M.\mu$, which is true for any set of random variables. □

Thus, SCA Scheme 1 parallel Monte Carlo algorithms have $O(K)$ expected time complexity and, comparing with the time complexity of sequential Monte Carlo algorithms, they achieve $O(P)$ speedup, i.e. linear speedup.

If $P=2$, then the expected time complexity of these parallel algorithms can be given as [26]

$$E[T(P, K)] \cong \frac{K\mu}{P} + \sqrt{\frac{K}{\pi P}}.\sigma \quad \text{as } K \to \infty. \tag{3}$$

4.2. SCA Scheme 2. In this scheme, we have $(K/P) = M \sim 10$ and we assign K PECs among P processors, as in Scheme 1. In Scheme 1, since M is large, the i.i.d. random variables $t_1, t_2, \cdots, t_P$ were considered as normal, whereas in this scheme, since M is smaller, such an approximation may not be acceptable.

THEOREM 3. *The* SIMD *and* MIMD *algorithms based on Scheme* 2 *have the expected time complexity*

$$\frac{K\mu}{P} \leqq E[T(P, K)] \leqq \frac{K\mu}{P} + \frac{(P-1)\sqrt{K}}{\sqrt{(2P-1)P}}.\sigma. \tag{4}$$

Proof. The upper bound follows from the distribution-free upper bound on the extreme order statistic of i.i.d. random variables given in [15, pp. 46–47] and the lower bound remains same as in Theorem 2. □

In practice, K is usually a large number (10^3 to 10^5 is not uncommon) and since $K/P \sim 10$, we will usually have $P \gg 1$. In this case, the upper bound in (4) can be

simplified to the following:

$$E[T(P, K)] \leqq \frac{K\mu}{P} + \sqrt{\frac{K}{2}} \cdot \sigma.$$

It can be easily seen that the lower bound on the speedup for SCA Scheme 2 parallel algorithms will be $O(\sqrt{P})$.

4.3. SCA Scheme 3. In this scheme, we assign one PEC to each processor. Thus the execution time of the parallel algorithms based on this scheme will be given by the extreme order statistic of K i.i.d. random variables represented by PEC times and we have the following.

THEOREM 4. *The* SCA *Scheme* 3 *parallel Monte Carlo algorithms have the expected time complexity*

$$\mu \leqq E[T(P, K)] \leqq \mu + \frac{(K-1) \cdot \sigma}{\sqrt{(2K-1)}}. \tag{5}$$

Proof. Follows directly by using the distribution-free upper bound on the extreme order statistic given in [15]. □

The asymptotic time complexity of these parallel algorithms will be given as

$$\mu \leqq E[T(P, K)] \leqq \mu + O(\sqrt{K}) \tag{6}$$

with asymptotic speedup over the corresponding sequential algorithms as

$$O(\sqrt{K}) \leqq S(K, K) \leqq K.$$

In a Scheme 3 SIMD algorithm, we would use a global counter to keep track of the completed PECs. The global counter will be updated by only one processor at a time and the remaining processors in the SIMD computer will be idle. Consequently, the time complexity result would get modified to the following:

$$\mu + K\tau \leqq E[T(K, K)] \leqq \mu + \frac{(K-1)\sigma}{\sqrt{(2K-1)}} + K\tau \tag{7}$$

where τ is the time required to update the global counter by a processor. Thus, the expected time complexity will turn out to be $O(K)$, whereas by neglecting such an updating time, the lower bound is a constant and the upper bound is $O(\sqrt{K})$. This demonstrates that the sequential program segments in an SIMD algorithm may increase the order of time complexity. In Scheme 3 MIMD algorithms, the global counter updating time may not contribute much time overhead, because this time will be overlapped with the PEC times of active processors.

It is interesting to compare the cumulative distribution function (cdf) of the time complexity of sequential Monte Carlo algorithms with the Scheme 3 parallel algorithms. In the sequential algorithms, the cdf will be given by the *K-fold convolution*, whereas in the parallel algorithms it is given by the Kth *power* [6]. Consequently, the variance of the time complexity of parallel Monte Carlo algorithms will turn out to be much smaller than that of the corresponding sequential algorithms. This is an attractive feature of these parallel algorithms.

4.4. SCA Scheme 4. In this scheme, we have $P > K$. We assign one PEC to each processor with a motivation to minimize the execution time of the parallel Monte Carlo algorithms by terminating the execution soon after we have K PECs completed

out of the initiated P PECs. Unfortunately, this may result in a *biased* estimate of the desired solution as discussed below.

In Monte Carlo methods, the execution time of PECs is usually related to either the number of terms considered from a finite or an infinite series (e.g. Neumann series used in solving linear equations [18]), or the lengths of random walks (e.g. studies in nuclear physics [37], molecular chemistry [19]). As a result, the execution time is related to the accuracy of the estimates obtained. In the present scheme, since we always choose the earliest completing K PECs, we are effectively limiting the number of terms from a series or truncating lengths of random walks. Consequently, in this scheme we may obtain *biased* estimates.

In [6], we have given an SIMD algorithm based on this scheme. We have also given two MIMD algorithms, one with and the other without interruption of processes, as suggested in [26].

The time complexity of Scheme 4 parallel Monte Carlo algorithms is represented by the Kth order statistic of P i.i.d. PEC times, with $P > K$. Unfortunately, distribution-free closed form expressions are not possible for such an order statistic. However, tables are available for normal distribution (see [15, p. 28]), gamma distribution, logistic distribution, etc. (see [15, pp. 226-227]). In § 6 we give the expected time complexity results for Scheme 4 parallel algorithms, when the PEC times are exponentially distributed i.i.d. random variables.

5. Dynamic computation assignment (DCA) schemes. In DCA parallel Monte Carlo algorithms, whenever a PEC (or a set of PECs) is completed on a processor, the decision as to whether or not to initiate another PEC (or another set of PECs) on that processor is made during the execution, based on some global criteria. The main motivation is to minimize their execution time. Here, we propose two basic schemes for such parallel algorithms. Scheme 1 can be used in the development of parallel Monte Carlo algorithms employing arbitrary number of processors. Scheme 2 is designed specifically for implementations with $P < K$. The closed form distribution-free time complexity results for these parallel algorithms can be obtained only when $P \ll K$ and hence we consider here only such cases in the time complexity analysis.

5.1. DCA Scheme 1. In this scheme, given P processors, first we initiate P PECs in parallel. Whenever a PEC is completed on a processor, we initiate another PEC on that processor, if K PECs have not already been completed. Obviously, in this scheme we will initiate more than the required K PECs. The details of the parallel Monte Carlo algorithms are given in [6].

Figure 1 illustrates a possible execution time profile for DCA Scheme 1 parallel algorithms (under various assumptions stated in § 3) in obtaining nine primary estimates with three processors, P1, P2, and P3. The time $t_j^{(i)}$ represents the time instant at which the jth PEC is completed by the ith processor. The behavior of the parallel algorithms, taking into account computations on all the three processors, is represented by the virtual processor P^*. For P^*, time t_i represents the time instant at which the ith PEC is completed. It is seen that, when the required number of PECs are completed, the incomplete PECs on the remaining $(P-1)$ processors get aborted. Consequently, in this scheme (as in SCA Scheme 4) we may obtain biased estimates. The aborted computation time is significant only in MIMD algorithms; in SIMD algorithms this time is not significant, because if these $(P-1)$ processors did not carry out the computations which later got aborted, any way they would have been idle. In order to eliminate the wasteful computations in the DCA Scheme 1 MIMD algorithms, we give DCA Scheme 2 in the next subsection.

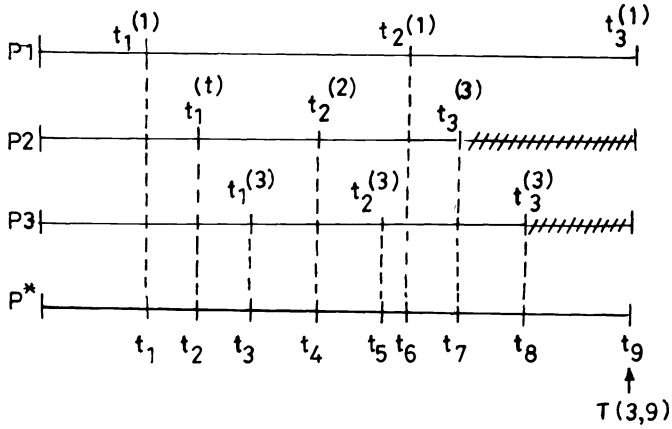


FIG. 1. *A possible execution time profile for* DCA *Scheme* 1 *parallel Monte Carlo algorithms.* P1, P2, P3: *Processors.* P*: *Virtual processor representing the execution time behavior of the parallel algorithms.* —— PEC *times.* -/// *aborted computation time.*

THEOREM 5. *The* DCA *Scheme* 1 *parallel Monte Carlo algorithms, carrying out K complete* PECs *with P processors, have the expected time complexity*

$$E[T(P,K)] \cong \frac{K\mu}{P} + \frac{(P-1)(\mu^2-\sigma^2)}{2P\mu}. \tag{8}$$

Proof. The time $t_j^{(i)}$, as shown in Fig. 1, represents the sum of j i.i.d. PEC times. Hence, the computational behavior of the ith processor is an ordinary renewal process [14, p. 25].

Since all the PEC times are i.i.d. random variables, a DCA Scheme 1 parallel algorithm consists of P independent ordinary renewal processes in simultaneous operation, all with the same probability distribution. Therefore, the behavior of the parallel algorithm, represented by P^*, can be considered as the behavior of the superposition of the constituent P ordinary renewal processes. Then, the theorem follows by using the result in [14, pp. 73-75]. □

Thus, DCA Scheme 1 parallel Monte Carlo algorithms have $O(K)$ expected time complexity and comparing with the time complexity of sequential algorithms, they achieve speedup approximately *equal* to P, i.e., ideal speedup. If K is large, as usually it will be, then the computing time on each processor can be considered as normally distributed and then by using the result in [14, p. 73] we have

$$\text{Var}[T(P,K)] = \text{Var}[T(1,K)]/P^2. \tag{9}$$

Again, this shows that the variance of the time complexity of parallel algorithms may be much smaller than that of corresponding sequential algorithms.

In DCA Scheme 1 MIMD algorithms, we will have a critical region to update the global counter for keeping track of completed PECs and to update the secondary estimate. If the critical region execution time for the constituent processes in such parallel algorithms is not to be neglected and the contention among them to execute the critical region is considered, then behavior of these algorithms can be modelled as a queueing system, similar to the models of time shared computer systems as shown in Fig. 2. Such a queueing system has been extensively studied as a model of single processor time shared computer systems (see [35, pp. 27-31], [23, pp. 144-154]), when

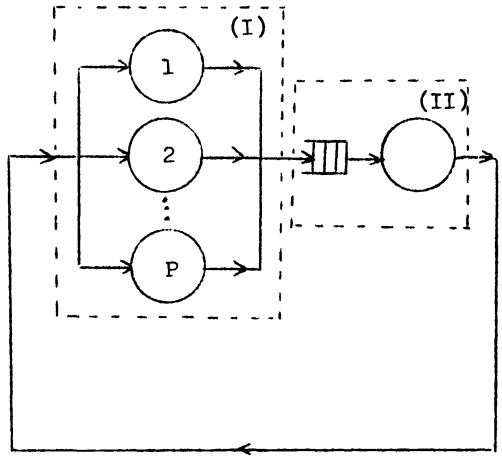


FIG. 2. *A queueing system model for* DCA *Scheme* 1 MIMD *algorithms.* A. *P Customers in the whole system: P processes.* B. *P Servers in system* I: PEC*s on P processors.* C. 1 *Server in system* II: *the critical region execution.* D. *At most P servers active at the same time in the entire system.*

the probability distributions of all the servers are exponential. Assuming that the mean critical region execution time is equal to τ_c, the steady-state probability that no process will be executing the critical region is given by

$$\pi_0 = \frac{1}{\sum_{j=0}^{P} (P! r^j/(P-j)!)} \quad \text{where } r = (\tau_c/\mu). \tag{10}$$

Since $P \ll K$, each process will execute a large number of PECs and therefore the steady-state solution is justifiable. Then, the expected time complexity of DCA Scheme 1 MIMD algorithms will be given as

$$E[T(P, K)] = \frac{\tau_c \cdot K}{(1-\pi_0)}. \tag{11}$$

5.2. DCA Scheme 2. This scheme is designed specifically for MIMD algorithms. In this scheme, given P processors, first we initiate P PECs in parallel. Whenever a PEC is complete, we initiate another PEC on that processor only if the required K PECs have not been already initiated. Thus, in this scheme we initiate only K PECs and therefore no PECs get aborted as in DCA Scheme 1.

Figure 3 illustrates a possible execution time profile the DCA Scheme 2 parallel Monte Carlo algorithms, assuming that nine primary estimates are to be computed using three processors. It is seen that, when the ninth PEC is initiated at time t_5^*, six PECs are already complete and two are in progress. When the longest of these three PECs completes at time t_9, we obtain the required nine primary estimates. Thus, in general, when Kth PEC is initiated, $(K-P)$ PECs are already complete and $(P-1)$ PECs are in progress. When the largest of these P PECs terminates, we obtain the required K primary estimates.

THEOREM 6. *The* DCA *Scheme* 2 MIMD *algorithms, carrying out* K PECs *with* P *processors* ($P \ll K$) *have the expected time complexity*

$$\tau \lesssim E[T(P, K)] \lesssim \tau + \frac{(P-1) \cdot \sigma}{\sqrt{(2P-1)}} \tag{12}$$

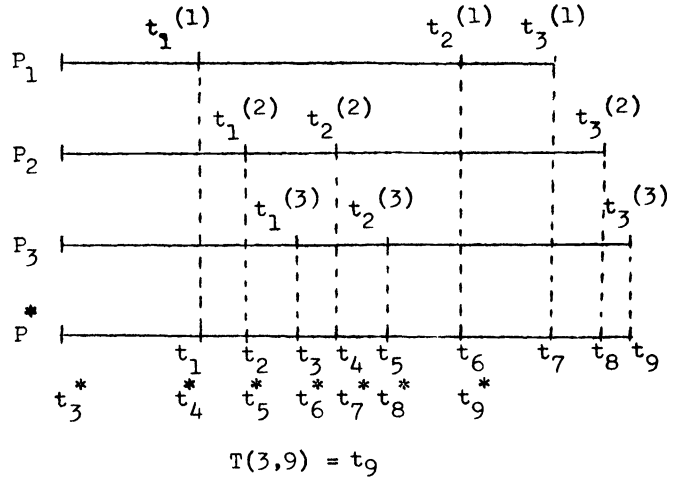


FIG. 3. *A possible execution time profile for* DCA *Scheme* 2 MIMD *algorithms.* $t_j^{(i)}$: *time at which jth* PEC *is completed by the ith processor.* t_i: *time at which ith* PEC *is completed in the parallel algorithm.* t_i^*: *time at which ith* PEC *is initiated in the parallel algorithm.*

where

$$\tau \cong \frac{K\mu}{P} + \frac{(P-1)(\mu^2-\sigma^2)}{2P\mu}.$$

Proof. When the Kth PEC is initiated, $(K-P)$ PECs are complete, say at time $t_{(P-K)}$. The time $t_{(P-K)}$ is a random variable representing the time instant at which $(K-P)$ renewals are completed in the superposed renewal process of the constituent P ordinary renewal processes, each ordinary renewal process comprising the i.i.d. PEC times. Therefore, by using the result in [14, p. 75], the expectation of $t_{(P-K)}$ will be given as

$$E[t_{(P-K)}] \cong \frac{(K-P)\mu}{P} + \frac{(P-1)(\mu^2-\sigma^2)}{2P\mu}. \tag{13}$$

Immediately after $t_{(P-K)}$, we have one new PEC and $(P-1)$ PECs already in progress. Then, the additional time required after $t_{(P-K)}$ to complete these P PECs, say t_P, will be given as

$$\begin{aligned} t_P &= \max\,[\text{residual times of } (P-1) \text{ PECs, lifetime of new PEC}] \\ &= \max\,[t_r^{(1)}, t_r^{(2)}, \cdots, t_r^{(P-1)}, t_m] \end{aligned} \tag{14}$$

where $t_r^{(i)}$ is a random variable representing the residual lifetime of ith PEC, and t_m is the random variable representing the lifetime of the new PEC. The time t_P is the extreme order statistic of independent, but *not* identically distributed, P random variables. Since distribution-free closed form expressions are not possible for the expectation of the right-hand side of (14), we obtain the upper bound on the expectation of t_P as

$$\begin{aligned} E[t_P] &\leqq \max\,[\text{lifetimes of } P \text{ PECs}] \\ &= \mu + \frac{(P-1)\,.\,\sigma}{\sqrt{(2P-1)}}. \end{aligned} \tag{15}$$

The lower bound on $E[t_P]$ is given as follows.

$$\begin{aligned} E[t_P] &= E[\max\{t_r^{(1)}, t_r^{(2)}, \cdots, t_r^{(P-1)}, t_m\}] \\ &\geqq \max\{E[t_r^{(1)}], E[t_r^{(2)}], \cdots, E[t_r^{(P-1)}], E[t_m]\} \\ &\geqq \mu. \end{aligned} \tag{16}$$

Now, the theorem follows from (14), (15) and (16). □

Thus, the DCA Scheme 2 MIMD algorithms have $O(K)$ time complexity and, comparing with sequential algorithms, they achieve speedup approximately equal to P.

The time complexity results for the various SCA and DCA schemes are summarized in Table 1.

6. An example. In this section, we illustrate the performance of the sequential and parallel Monte Carlo algorithms, when the PEC times are standardized i.i.d. exponential random variables; i.e. their mean $\mu = 1$, standard deviation $\sigma = 1$, and their probability distribution function (pdf) given by

$$f(t) = e^{-t} \qquad (t \geqq 0). \tag{17}$$

Then the pdf of the time complexity of sequential Monte Carlo algorithms will be given by

$$\begin{aligned} F_{(1,K)} &= K\text{-fold convolution of } f(t) \\ &= \frac{t^{(K-1)} \cdot e^{-t}}{(K-1)!}. \end{aligned} \tag{18}$$

Since in practice usually K is large, the random variable $T(1, K)$ will be asymptotically normally distributed. Further, we have

$$E[T(1, K)] = K, \quad \text{and} \tag{19}$$

$$\text{Var}[T(1, K)] = K. \tag{20}$$

6.1. Performance of SCA schemes. The performance of Schemes 1 and 2 found out by using (2), (4), and (5) is given in Table 1.

In Scheme 3, the cdf of the execution time of the parallel algorithms will be given by

$$F_{(K,K)} = [1 - e^{-t}]^{(K-1)} \tag{21}$$

and their pdf as

$$f_{(K,K)} = K e^{-t}[1 - e^{-t}]^{(K-1)}. \tag{22}$$

Further, it can be easily derived that

$$E[T(K, K)] = \sum_{i=1}^{K} i^{-1}$$

$$= H_K \tag{23a}$$

$$\cong \log_e K \qquad \text{(if } K \text{ is very large)} \tag{23b}$$

where H_K is the Kth harmonic number. Also, we would obtain

$$\text{Var}[T(K, K)] = \sum_{i=1}^{K} i^{-2} \tag{24}$$

TABLE 1

PEC (*primary estimate computation*)-*time distribution-free time complexity results for the various schemes for parallel Monte Carlo algorithms.*

Scheme		Expected time complexity
SCA Scheme 1	$P \ll K$	$\frac{K\mu}{P} \leqq E[T(P,K)] \leqq \frac{K\mu}{P} + \frac{\sqrt{K}\cdot\sigma}{2}$ If $P=2$, $E[T(P,K)] \cong \frac{K\mu}{P} + \sqrt{\frac{K}{\pi P}}\cdot\sigma$ as $K\to\infty$
SCA Scheme 2	$\frac{K}{P} \sim 10$	$\frac{K\mu}{P} \leqq E[T(P,K)] \leqq \frac{K\mu}{P} + \frac{(P-1)\sqrt{K}}{\sqrt{(2P-1)P}}\cdot\sigma$
SCA Scheme 3	$P=K$	$\mu \leqq E[T(P,K)] \leqq \mu + \frac{(K-1)}{\sqrt{2K-1}}\cdot\sigma$
SCA Scheme 4	$P>K$	PEC-time distribution-free time complexity results are not possible.
DCA Scheme 1		$E[T(P,K)] \cong \frac{K\mu}{P} + \frac{(P-1)(\mu^2+\sigma^2)}{2P\cdot\mu}$
DCA Scheme 2		$\tau \lesssim E[T(P,K)] \lesssim \tau + \frac{(P-1)}{\sqrt{2P-1}}\cdot\sigma$ where $\tau \cong \frac{K\mu}{P} + \frac{(P-1)(\mu^2-\sigma^2)}{2P\mu}$

K = The desired total number of primary estimates of a problem variable.
M = mean of i.i.d PEC times, σ = standard deviation of i.i.d. PEC times.
P = number of processors.
Note. The time complexity of a sequential Monte Carlo algorithm is given as $E[T(1,K)] = K\cdot\mu$.

and if K is very large, we can use the result in [22, p. 75] and obtain

$$\text{Var}[T(K,K)] = \frac{\pi^2}{6}. \tag{25}$$

Thus for SCA Scheme 3, the expected time complexity increases logarithmically as against linear increase in sequential Monte Carlo algorithms. Moreover, the variance of the parallel Monte Carlo algorithms is bounded for large values of K, whereas it increases linearly for the sequential Monte Carlo algorithms.

For SCA Scheme 4, we had noted that distribution-free time complexity results are not possible. However, we can determine the expected time complexity for the present example. It can be easily shown that the Kth order statistic of P i.i.d. PEC times considered here is given as

$$E[T(P,K)] = \sum_{i=(P-K+1)}^{P} i^{-1}$$

$$= H_P - H_{(P-K)} \tag{26}$$

$$\cong \log_e P - \log_e (P-K) \quad \text{if } (P-K) \gg 1. \tag{27}$$

The variance of the time complexity will be given by

$$\text{Var}[T(P,K)] = \sum_{i=(P-K+1)}^{P} i^{-2}. \tag{28}$$

Thus, it is seen that the expectation as well as the variance of the time complexity of SCA Scheme 4 parallel Monte Carlo algorithms can be reduced by increasing the number of processors.

6.2. Performance of DCA schemes. The performance of DCA Scheme 1 using (8) will be given as

$$E[T(P, K)] = \frac{K}{P}. \tag{29}$$

This implies that we can achieve speedup exactly equal to P, the one which is always desired and rarely achieved in practice.

It can be easily proved that the expected time complexity of DCA Scheme 2 will be given by

$$E[T(P, K)] = \frac{K}{P} + H_P. \tag{30}$$

The expected time complexity results given in this section are summarized in Table 2.

7. Computational results. In this section, we present some computational results illustrating the performance of the schemes for parallel Monte Carlo algorithms given in §§ 4 and 5. We consider the Laplace's equation defined on a rectangle,

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = 0, \qquad 0 < x < 6, \quad 0 < y < 10, \tag{31}$$

TABLE 2

Expected time complexity of parallel algorithms based on various SCA *and* DCA *schemes, when* PEC*-times are i.i.d. exponential random variables with* $\mu = 1$ *and* $\sigma = 1$. *Time complexity of sequential algorithm* $= K$.

Scheme		Time complexity
SCA Scheme 1	$P \leqq K$	$\frac{K}{P} \leqq E[T(P, K)] \leqq \frac{K}{P} + \frac{\sqrt{K}}{2}$ If $P = 2$, $E[T(P, K)] \cong \frac{K}{2} + \frac{\sqrt{K}}{\sqrt{2\pi}}$.
SCA Scheme 2	$\frac{K}{P} \sim 10$	$\frac{K}{P} \leqq E[T(P, K)] \leqq \frac{K}{P} + \frac{(P-1)\sqrt{K}}{\sqrt{(2P-1)P}}$ If $P \gg 1$, then $\frac{K}{P} \leqq E[T(P, K)] \leqq \frac{K}{P} + \sqrt{\frac{K}{2}}$
SCA Scheme 3	$P = K$	$E[T(P, K)] = H_K \cong \log_e K$
SCA Scheme 4	$P > K$	$E[T(P, K)] = H_P - H_{(P-K)}$ $\cong \log_e P - \log_e (P - K)$
DCA Scheme 1	$P \ll K$	$E[T(P, K)] = \frac{K}{P}$
DCA Scheme 2	$P \ll K$	$E[T(P, K)] = \frac{K}{P} + H_P$

Note. H_n represents the nth harmonic number.

with boundary conditions

$$U(x, 0) = 10, \qquad U(x, 10) = U(0, y) = U(6, y) = -10.$$

This problem is selected for two reasons: (i) the analytic solution to this problem can be easily obtained and the Monte Carlo solutions could be compared, and (ii) the same equation has been considered in [34].

In solving PDEs by Monte Carlo methods, the solution at a point in the given region can be obtained independently of the solution at other points [2], [34]. We have selected three representative points in the region given in (31): the first point as (1, 1) near the boundary, the second point (3, 5) at the centre of the region and the third point (2, 3) in between these two points. In the method described in [34], the given region is replaced by a square mesh, as in a finite-difference method. Figure 4 shows the given region divided into a mesh, with mesh size $h = \frac{1}{4}$. Now a set of random walks are defined on the set of states comprising [10]: (i) one transient state corresponding to each of the internal nodes (i.e. the unknowns), and (ii) one absorbing state for each of the boundary nodes. In solving (31), the state transition probabilities for a step from any one internal node to its four neighboring nodes turns out to be equal to 0.25. Equation (31) was solved on a sequential computer for obtaining solutions at

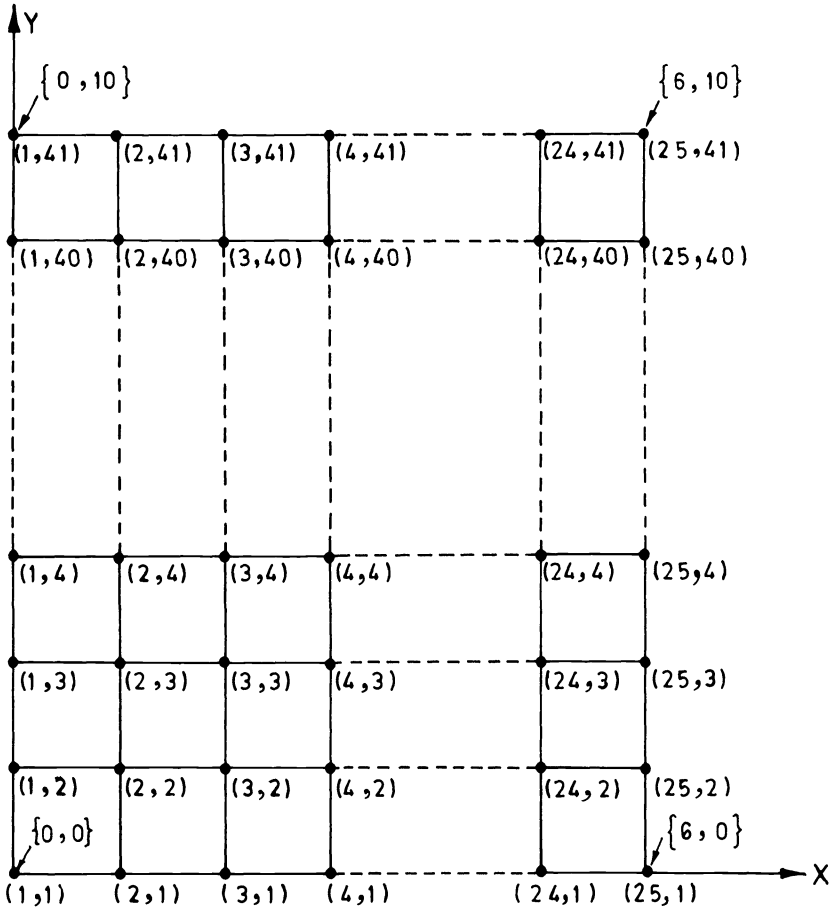


FIG. 4. *The region for Laplace's equations with mesh, mesh size* $h = \frac{1}{4}$. (m, n): *mesh points.* $\{x, y\}$: *co-ordinates.*

the selected three points and the Monte Carlo solutions were in good agreement with the analytic and numerical solutions when 2048 primary estimates were used (see [6]). The random walk duration statistics were collected for solutions at each of the selected points. Since parallel computers were not accessible, we have used the statistics collected for estimating the performance of various schemes for parallel algorithms. We consider the computational times of 2048 primary estimates on parallel computers, with the number of processors varying from 2, 4, 8, $\cdots$, 2048 in logarithmic steps, the computational time being in terms of the number of random walk steps (i.e. the duration of random walks) in a primary estimate computation.

7.1. SCA schemes. The first three SCA schemes discussed in § 4 are considered to be as follows:

1. Scheme ($P \ll K$): for $P = 2, 4, 8, \cdots, 128$.
2. Scheme ($K/P \sim 10$): for $P = 128$, 256, 512, and 1024.
3. Scheme ($P = K$): for $P = 2048$.

Now, we use the sample mean and the standard deviation of the durations of random walks to obtain the bounds on the speedup of the above three schemes, using the results in Theorems 2, 3 and 4. Figures 5, 6 and 7 depict the upper and lower bounds on the speedup expected in solving (31) at points (1, 1), (2, 3), and (3, 5), respectively, on a log-log scale. It is seen that until about $P = 64$ there is no significant difference between the upper and lower bounds. The difference increases rapidly as the number of processors are increased beyond this point. At $P = 128$, the lower bounds obtained by Scheme 1 and Scheme 2 are indicated by points A and B, respectively.

We estimate the performance of SCA parallel algorithms based on the following. It is known that given N i.i.d. random processes operating in parallel, the characteristics of the ensemble can be derived from successive observations on a single source, due to the ergodicity principle. In the present case, we use the data generated during Monte Carlo solutions on a sequential computer and estimate the performance of SCA schemes as follows. We divide the data about random walk durations into P disjoint sets with only M *successive* samples in each set. The execution time on a sequential computer is represented by the sum of the durations of all 2048 random walks in a run, whereas the execution time on a parallel computer is given by the maximum of the sum of the durations of random walks for each of the P sets. The speedup estimated for Monte Carlo solutions at the chosen three points using by the various SCA schemes is given in Figs. 5, 6, and 7, for two runs.

We make the following observations from Figs. 5, 6 and 7. The estimated speedup follows closely the lower bound, until about $P = 32$, but being always less than this bound. Here, it may be expected that the estimated performance should lie between the upper and lower bounds. However, it may be recalled that the speedup is defined as the ratio of the *expectations* of the execution times on sequential and parallel computers [32], and here we have presented the results only for two runs. From $P = 64$ to $P = 2048$, the estimated speedup is almost always within the upper and lower bounds. In many cases, the slope of the estimated speedup decreases as the number of processors increases, as expected.

7.2. DCA schemes. The speedup using the two DCA schemes from § 5 is computed using Theorems 5 and 6 by substituting the sample mean and the standard deviation of the durations of random walks, and the results are shown in Figs. 5, 6 and 7. It is seen that for Scheme 1 the speedup closely follows a linear speedup. Further, for the solution at point (1, 1) the speedup is *greater* than P, due to the fact that in this case $\mu < \sigma$ (refer to (8)). For Scheme 2, the upper bound on the speedup is the same as

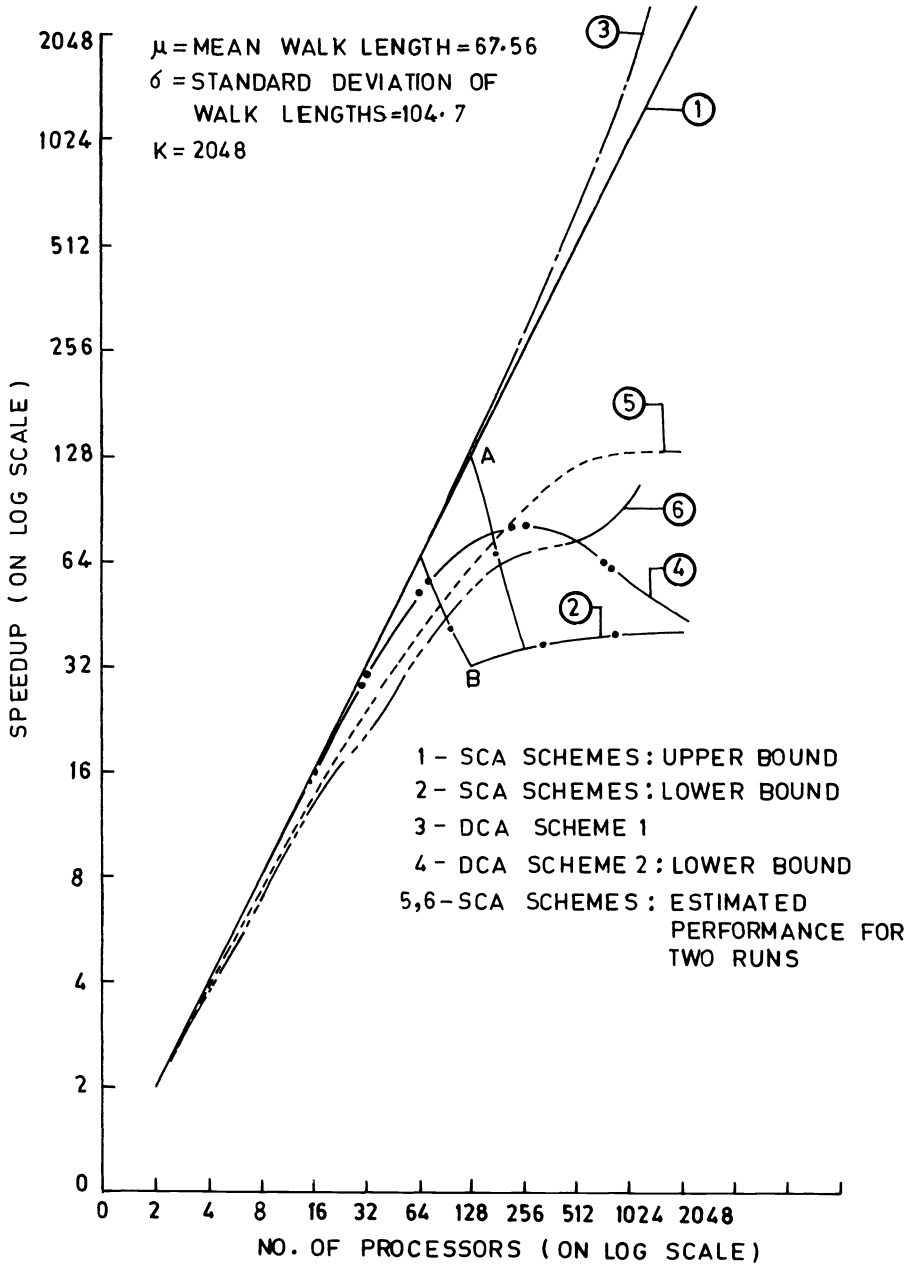


FIG. 5. *Performance of various schemes for parallel Monte Carlo algorithms for solving Laplace's equation at point* (1, 1) *in the given region.*

that of Scheme 1 by Theorem 6. The lower bound on the speedup starts dropping after about $P > 256$, due to the denominator expression in (8).

In order to estimate the performance of a DCA scheme, it should be noted that the behavior of each processor can be simulated properly only by considering successive PECs and the last PEC in these PECs may be an aborted PEC. Consequently, the completed K PECs on P processors, which determine the execution time of a simulated

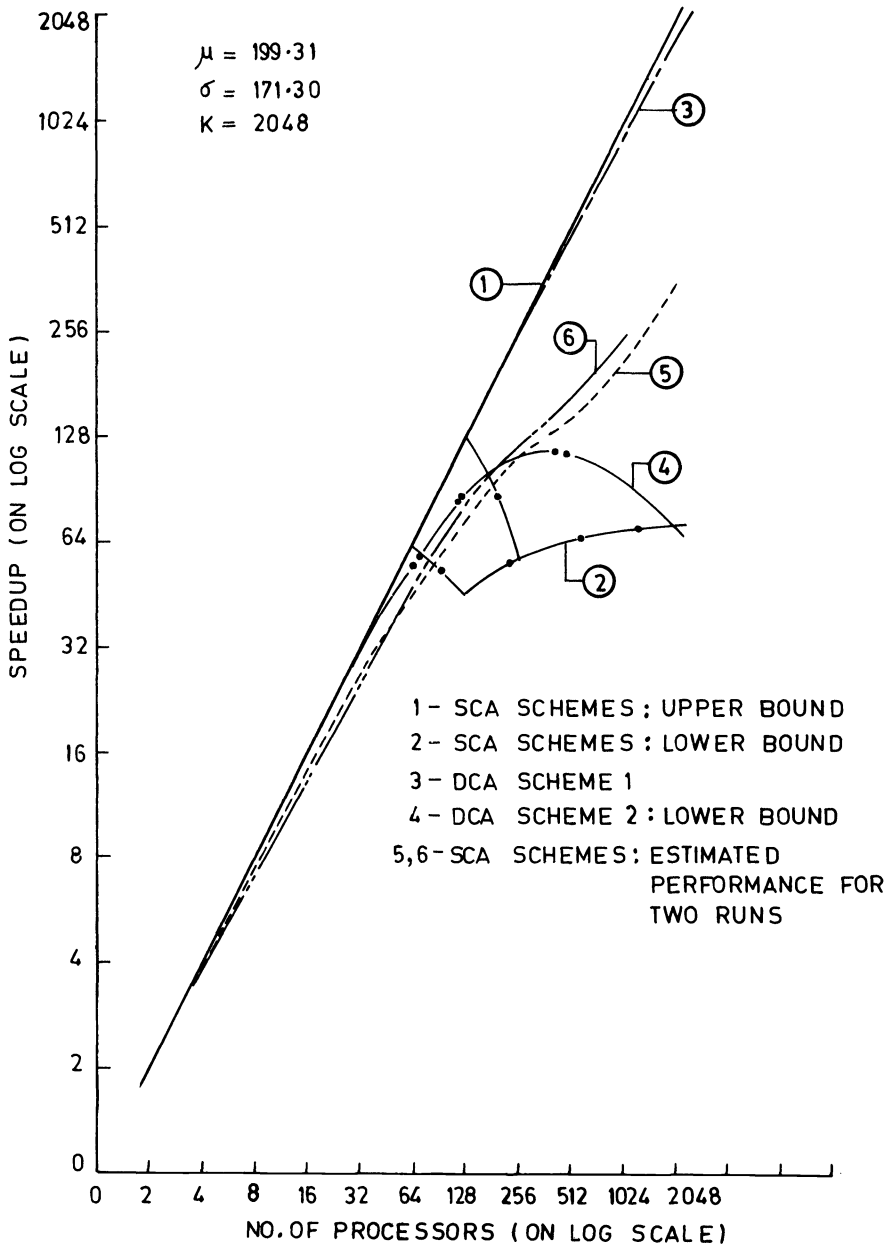


FIG. 6. *Performance of various schemes for parallel Monte Carlo algorithms for solving Laplace's equation at point* (2, 3) *in the given region.*

DCA scheme, may not necessarily be consecutive. However, the execution time of a sequential Monte Carlo algorithm has to be obtained by using only the data for consecutive K PECs. Thus, the requirements for estimating the execution time of sequential Monte Carlo algorithm and its corresponding DCA parallel algorithm are incompatible. Hence, the performance of DCA schemes for a given run cannot be estimated.

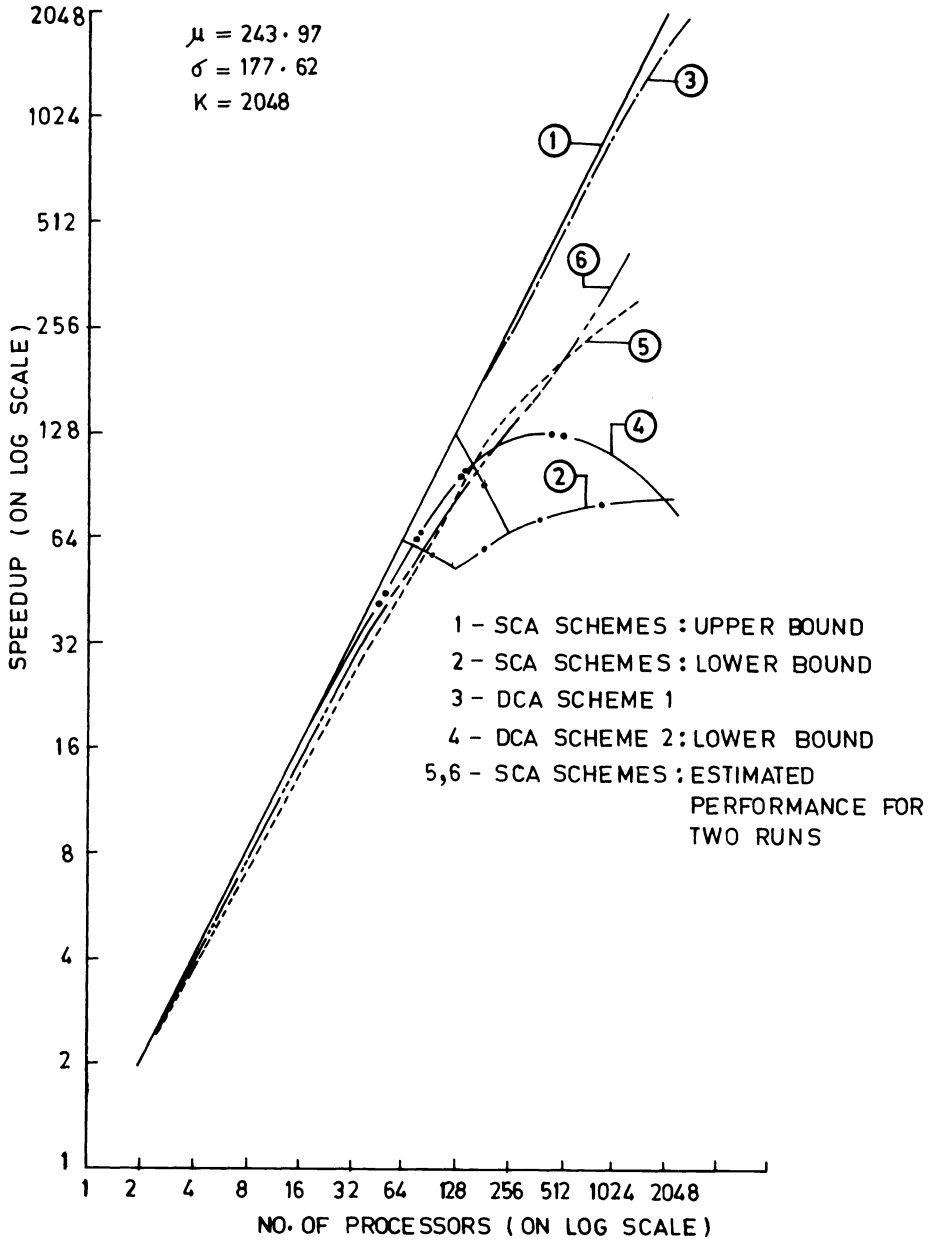


FIG. 7. *Performance of various schemes for parallel Monte Carlo algorithms for solving Laplace's equation at point* (3, 5) *in the given region.*

8. Conclusion. In this paper, we have demonstrated that the potential of intrinsic parallelism in Monte Carlo methods, which had remained essentially untapped so far, can be exploited for implementing these methods efficiently on parallel computers.

We have shown that the primary estimate computation (PEC) times in these methods turn out to be independent identically distributed random variables. This crucial result is shown to greatly influence the design and analysis of parallel Monte Carlo algorithms.

We have identified the intrinsic parallelism in these methods at four levels. The potential of this intrinsic parallelism is shown to be equally well suited for exploitation on SIMD as well as MIMD computer architectures. The computations involved have large module granularity, little concurrency control requirement and they do not demand specialized interprocessor interconnection networks.

Since the PEC times primarily determine the time complexity of a Monte Carlo method, we have proposed two basic schemes for developing parallel Monte Carlo algorithms utilizing intrinsic parallelism between PECs. In static computation assignment (SCA) schemes we assign a fixed number of PECs to the processors before any PECs are initiated, whereas in dynamic computation assignment (DCA) schemes the decision as to whether or not to initiate another PEC on a processor, whenever it completes a PEC, is made during the execution based on some global criteria. The DCA parallel Monte Carlo algorithms are more adaptable to the variations in the PEC times. The SCA and DCA schemes for parallel Monte Carlo algorithms can be applied for almost any Monte Carlo methods. In [6], we have illustrated their application in developing parallel algorithms for Monte Carlo methods used in solving linear algebraic equations and partial differential equations, and the details of the parallel algorithms are also given. In this paper, we have derived the time complexity results of these schemes. The analyses of SCA schemes were carried out using some results from order statistics, whereas those of DCA schemes were carried out using these results as well as some results from queueing theory and renewal theory. The time complexity results obtained for these schemes are independent of PEC time distributions and therefore are universal. The importance of such distribution-free results stems from the fact that for many Monte Carlo methods it is almost impossible to obtain the PEC time distributions a priori (see [6, pp. 92–165]). These schemes are shown to achieve either $O(P)$ or almost equal to P speedup, when the number of processors (P) is much smaller than the required number of primary estimates. However, as P approaches K this is no longer the case; the slope of the speedup decreases as the number of processors increases. We have shown that when the PEC times are standardized exponential random variables, it is possible to obtain exact time complexity results, rather than bounds.

Finally, we have illustrated the performance of the various schemes for parallel Monte Carlo algorithms with the solution of Laplace's equation.

The treatment in the paper assumes that "ideal" random number generators are used to drive Monte Carlo computations. However, since pseudorandom number generators are usually employed in practice, we have discussed some problems in pseudorandom number generation in parallel computations in [14], [15], [36].

We hope that the results in this paper, in [6], and our recent work [8]–[10] generates interest in the design and analysis of parallel Monte Carlo algorithms.

Acknowledgments. The authors would like to thank Dr. S. P. Nawathe, Prof. M. N. Vartak, Prof. S. L. Mehndiratta and Prof. S. Arunkumar of the Indian Institute of Technology, Bombay, for their helpful suggestions during this research work. The authors would also like to thank the referees for useful comments.

REFERENCES

[1] G. M. BAUDET, *The design and analysis of algorithms for asynchronous multiprocessors*, Ph.D. dissertation, Carnegie-Mellon University, Pittsburgh, 1978.

[2] V. C. BHAVSAR AND V. V. KANETKAR, *A multiple microprocessor system* (MMPS) *for the Monte Carlo solution of partial differential equations*, in Advances in Computer Methods for Partial Differential Equations, vol. 2, R. Vichnevetsky, ed., IMACS, New Brunswick, Canada, 1977, pp. 205–213.

[3] V. C. BHAVSAR AND A. J. PADGAONKAR, *Effectiveness of some parallel computer architectures for Monte Carlo solution of partial differential equations*, in Advances in Computer Methods for Partial Differential Equations, vol. 3, R. Vichnevetsky and R. S. Stepleman, eds. IMACS, New Brunswick, Canada, 1979, pp. 259-264.

[4] V. C. BHAVSAR AND J. R. ISAAC, *Some parallel algorithms for Monte Carlo solution of linear equations*, in Proc. 15th Annual Convention of Computer Society of India, Bombay, Feb. 8-11, 1980, vol. III, pp. 3.76-3.81.

[5] V. C. BHAVSAR, *Some parallel algorithms for Monte Carlo solutions of partial differential equations*, in Advances in Computer Methods for Partial Differential Equations, vol. 4, R. Vichnevetsky and R. S. Stepleman, eds. IMACS, New Brunswick, Canada, 1981, pp. 135-141.

[6] ———, *Parallel algorithms for Monte Carlo solutions of some linear operator problems*, Ph.D. thesis, Department of Electrical Engineering, Indian Institute of Technology, Bombay, Nov. 1981.

[7] V. C. BHAVSAR AND J. R. ISAAC, *Design and analysis of parallel algorithms for Monte Carlo methods*, in Proc. 10th IMACS World Congress on "System Simulation and Scientific Computation," Montreal, Canada, Aug. 8-13, 1982, IMACS, New Brunswick, Canada, 2, pp. 323-325.

[8] V. C. BHAVSAR, VLSI *algorithms for Monte Carlo solution of linear equations*, presented at the APICS Annual Computer Science Seminar, St. Francis Xavier Univ., Antigonish, N.S., Canada, 4-5, Nov. 1983.

[9] ———, VLSI *algorithms for Monte Carlo solutions of partial differential equations*, in Advances in Computer Methods for Partial Differential Equations, vol. 5, R. Vichnevetsky and R. S. Stepleman, eds., IMACS, New Brunswick, Canada, 1984, pp. 268-276.

[10] V. C. BHAVSAR AND U. G. GUJAR, VLSI *architectures and their applications to Monte Carlo simulation*, in Proc. IEEE International Conference on "Computers, Systems and Signal Processing," Indian Institute of Science, Bangalore, India, 3 (1984), pp. 1252-1255.

[11] K. BINDER, ed., *Monte Carlo Methods in Statistical Physics*, Springer-Verlag, Berlin, 1979.

[12] K. C. BOWLER AND G. S. PAWLEY, *Molecular dynamics and Monte Carlo simulations in solid-state and elementary particle physics*, Proc. IEEE, 72, 1 (1984), pp. 42-55.

[13] E. G. COFFMAN, JR., ed., *Computer and Job-Shop Scheduling Theory*, John Wiley, New York, 1976.

[14] D. R. COX, *Renewal Theory*, Methuen, London, 1962.

[15] H. A. DAVID, *Order Statistics*, John Wiley, New York, 1970.

[16] M. J. GONZALEZ, JR., *Deterministic processor scheduling*, ACM Computing Surveys, 9 (1977), pp. 173-204.

[17] E. J. GUMBEL, *Statistical theory of extreme values (main results)*, as Chapter 6 in Contributions to Order Statistics, A. E. Sarahan, and B. G. Greenberg, eds., John Wiley, New York, 1962.

[18] J. H. HALTON, *A retrospective and prospective survey of the Monte Carlo method*, SIAM Rev., 12 (1970), pp. 1-63.

[19] J. M. HAMMERSLEY AND D. C. HANDSCOMB, *Monte Carlo Methods*, Methuen, London, 1964.

[20] L. S. HAYNES, R. L. LAU, D. P. SIEWIOREK AND D. W. MIZELL, *A survey of highly parallel computing*, Computer, 15, 1 (1982), p. 9-24.

[21] D. HELLER, *A survey of parallel algorithms in numerical linear algebra*, SIAM Rev., 20 (1978), pp. 740-777.

[22] D. E. KNUTH, *The Art of Computer Programming, vol. 1, Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1968.

[23] H. KOBAYASHI, *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology*, Addison-Wesley, Reading, MA, 1978.

[24] E. V. KRISHNAMURTHY AND S. K. SEN, *Computer-Based Numerical Algorithms*, Affiliated East West Press, New Delhi, 1976.

[25] D. J. KUCK, *A survey of parallel machine organization and programming*, Computing Surveys, 9, 1 (1977), pp. 29-59.

[26] H. T. KUNG, *Synchronized and asynchronous parallel algorithms for multiprocessors*, in Algorithms and Complexity: New Directions and Recent Results, J. F. Traub, ed., Academic Press, New York, 1979, pp. 153-200.

[27] ———, *The Structure of Parallel Algorithms*, Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, CMU-CS-79-149, 1979. Also in Advances in Compters, M.C. Yovits, ed., Academic Press, New York, vol. 19, 1980.

[28] L. LAMBROU, V. C. BHAVSAR, AND U. G. GUJAR, *On the discrepancy of pseudo-random number sequences generated by the linear congruential method*, in Proc. Annual APICS Computer Science Seminar, Dept. of Mathematics, Statistics and Computing Science, Dalhousie University, Halifax, N.S., Canada, Nov. 1985, pp. 53-68.

[29] L. A. LAMBROU, *Pseudo-Random Number Sequences for Parallel Computers*, School of Computer Science, Univ. of New Brunswick, Fredericton, N.B., Canada, M.Sc. Project Report, TR86-033, Feb. 1986.

[30] M. METROPOLIS AND S. M. ULAM, *The Monte Carlo method*, J. Amer. Statist. Assoc., 44 (1949), pp. 335-341.

[31] H. NIEDERREITER, *Quasi-Monte Carlo methods and pseudorandom numbers*, Bull. Amer. Math. Soc., 84 (1978), pp. 957-1041.

[32] J. T. ROBINSON, *Some analysis techniques for asynchronous multiprocessor algorithms*, IEEE Trans. Software Engrg., SE-5, 1 (1979), pp. 24-31.

[33] R. F. ROSIN, *An Algorithm for concurrent random walks on highly parallel machines*, Cooley Electronics Lab., Univ. Michigan, Ann Arbor, TR 151, 1964.

[34] E. SADEH AND M. A. FRANKLIN, *Monte Carlo solution of partial differential equations by special purpose digital computer*, IEEE Trans. Comput., C-23 (1974), pp. 389-397.

[35] A. L. SCHERR, *An Analysis of Time-Shared Computer Systems*, MIT Press, Cambridge, 1966.

[36] YU. A. SCHREIDER, ed., *The Monte Carlo Method*, Pergamon Press, Oxford, 1966.

[37] J. SPANIER AND E. M. GELBARD, *Monte Carlo Principles and Neutron Transport Problems*, Addison-Wesley, Reading, MA, 1969.

SIAM. J. SCI. STAT. COMPUT.
Vol. 8, No. 1, January 1987
© 1987 Society for Industrial and Applied Mathematics

ADAPTING A NAVIER–STOKES CODE TO THE ICL-DAP*

CHESTER E. GROSCH†

Abstract. This is a report of the results of an experiment: to adapt a Navier-Stokes code, originally developed on a serial computer, to concurrent processing on the ICL Distributed Array Processor (DAP). In this paper the algorithm used in solving the Navier-Stokes equations is briefly described. The architecture of the DAP and DAP Fortran is also described. The modifications of the algorithm so as to fit the DAP are given and discussed. Finally, performance results are given and conclusions are drawn.

Key words. Navier-Stokes, parallel computing, ICL-DAP

AMS(MOS) subject classifications. 76D05, 35Q10, 68C25

1. Introduction. The objective of the research reported here was to adapt a Navier-Stokes code, originally developed on a serial computer, to concurrent processing on the ICL-DAP and to measure the performance of the concurrent version of the code.

Despite the fact that the most powerful existing computers can perform several hundred MFLOPS (million floating point operations per second), they are clearly inadequate for many important applications. In part, this inadequacy is due to the fact that it is generally rather difficult to fully use the vector capabilities of these "supercomputers." In practice, average processing rates for many codes are in the range of ten to twenty MFLOPS (see, for example, [2]). Even if the average processing efficiency could be increased so as to more nearly reflect the potential processing power of the vector processors, the processing rates would still be inadequate for many fluid dynamic applications.

Computational fluid dynamics requires both very large amounts of fast primary storage, even larger amounts of slow, secondary storage and high average processing rates. Different individuals and groups have independently estimated that current needs for some fluid dynamic calculations are, on average, 10^3 MFLOPS processing rates, 32×10^6 words of fast primary storage, and 2.56×10^8 words of slow secondary storage. Even higher average processing rates and larger primary and secondary storage will be needed in the near future.

It appears that future supercomputers will be multiprocessors; in fact some have already appeared. There are a number of important, unresolved questions concerning these multiprocessor computers. Among these issues are: should they consist of a few, rather powerful processors or many, very much less powerful processors, or something in between? Should the new computers be SIMD or MIMD? There is a natural expectation that the multiprocessors with a few, powerful processors will have an MIMD architecture and that the others will have SIMD architectures. It seems sterile at this point to argue what, taking into account total processing power, cost, ease of programming, etc., is the "best" combination of number of processors and power per processor. Rather, carrying out experiments with existing multiprocessor computers would appear to be of greater value.

* Received by the editors December 2, 1985; accepted for publication (in revised form) April 10, 1986.

† Old Dominion University, Norfolk, Virginia 23508; British Maritime Technology, Teddington TW11 0JJ, England; Institute for Computer Applications in Science and Engineering, Hampton, Virginia 23665-5225. This research was supported by the National Aeronautics and Space Administration under NASA contracts NAS1-17070 and NAS1-18107 while the author was in residence at ICASE, NASA Langley Research Center, Hampton, Virginia 23665-5225.

One approach to concurrent processing for computational fluid dynamics could be to develop highly parallel algorithmic modules or kernels to perform certain computational tasks. Complete computational fluid dynamic algorithms would then be built by combining these modules. This approach has the advantage that "good," highly parallel algorithms well suited to the architecture would be used. It has the disadvantage that much of the algorithmic and code development effort of the past would be unusable and existing production codes would have to be abandoned. It would, conceivably, be many years before new production codes would be available.

An alternate approach could be to translate effective algorithms, now embodied in codes for single processor computers, onto the multiprocessor computer. This has the advantage of using existing, tested algorithms. The disadvantage is that, because the algorithm may not "fit" the multiprocessor architecture, this may be very difficult or even impossible. If such translation is possible, the algorithm may not execute effectively on the multiprocessor, that is the computational cost may be very high.

There have been a substantial number of theoretical studies of the performance of algorithms on parallel computers but far fewer actual experimental studies (see [11] for a comprehensive, up-to-date review). Most of the experimental studies of applications discussed by Ortega and Voigt (see [11, § 5]) are in the area of fluid dynamics. Apart from some early studies using Illiac IV, nearly all of this work has concentrated on the use of the Cyber 200 and Cray series of computers. The ICL-DAP is one of the few working, both in software as well as hardware, SIMD computers in existence. The MPP is another, with an architecture very similar to that of the DAP. Another large, parallel SIMD architecture is that of the Connection Machine computer. Both of these computers have a large number of single bit processors. The MPP has the same nearest neighbor connection as the DAP. The Connection Machine computer has a variable interconnection network, with a nearest neighbor connection as one possibility. It appears that there are no studies of the implementation of Navier-Stokes algorithms on either of these computers. Insofar as MIMD computers are concerned, there are even fewer actual studies of the implementation of Navier-Stokes algorithms. The exceptions are some applications using the Denelcor HEP cited by Ortega and Voigt [11].

Given the paucity of studies of complex applications to physical problems using parallel computers, it seems reasonable to carry out some now in order to measure the performance of these parallel architectures. Although there has been increasing interest in MIMD architectures and the appearance of a number of them, it is by no means clear that they are intrinsically superior to SIMD architectures for fluid dynamic applications. Both SIMD and MIMD architectures have intrinsic bottlenecks which can be revealed by experimental studies of their use in applications as well as by theoretical analysis. In view of these considerations, it seems worthwhile to measure the actual performance of an application code on a working SIMD computer. Even if one agrees that measured performance on an actual parallel processor is the true measure of computing power, there can be disagreements as to what performance is to be measured.

The time to perform a single arithmetic operation, together with the number of processors, can be used to calculate an upper bound on performance. Such a measure is widely held to be unrealistic because it does not include any of the omnipresent overhead. A different approach is to use, as a test problem, the evaluation of a slightly more complex expression such as a vector dyad, the sum or product of two vectors or a vector triad, the sum of a vector and the product of two vectors, etc. (see Hockney and Jesshope [7] and Hockney [8] for examples). While certainly more realistic than

using the single operation time, the use of these test problems can be criticized because they fail to measure the overhead associated with a complex algorithm and its embodiment in a specific program in a particular programming language. Thus, these types of tests can be said to give only a measure of the maximum performance of a microsegment of a code. The average performance of a complex scientific code is probably quite different, and less than that of some microsegment. Another approach is to run a specific algorithm on a particular computer. Despite the fact that this is a very specific experiment this approach has some advantages; namely it allows an objective measurement of the performance of a complete algorithm, albeit a specific one, expressed in a specific parallel language, and executed on a specific parallel processor; and it can also yield subjective evidence as to how well an algorithm fits the architecture, how difficult it was to program in the parallel language, and so on.

The work reported here was just such an experiment. A Navier-Stokes code, embodying a compact, finite difference form of a vorticity, velocity formulation of the Navier-Stokes equations (Gatski, Grosch and Rose [3]) was reprogrammed in DAP Fortran and run on the ICL DAP. The Navier-Stokes formulation and algorithm are nonstandard but appear to be about as complicated as the standard ones. In order for this to be a realistic study, the entire code was implemented. This includes initialization, input/output, calculation of the time-dependent boundary conditions, calculation of energy, vorticity and enstrophy norms, as well as the solution for the velocity and vorticity fields. This was done so as not to overlook any hidden bottlenecks. Although the numerical scheme and the DAP have been described elsewhere, they are briefly described in §§ 2 and 3 for the sake of completeness. The implementation is described in § 4. The results are given in § 5. Finally, § 6 contains some concluding remarks.

2. The algorithm. The basic algorithm used here to solve the Navier-Stokes equations was first described, along with the results of some test cases, by Gatski, Grosch and Rose [3]. It has since been used by McInville, Gatski and Hassan [10] to study the instability and subsequent nonlinear evolution of a shear layer; by Gatski and Grosch [4] to study the flow past an open cavity in a boundary; and finally is being used to study the separating flow past a backward-facing step and the impulsive start of elliptic cylinders [5]. As mentioned above, it is briefly described here for the sake of completeness.

The Navier-Stokes equations for the two-dimensional, time-dependent flow of a viscous, incompressible fluid may be written, in dimensionless variables, as:

$$u_x + v_y = 0, \tag{2.1}$$

$$v_x - u_y = \zeta, \tag{2.2}$$

$$\zeta_t + u\zeta_x + v\zeta_y = \mathrm{Re}^{-1}(\zeta_{xx} + \zeta_{yy}), \tag{2.3}$$

where $\vec{u} = (u, v)$ is the velocity, ζ is the vorticity and Re is the Reynolds number.

The finite difference scheme used to approximate these equations is based on a compact difference method described by Rose [13] and Philips and Rose [12]. These schemes involve only variables within and on the boundaries of a single cell and are second order accurate, with the accuracy independent of the local cell Reynolds number. Apart from the independence of the accuracy on the magnitude of the cell Reynolds number, these compact schemes have certain other advantages in that it is quite simple and straightforward to use nonuniform grids and to impose boundary conditions.

The compact difference approximation to equations (2.1) to (2.3) can be briefly described. Consider a rectangular cell, centered at (x_i, y_j), with length Δx and height

Δy, see Fig. 1, and let $U_{ij}^n \equiv u(i\Delta x, j\Delta y, n\Delta t)$, for i, j, and n integers. The variables shown on cell faces of Fig. 1 are the averages of the continuous variable over that face and that at the center of a cell is the average over the entire cell. Define the difference and average operators by:

$$\delta_x U_{i,j}^n \equiv (U_{i+1/2,j}^n - U_{i-1/2,j}^n)/\Delta x, \tag{2.4a}$$

$$\mu_x U_{i,j}^n \equiv (U_{i+1/2,j}^n + U_{i-1/2,j}^n)/2. \tag{2.4b}$$

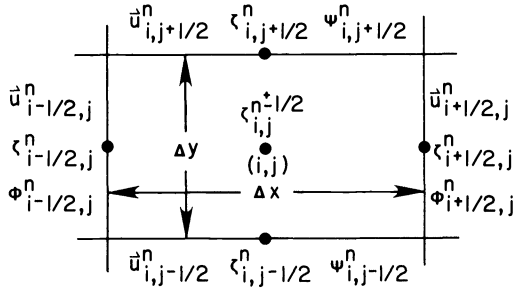


FIG. 1. *Typical computational cell and the data values associated with it.*

Then (2.1) to (2.2), hereafter the velocity equations, are replaced by:

$$\delta_x U_{i,j}^n + \delta_y V_{i,j}^n = 0, \tag{2.5}$$

$$\delta_x V_{i,j}^n - \delta_y U_{i,j}^n = \zeta_{i,j}^{n-1/2}, \tag{2.6}$$

$$\mu_x U_{i,j}^n = \mu_y U_{i,j}^n, \tag{2.7}$$

$$\mu_x V_{i,j}^n = \mu_y V_{i,j}^n. \tag{2.8}$$

Assuming that ζ is known, the boundary conditions for the Cauchy-Riemann equations (2.1) and (2.2) are that one, and only one, component of the velocity is specified at each point of the boundary. For the discrete equations (2.5)-(2.8) it is easy to see that if there are N by M cells, the number of unknowns is $2(2NM + N + M)$, the number of equations is $4NM$, and the number of boundary conditions is $2(N + M)$ so that the algebraic system is determined.

The vorticity transport equation (2.33) is approximated by the system:

$$[\delta_t + (\mu_x U_{i,j}^n)\delta_x + (\mu_y V_{i,j}^n)\delta_y]\zeta_{i,j}^n = \mathrm{Re}^{-1}[\delta_x \phi_{i,j}^n + \delta_y \psi_{i,j}^n], \tag{2.9}$$

$$\delta_x \zeta_{i,j}^n = (\mu_x - \tfrac{1}{2}\Delta x q(\theta_x)\delta_x)\phi_{i,j}^n, \tag{2.10}$$

$$\delta_y \zeta_{i,j}^n = (\mu_y - \tfrac{1}{2}\Delta y q(\theta_y)\delta_y)\psi_{i,j}^n, \tag{2.11}$$

$$\mu_t \zeta_{i,j}^n = \mu_x \zeta_{i,j}^n = \mu_y \zeta_{i,j}^n. \tag{2.12}$$

Here, θ_x and θ_y are local cell Reynolds numbers given by

$$\theta_x = U_{i,j}^n \,\mathrm{Re}\, \Delta x/2, \tag{2.13}$$

$$\theta_y = V_{i,j}^n \,\mathrm{Re}\, \Delta y/2, \tag{2.14}$$

$$q(\theta) \equiv (\coth \theta) - (1/\theta). \tag{2.15}$$

As was shown by Gatski et al. [3], the parameter $q(\theta)$ serves to reduce the truncation error. Of course $q(\theta)$ is not computed using the definition, equation (2.15), as that would be prohibitively expensive because of the cost of calculating coth (θ).

Instead the approximation

$$q(\theta) = \theta/3 \qquad \text{for } \theta \text{ small,} \tag{2.16a}$$

$$q(\theta) = (\operatorname{sgn}\theta) - (1/\theta) \qquad \text{for } \theta \text{ large,} \tag{2.16b}$$

is used, with $\operatorname{sgn}\theta \equiv \theta/|\theta|$.

The solution procedure for these finite difference equations is

Step (1). Assume that $\zeta_{i,j}^{n-1/2}$ is known. Then, with one component of $\vec{u}$ prescribed on the boundary, solve (2.5) to (2.8) by relaxation.

Step (2). Determine the vorticity boundary conditions. At inflow boundaries ζ is prescribed. At solid boundaries, an increment of vorticity is created so that the tangential velocity component will equal the speed of the boundary. At outflow and freestream boundaries the vorticity is determined by the flux condition,

$$\frac{D\zeta}{Dt} \equiv \frac{\partial \zeta}{\partial t} + (\vec{u} \cdot \nabla)\zeta = 0. \tag{2.17}$$

Step (3). Solve (2.9) to (2.11) using an ADI scheme, described in detail by Gatski et al. [3]. Then use equation (2.12) to advance ζ in time, yielding $\zeta_{i,j}^{n+1/2}$.

Further details, as needed, will be given in § 4, below, wherein the implementation on the DAP is described.

3. The ICL–DAP and DAP FORTRAN. A brief description of the DAP architecture is given here in order to clarify the way in which the Navier-Stokes code, described in § 2, was adapted to the DAP architecture. Both the hardware and software are described in order that one can appreciate some of the most important features of the DAP that contribute to its limitations and advantages. These features include, among others, the use of the control unit, the advantages and disadvantages of a bit level instruction set to implement high level instructions, the data storage scheme, and the local and global communication paths. Particular emphasis is put on the software. The Fortran functions have some rather surprising performance rates; particularly the data movement and global reduction functions. As will be seen, this large suite of efficient high level software is the major reason that the DAP can be routinely used for scientific applications rather than remain an experimental architecture. A detailed description of the DAP is given by Hockney and Jesshope [7] and in the ICL references which they cite.

The DAP is an SIMD computer with a processor array, a control unit, and an access control unit, see Fig. 2. The complete processor is embedded within an ICL 2900 system and can be used as a conventional storage unit by the 2900. Access to the DAP is through the DAP access control unit. The 2900 initiates the execution of a DAP program by passing control to the DAP master control unit. Once this is done, the DAP retains control; the 2900 cannot interrupt program execution.

The DAP control unit is similar to the control unit of a conventional serial computer. There is an instruction counter, instruction register, instruction buffer, modifier register, and eight control registers. The instruction buffer can hold up to 60 instructions, each of which is 32 bits long. The modifier register acts as a base register for memory references and the control registers are used for data and/or instruction modifications. The DAP processes data in a bit serial mode. A 200 ns fetch cycle, in which two instructions are fetched, and a 200 ns execute cycle are required for an operation on a single bit. More complex operations such as integer addition, floating point addition, etc. are microprogrammed using the primitive hardware operations. This is facilitated by the use of a hardware loop capability, through the instruction

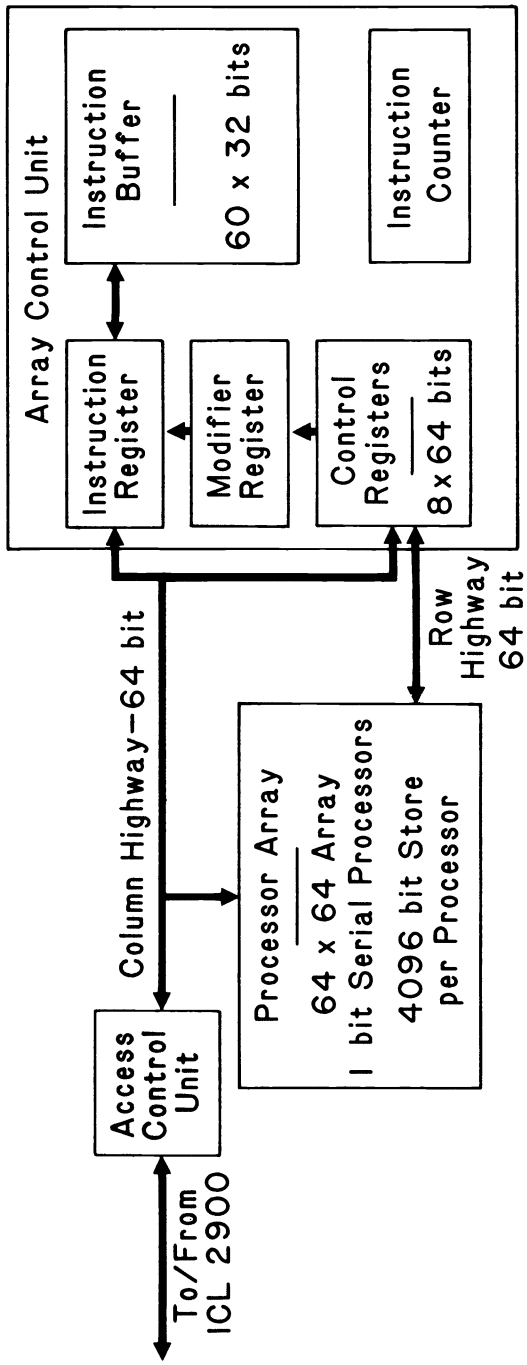


FIG. 2. *Schematic diagram of the* DAP *architecture.*

buffer, which permits the rate of instruction execution to be reduced, asymptotically, to one per cycle. The execution time for typical high level instruction such as a floating point addition is quite long, requiring from about ten to several hundred microseconds. The computational power of the DAP results from the fact that the arithmetic is performed in a processor array, not a single processor.

The DAP processor array consists of 4096 single bit processors arranged in a 64 by 64 array. Each processor has a one bit full adder, three one bit registers, an input multiplexer, an output multiplexer, as well as a 4096 bit memory. Thus the processor array has a total of two megabytes of memory. This memory is used to store both data and DAP code, so that in practice the theoretical maximum of 128 words, each of 32 bits, per processor is usually reduced to about 100 words of 32 bit data per processor. The DAP has three basic data modes; matrices which are 64 by 64 arrays, vectors having 64 or more elements, and scalars.

The storage of instructions and data can be easily visualized by considering the DAP memory as a three-dimensional array of 64 rows, 64 columns, and 4096 planes. The 32 bit instructions are stored in planes with two per row. Matrices are stored on end, so to speak, with each plane containing one bit of the words. Thus matrices of 32 bit floating point words require 32 planes but logical matrices only a single plane. Vectors with 64 elements are, on the other hand, stored in one plane with one element per row. Scalars are treated similarly to vectors.

The DAP array has both global and local communication paths. The global paths are one bit wide row and column highways (buses). The single bit column highways interface with the DAP access controller and thus provide the data paths connecting the DAP to the 2900 host. The column highways are also connected to the instruction register and master control registers. Instructions are fetched from the DAP memory along the column highways and are sent to the instruction register. Data and instructions can also be moved along the column highways to the master control registers where they can be modified. In contrast the one bit wide row highways are only connected to the master control registers and are only used to transmit data.

Apart from the data paths internal to the processor, each processor in the array is connected by four, one bit wide data paths to the four nearest neighbor processors. Thus instructions may refer to data elements in nearest neighbor processors. The edge connections can be defined by software in two ways: planar or cyclic. With a planar connection, data passed out at an edge is lost and input at an edge is defined to be zero. The cyclic connection defines, as nearest neighbors, the first and last elements of rows or columns or both.

ICL provides a DAP assembly language and a DAP Fortran, see ICL Manual, DAP: Fortran Language [1]. DAP Fortran is an extension of standard Fortran and incorporates matrix and vector instructions. There is usually very little advantage to using the assembly language instead of DAP Fortran, as the Fortran routines are highly optimized and the overhead associated with them is only of the order of 10%. Some typical Fortran statements, their execution times, and processing rates are listed in Table 1.

The processing rate is found by dividing the number of operations, 4096 for matrices and 64 for vectors, by the execution time. Although this is quite straightforward for a simple arithmetic operation, such as the addition of two matrices, there is a difficulty when we consider data transfer operations. Traditional complexity analysis, that is operation counting, has ignored the cost (time) of data movement. While this may be justified for conventional architectures, it is certainly not always true for processor arrays such as the DAP. For example, Grosch [6] has shown that the data

TABLE 1

Examples of DAP *Fortran statements, execution times, and the corresponding processing rates. Here X, Y, and Z are* 64 *by* 64 *matrices of* 32 *bit floating point numbers; L is a* 64 *by* 64 *single bit logical matrix, i.e. a mask; V is a* 64 *element vector of* 32 *bit floating point numbers; S is a* 32 *bit floating point scalar.*

DAP Fortran statement	Execution time (microseconds)	Processing rate (MFLOPS)
$Z = X$	17	241
$Z = X + Y$	175	23
$Z(L) = X + Y$	179	23
$Z = X * Y$	274	15
$Z = X / Y$	386	11
$Y = \text{SQRT}(X)$	194	21
$Y = \text{EXP}(X)$	414	10
$Y = \text{SIN}(X)$	862	5
$Y = \text{TRAN}(X)$	714	6
$X = \text{MATC}(V)$	31	132
$X = \text{MATR}(V)$	31	132
$Z = \text{MERGE}(X, Y, L)$	50	82
$Y = \text{SHEP}(X, 32)$	227	577
$Y = X(,-)$	23	178
$Y = \text{SHEC}(X)$	23	178
$Z = X(+,) + Y(,-)$	221	56
$Z = X(+,+) + Y(-,+)$	267	77
$S = \text{MAXV}(X)$	56	73
$S = \text{MAXV}(X, L)$	57	72
$S = \text{SUM}(X)$	450	9
CALL CONVFME (X)	2252	2

transfer cost (time) can be the major portion of the cost in the implementation of relaxation and direct Poisson solvers on processor arrays with architectures similar to the DAP. Because of the importance, and possible dominance, of the data transfer cost, all data transfers are counted as floating point operations and the time to complete them is included in the total operation time. Thus, the first Fortran statement, a matrix transfer, is counted as 4K floating point operations with an execution time of 17 μs, yielding a processing rate of 241 MFLOPS.

Logical matrices may be used as masks in arithmetic operations. The third Fortran statement in Table 1 is an example of this capability. The index, L, is a 64 by 64 logical matrix of one bit elements. The results of the right-hand side arithmetic operation, here an addition, are stored in those elements of Z for which the corresponding elements of L are true.

DAP Fortran provides an extensive set of built-in functions, which belong to one of three types: computational, aggregate, and error management. The computational functions include most of the standard Fortran functions, ABS, SQRT, EXP, SIN, and so forth. The execution times and processing rates for some of these are listed in Table 1. These timings show the advantages of bit level processing. For example, a SQRT is calculated in only about 111% of an addition time because a bit level Newton iteration scheme is used. Similarly, the EXP is evaluated in little more than one divide time. Although only examples of matrix functions are given in Table 1, the arguments of the computational functions may be of any mode (matrix, vector, or scalars) and the result will be of the same mode.

The aggregate functions perform nonnumeric operations, that is data manipulations, on vectors or matrices. Although generally not considered in standard complexity analysis, efficient implementation of the aggregate functions can be the "sine qua non" for efficient use of processor arrays. The implementation of this class of functions on the DAP is very efficient, for example the TRAN function, which transposes a matrix, is completed in about four addition times.

Mixed mode expressions, containing scalars and vectors or scalars and matrices, are permitted in DAP Fortran. The scalars are expanded to either vectors or matrices in context. The vector-matrix combination is invalid because of the ambiguity in interpretation; the vector could be expanded either as a matrix of column vectors or of row vectors. The programmer must, in order to have a vector-matrix combination, use either MATC which generates a matrix of column vectors or MATR which generates a matrix of row vectors.

The MERGE function sets the elements of matrix Z equal to the corresponding elements of X, for those elements of L which are true, and to the corresponding elements of Y, for those elements of L which are false. This function can be used to implement conditional, data dependent calculations. For example, a branch instruction with two branches can be implemented by (1) calculating the results of the first branch and storing them in matrix X; (2) calculating the results of the second branch and storing them in Y; (3) generating the mask using the data test with, say, true denoting the first branch and false the second branch; (4) finally, calling MERGE.

By convention the first index of a matrix, the I of $X(I, J)$, labels the rows and the second labels the columns, see Fig. 3. The DAP convention labels the first row of the array as the North and the first column as the West. These conventions are used in the shifting functions, SHEP(X, N) for example. This function shifts (SH) the X matrix N columns to the East (E), i.e., the first column of X becomes the $N+1$ column. The P denotes a planer shift so that the last N columns of X are shifted out of the array and the first N columns are filled with zeros. If N is not given it will be

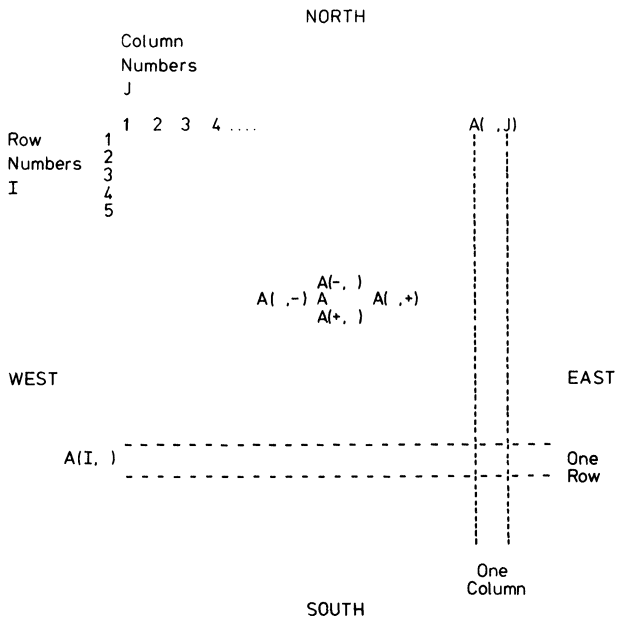


FIG. 3. *Matrix storage and indexing conventions on the* DAP.

taken to be one, and if N is greater than or equal to 64, the shift will be modulo 64. The SHEC function performs a cyclic shift in the East-West direction on the matrix. Single row or column shifts can also be accomplished by using the + or − notation. For example, $X(,-)$ is equivalent to SHEP(X). When using the ± notation the shifts are planer by default, but a GEOMETRY declaration can be used in any subroutine to declare the East-West or North-South shifts, or both, to be cyclic.

The MAXV function returns the maximum element of X, with an option to test only those elements of X for which the corresponding elements of L are true. The use of bit level processing yields high efficiency for this function. Finally, the SUM function calculates the sum of all of the elements of X. It is noteworthy that this requires an execution time which is slightly less than two addition times, rather than the 12 ($4096 = 2^{12}$) that might have been expected.

Figure 4 shows in block form the structure of a typical program. There can be more than one DAP entry subroutine in a Fortran program, as is shown in this figure. The call to the DAP entry subroutine must not have any arguments; all data is transferred via COMMON blocks. Immediately after entering the DAP entry routine and, again, before returning to the calling routine, the data in COMMON must be converted. This is because the data storage of standard Fortran is different from that of the DAP, as described above. The conversion routines, for example CONVFME (see Table 1), are extremely expensive in time and so the number of transfers between standard and DAP Fortran should be minimized. In particular, this affects the way in which input

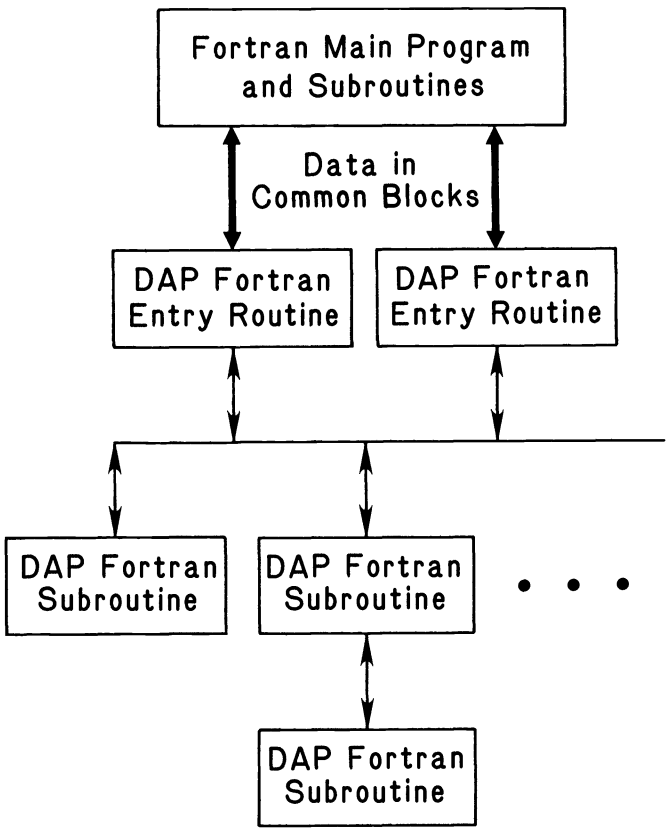


FIG. 4. *Structure of a typical Fortran program for the* DAP.

and output must be handled. Because of the difference in data storage format, there is no input or output from the DAP. The program must return from the DAP segment, at the cost of a conversion, perform the input and/or output and return to the DAP, at the cost of another conversion. Because of this overhead cost of conversion, input and output calls must be minimized. The data can be stored and output, say, done after final return from the DAP.

4. Implementation. The data structure of this algorithm, as presented in § 2, does not quite fit the DAP architecture. This is because the difference scheme is a compact one with the dependent variables defined on the cell edges rather than at the corners of a cell as in more familiar schemes. For an N by N array of cells there are N by $N+1$ cell sides and $N+1$ by N cell tops. So with the velocity, for example, defined on the cell edges the data arrays do not map directly onto an N by N array of processors.

On the other hand, there are certain advantages to using the compact difference scheme on the DAP. First, the application of boundary conditions is quite simple. There is no need to introduce "ghost" points outside of physical boundaries. Second, all derivatives are evaluated using variables which can be seen to be nearest neighbors on the computational grid (see § 2). Thus, the amount of long range communication is minimized.

The adaptation of this algorithm to the DAP architecture can be simplified by the introduction of box variables to represent the velocity field. The center of a cell is at (i, j). The box variables, $\vec{P}$, are defined at the corners of the cells, points $(i \pm 1/2, j \pm 1/2)$. They are related to the velocity $\vec{U}$ by

$$\vec{U}_{i,j\pm 1/2} = (\vec{P}_{i+1/2,j\pm 1/2} + \vec{P}_{i-1/2,j\pm 1/2})/2, \tag{4.1}$$

$$\vec{U}_{i\pm 1/2,j} = (\vec{P}_{i\pm 1/2,j+1/2} + \vec{P}_{i\pm 1/2,j-1/2})/2. \tag{4.2}$$

The (P, Q) variables are not components of the velocity, $\vec{U} = (U, V)$. They are simply auxiliary variables related to the velocity field, averaged over the cell sides, by (4.1) and (4.2). In the vorticity equations, (2.9) to (2.15), these equations, (4.1) and (4.2), are used to calculate the velocity field.

It is obvious that (2.7) and (2.8) are satisfied identically for any set of box variables. For the cell (i, j), (2.5) and (2.6) are

$$AP = Z \tag{4.3}$$

where

$$\vec{P}_{ij} = (P_{ij}, Q_{ij})^T, \tag{4.4}$$

and

$$P = \begin{pmatrix} \vec{P}_{i+1/2,j-1/2} \\ \vec{P}_{i+1/2,j+1/2} \\ \vec{P}_{i-1/2,j+1/2} \\ \vec{P}_{i-1/2,j-1/2} \end{pmatrix}, \qquad Z \equiv \begin{pmatrix} 0 \\ 2(\Delta y)_i \zeta_{i,j}^{n-1/2} \end{pmatrix}, \tag{4.5a, b}$$

$$A = \begin{pmatrix} \lambda_{i,j} & -1 & \lambda_{i,j} & 1 & -\lambda_{i,j} & 1 & -\lambda_{i,j} & -1 \\ 1 & \lambda_{i,j} & -1 & \lambda_{i,j} & -1 & -\lambda_{i,j} & 1 & -\lambda_{i,j} \end{pmatrix}, \tag{4.6}$$

$$\lambda_{i,j} \equiv (\Delta y)_i/(\Delta x)_j \tag{4.7}$$

is the aspect ratio of cell (i, j).

The box variables lie at the vertices of the computational grid. The storage pattern used on the DAP is to store $\vec{P}_{i-1/2,j+1/2}$, $\zeta_{i,j}$, and $\lambda_{i,j}$ in memory of processor (i, j).

Thus with a 64×64 array of processors there is an array of 63×63 cells and for each cell we have (4.3).

The set of equations is solved by an iteration scheme which was originally proposed by Kaczmarz [9] and generalized by Tanabe [14]. If $P^{(k)}$ is the value after the kth iteration, then the residual after the kth iteration, $R^{(k)}$, is given by

$$R^{(k)} = AP^{(k)} - Z. \tag{4.8}$$

The next iteration is

$$P^{(k+1)} = P^{(k)} - \omega A^T (AA^T)^{-1} R^{(k)}, \tag{4.9}$$

which, if $\omega \equiv 1$, would give

$$R^{(k+1)} \equiv 0. \tag{4.10}$$

For $\omega \neq 1$, this is an SOR scheme. On a serial computer the array of computational cells is swept over, applying (4.9) to each, until the maximum residual is reduced to the desired level.

The key to the adaptation of this relaxation scheme to the DAP is the realization that each $\vec{P}$ is updated four times in a sequential sweep over the array of cells. For example, see Fig. 5, if the sweep is across the columns and then down the rows, $\vec{P}_{i+1/2,j+1/2}$ is changed during the relaxation of cell (i,j) first, then of cell $(i,j+1)$, then cell $(i+i,j)$, and finally cell $(i+1,j+1)$. In each of these cases $\vec{P}_{i+1/2,j+1/2}$ lies at a different corner of the cell being relaxed. It is therefore clear that the cell iteration scheme for the box variables is a four "color" scheme.

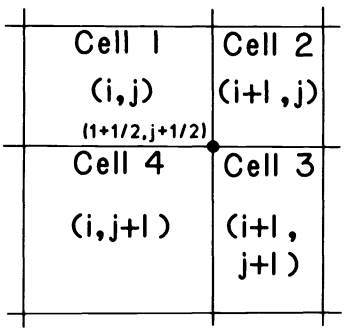


FIG. 5. *Computational cells bordering the grid point* $(i+\frac{1}{2}, j+\frac{1}{2})$.

The DAP scheme is therefore to relax all of the $\vec{P}$'s four times; in the first pass a particular $\vec{P}$ is treated as lying in the lower right-hand corner of the cell and is labelled 1; in the next pass, labelled 2, as lying in the lower left-hand corner; in the third, labelled 3, as lying in the upper left-hand corner; and finally in the last pass, labelled 4, as being in the upper right-hand corner of the cell.

In detail, the DAP algorithm is implemented by

(1) Computing the residuals, $R^{(k)}$, for all cells using (4.8).

(2) Computing the correction to $P^{(k)}$ for all cells, as given in (4.9). Note that the coefficients in the matrices A and

$$B \equiv A^T (AA^T)^{-1} \tag{4.11}$$

are precomputed once and for all and are stored.

(3) Correct and restore the P's.

This sequence must be completed four times in order to complete a sweep. The only difference between these sequences is the assignment of the data in a particular processor memory to one of the four logical positions in a computational cell. This is easily done using logical masks which also mask out boundary values. The overhead caused by these data transfer is about 7% of the total.

Extensive experiments, using test problems with known solutions, have been carried out on serial and vector computers and on the DAP. Constant grid sizes in both directions were used and the acceleration parameter, ω, was varied in order to find the value of ω giving a minimum spectral radius. The Kaczmarz iteration scheme was applied to the original set of equations, (2.5)-(2.8), and the box scheme, (4.3) to (4.7). On the serial computers both the original and box variable forms were relaxed using lexographic, red-black, and four color orderings. A vectorized version of the original scheme using red-black ordering was implemented on a CYBER 205. Only the four color scheme with box variables was implemented on the DAP. In all cases the results were nearly identical; the minimum spectral radius is approximately 0.96 and occurs at $\omega \approx 1.95$ for $\Delta x = \Delta y = 1/63$.

Figure 6 shows the kernel of the DAP relaxation routine. The subroutine call statement, COMMON statements, declaration of temporary arrays, shifts of the data into temporary arrays, etc. are not shown in this figure. The TP's, TQ's, and R's are temporary matrices of real numbers. The P, Q, matrices are the first and second components of the box variables, and ZT is defined by equation (4.5). Finally the ZLT matrix contains the values of $\lambda_{i,j}$ and CT is a matrix of coefficients obtained from equation (4.11). The acceleration parameter ω is included in CT.

The residuals for all cells are computed in the first block of six DAP Fortran statements. These are stored in R1 and R2. The next set of four statements computes the terms in the correction, see (4.9), to the current values. The box variables are finally updated in the next eight statements.

```
TP1 = P1 - P3
TP2 = P2 - P4
TQ1 = Q1 - Q3
TQ2 = Q2 - Q4
R1 = ZL1*(TP1+TP2)-TQ1+TQ2
R2 = TP1 - TP2+ZLT*(TQ1+TQ2)-ZT

TP1 = CT*R1
TQ1 = ZLT*TP1
TP2 = CT*R2
TQ2 = ZLT*TP2

P1 = P1-TQ1-TP2
Q1 = Q1+TP1-TQ2
P2 = P2-TQ1+TP2
Q2 = Q2-TP1-TQ2
P3 = P3+TQ1+TP2
Q3 = Q3-TP1+TQ2
P4 = P4+TQ1-TP2
Q4 = Q4+TP1+TQ2
```

FIG. 6. *Kernel of the relaxation subroutine in* DAP *Fortran.*

Figure 7 shows the DAP Fortran code used to compute the maximum residual. The residual matrices are zeroed, and the absolute values of the components of the residual vector are computed, as above. The matrix MR masks off the meaningless values generated at the bottom and right-hand sides of the array. The R1 and R2 matrices are merged to form a matrix of maximum values. The MAXV function is then used to extract the maximum residual.

```
R1 = 0.0
R2 = 0.0
TP1 = P(+,+)−P
TP2 = P(,+)−P(+,)
TQ1 = Q(+,+)−Q
TQ2 = Q(,+)−Q(+,)
R1(MR) = ABS(ZL*(TP1+TP2)−TQ1+TQ2)
R2(MR) = ABS(TP1−TP2+ZL*(TQ1+TQ2)−Z)
ERR = MAXV(MERGE(R1,R2,R1.GT.R2))
```

FIG. 7. DAP *Fortran code to compute the maximum value of the residuals.*

Apart from the branch instructions in the DO loops and GO TO, there are only three scalar operations in this subroutine. The first of these sets an error flag to zero, the next is used to test the maximum residual against the convergence tolerance. If the iteration does not converge in a specified number of iterations, the last of the scalar operations sets the error flag to unity.

The calculation of the velocity field is the first major piece of the algorithm; the time stepping of the vorticity is the other. The first step in the vorticity calculation is to set the vorticity boundary conditions, as defined in § 2. This is relatively expensive because it involves only vector operations to generate the vectors of boundary values. A typical segment of the code is shown in Fig. 8.

```
U2 = P(63,)
U3 = P(62,)
U2 = U2+U2(+)
U3=0.25*(U3+U3(+))
ZBOT = (U3−U2)/VEC(DY(63))
```

FIG. 8. DAP *Fortran code to calculate the boundary values of the vorticity at a solid wall.*

Here (2.6) is used to calculate the vorticity at a solid boundary at $y=0$. It has been assumed that the heights, Δy, of the first two cells bordering the boundary are equal. This gives

$$\zeta(\text{Boundary}) = (\tfrac{1}{2}u(x, 2\Delta y) - 2u(x, \Delta y))/\Delta y. \tag{4.12}$$

The fact that the velocity is the average of two adjacent box variables has also been used to simplify the coefficients in this piece of code. In this figure U2, U3, and ZBOT are 64 element vectors. ZBOT is the vector of boundary values of the vorticity. Note that U2 and U3 are formed by the reduction operation of extracting a row from a matrix. In contrast, VEC(DY(63)) forms a vector by expanding the scalar DY(63). Similar code segments involving vectors are used to calculate the boundary values of the vorticity on the other three boundaries.

Once the boundary values of the vorticity have been calculated, one can proceed to the solution of the advection diffusion equation for the vorticity, equations (2.9) to (2.15). As was discussed by Gatski, Grosch and Rose [3] and Gatski and Grosch [4], [5], $\zeta^{n+1/2}$ is eliminated between (2.9) and (2.12). This gives an implicit system for ζ^n, which is then solved by an ADI method involving two complete sweeps. In the first sweep one solves for the set $\{\zeta^n_{i+1/2,j}, \phi^n_{i+1/2,j}\}$, $i=0, 1, \cdots$ for all j, using the values of $\zeta^{n-1/2}_{i,j}$. Next the set $\{\zeta^n_{i,j+1/2}, \psi^n_{i,j+1/2}\}$, $j=0, 1, \cdots$ for all i, is solved for. Using these values and $\zeta^{n-1/2}_{i,j}$, we solve for the next, and last, approximation to $\{\zeta^n_{i+1/2,j}, \phi^n_{i+1/2,j}\}$.

In the first pass the set $\{\zeta^n_{i+1/2,j}\}$ is the solution of a tridiagonal system, i.e.,

$$a_{i,j}\zeta^n_{i-1/2,j} + b_{i,j}\zeta^n_{i+1/2,j} + c_{i,j}\zeta^n_{i+3/2,j} = q_{i,j} \tag{4.13}$$

for $i=1, 2\cdots$, and fixed j. Taking $j=1, 2\cdots$ gives a set of N tridiagonal equations, each for N variables. Next we solve for the set $\{\phi_{i+1/2,j}\}$. It can be seen that the coefficients $\{a_{ij}, b_{ij}, c_{ij}\}$, for both passes are functions of the velocity and the parameters $(\Delta t/\Delta x)$, $(\Delta t/\mathrm{Re}\,\Delta x^2)$, and the cell Reynolds numbers. The forcing terms, the $\{q_{ij}\}$, must be recomputed after each pass. For example, when solving for the $\{\zeta^n_{i+1/2,j}\}$, the q's are functions of the $\{\zeta^n_{i,j+1/2}, \psi^n_{i,j+1/2}\}$. Once the vorticity at time level Δt is computed using this ADI scheme, the time advance of the vorticity can be completed by using the fact that the time average equals the space average, see (2.12).

Each pass of the implicit ADI step requires that the coefficients and forcing terms of the tridiagonal systems be computed. This can be done concurrently for all N systems, i.e., all systems in rows or all systems in columns. A portion of the DAP Fortran code to accomplish this is shown in Fig. 9. The average velocity in the cells is computed in the first four statements. The next seven statements calculate various terms needed to compute the a's, b's, and c's. They are combined in the next block of four statements to form α^-, β^-, α^+, and β^+. The next three statements compute the remaining temporaries required to set the forcing terms, the q_{ij} of (4.13), for all of the equations. Note that two of these statements evaluate the advective and diffusive terms for the orthogonal direction. These terms are then combined to form the q_{ij} matrix. But note that two additional vector operations are required to set the boundary conditions. Finally, the α's and β's are combined to give the coefficient matrices for the tridiagonal systems.

The block of code given in Fig. 9 constitutes one pass in one direction and, as such, is about one quarter of the total amount of code to implement the ADI scheme for (2.9)-(2.15). This segment of DAP Fortran, to compute the vorticity at the tops and bottoms of cells at time level n, is not completely optimized. There are several uses of the MATC and MATR functions to expand vectors to matrices. These matrices could have been precomputed and stored, but it was decided to minimize the storage used at the cost of a few expansions and matrix arithmetic operations. It is also possible, by judicious redefining of some of the temporary matrices, to save two or three matrix additions in this code segment. However it was decided to forego this in the interest of clarity of presentation.

There are two matrix functions in the DAP Fortran code shown in Fig. 10. The first of these is QTHETD. The function implements the $q(\theta)$ function as defined in (2.16a, b). It is programmed using the MERGE for evaluating a two branch function as described in § 3. The second matrix function is TRIIED, the tridiagonal solver.

The DAP Fortran routine TRIIED is given in Fig. 9. There are 64 sets of 64 equations for the vorticity values on the tops of the cells. These equations are solved by the cyclic elimination algorithm, which is the cyclic reduction algorithm (see

```
U = P+P(+,)
U = 0.25*(U+U(,+))
V = Q+Q(+,)
V = 0.25*(V+V(,+))

T1 = 0.5/MATC(RLY)
T2 = MATC(RKY)
QT = QTHETD(0.5*RE*V*MATC(DY))
CC = QT-1.0
QQ = QT+1.0
AA = 1.0+V*MATC(RLY)
BB = 1.0-V*MATC(RLY)

ALFAM = T1(-,)*(CC(-,)*AA(-,)+T2(-,))
BETAM = T1(-,)*(CC(-,)*BB(-,)-T2(-,))
ALFAP = T1*(QQ*AA+T2)
BETAP = T1*(QQ*BB-T2)

CC = 0.5/MATR(DX)
AA = ZETA-DT*CC*(U*(ZX(,+)-ZX)-(PHI(,+)-PHI)/RE)
BB = ZETA(-,)-DT*CC(-,)*(U(-,)*(ZX(-,+)-ZX(-,))-(PHI(-,+)-PHI(-,))/RE)

QQ = 2.0*((1.0-QT(-,))*T1(-,)*BB+(1.0+QT)*T1*AA)
QQ(1,) = ZTOP
QQ(64,) = ZBOT

AA = BETAP
AA(1,) = 0.0
AA(64,) = 0.0
BB = ALFAP-BETAM
BB(1,) = 1.0
BB(64,) = 1.0
CC = -ALFAM
CC(1,) = 0.0
CC(64,) = 0.0

ZY = TRIIED(AA,BB,CC,QQ)
```

FIG. 9. *A portion of the* DAP *Fortran code to carry out the* ADI *calculation.*

```
      MATRIX FUNCTION TRIIED(A,B,C,Q)
      REAL A(,), B(,), C(,), Q(,)
      K = 1
      DO 100 L=1,6
      A = A/B
      C = C/B
      Q = Q/B
      Q = Q-A*SHNP(Q,K)-C*SHSP(Q,K)
      B = 1.0-A*SHNP(C,K)-C*SHSP(A,K)
      A = -A*SHNP(A,K)
      C = -C*SHSP(C,K)
100   K = K+K
      TRIIED = Q/B
      RETURN
      END
```

FIG. 10. *Tridiagonal equation systems solver in* DAP *Fortran.*

Hockney and Jesshope [7]) applied to all of the equations. This eliminates the back substitution phase of the algorithm.

This subroutine is reasonably efficient but does contain some hidden defects which are inherent in the algorithm. The shift parameter, K, takes on the successive values 2^k for $k = 0, 1, \cdots, 5$, so that in each pass 2^k rows of data are shifted off the array and, more importantly, 2^k rows of zeros are shifted onto the array. Thus, although all of the processors are active all of the time, some of these processors are not doing useful work because they are multiplying by zero or adding a zero. This must be taken into account in any fair calculation of processing rate.

The final calculation in a time step is the computation of the total energy, vorticity and enstrophy (the squared vorticity), as diagnostic measures. This is a simple, straightforward computation using the SUM function. Details will not be given.

5. Results. The numerical algorithm, the DAP architecture and DAP Fortran, and the adaptation of the algorithm to this architecture were presented in previous sections. Timing and performance results are given in this section.

The host program is a standard Fortran program in which COMMON blocks containing box variables, P and Q, the vorticity, the grid sizes, etc. are defined. The host program also handles the input and output, initialization of variables and the calling of the DAP Entry routine. There is only a small amount of computation in the host program and therefore the amount of time spent in this section is insignificant. This is evident once it is realized that the DAP Fortran section is run for a hundred, perhaps several hundred, time steps in a typical case.

In the DAP Entry routine all of the variables which were passed through COMMON are converted from Fortran to DAP format. This is a fairly costly operation: a matrix conversion takes 2252 μs, and a vector or scalar conversion about 50 μs. In total, 7 matrices, 28 vectors, and 23 scalars are converted on entry to the DAP program. This takes 18.3 ms which is a little over 100 addition times. This is a significant overhead only if it is repeated every time step. Because tens, or hundreds, of time steps are taken for each call to the DAP Entry routine, this overhead, and that of conversion upon leaving, is quite unimportant. The only other initialization task in this entry routine is to generate the one bit logical masks. The total time for this is only about 150 μs; less than one addition time. Again this is negligible.

From this discussion it should be clear that the only substantial costs are those associated with the application of the numerical algorithm for each time step. The total overhead in the host Fortran program and in the data conversions in the DAP Entry routine can be shown to be less than the cost of three iterations in the relaxation routine. In the calling sequence in the Entry routine there are only four tests on flag variables and seven scalar arithmetic operations per time step and the time to do this is less than one fifth of that to carry out one relaxation iteration; a typical time step would require about fifty iterations.

The required sequence of calls to the various DAP Fortran routines in order to advance the Navier-Stokes algorithm one time step is: RELAXD (solves for the velocity field by relaxation), FIXZBD (calculates boundary values of the vorticity), ZCALCD (advances the vorticity in time); and FNORMD (calculates the total energy, vorticity, and enstrophy of the field). In ZCALCD there are calls to the matrix functions discussed previously, QTHETD and TRIIED. The function TRIIED solves tridiagonal equations distributed over columns. There are also calls in ZCALCD to the matrix function TRIJED which solves tridiagonal equations distributed over rows.

TABLE 2
Summary of results.

Subprogram	Execution time (msec)	Proportion of transfer time (%)	Processing rate (MFLOPS)
RELAXD	30.2*	6.9	20.2
FIXZBD	1.4	5.9	0.3
ZCALCD	222.4	3.0	10.1
FNORMD	5.4	2.8	15.1
QTHETD	1.6	2.1	5.0
TRIIED	32.0	27.2	10.1
TRIJED	32.0	27.2	10.1

* Per iteration.

Table 2 contains a list of these subprograms; the execution time for each, that for one iteration in the case of RELAXD; and the processing rate. The processing rate is determined by taking the ratio of the number of effective arithmetic operations to the total execution time for the subprogram. In counting the number of effective arithmetic operations only those operations which truly contribute to the solution are counted. In those cases for which some of the processors are not performing useful work, as in TRIIED as described previously, those processors are not counted. Data transfer operations, vital as they are, were not counted as floating point operations. However, the time required to do these transfers was included in the total execution time.

One can see that there is a wide range in both execution time, from 1 to over 200 msec, and processing rates, 0.3 to 20 mflops, for these subprograms. The amount of time spent on data transfers is quite modest for most of the subprograms; in general 2% to 7% except for TRIIED and TRIJED.

There are two reasons why the data transfer overhead is generally so unimportant. The first is that the basic algorithm does not contain many data transfers and these transfers are only between nearest neighbors. The local nature of the data transfers is due to the fact that the differencing scheme is a compact one. The second reason is that the DAP has very efficient data transfer hardware, yielding rates in the neighborhood of 200×10^6 words per second. In contrast to most of the subroutines, 27% of the execution time is used by data transfers in the tridiagonal solvers. These subroutines are mostly data transfer operations; in fact only 18% of the operations are arithmetic operations, but these transfer operations take up only about one fourth of the time because of the efficient implementation of data movements on the DAP.

The subroutine FIXZBD, used to calculate the boundary values of the vorticity, is very inefficient with a processing rate of only 0.3 mflops. This is because there are only vector operations in this subroutine and the vector processing rate is 1/64 of that for matrix operations. Fortunately, the execution time of FIXZBD is a small fraction, approximately 0.5%, of the total execution time if a time step requires only one iteration and about 0.04% of the total excution time if there are 100 iterations per time step. It is obvious that: FIXZBD uses only vector operations; has a very low efficiency; and has a negligible effect on the overall efficiency of the code. It would seem that this conclusion concerning the setting of boundary conditions is independent of the structure of the algorithm.

Subroutine ZCALCD, within which the vorticity is advanced in time, consists of two principal operational parts: the generation of the coefficients for the tridiagonal

systems and the solution of these tridiagonal systems. Within this subroutine there are a number of vector and scalar operations, but the time required to evaluate them is only about 14% of the total. Generation of the coefficients for the tridiagonal systems takes about 42% of the execution time so that even if the tridiagonal solvers ran in zero time the execution time would only decrease by a factor of two and the processing rate double. The effective processing rate of 10 megaflops for this routine is due to, first the vector and scalar operations required to generate the coefficients of the tridiagonal systems and, second, the large number of data transfers required in the tridiagonal solvers. The average processing rate is approximately half the maximum possible rate. Given the necessity of treating boundaries and solving the tridiagonal systems, there does not seem to be any more efficient way to handle these problems than that adopted here.

6. Concluding remarks. The adaptation of this Navier-Stokes algorithm to the DAP architecture was quite straightforward and fairly successful. This particular algorithm was chosen because it has been found, when implemented on serial and vector computers, to be flexible, accurate and to have certain advantages with respect to the use of nonuniform grids and the implementation of boundary conditions. It is now being used in a number of application studies and a three-dimensional version is being tested. Although the algorithm is not a standard one, velocity-vorticity variables are used with compact difference schemes, its complexity is comparable to that of more standard algorithms. The very nature of the problem, the numerical solution of the Navier-Stokes equations—a set of nonlinear partial differential equations—dictates that any algorithm must have some high degree of complexity.

This class of applications, as mentioned in the introduction, require millions, perhaps tens of millions of words of memory and, in many applications, in excess of 10^{11} floating point operations per case. The "traditional" bottlenecks of computational fluid dynamics are the lack of sufficient fast memory and a relatively low floating point processing rate. The invention of fast algorithms does not eliminate the need for fast hardware. This experiment was intended to explore the implementation of an algorithm for computational fluid dynamics on one member of the class of SIMD architectures.

Overall this adaptation was quite successful. Within the constraint of fitting the computational mesh to the array size, there are, essentially, no bottlenecks in the relaxation portion of the algorithm. With the exception of some of the boundary processors which are masked off because of the boundary conditions, it was possible to keep all of the processors doing useful work by using a four color relaxation scheme. There are no vector operations in the relaxation subroutine and only one scalar operation per iteration; the comparison of the maximum residual to the error tolerance. It might have been assumed that the determination of the maximum residual would be a major bottleneck. Such is not the case, as the maximum residual is found using only a MERGE and a MAXV function. The total execution time is 107 μsecs which is little more than half of an addition time.

The absence of bottlenecks in the adaptation of the relaxation scheme is strikingly illustrated by the processing rate of subroutine RELAXD. The maximum possible arithmetic processing rate of the DAP is 23 megaflops and RELAXD has a processing rate of a little over 20 megaflops, about 88% of the maximum. This good performance is due to the fact that the relaxation algorithm is an excellent fit to the architecture and that the DAP has very efficient software for the reduction functions, in this case the MAXV function.

The other principal subroutine, ZCALCD, embodies the ADI solver for the implicit

formulation of the vorticity transport equation. It is not nearly as efficient as the relaxation routine; with a processing rate of 10 megaflops it is only 43% of the maximum. This routine, and in particular the tridiagonal solver routines at its core, is the nearest thing to a bottleneck in the entire code. The major difficulty is that the cyclic elimination algorithm requires many data transfers and, in the latter stages of the algorithm, many operations on zero data. There does not seem to be any more efficient parallel tridiagonal$_i$ algorithm than the one used here. It is fortunate that although the processing rate of ZCALCD is somewhat low, the execution time of 220 msec is only about that of seven iterations of the relaxation scheme. Thus this part of the algorithm is only a minor bottleneck.

The overall processing rate depends on the number of iterations per time step. It is clear that if the physics of the flow is such that the field is evolving rapidly then, with a modest size time step, there will be a large change in the velocity per time step. This will require a substantial number of iterations in RELAXD per time step. The alternative is, of course, to choose a small enough Δt so that the number of iterations is small. Considering that the relaxation routine in the most efficient of all of the routines, extra iterations for a larger time step would appear to be a reasonable tradeoff.

This is apparent when one considers some reasonable scenarios. If there is one iteration per time step, the execution time per step is 0.26 seconds and the processing rate is 11.3 mflops. If Δt were increased and there were 100 iterations per time step then the execution time would increase to 3.25 seconds per time step, but the processing rate would also increase to 19.5 mflops.

A good deal of the success in adapting this algorithm to the DAP is due to the fact that it is for a two-dimensional problem and the computational array size was chosen to fit the size of the DAP array. The DAP memory is quite small, a maximum of 128 words, each of 32 bits, per processor. Because the instructions are also stored in the DAP store, the maximum is never available for data storage. Generally only about 3000 bit planes can be used as data memory.

Based on the amount of storage available, two-dimensional problems with up to four times the number of grid points could have been run. The DAP software package includes routines for manipulating array sized segments of data from "big" matrices, i.e., those larger than 64 by 64. In principle there is no barrier to applying this algorithm to these larger two-dimensional problems. Certainly the spectral radius would increase as the grid size was reduced; thus the number of iterations per time step would also increase but the processing rate ought to remain nearly the same. The effective processing rate of the tridiagonal solver would also decrease somewhat as the problem size increased because there would be more data transfers and operations on zero data. However, based on the results reported here and the known efficiency of the "big" matrix manipulation software, one could expect the overall performance on moderately larger two-dimensional problems to be quite similar to that reported in this paper.

This is certainly not true for very large two-dimensional and three-dimensional flow problems. In order to handle these problems, the external memory would have to be used. This would require more complex data structures and very many data transfers from the DAP memory to external memory, each requiring a conversion. This data movement overhead would reduce the efficiency and processing rate by a substantial amount.

The DAP is currently embodied in mid-1970s technology. There are three basic ways in which its hardware could be updated. First the cycle time could be reduced, second the array size could be increased (the MPP has 16K processors and the Connection Machine has 64K), and third, the memory per processor could be increased.

Based on the experiences reported here and analysis of, in particular, three-dimensional flow problems, it is suggested that: the first priority should be a much larger, at least an order of magnitude, memory per processor; next more processors; and finally a smaller cycle time.

There is one further observation of interest. The DAP Fortran code reveals blocks of DAP instructions whose structure suggests a way in which this algorithm might be adapted to an MIMD machine. On an MIMD machine, there would appear to be a basic requirement for a "good" program: that there be reasonably large blocks of code which do not require synchronization or have likely memory conflicts. The size of the code blocks is to be interpreted to mean the product of, say, the numbers of arithmetic operations per grid point and the number of grid points in the field.

Consider, first, the relaxation routine. This can be partitioned in at least three ways; geometrically, by color, and in combination. One could split the array of cells into subarrays and relax these without synchronization. This would be a form of chaotic block relaxation. This would probably converge under some conditions, and deserves some study. Color partitioning among four processors would be a simple extension of the algorithm which was implemented on the DAP, but would require synchronization. A combined scheme might be quite effective.

If we consider the DAP Fortran code for the ADI scheme, Figs. 9 and 10, we can see that there are blocks of code which contain independent instructions. For example in Fig. 9, the calculation of U and V are independent. If the next block of code is rearranged so that QT is computed first, then the other six statements are all independent of each other. The next block of four statements are also independent, and so on. Finally the tridiagonal systems are all independent.

One can thus see that there are, on the average, blocks of 4 or 5 independent statements in the code. Each of these performs the same calculation in all processors. These calculations can thus be split up among the independent processors of an MIMD computer. Without considering the details of the MIMD machine it is difficult to say more, but it appears that this algorithm might be a good candidate for MIMD architectures. However, there clearly could be major problems because of the requirements of large data transfers and synchronization and convergence testing for the relaxation routine. These potential problems require analysis and experimentation on one or more MIMD architectures.

REFERENCES

[1] DAP: *Fortran Language*, Technical Publication 6918, International Computers Limited, 1981.

[2] J. J. DONGARRA, *Performance of various computers using standard linear equations software in Fortran environment*, Argonne National Laboratory Technical Memo, 23, 1984.

[3] T. B. GATSKI, C. E. GROSCH AND M. E. ROSE, *A numerical study of the two-dimensional Navier-Stokes equations in vorticity-velocity variables*, J. Comput. Phys., 48 (1982), pp. 1-22.

[4] T. B. GATSKI AND C. E. GROSCH, *A numerical study of the two- and three-dimensional unsteady Navier-Stokes equations in velocity-vorticity variables using compact schemes*, Proc. Ninth International Conference on Numerical Methods in Fluid Dynamics, Springer-Verlag, New York-Heidelberg-Berlin-Tokyo, 1985, pp. 235-239.

[5] ———, *Embedded cavity drag in steady laminar flow*, AIAA J., 23 (July 1985), pp. 1028-1037.

[6] C. E. GROSCH, *Performance analysis of Poisson solvers on array computers*, in Infotech State of the Art Report: Supercomputers, C. Jesshope and R. W. Hockney, eds., 2, 1979, Infotech International, pp. 147-181.

[7] R. W. HOCKNEY AND C. JESSHOPE, *Parallel Computers: Architecture, Programming and Algorithms*, Adam Hilger, Ltd., Bristol, 1981.

[8] R. W. HOCKNEY, $(r_\infty, h_{1/2}, s_{1/2})$ *measurements on the* 2-CPU *Cray* X-MP, Univ. Reading, Dept. Comput. Sci., Report RCS 185, 1985.

[9] S. KACZMARZ, *Angenaherte auflosung von systemen linearer gleichungen*, Bull. Acad. Polon, Sci. Lett., A (1937), pp. 355-357.

[10] R. M. MCINVILLE, T. B. GATSKI AND H. A. HASSAN, *Analysis of large vortical structures in shear layers*, AIAA J., 23 (1985), pp. 1165-1171.

[11] J. M. ORTEGA AND R. G. VOIGT, *Solution of partial differential equations on vector and parallel computers*, SIAM Rev., 27 (1985), pp. 149-240.

[12] R. B. PHILIPS AND M. E. ROSE, *Compact finite difference schemes for mixed initial-boundary value problems*, SIAM J. Numer. Anal., 19 (1982), pp. 698-719.

[13] M. E. ROSE, *A "unified" numerical treatment of the wave equation and the Cauchy-Riemann equations*, SIAM J. Numer. Anal., 18 (1981), pp. 372-376.

[14] K. TANABE, *Projection method for solving a singular system of linear equations and its applications*, Numer. Math., 17 (1971), pp. 203-214.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 1, January 1987

© 1987 Society for Industrial and Applied Mathematics

REDUCTION OF THE EFFECTS OF THE COMMUNICATION DELAYS IN SCIENTIFIC ALGORITHMS ON MESSAGE PASSING MIMD ARCHITECTURES*

JOEL H. SALTZ†, VIJAY K. NAIK† AND DAVID M. NICOL†

Abstract. The efficient implementation of algorithms on multiprocessor machines requires that the effects of communication delays be minimized. The effects of these delays on the performance of a model problem on a hypercube multiprocessor architecture is investigated, and methods are developed for increasing algorithm efficiency. The model problem under investigation is the solution by red-black Successive Over Relaxation of the heat equation; most of the techniques to be described here also apply equally well to the solution of elliptic partial differential equations by red-black or multicolor SOR methods.

This paper identifies methods for reducing communication traffic and overhead on a multiprocessor and reports the results of testing these methods on the Intel iPSC Hypercube. We examine methods for partitioning a problem's domain across processors, for reducing communication traffic during a global convergence check, for reducing the number of global convergence checks employed during an iteration, and for concurrently iterating on multiple time-steps in a time dependent problem. Our empirical results show that use of these methods can markedly reduce a numerical problem's execution time.

Key words. parallel processing, scientific computation, communication delay, hypercube architecture, convergence checking

AMS(MOS) subject classifications. 34A50, 35-04, 68B20

1. Introduction.

The efficient implementation of algorithms on multiprocessor machines requires that the effects of communication delays be minimized. Reduction of communication delay effects in message passing machines may be brought about by restructuring algorithms. Ways in which the effect of communication delays can be reduced by such restructurings include: (1) reducing the quantity of information that must be communicated, (2) reducing the frequency with which messages must be sent, (3) overlapping communication with computation. The above goals may not in practice be mutually compatible. The relative importance of the three aspects of communication delay reduction will depend on the architecture under consideration [9], [10], [13].

The effects of these delays on the performance of a model problem on a hypercube multiprocessor architecture are investigated, and methods are developed for increasing algorithm efficiency. A hypercube multiprocessor [12], [7] is a collection of processors or nodes connected by a communication network with a hypercube topology. A hypercube has 2^m identical nodes where m represents the dimension of the hypercube. Each node in a hypercube is connected to m other neighbors. Nodes are assigned addresses from 0 to $2^m - 1$. Two nodes of a hypercube are connected when the binary expansion of the nodes' addresses differs in one bit position.

The model problem under investigation is the solution by red-black Successive Over Relaxation [14] of the heat equation; most of the techniques to be described here also apply equally well to the solution of elliptic partial differential equations by red-black or multicolor SOR methods. The model problem is solved on an N-dimensional hypercube by decomposing the domain into 2^N regions and assigning

* Received by the editors December 9, 1985; accepted for publication (in revised form) April 23, 1986. This research was supported by the National Aeronautics and Space Administration under NASA contract NAS1-17070 and NAS1-18107 while the authors were in residence at ICASE, NASA, Langley Research Center, Hampton, Virginia 23665.

† Institute for Computer Applications in Science and Engineering, NASA, Langley Research Center, Hampton, Virginia 23665.

one region to each processor. The regions are chosen either as strips or as rectangles and are mapped onto the architecture using a Gray code [3], [8]. Due to the grey code mapping, processors assigned adjacent regions of the domain are directly connected.

The two sources of communication delays in a simple iterative method such as SOR are the need to exchange information between the boundaries of the subdomain assigned to each processor and the need to check convergence. We examine a variety of factors which help determine the interprocessor communcation costs. In this paper methods of reducing both of these sources of communication delays are proposed.

The time required to transmit a packet of information from one processor to another to which it is directly linked may be approximately expressed as

$$\frac{b}{\alpha}+\beta$$

where b is the number of bytes contained in the message, α is bandwidth of the communication channel, and β is the overhead for sending a message. When $1/\alpha$ is considerably smaller than β, the time required to send a small message is only weakly dependent on the message size. In this case, there may be significant performance advantages in arranging an algorithm so that information that must be transmitted is sent in large quantities.

In the Intel hypercube used in the experiments described in this paper, $\beta \gg 1/\alpha$. In architectures in which it is possible to mask communication with computations, algorithms can be designed to allow computations to proceed despite high communication latencies [10]. In the Intel hypercube, this type of reorganization is not helpful, as communication may not be overlapped with computation to any appreciable extent. Given a multiprocessor with these characteristics, reducing the number of messages that must be sent by processors is consequently the main goal of the work to be described. In §§ 3, 4 and 5 methods are described that may be used to reduce the number of messages that must be sent by each processor and to consequently reduce the effect of the communication overhead β. Section 3 explores the consequences on the hypercube performance of the simple observation that the number of messages that must be sent by a processor when a domain is partitioned into strips is at most two, while a processor may have to send four messages when a domain is partitioned into square regions. Allowing iterations to sweep over more than one timestep is a method used in § 5 to decrease the number of messages that must be sent.

In a hypercube, it is possible to perform communications that combine results from all processors and to disseminate the results thus obtained in a time that grows logarithmically in the number of processors involved. When communication overheads are large, convergence testing may be quite costly despite this logarithmic growth. In § 4.1 two logarithmic methods for combining the results obtained from local convergence checks are compared. In both methods, for all but very small hypercubes, the communication delays resulting from convergence checking are comparable with or greater in size than the delays arising from the communication of the boundary variable values.

In both of the above schemes, communications for global convergence checking occur after each iteration. Two methods are proposed and tested for reducing the frequency with which communication is required for global convergence checking. The first method, discussed in § 4.2, checks for global convergence only when certain necessary conditions are fulfilled. One necessary condition is that all subdomains have detected convergence at some point in their computations. Other necessary conditions result from the fact that global convergence requires all processors to detect local

convergence at a given iteration. In § 4.3 a method is proposed that utilizes a logarithmic method for checking for convergence but employs a statistical methodology to schedule convergence checks only at critical iterations.

In the following sections experimental results will be presented that pertain to the solution of a model problem. The partial differential equation being solved is the heat equation on the unit square with Dirichlet boundary conditions and with the first two modes used as the initial condition. The heat equation is solved using optimally over-relaxed red-black SOR on 64 by 64, 128 by 128, 256 by 256 point meshes with timesteps of 0.004, 0.002 and 0.001, respectively. All experimental results were obtained using a 64 processor Intel iPSC hypercube.

2. Effect of problem size on performance. As has been widely reported [2], [6], in order to obtain better performance from the multiprocessors, one should have a balance between computation and communication costs. If the communication costs are too high compared with the costs of computations, then the performance is bound to deteriorate. The obvious way to improve the performance is to increase the size of the subdomain assigned to each processor, which increases computation time without increasing the communication costs proportionately. In Fig. 1, we depict the performance of the system in terms of efficiency, as the domain size is varied from 64 by 64 through 256 by 256 grid sizes. Here the *efficiency* of an N-processor system is defined as the ratio of the time taken to solve the problem on one processor to the time taken to solve the problem on N processors multiplied by N. As expected, the efficiency drops as the number of processors is increased, but the rate at which it drops becomes more gradual as the grid size is increased. In all the experiments performed here the domain was subdivided into strips, and convergence checking was performed after

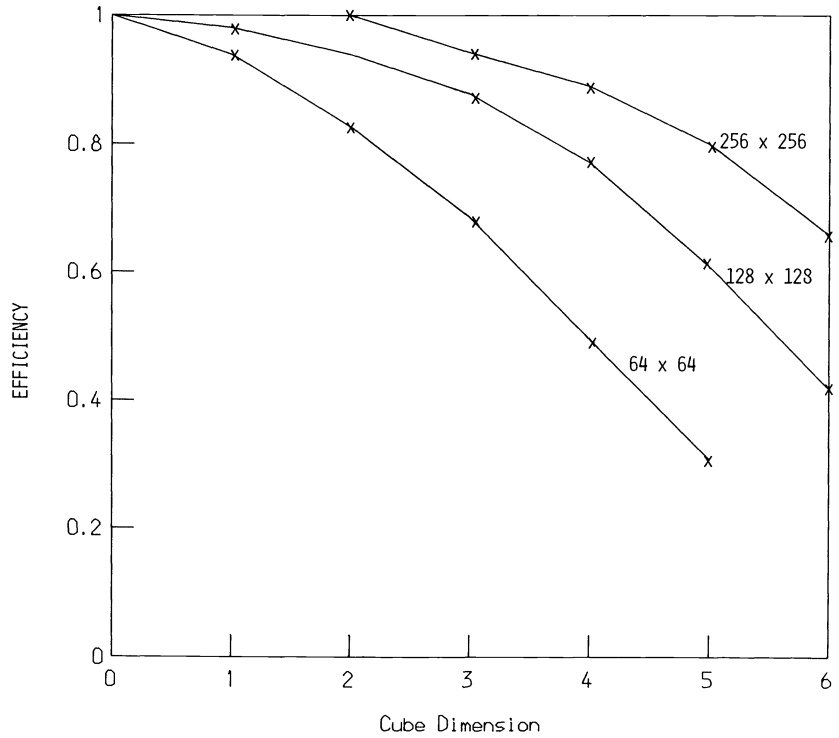


FIG. 1. *Effect of domain size on parallel efficiency.*

each iteration. The model problem was solved for five timesteps, and the efficiencies were computed by measuring the elapsed time to find the solutions of the first five timesteps.

3. The effect of domain partitioning on performance. The cost of communicating information from one processor to another in a hypercube multiprocessor is a function of the amount of data that must be sent, the number of packets of data into which the data is placed, and the logical distance of the processors from one another in the hypercube. The domain of a PDE may be decomposed into regions with a variety of shapes, with certain shapes provably optimal with respect to minimizing the number of variable values that must be communicated across boundaries.[1] The regions may then be mapped onto a hypercube in a way that attempts to minimize the number of intermediate nodes that messages must traverse in going from the boundary of one region to another [8].

We considered domain decompositions consisting of strips and rectangles; such shapes are easier to program and can be mapped onto a hypercube so all processors that need to send messages to one another are directly linked. It is easily demonstrated that in a domain divided into rectangles, less information must be transmitted across boundaries during each iteration than would be the case with a domain divided into strips. On the other hand, in a domain divided into rectangles, regions may have four neighbors, while when the domain is divided into strips, each region may have no more than two neighbors.

We examine the trade-off in costs between the division of a domain into rectangles and the division of a domain into strips in C color SOR [1]. Assume a 2^n by 2^n point domain and 2^m processors. The domain may be divided into 2^m $2^{n-\lfloor m/2 \rfloor}$ by $2^{n-\lceil m/2 \rceil}$ rectangles, or alternatively into 2^n by 2^{n-m} strips. In C color SOR the values of points are adjusted one color at a time. Each time a color is adjusted, all rectangles in the interior of the domain must send four packets. If the number of colors used in the SOR sweeps does not evenly divide the number of points on a side of a rectangle or strip, the number of values to be communicated may differ by one between sweeps over different colors.

Rectangles in the interior of a domain, i.e., those with four neighbors during the course of each iteration, must send $2C$ packets of average size $2^{n-\lfloor m/2 \rfloor}/C$ and must send $2C$ packets of average size $2^{n-\lceil m/2 \rceil}/C$. Strips in the interior of a domain, i.e. those with two neighbors, must send $2C$ packets of average size $2^n/C$.

The comparison of costs between strips and rectangles depends on the overhead for sending each message, on the per-byte cost of transmitting information, on the number of processors, and on the size of the domain. Figure 2 depicts a comparison between local communication costs when the model problem is solved with domains of varying size using 64 processors. For domains of size 256 by 256 or smaller, the use of strips led to smaller communication delays than the use of rectangles, while for a domain with 512 by 512 mesh points the use of rectangles led to the smaller delays. Figure 3 depicts the local communication costs for rectangles compared to the local communication cost for strips for varying numbers of processors in a 64 by 64 point domain. Note that the communication cost for rectangles exceeds that for strips when at least eight processors are utilized. When fewer than eight processors are used, the number of packets that must be sent by each processor in a domain divided into rectangles is equal to the number of packets that must be sent in a strip divided domain.

[1] D. Reed, M. Patrick and L. Adams, to be submitted to IEEE Transactions on Computers.

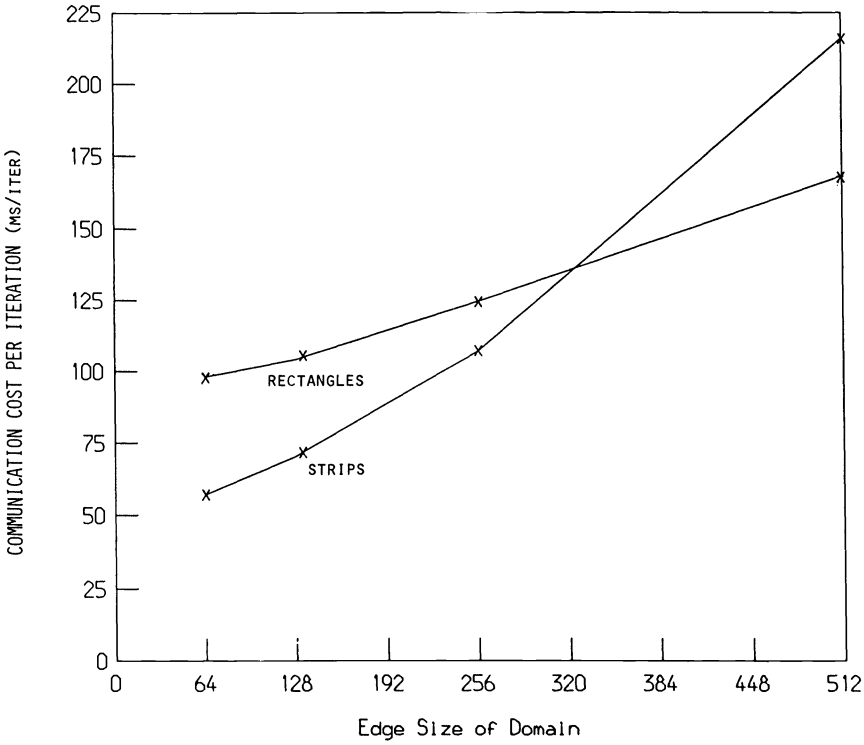


FIG. 2. *Effect of subdomain shape on communication cost per iteration.*

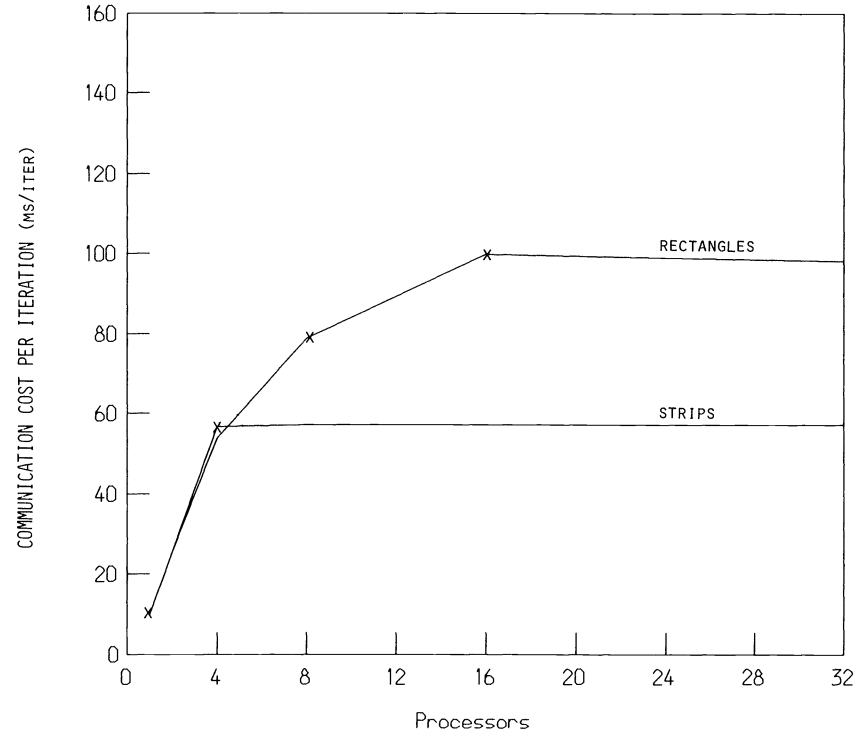


FIG. 3. *Effect of subdomain shape on boundary variable communication cost.*

Moreover, in this case, the amount of information that must be sent by each processor is smaller or equal when the domain is partitioned into rectangles instead of strips. The above local communication costs were measured in the following way. The model problem was solved for a given domain size for 50 iterations over 3 timesteps, and then for the same domain size the program was run without sending any messages. The difference in the execution times between the program runs that sent messages and those that did not was used to give an estimate of time spent in communicating boundary information. Note that neither of the program runs performed any communication for convergence checking.

4. Convergence checking schemes. In this section it will be shown that the communication required in testing for global convergence at the end of an iteration may be quite costly. Several methods for reducing this delay are then experimentally examined.

Our experiments presumed that convergences had been achieved when

$$\|X_i - X_{i-1}\|_\infty \leqq \varepsilon$$

where X_i is the vector valued solution approximation after the ith iteration, and ε is our tolerance. The $\|\cdot\|_\infty$ norm above yields the maximal absolute difference between components of X_i and X_{i-1}. Convergence checking in a multiprocessor involves two distinct costs. The first is simply the time required to compute the component differences. The second cost is the time required for the processors to communicate and combine their respective local convergence results to determine whether global convergence is achieved at the end of an iteration. This latter cost is a function of the multiprocessor communication delay and the scheme used to combine the component differences. We will first look at ways to combine differences during a global convergence check on the hypercube. We will then discuss two different ways of reducing the *number* of convergence checks used during an iterative solution.

4.1. Combination methods. We compared two different ways of combining component differences during a convergence check. Both of these methods first require each processor to find the maximal component difference over its own piece of the domain. Each processor then sets its *convergence flag* equal to 1 if this maximal difference over the processor's domain is less than ε; the flag is otherwise 0. Clearly convergence is achieved if and only if every processor's convergence flag is 1. The two methods differ in how they cause processors to exchange and combine these flags. Both schemes have time complexity $O(m)$, m being the dimension of the hypercube.

The first scheme, to be called the tree method, requires $2m$ stages. In m stages, a logical AND is taken of all convergence flags and the resulting flag ends up in node 0. If the value of this logical AND is 1, global convergence has occurred; otherwise global convergence has still not been detected. During each state k, $0 \leqq k < m$ of communication, each of $2^m/(k+2)$ processors sends to another processor a flag that indicates its current knowledge of whether global convergence has occurred. Let $\langle d_{m-1}, \cdots, d_0\rangle$ be the binary expansion of the address of a node, and let $\bar{d}_k$ represent the complement of binary digit d_k. Let $\text{Flag}^k_{\langle d_{m-1}, d_k, \cdots, d_0\rangle}$ represent the current knowledge of global convergence in the node with address $\langle d_{m-1}, d_k, \cdots, d_0\rangle$ at the beginning of communication stage k. In stage k, processor $\langle d_{m-1}, d_k, \cdots, d_0\rangle$ will send $\text{Flag}^k_{\langle d_{m-1}, d_k, \cdots, d_0\rangle}$ to processor $\langle d_{m-1}, \cdots, \bar{d}_k, \cdots, d_0\rangle$ if $d_0 = \cdots = d_{k-1} = 0$ and $d_k = 1$. Upon the receipt of the flag, processor $\langle d_{m-1}, \cdots, \bar{d}_k, \cdots, d_0\rangle$ will set $\text{Flag}^{k+1}_{\langle d_{m-1}, \bar{d}_k, \cdots, d_0\rangle} = \text{Flag}^k_{\langle d_{m-1}, \cdots, \bar{d}_k, \cdots, d_0\rangle}$ AND $\text{Flag}^k_{\langle d_{m-1}, d_k, \cdots, d_0\rangle}$. This process terminates at Node 0. Node 0 at this point distributes the information on whether global convergence

has occurred. Distributing the convergence results also require m stages of communication.

Another method of checking for convergence can be accomplished in m stages of communication. This procedure is similar in principle to the cascade method for computing sums [5] and will be called the cascade method for checking convergence. In this method, during each stage of communication each processor P sends a flag Flag_P^k indicating its current knowledge of whether global convergence has occurred to another processor. At the end of the m stages, the resulting flag obtained in all processors is the logical AND of all convergence flags.

The cascade process functions as follows: Let $\langle d_{m-1}, \cdots, d_0\rangle$ be the binary expansion of the address of a node. In stage k, $0 \leqq k < m$, processor $\langle d_{m-1}, d_k, \cdots, d_0\rangle$ will send $\text{Flag}^k_{\langle d_{m-1}, d_k, \cdots, d_0\rangle}$ to processor $\langle d_{m-1}, \cdots, \bar{d}_k, \cdots, d_0\rangle$. Upon receipt, processor $\langle d_{m-1}, \cdots, \bar{d}_k, \cdots, d_0\rangle$ will set $\text{Flag}^{k+1}_{\langle d_{m-1}, \bar{d}_k, \cdots, d_0\rangle}$ equal to the logical AND of flags $\text{Flag}^k_{\langle d_{m-1}, \cdots, \bar{d}_k, \cdots, d_0\rangle}$ and $\text{Flag}^k_{\langle d_{m-1}, d_k, \cdots, d_0\rangle}$. This process terminates after m stages, and at this point the flag $\text{Flag}^m_{\langle d_{m-1}, d_k, \cdots, d_0\rangle}$ in each processor is the logical AND of all convergence flags. The flow of data corresponding to the two convergence checking processes is depicted in Fig. 4. This figure illustrates that the tree method has a single process detect and then report global convergence; the cascade method requires all processors to calculate the global convergence state.

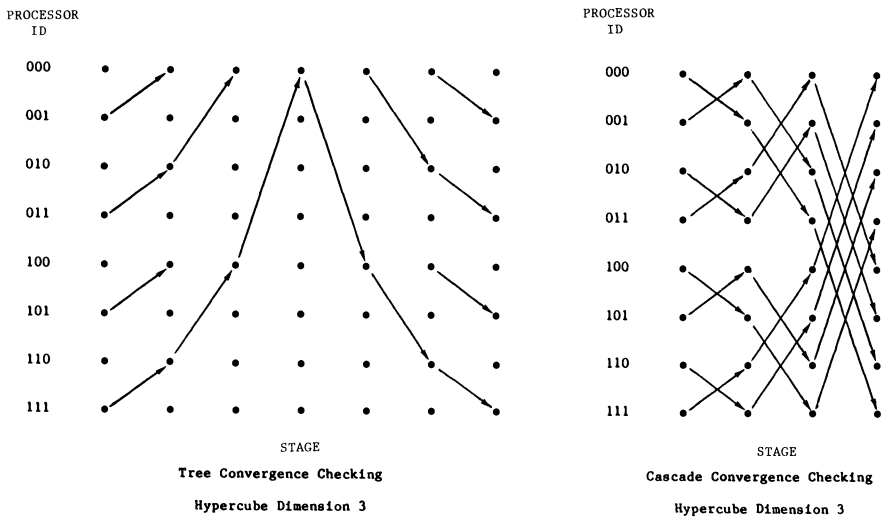


FIG. 4

The cost per iteration of the tree and cascade convergence checking methods is depicted in Fig. 5, along with the cost per iteration of local communication between strips and the per-iteration computation cost for a 64 by 64 mesh point model problem. The cost of convergence checking is estimated by running the model problem for three timesteps, 50 iterations per timestep both with and without the convergence checking methods and comparing the run times. When the convergence checking methods were utilized, a minor modification in the program caused the convergence results thus obtained to be ignored. The additional time required for convergence checking information could hence be ascertained by comparing the timings of programs that otherwise performed identical computations.

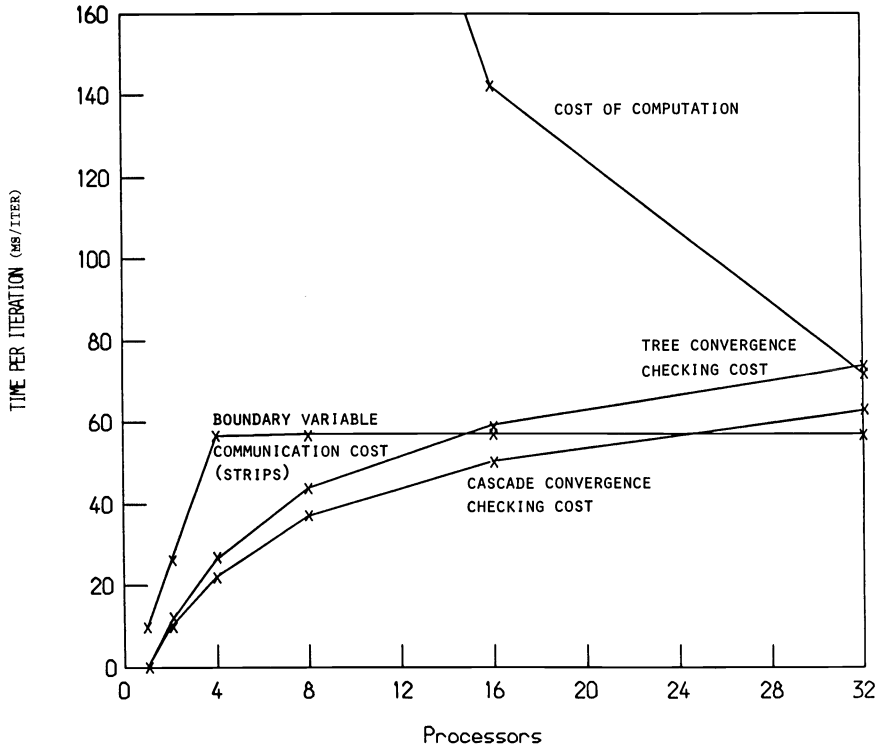


FIG. 5. *Comparison of boundary variable communication cost, convergence checking cost and computation cost.*

The cascade convergence checking method requires each processor to send and to receive four messages per iteration when a cube with 16 processors is used. During each iteration, nodes assigned strips that are on the interior of the domain must send and receive four messages containing boundary variables values, two for the sweep over red points and two for the sweep over black points. When a cube with 16 processors is utilized, the communication cost of sending boundary variable values and of sending convergence flag information is indeed comparable. The cost of sending the boundary variable values is slightly greater, presumably due to larger amounts of data per message.

The cost of the tree method of convergence checking is only slightly greater than the cost of the cascade method despite the fact that it requires twice as many stages. In the tree method, during each stage of the computation a processor is called upon either to send *or* receive a message, while in the cascade method each processor must both send and receive a message during each stage. Further experimentation has indicated that, in a number of contexts, there appears to be a substantial time penalty associated with requiring a processor to both send and receive a message during a given stage of communication.

4.2. Asynchronous convergence checking. This section discusses one means of reducing the number of convergence checks required during an iterative solution. This scheme is called *asynchronous convergence checking*, or the ACC scheme. The ACC's basic idea is to check for global convergence only when certain necessary conditions are fulfilled. One necessary condition is that all subdomains have detected convergence at some point in their computations. Other necessary conditions, to be expanded upon

later, result from the fact that global convergence requires that all processors detect local convergence at a given iteration.

Discussion of the ACC is facilitated by a few definitions. At the end of an iteration, a processor's subdomain is in one of two states: *nonconverged*, or *presumptively converged*. This nomenclature emphasizes that convergence over a processor's subdomain does not guarantee global convergence at that iteration, nor does it prohibit a subdomain from oscillating between nonconvergence and presumptive convergence. Global convergence is achieved if and only if all the subdomains achieve presumptive convergence at the end of the same number of iterations. If the computations were to continue, all subdomains would be expected to remain in this state indefinitely. A *convergence sequence* for a subdomain is a maximal sequence of iterations during which the subdomain is presumptively converged; the first iteration of the sequence is the *convergence sequence header*. Thus a convergence sequence header identifies an iteration where a subdomain passes from nonconvergence to presumptive convergence (startup anomalies are accounted for by *defining* each subdomain to be nonconverged before the first iteration). No further convergence headers are produced by that subdomain until it passes out of and then back into presumptive convergence. A subdomain's oscillation between nonconvergence and presumptive convergence gives rise to a series of convergence sequences, each with a distinct header. It is important to note that if global convergence is achieved at iteration j, then j is a convergence sequence header for at least one subdomain. The ACC method looks for global convergence only at certain iterations which serve as convergence sequence headers of one or more subdomains, but not at all the convergence sequence headers found in the system. The global convergence test is made centrally, in a processor known as the *c-host*. The high cost of communication between Intel hypercube nodes and their *system host* led us to designate a hypercube node as c-host for global convergence checks.

Under the ACC method, the c-host makes an informed guess at when the global convergence might have been achieved; after communicating with the other processors, the c-host either confirms or rejects the guess. A rejection is followed by another guess. This process is continued until the confirmation takes place. Implementation of ACC requires each processor to always maintain its subdomain's current convergence state, and the convergence sequence header if the subdomain is presumptively converged. We now separately describe the three component parts of the method, the *initial guess*, the *processor guess response*, and the *guess confirmation/generation*.

Initial guess. As soon as a subdomain is presumptively converged for the first time, its processor reports the corresponding convergence sequence header to the c-host. The processor then continues with the computation without waiting for convergence information from other processors. The c-host makes the first guess after receiving such a message from each processor; let $\{j_1, \cdots, j_k\}$ be the received header values. The c-host optimistically guesses that global convergence is achieved as early as is possible, at iteration $j_{\max} = \max\{j_1, \cdots, j_k\}$. The c-host then tells each processor that $j_{\max}$ is a potential point of global convergence.

Processor guess response. Suppose a processor receives a guess (not necessarily the first guess) j_G from the c-host. At iteration j_p, $j_p \geqq j_G$, the processor sends back the *current* convergence sequence header if j_p is part of a convergence sequence (an outdated convergence sequence header is thus effectively overwritten in the processor's memory). A message is sent immediately after such a j_p is found.

Guess confirmation/generation. As described above, each processor responds to a guess j_G by returning a convergence sequence header to the c-host. The c-host either confirms or rejects the guess as soon as every processors' response is received. Letting

$j_{\max}$ be the maximal value among these responses, the guess j_G is confirmed if $j_G = j_{\max}$; in this case global convergence is achieved at iteration j_G, and the c-host instructs all processors to stop iterating. If $j_G < j_{\max}$, then j_G is rejected, and the value $j_{\max}$ is sent to all processors as the next guess.

We can show that the ACC identifies an iteration achieving global convergence; furthermore, if the solution cannot drift out of global convergence (even temporarily), then the ACC is guaranteed to identify the first iteration achieving global convergence. The first claim is proven by contradiction. Suppose that ACC confirms iteration j_A as a point of global convergence, but that some processor P has a nonconverged subdomain at iteration j_A. By the confirmation process described above, j_A is the maximum header value in response to a guess j_G, and $j_A = j_G$. Consider P's response to this guess, at (say) iteration j_p. Clearly $j_p \geqq j_G$, since P does not respond until it has iterated at least to j_G and is presumptively converged. Furthermore, $j_p > j_G$ since P's subdomain is nonconverged at iteration $j_A = j_G$. But j_A is the maximum among all responses to j_G, so $j_p \leqq j_G$, a contradiction. We also show that ACC finds the first globally converged iteration if the solution never diverges from global convergence. Suppose that global convergence is achieved first at iteration j_f, and that the ACC first detects global convergence at j_A. For the sake of contradiction, suppose that $j_A \neq j_f$. Since ACC does detect global convergence, we must have $j_A \geqq j_f$; for the sake of contradiction, suppose that $j_A > j_f$. j_A is a convergence sequence header from some processor P; thus P's subdomain was *not* converged at iteration $j_A - 1 \geqq j_f$. This is a contradiction, since we have assumed that global convergence at j_f implies convergence in P for all iterations $j \geqq j_f$. Thus $j_A = j_f$, showing that ACC finds the first globally converged iteration.

The ACC scheme is asynchronous in that the processors never synchronize waiting for convergence information and in that the processors do not necessarily send the messages to the c-host at the end of the same iteration. Unlike asynchronous chaotic methods, we still presume that the processors synchronize with their neighbors at each iteration. The communication costs required by the ACC method are quite small. The ACC requires substantially fewer messages than standard convergence checking; furthermore, on appropriate architectures the communication may be overlapped with computation. From this aspect, the ACC is clearly superior to standard convergence checking. However, the ACC does incur two additional computational costs. The first (and minimal) cost is execution of the ACC logic; the second cost occurs because each processor continues to iterate until it is told to stop. Thus each processor "overshoots," doing slightly more computation even after global convergence is achieved. In practice, neither of these costs proved to be significant; the improvement over standard convergence checking is substantial, as discussed in § 5.

Further improvements in the ACC scheme are possible. With a cube of large dimension, we can distribute the c-host function by forming clusters of smaller cubes; ACC is then applied locally at a cluster. A cluster reaching "global" convergence is logically equivalent to presumptive convergence in a subdomain; a central c-host would determine global convergence using ACC at a global level. Another improvement is achieved by checking a subdomain's convergence only at selected iterations. The computational cost of checking convergence is saved, at the risk of doing more iterations than are required. In a similar vein, the scheme discussed in the next section formally schedules convergence checks and balances the benefits of skipping checks with its risks.

4.3. Maximized expected useful work. A second method of reducing convergence checking costs *schedules* convergence tests at critical iterations. The scheduling methodology is that of decision making under uncertainty. It is intended for use when

only inexact information about convergence behavior can be presumed. This gives it a particular advantage for use by a multiprocessor operating system on general numerical problems. Thus, the "uncertainty" considered within the scheduling mechanism is due to the generality of problem treated by the mechanism, and to the lack of knowledge about the solution's behavior.

Upon the completion of a scheduled test, the next convergence test is scheduled on the basis of the cost of testing convergence and the costs of scheduling the next test "too far" in the future after convergence has been achieved. The iteration chosen is the one maximizing the "expected useful work" per unit time and is thus dubbed the MEW method.

The MEW method entails a certain amount of mathematical formalism. First, we define the ith iteration error estimate E_i:

$$E_i = \|X_i - X_{i-1}\|_\infty.$$

We model the convergence behavior of an iterative method by assuming that

$$E_n \leqq E_1 \cdot e^{-\lambda \cdot (n-1)}. \tag{1}$$

We say that the solution has converged at iteration n if $E_n \leqq \varepsilon$ for our tolerance ε. The key issue in this formulation is the estimation of λ. If the exact value of λ were known, then the first converged iteration is found by solving for n in the equation $\varepsilon = E_1 \cdot e^{-\lambda \cdot n}$. Since the exact value of λ is not known, it must be estimated.

Our treatment of λ is Bayesian (the reader unfamiliar with Bayesian estimation can consult [11] or any standard statistical text). We view a convergence test as a statistical observation of λ. The *observation* of λ created by calculating E_j is derived from relation (1):

$$\hat{\lambda} = \frac{1}{(j-1)} \cdot (\ln(E_j) - \ln(E_1)). \tag{2}$$

Our uncertainity about future convergence convergence behavior is modeled by assuming that $\hat{\lambda}$ is a normal random variable $N(\lambda, \sigma_s^2)$. λ here is the true unknown convergence rate, and σ_s^2 is a sampling variance which we will also estimate. We again emphasize that the normality assumption models our uncertainty about the problem itself, as well as the problem and solution method behavior. This assumption describes what we are willing to believe about the true value of λ, in the absence of any further information.

We furthermore suppose that we have some prior knowledge of what λ might be. This prior knowledge is encoded with a normal (prior) probability distribution $N(\lambda_{\text{pr}}, \sigma_{\text{pr}}^2)$ describing the likelihood of λ taking any particular value. Given the parameters $\lambda_{\text{pr}}, \sigma_{\text{pr}}^2$ and an observation $\hat{\lambda}$, Bayes' theorem says that the posterior distribution of λ is normal $N(\lambda_{\text{pt}}, \sigma_{\text{pt}}^2)$ where

$$\lambda_{\text{pt}} = \left(\frac{\sigma_{\text{pr}}^2}{\sigma_{\text{pr}}^2 + \sigma_s^2}\right) \cdot \hat{\lambda} + \left(\frac{\sigma_s^2}{\sigma_{\text{pr}}^2 + \sigma_s^2}\right) \cdot \lambda_{\text{pr}}$$

and

$$\sigma_{\text{pt}}^2 = \sigma_{\text{pr}}^2 \cdot \frac{\sigma_s^2}{\sigma_{\text{pr}}^2 + \sigma_s^2}.$$

The posterior distribution incorporates our prior knowledge of λ with the additional information afforded by $\hat{\lambda}$. The scheduling of our next convergence test (presuming $E_j > \varepsilon$) depends in part on this distribution.

We next examine the mechanics of our convergence test scheduling, temporarily deferring discussion of prior determination and the estimation of σ_s^2. Suppose we have a prior distribution of λ, and we make a convergence test at iteration j. We then calculate λ_{pt} and σ_{pt}^2 as described. From iteration j, we view the probable future behavior of convergence at iteration $j+d$ *as though* we are at the first iteration. That is, we presume that the convergence model for iterations $j+d$, $d>0$, is

$$E_{j+d} \leqq E_j \cdot e^{-\lambda \cdot d}.$$

It can be shown that sensitivity to measurement errors in λ is reduced by using this modification. The probability of not observing convergence at iteration $j+d$ is easily seen to be identical to the probability that λ is less than the threshold $T_j(d)$, where

$$T_j(d) = \frac{1}{d} \cdot \ln\left(\frac{E_j}{\varepsilon}\right).$$

Appealing to the normal structure of the posterior distribution, we thus have

$$\text{Prob}\{\lambda < T_j(d)\} = \Phi\left(\frac{T_j(d) - \lambda_{pt}}{\sigma_{pt}}\right)$$

where Φ is the standard normal cumulative distribution function. We use the probability above as an estimate of the probability that we will not have converged by iteration $j+d$.

We can now detail the scheduling decision. Let I be the delay cost of performing a convergence test, and let D be the time required to perform one iteration without a convergence test. If we schedule the next convergence test at iteration $j+d$, the total time required to do d iterations and perform the test is $d \cdot D + I$. Then the average *required* number of iterations per unit time achieved by this decision is

$$\sum_{i=1}^{d} \Phi\left(\frac{T_j(i) - \lambda_{pt}}{\sigma_{pt}}\right) \Big/ d \cdot D + I. \tag{3}$$

We find the $d = d_{max}$ which maximizes the expression above and schedule the next convergence test at iteration $j + d_{max}$. Maximization of expression (3) balances the cost of testing convergence with the cost and uncertainty of doing more iterations than are required. As a function of d, expression (3) has at most one local maximum which is easily found. The convergence test scheduling decision at iteration $j + d_{max}$ uses the $N(\lambda_{pt}, \sigma_{pt}^2)$ distribution as its prior.

The mechanics of our scheduling policy illustrate how we deal with uncertainty about λ. This policy is also dependent on quantities we now discuss: the sampling variance σ_s^2 and the initial prior distribution of λ. There are situations where significant prior knowledge of convergence behavior is known. The model problem is time-dependent, so we need to solve the equations at each of a number of time steps. If the same iterative method is used each time iteration, the *asymptotic* convergence rates will be the same for all the time iterations. However, the effective convergence rate may vary since initial errors will be different. The convergence behavior of the method at time steps in the near past is a good predictor of the convergence behavior in the near future. In fact, after the first three time steps, we were able to construct very reasonable priors *before* beginning a time step's iterations. We simply used the last timestep's effective λ as the prior mean (found by solving $\varepsilon = E_1 \cdot e^{-\lambda \cdot (N-1)}$ for λ, knowing that exactly N iterations were required); we used the sample variance of the last three timesteps' effective λ's for our prior variance. The Bayesian formulation can also exploit user experience with the solution method's convergence; this experience could be summarized as a prior distribution.

At the beginning of the computation we might well presume no prior knowledge of the convergence behavior. We gain some insight into this behavior by testing for convergence after each of the first few iterations. As before, a convergence test is viewed as an observation of λ. If we could assume that each observation is independent of any other, we could then use the sample mean as the initial prior mean λ_{pr}, and the sample variance as both the sample variance σ_s^2 and the prior variance σ_{pr}^2. However, successive errors E_i and E_{i+1} are not independent. Their correlation leads to a biased estimation of λ_{pr} and the underestimation of σ_{pr}^2. To compenstate for this conflict of mathematical assumption and practical reality, we devised the *constrained projection* rule. This rule states that if convergence is tested at iteration j, the next convergence test must be scheduled before iteration $2 \cdot j + 1$. This rule forces additional convergence tests at the beginning of the computation, and affords protection from wildly optimistic scheduling decisions. We thus used the sample statistics to construct our prior information, but then protected ourselves from a bad prior with the constrained projection rule. In our experience, this rule was effectively invoked only at the beginning of the computation. After this startup period, our underlying assumption of independence between observations of λ is better satisfied, and the statistics are more accurate. The variance σ_s^2 is then reasonably taken to be the sample variance of the observed λ's to date.

The MEW method is an excellent vehicle for encapsulating both our prior knowledge, and the knowledge gained about convergence as the solution progresses. Furthermore, it is simple to program, and its sensitivity to changes in the problem or problem distribution across processors lies only in the parameters I and D. Yet the sensitivity to these key parameters and measured behavior variation makes the MEW method more attractive than a simple "test every q iterations" approach, an approach left with the difficult problem of determining q. Most importantly, the empirical study described in the next section shows that MEW is quite effective in reducing convergence checking delay.

5. Convergence checking performance. The effects of employing the three different convergence schemes on the algorithm performance (on a 128 by 128 grid) is depicted in Fig. 6. The model problem was solved on a cube with 1, 2, 4, 8, 16, 32, and 64 processors; our implementation of ACC dedicated one node to the c-host function, so that we did not test this method with 64 processors (the maximum cube size on our system). In each test the domain was subdivided into strips of equal size, assigned one to each processor so that adjacent strips were mapped onto adjacent processors. In Fig. 6, the measured performance in terms of time-steps advanced per second is plotted as a function of the number of processors. Figure 6 illustrates that the performance of all three convergence schemes degrades with an increasing number of processors. The standard convergence scheme checking scheme depicted here involves the use of the tree method of global convergence checking used each iteration. The tree method of convergence checking was utilized as scheduled in the MEW scheme. Results obtained for the cascade method of convergence checking are quite similar to those depicted here. The standard convergence scheme's deterioration is rapid, while the other two schemes degrade gradually. The difference between the MEW and ACC schemes is not significant. Both have essentially the same communication costs, as very few ACC guesses and MEW checks were required on each timestep. The ACC overshot after global convergence was between three and five percent. The MEW scheme has a lower computation cost than ACC because it skips local convergence checking on some iterations altogether. The ACC scheme checks the state of each

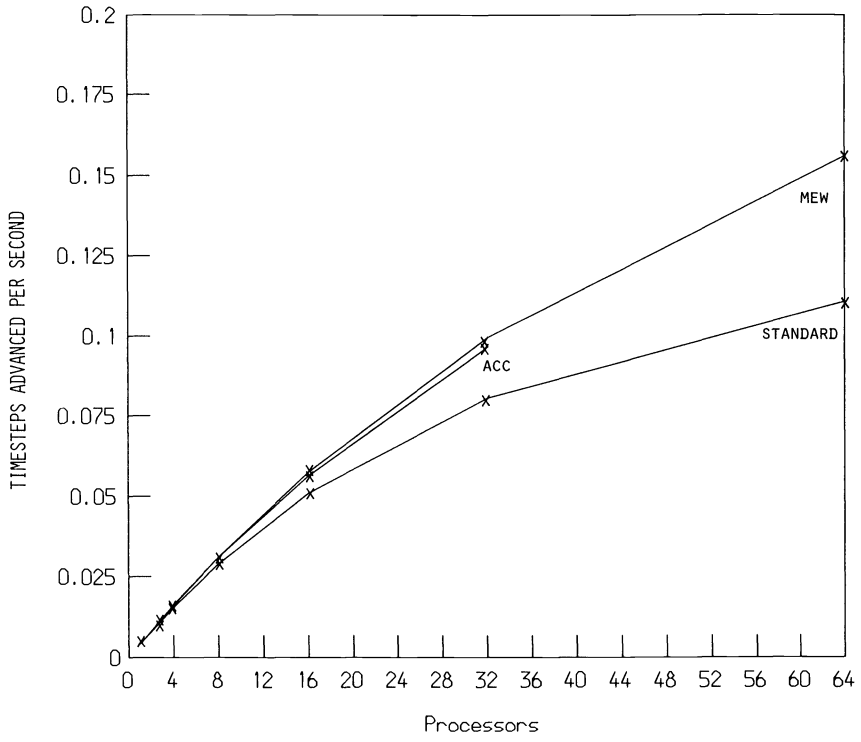


FIG. 6 *Effect of convergence checking schemes on algorithm performance.* (*Grid size* 128 × 128, *Subdomain Shape*: *Strips.*)

subdomain every iteration, although we observed a 10% improvement by checking every other iteration. The difference in local convergence checking accounts for the slight difference in the two schemes' performance. All three methods showed similar effects on the algorithm performance when the grid size was changed to 64 by 64 or 256 by 256.

6. Reduction of communication delays resulting from windows. Reduction of communication delays in the iterative solution of the linear equations produced by a discretization of a time dependent problem can be effected by iteratively solving more than one timestep during a given stage of the computations. The boundary variable values from more than one timestep can be sent in a packet, thus reducing the effect of the message transmission overhead.

In an iterative solution of a time dependent problem, one generally iterates over each timestep individually until convergence at that timestep is detected. One may instead iterate over more than one timestep during each stage of a computation. Assume that we are iterating over variables at timesteps $t_1, \cdots, t_n$. During each iterative sweep, all variables are updated in each of the timesteps included in the window. Following the sweep, the right-hand sides of equations at timesteps $t_2, \cdots, t_n$ are updated to account for changes in the variable values at the earlier timesteps. Convergence is checked at t_1, and when global convergence is detected at this time the window shifts up one timestep to encompass $t_2, \cdots, t_{n+1}$. It has been shown [9] that the asymptotic rate of convergence of SOR implemented with windows is equivalent to that of SOR applied to each timestep individually. In other words the total number of sweeps over each timestep required for a given degree of error reduction does not, in any asymptotic

sense, change with the window size. For finite difference equations in which time discretization is by Crank–Nicholson or backwards Euler methods, the operation count for each sweep over a timestep is minimally effected by the use of windows. In practice, the computational work required to solve a problem increases quite gradually with window size.

Communication costs are reduced in two ways when one iterates over windows of timesteps in a time-dependent problem. The first is the previously stated fact that fewer but larger packets need be sent for the transmission of boundary variable data. Because convergence need be checked only at the lowest timestep in a window, the number of global communications required to check for convergence is reduced by a factor of 1/window-size as long as the total number of sweeps over each timestep does not change with window size, as is approximately the case for small windows. Thus if the total number of sweeps over each timestep were independent of window size and the cost per packet were independent of the size of the packet, the overall cost of communication would be reduced by a factor of 1/window-size. Finally, some computation time is saved because the computations required for local convergence checking need only be carried out at the lowest timestep in a window.

Algorithm performance is improved by the use of windows of a relatively small size. A number of experiments were carried out to demonstrate this; the results of one set are shown in Fig. 7. The model problem was solved on a 64 by 64 point domain, and the rate of computation resulting from the use of windows of sizes 1, 2 and 3 is shown. A notable improvement in performance is seen in comparing windows of sizes 1 and 2; a less marked improvement in performance is seen when a window of size 3 was utilized. This result is expected as the computational cost increases as window

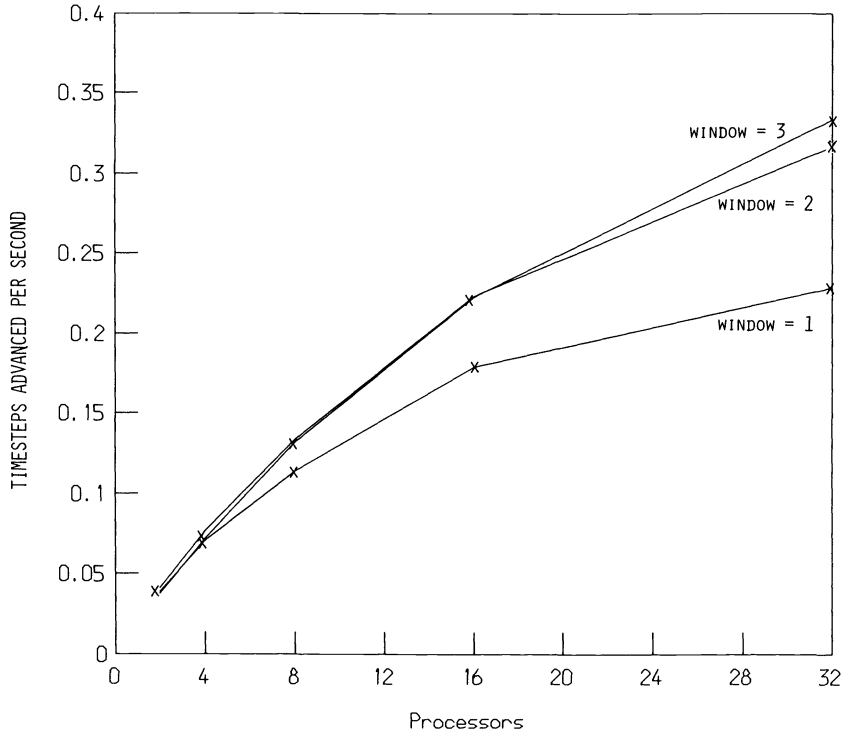


FIG. 7. *Effect of windows on performance.* (*Grid*: 64×64, *Shape*: *Strips, Standard Conv. Checking.*)

size is increased, and communication delays can be reduced by no more than a factor of 1/window size.

The use of windows decreases the cost of communicating boundary variables and communication flags at the cost of increased storage requirements and an increase in the computation required for each timestep. The methods for reducing the costs of global convergence checking described here have minimal costs and storage requirements. It is hence natural that the methods should be used together. In Fig. 8 are depicted results for the model problem solved on a 128 by 128 point domain using windows of sizes 1 and 2 with Bayesian MEW convergence test scheduling and using windows of sizes 1 and 2 with standard convergence testing. The use of both windowing and Bayesian convergence test scheduling together led to additional improvements in performance over that obtained through the use of either separately.

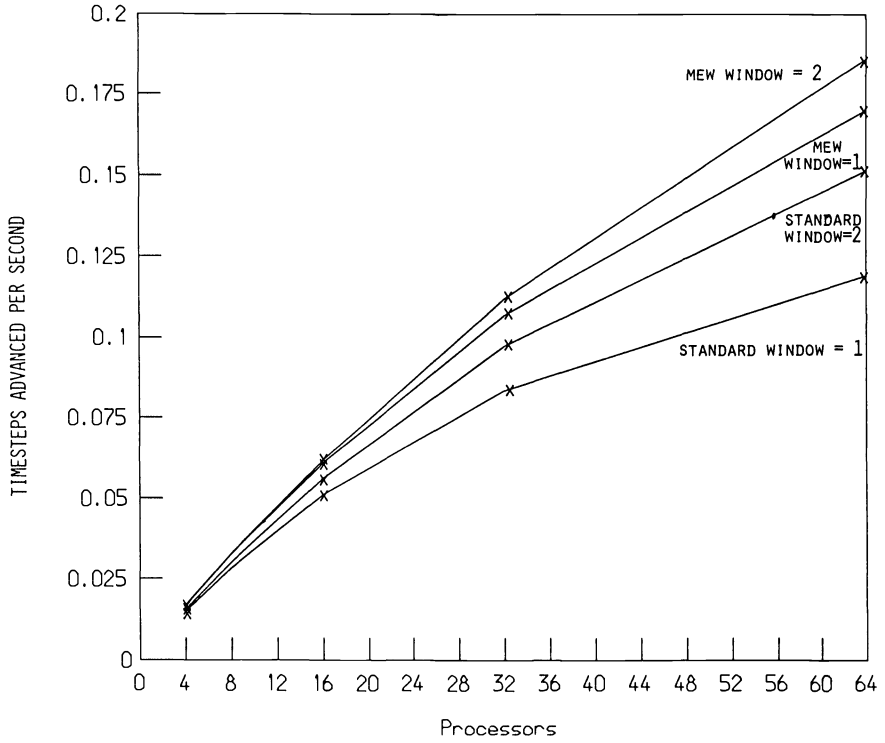


FIG. 8. *Effect of windows and of convergence checking schemes on performance.* (*Grid*: 128×128, *Shape*: *Strips.*)

7. Conclusion. The sources of communication delays in the solution of a model problem by red-black SOR have been identified and their relative contribution quantified under a number of circumstances. A number of methods have been proposed and tested to reduce the effects of each of these sources of communication delay.

In a message passing multiprocessor whose overhead for communication β is substantially larger than $1/\alpha$, the time required to send a byte, there is considerable motivation to reduce the number of messages that must be sent. In § 3 the discussion and experimental tests on domain partitioning indicate that improvements in performance may in many circumstances be obtained by reducing the number of messages that must be sent even when the total number of bytes to be sent must increase. In § 5

it is seen that through the use of windows the number of messages that must be sent is decreased. In this case the trade-off is a small increase in the cost of computation, and again an overall performance improvement is noted.

Checking global convergence when message overhead is high is quite costly, and methods were described in § 3 to perform tests efficiently and to reduce the number of such tests required. It is demonstrated in § 5 that the effects of using windows and of using methods that reduce the effects of convergence costs can have a complementary effect of performance.

It should be noted that convergence testing is the only nonlocal communication in red-black SOR. Both the ACC and the MEW method greatly reduce global communication and are consequently expected to be quite useful in architectures where the interprocessor connectivity is more restricted, such as a ring or a mesh multiprocessor. In these architectures, combining results from all processors may be quite expensive for large machines.

Acknowledgments. We would like to thank the Mathematics and Statistics Research Section at Oak Ridge National Laboratory for generously granting us use of their Intel Personal Super-Computer. We are particularly indebted to Mike Heath and Tom Dunigan for assistance during the course of this work. We are grateful to Bob Voigt for constant support and encouragement.

REFERENCES

[1] L. M. ADAMS AND J. M. ORTEGA, *A multi-color* SOR *method for parallel computation*, Proc. of the International Conference on Parallel Processing, K. E. Batcher, W. C. Meilander and J. L. Potter, eds., August 24-27, 1982, pp. 53-56.

[2] G. C. FOX, *Concurrent processing for scientific calculations*, COMPCON84, 28th IEEE Computer Society International Conference, 1984, pp. 70-73.

[3] E. M. REINGOLD, J. NIEVERGELT AND N. DEO, *Combinatorial Algorithms*, Prentice-Hall, Englewood Cliffs, NJ, 1977, pp. 173-189.

[4] C. L. SEITZ, *The cosmic cube*, Comm. ACM, 28 (1985), pp. 22-33.

[5] R. HOCKNEY AND C. JESSHOPE, *Parallel computers: Architecture Programming, and Algorithms*, Adam Hilger Ltd., Bristol, UK, 1981.

[6] J. ORTEGA AND R. VOIGT, *Solution of partial differential equations on vectors and parallel computers*, SIAM Rev., 27 (June 1985), pp. 149-240.

[7] J. RATTNER, *Concurrent processing: A new direction in scientific computing*, AFIPS Conference Proceedings 1985 National Computer Conference, Vol. 54, pp. 157-166.

[8] Y. SAAD AND M. H. SCHULTZ, *Topological properties of hypercubes*, Research Report YALEU/DCS/RR-389, Yale University, New Haven, CT, 1985.

[9] J. H. SALTZ, *Parallel and adaptive algorithms in scientific and medical computing: robust method for the solution of partial differential equations on multiprocessor machines*, Ph.D. thesis, Duke University, 1985.

[10] J. H. SALTZ AND V. K. NAIK, *Towards developing robust algorithms for solving partial differential equations on* MIMD *machines*, ICASE Report No. 85-39, NASA Contractor Report No. 177979, 1985.

[11] S. A. SCHMITT, *An Elementary Introduction to Bayesian Statistics*, Addison-Wesley, Reading, MA, 1969.

[12] C. L. SEITZ, *The cosmic cube*, Comm. ACM, 28 (1985), pp. 22-33.

[13] R. G. VOIGT, *Where are the parallel algorithms*, 1985 National Computer Conference Proceedings, AFIPS Press, Reston, Virginia, 1985, pp. 329-334.

[14] D. M. YOUNG, *Iterative Solutions of Large Linear Systems*, Academic Press, New York, 1971.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 1, January 1987

© 1987 Society for Industrial and Applied Mathematics

TIMELY COMMUNICATIONS

Under the "timely communications" policy for the SIAM Journal of 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.

A FAST ALGORITHM FOR TRUMMER'S PROBLEM*

A. GERASOULIS†, M. D. GRIGORIADIS† AND LIPING SUN†

Abstract. We describe an algorithm of $O(n(\log n)^2)$ arithmetic time complexity for computing the product **By**, where **y** is a given vector and **B** is Trummer's matrix defined by $b_{ij}=1/(c_i-c_j)$ if $i\neq j$, and $b_{ii}=0$ if $i=j$, for all $i,j=1,\cdots,n$. The c_i, $i=1,\cdots,n$ are assumed to be distinct. This problem arises in the computation of conformal mappings, the Riemann zeta function and Cauchy singular integrals.

Key words. computational complexity, matrix vector multiplication, Fast Fourier Transforms, polynomial interpolation, polynomial evaluation

AMS (MOS) subject classifications. 68C25, 65F30

1. Introduction. Consider an n-order matrix **B** defined by $b_{ij}=1/(c_i-c_j)$ if $i\neq j$, and $b_{ij}=0$ if $i=j$, for all $i,j=1,\cdots n$, where c_i, $i=1,\cdots,n$ are distinct. Let **y** be any given n-vector. Golub [4], [5] states *Trummer's problem* as follows:

> Give an algorithm for computing **By** in less than $O(n^2)$multiplications.
> If this is impossible, show that it cannot be done.

The problem arises in various applications such as the computation of conformal mappings (Trummer [8], Kerzman and Trummer [6]), multiple evaluation of the Riemann zeta function (Odlyzko and Schönhage [7]), and the numerical solution of Cauchy singular integral equations (Gerasoulis [3]). In these instances n may be very large and a fast evaluation of **By** is not only of theoretical but also of practical interest, particularly if **By** is to be repeatedly computed.

In [2], Gastinel presents a method for finding the inverse of a generalized Hilbert matrix by evaluating an interpolation polynomial. This is substantially different from Trummer's problem. However, Gastinel's result suggests an approach which reduces Trummer's problem to one of finding an interpolation polynomial and its derivatives, and their evaluation at the n points c_i. We describe such a reduction below and outline an $O(n(\log n)^2)$ algorithm for computing **By**.

2. Equivalence of Trummer's problem and polynomial evaluation. Consider the nth degree polynomial

$$g(x)=\prod_{i=1}^{n}(x-c_i) \tag{1}$$

* Received by the editors January 27, 1986; accepted for publication April 2, 1986. This material is based upon work supported by the National Science Foundation under grants DMS-8506464 and MCS-8113503. A draft of this paper was made available to the NA.DIS@SU-SCORE distribution list on October 30, 1985.

† Department of Computer Science, Rutgers University, New Brunswick, New Jersey 08903.

and determine the polynomial $h(x)$ whose degree is at most $n-1$ from

$$\frac{h(x)}{g(x)} = \sum_{i=1}^{n} \frac{y_i}{x - c_i}. \tag{2}$$

Setting $x = c_j$ in (2) we have

$$h(c_j) = y_j g'(c_j), \qquad j = 1, \cdots, n \tag{3}$$

as was previously observed by Gastinel [2]. Now, multiplying (2) by $g(x)$, and differentiating, we obtain

$$h'(x) = g'(x) \sum_{i=1}^{n} \frac{y_i}{x - c_i} - g(x) \sum_{i=1}^{n} \frac{y_i}{(x - c_i)^2} \tag{4}$$

which may be rewritten by extracting the diagonal from the sums, i.e.,

$$h'(x) = D_j(x) + g'(x) \sum_{\substack{i=1 \\ i \neq j}}^{n} \frac{y_i}{x - c_i} - g(x) \sum_{\substack{i=1 \\ i \neq j}}^{n} \frac{y_i}{(x - c_i)^2} \tag{5}$$

where

$$D_j(x) = \frac{g'(x) y_j}{x - c_j} - \frac{g(x) y_j}{(x - c_j)^2}. \tag{6}$$

Setting $x = c_j$ in (5), and recalling that $g(c_j) = 0$, we have

$$h'(c_j) = D_j(c_j) + g'(c_j) \sum_{\substack{i=1 \\ i \neq j}}^{n} \frac{y_i}{c_j - c_i}, \qquad j = 1, \cdots, n. \tag{7}$$

Expanding (6) in Taylor's series

$$\begin{aligned} D_j(x) &= \frac{y_j}{(x - c_j)} \left(g'(x) - \frac{g(x)}{x - c_j} \right) \\ &= \frac{y_j}{(x - c_j)} \left(g'(c_j) + (x - c_j) g''(c_j) + \cdots - g'(c_j) - \frac{1}{2}(x - c_j) g''(c_j) - \cdots \right) \\ &= y_j \left(\frac{1}{2} g''(c_j) + (x - c_j) \times (\text{higher order terms}) \right), \end{aligned}$$

and setting $x = c_j$, we obtain

$$\lim_{x \to c_j} D_j(x) = D_j(c_j) = \tfrac{1}{2} y_j g''(c_j). \tag{8}$$

It is now easy to see that (7) is equivalent to $\mathbf{By} = \mathbf{x}$, where

$$x_j = \frac{h'(c_j) - D_j(c_j)}{g'(c_j)}, \qquad j = 1, \cdots, n. \tag{9}$$

Hence, Trummer's problem is equivalent to evaluating $\mathbf{x}$ from (9).

3. A fast algorithm for the evaluation of x. In what follows we assume without loss of generality that $n = 2^k$ for some $k \geq 1$. We adopt the straight-line model of computation for RAM programs, where we consider each arithmetic operation to be performed in $O(1)$ time (see e.g. Aho et al. [1, pp. 19–22]). The vector **c** denotes the given points $(c_1, \cdots, c_n)$.

Procedure Trummer(n, **c**, **y**); **return x**;

1. Compute the coefficients d_i of $g(x) = \sum_{i=0}^{m} d_i x^i$, starting from its form (1) and by proceeding upward in a binary tree with n leaves. This may be done in $O(n(\log n)^2)$ time (Aho et al. [1, Thm. 8.14, p. 299]);
2. Compute the coefficients of $g'(x)$ and $g''(x)$ in $O(n)$ time each;
3. Evaluate $g'(c_j)$ and $g''(c_j)$, $j = 1, \ldots, n$, in $O(n(\log n)^2)$ time (Aho et al. [1, Cor. 2, p. 294]);
4. Compute $h(c_j) = y_j g'(c_j)$, $j = 1, \cdots, n$, in $O(n)$ time;
5. Find the polynomial $h(x)$ which interpolates the points $(c_j, h(c_j))$, $j = 1, \ldots, n$, in $O(n\ (\log n)^2)$ time (Aho et al. [1, Thm. 8.14, p. 299]);
6. Compute the coefficients of $h'(x)$ in $O(n)$, and evaluate $h'(c_j)$, $j = 1, \ldots, n$, in $O(n(\log n)^2)$ time, in a manner similar to steps 2 and 3;
7. Compute $D_j(c_j) = \frac{1}{2} y_j g''(c_j)$, $j = 1, \cdots, n$, in $O(n)$ time;
8. Compute $x_j = (h'(c_j) - D_j(c_j))/g'(c_j)$, $j = 1, \cdots, n$, in $O(n)$ time;

end *TRUMMER*;

The total cost of steps 1–8 is $O(n(\log n)^2)$ time and $O(n \log n)$ space.

The issues of numerical stability and practical implementations of this *general* procedure, for an arbitrary set of points c_i, remain to be addressed. However, in many instances one is able to take advantage of some structure of the given points to simplify the algorithm. This may allow further improvements in time compexity and may also ensure adequate numerical accuracy for particular applications. For instance, if the c_i are the Chebyshev points, a situation which frequently arises in quadrature approximations of singular integrals, then the complexity of the algorithm reduces to $O(n \log n)$. Our preliminary computational experience with these points indicates that the algorithm provides satisfactory accuracy for large n. Even in single precision floating point arithmetic (on the DECSYSTEM-2060T), the maximum error does not exceed 10^{-5} for all $n \leq 2^{13}$. Details of this experimentation and several applications are presented in Gerasoulis [3].

Acknowledgment. We thank Professor Gene Golub for suggesting the problem and for providing us with reference [2].

REFERENCES

[1] A. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] N. GASTINEL, *Inversion d'une matrice generalisant la matrice de Hilbert*, Chiffres, 3 (1960), pp. 149–152.

[3] A. GERASOULIS, *A fast algorithm for the multiplication of generalized Hilbert matrices with vectors*, Technical Report LCSR-TR-79, Dept. Comp. Sci., Rutgers Univ., New Brunswick, NJ, 1986.

[4] G. GOLUB, *Trummer's problem*, Communication to NA.DIS@SU-SCORE, July 22, 1985.

[5] ———, *Trummer's problem*, SIGACT News, ACM Special Interest Group on Automata and Computability Theory, 17 (1985), p. 17.2–12.

[6] N. KERZMAN AND M. TRUMMER, *Numerical conformal mapping via the Szegö kernel*, J. Comput. Appl. Math., 14 (1986), pp. 111-123.

[7] A. M. ODLYZKO AND A. SCHÖNHAGE, *Fast algorithms for multiple evaluations of the Riemann zeta function*, Technical Report, AT & T Bell Laboratories, Murray Hill, NJ, 1986.

[8] M. TRUMMER, *An efficient implementation of a conformal mapping method using the Szegö kernel*, SIAM J. Numer. Anal., 23 (1986), pp. 853-872.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 1, January 1987
© 1987 Society for Industrial and Applied Mathematics

A SIMPLIFIED TVD FINITE DIFFERENCE SCHEME VIA ARTIFICIAL VISCOSITY*

STEPHEN F. DAVIS†

Abstract. In this paper we show that the total variation diminishing (TVD) finite difference scheme which was analysed by Sweby [8] can be interpreted as a Lax-Wendroff scheme plus an upwind weighted artificial dissipation term. We then show that if we choose a particular flux limiter and remove the requirement for upwind weighting, we obtain an artificial dissipation term which is based on the theory of TVD schemes, which does not contain any problem dependent parameters and which can be added to existing MacCormack method codes. Finally, we conduct numerical experiments to examine the performance of this new method.

Key words. finite difference methods, hyperbolic partial differential equations

AMS(MOS) subject classifications. 65M05, 76-08

1. Introduction. A major result of recent research into numerical methods for the solution of systems of hyperbolic conservation laws has been the development of second order accurate total variation diminishing (TVD) finite difference schemes. These schemes have a number of attractive properties including the fact that they resolve discontinuous solutions well, they do not exhibit spurious oscillations and, under certain circumstances, they can be proven to converge.

Unfortunately, the apparent complexity of these schemes has thus far discouraged their widespread adoption. Instead, most applications programs use standard methods such as the MacCormack variant of the Lax-Wendroff method and add additional artificial dissipation to damp the spurious wiggles that occur near discontinuities. These dissipation terms are usually chosen in an ad hoc fashion and usually contain problem-dependent parameters which must be fine tuned before the method will work.

In this paper we attempt to remedy this situation. In particular, we interpret the TVD finite difference scheme which was analysed by Sweby [8] as a Lax-Wendroff scheme plus an upwind weighted artificial dissipation term. We then attempt to simplify this artificial dissipation term by removing the requirement that it be upwind weighted. The result is a dissipation term which is based on the theory of TVD schemes and which does not contain any free parameters.

An outline of this paper is as follows. In § 2 we briefly review the theory of TVD finite difference schemes. Using this theory, in § 3, we show that a TVD scheme can be constructed by adding an artificial viscosity-like term to a standard finite difference scheme. Numerical experiments are performed to examine the performance of this new scheme. These results are extended to systems of hyperbolic equations in § 4 and additional numerical experiments are performed. Section 5 contains some closing remarks.

2. Total variation diminishing finite difference schemes. We consider the initial value problem for a scalar conservation law. That is

$$u_t + f(u)_x = u_t + a(u)u_x = 0, \quad a(u) = \frac{df}{du}(u), \quad t > 0$$
$$u(x, 0) = u_0(x), \qquad -\infty < x < \infty \tag{2.1}$$

* Received by the editors July 10, 1984, and in revised form April 22, 1985. This research was supported by the National Aeronautics and Space Administration under NASA contract NAS1-17070 while the author was in residence at ICASE, NASA Langley Research Center, Hampton, Virginia 23665.

† Applied Mathematics Branch, Naval Surface Weapons Center, White Oak, Silver Spring, Maryland 20903.

where $u_0(x)$ is assumed to have bounded total variation. A weak solution to this problem has the following monotonicity properties:

(1) No new extrema in $u(x)$ may be created.

(2) The value of a local minimum is nondecreasing and the value of a local maximum is nonincreasing.

The total variation of the solution to (2.1) at time t is defined by the formula

$$TV(u(x, t)) = \sup \sum_k |u(x_{k+1}, t) - u(x_k, t)|, \tag{2.2}$$

where the supremum is taken over all partitions of the real line.

It follows from this monotonicity property that the total variation in x of $u(x, t)$ does not increase in t. That is,

$$TV(u(t_2)) \leqq TV(u(t_1)) \quad \text{for all } t_2 \geqq t_1. \tag{2.3}$$

Much recent research has been devoted to the construction of finite difference schemes that satisfy a discrete version of equation (2.2). We briefly describe this work below.

Consider explicit finite difference schemes in conservation form which approximate (2.1) and which we denote by

$$U^{n+1} = L \cdot U^n. \tag{2.4}$$

A scheme is called *total variation diminishing* if

$$TV(U^{n+1}) = TV(L \cdot U^n) \leqq TV(U^n). \tag{2.5}$$

In addition, a scheme is called *monotonicity preserving* if the finite difference operator L is monotonicity preserving; that is, U^n a monotone mesh function implies that $L \cdot U^n$ is also a monotone mesh function. The following result is due to Harten [3].

THEOREM 2.1 (Harten). *A total variation diminishing scheme is monotonicity preserving.*

This says that TVD schemes will not produce spurious oscillations. This is their chief attraction.

Another reason why TVD schemes are attractive is that it is very useful to have a bound on the total variation of the solution when proving convergence of nonlinear difference schemes (cf. [4], [7]). Equation (2.5) provides such a bound but convergence proofs are beyond the scope of this paper.

The scheme (2.4) can be written in the form

$$U_k^{n+1} = U_k^n - C_{k-1/2}\Delta U_{k-1/2}^n + D_{k+1/2}\Delta U_{k+1/2}^n \tag{2.6}$$

where

$$\Delta U_{k+1/2}^n = U_{k+1}^n - U_k^n \tag{2.7}$$

and $C_{k-1/2}$ and $D_{k+1/2}$ are functions of U^n. The choice of $C_{k-1/2}$ and $D_{k+1/2}$ is not unique. If a particular scheme can be rearranged so that C and D satisfy the hypotheses of the following lemma, the scheme can be proved to be TVD.

LEMMA 2.2 (Harten). *If the coefficients C and D of* (2.6) *satisfy the inequalities*

$$\begin{aligned} &0 \leqq C_{k+1/2}, \quad 0 \leqq D_{k+1/2}, \\ &0 \leqq C_{k+1/2} + D_{k+1/2} \leqq 1, \end{aligned} \tag{2.8}$$

then the scheme (2.6) *is total variation diminishing.*

In the next section we derive a scheme and use this lemma to prove that it is TVD.

3. Scalar equations. For ease of presentation, we first consider the scalar linear equation

$$u_t + au_x = 0, \qquad a = \text{const} > 0. \tag{3.1}$$

Sweby [8] considers the solution to this problem using a scheme of the form

$$U_k^{n+1} = U_k^n - \nu\{1 + \tfrac{1}{2}(1-\nu)[\phi(r_k^+)/r_k^+ - \phi(r_{k-1}^+)]\}\Delta U_{k-1/2}^n \tag{3.2}$$

where

$$\nu = \frac{a\Delta t}{\Delta x}, \qquad r_k^+ = \frac{\Delta U_{k-1/2}^n}{\Delta U_{k+1/2}^n} \tag{3.3}$$

and $\phi(r_k)$ is the flux limiter. This is a scheme of the form (2.6) with

$$\begin{aligned} C_{k-1/2} &= \nu\{1 + \tfrac{1}{2}(1-\nu)[\phi(r_k^+)/r_k^+ - \phi(r_{k-1}^+)]\}, \\ D_{k+1/2} &= 0. \end{aligned} \tag{3.4}$$

A sufficient condition for the scheme (3.2) to be TVD is that $\nu \leqq 1$ and

$$|\phi(r_k^+)/r_k^+ - \phi(r_{k-1}^+)| \leqq 2. \tag{3.5}$$

Sweby also specifies that $\phi(r) > 0$ and that $\phi(r) = 0$ for $r \leqq 0$. Under these additional restrictions the bound (3.5) becomes

$$0 \leqq \frac{\phi(r)}{r} \leqq 2, \qquad 0 \leqq \phi(r) \leqq 2. \tag{3.6}$$

If $\phi(r) = 1$, scheme (3.2) reduces to the centered difference Lax-Wendroff method. If $\phi(r) = r$, scheme (3.2) reduces to the second order upwind Warming and Beam [10] method.

The region defined by (3.6) is shown in Fig. 1 along with the limiters corresponding to the Lax-Wendroff and Warming-Beam methods. Since these schemes are known to produce spurious wiggles in solutions with strongly varying gradients, it is not surprising that these schemes are not uniformly within the TVD region.

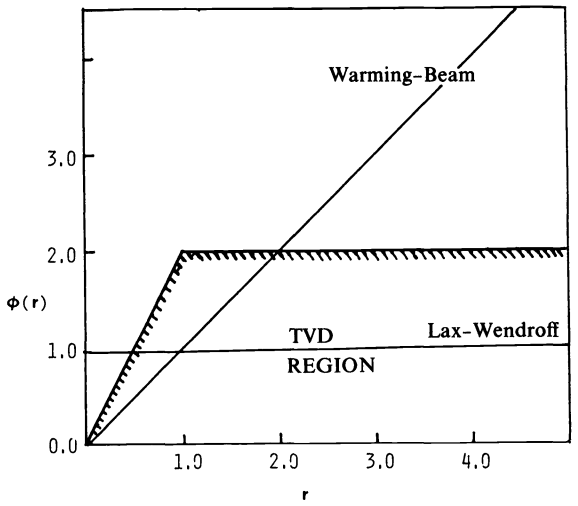


FIG. 1. TVD *region.*

Since the Lax-Wendroff method does not require one to determine an upwind direction and since many production computer codes are based on the Lax-Wendroff method or its variants, we wish to examine the possibility of adding terms to these codes to obtain a TVD scheme.

If we add a term of the form

$$K^+_{k+1/2}(r^+_k)\Delta U^n_{k+1/2} - K^+_{k-1/2}(r^+_{k-1})\Delta U^n_{k-1/2} \tag{3.7}$$

to the Lax-Wendroff scheme

$$U^{n+1}_k = U^n_k - \frac{\nu}{2}(\Delta U^n_{k+1/2} + \Delta U^n_{k-1/2}) + \frac{\nu^2}{2}(\Delta U^n_{k+1/2} - \Delta U^n_{k-1/2}) \tag{3.8}$$

and rearrange this to the form (3.2), we get

$$U^{n+1}_k = U^n_k - \left[\nu\left\{1 + \frac{1}{2}(1-\nu)[1/r^+_k - 1]\right\} - \left\{\frac{K^+_{k+1/2}}{r^+_k} - K^+_{k-1/2}\right\}\right]\Delta U^n_{k-1/2}. \tag{3.9}$$

That is,

$$\begin{aligned} C_{k-1/2} &= \nu\left\{1 + \frac{1}{2}(1-\nu)[1/r^+_k - 1]\right\} - \left\{\frac{K^+_{k+1/2}}{r^+_k} - K^+_{k-1/2}\right\}, \\ D_{k+1/2} &= 0, \end{aligned} \tag{3.10}$$

where the $^+$ superscript denotes terms which are added when $a > 0$. Note that the form (3.7) preserves conservation.

A comparison of (3.9) with (3.2) shows that we can obtain Sweby's scheme if we choose

$$K^+_{k+1/2} = \frac{\nu}{2}(1-\nu)[1 - \phi(r^+_k)]. \tag{3.11}$$

Next we consider the equation

$$u_t + au_x = 0, \qquad a = \text{const} < 0. \tag{3.12}$$

For this problem Sweby's scheme takes the form

$$U^{n+1}_k = U^n_k + \nu\left\{-1 + \frac{1}{2}(1+\nu)\left[\phi(r^-_{k+1}) - \frac{\phi(r^-_k)}{r^-_k}\right]\right\}\Delta U^n_{k+1/2} \tag{3.13}$$

where

$$r^-_k = \frac{\Delta U^n_{k+1/2}}{\Delta U^n_{k-1/2}}, \qquad \nu = \frac{a\Delta t}{\Delta x} < 0$$

and our modified Lax-Wendroff scheme takes the form

$$U^{n+1}_k = U^n_k + \nu\left[\left\{-1 + \frac{1}{2}(1+\nu)\left[1 - \frac{1}{r^-_k}\right]\right\} + \left\{K^-_{k+1/2} - \frac{K^-_{k-1/2}}{r^-_k}\right\}\right]\Delta U^n_{k+1/2}. \tag{3.14}$$

This reduces to Sweby's scheme if we choose

$$K^-_{k+1/2} = \frac{\nu(1+\nu)}{2}[\phi(r^-_{k+1}) - 1]. \tag{3.15}$$

A comparison of equations (3.11) and (3.15) shows how the sign of the coefficient a changes the method. In particular, we see that as a changes sign, the definition of

r changes and the form of the term involving the Courant number ν changes. We can combine these two cases into a Lax-Wendroff method with an upstream weighted artificial dissipation term as follows. Put the scheme into the form

$$U_k^{n+1} = U_k^n - \frac{\nu}{2}(U_{k+1}^n - U_{k-1}^n) + \frac{\nu^2}{2}(U_{k+1}^n - 2U_k^n + U_{k-1}^n)$$
$$+ [K_{k+1/2}^+(r_k^+) + K_{k+1/2}^-(r_{k+1}^-)](U_{k+1}^n - U_k^n) \tag{3.16}$$
$$- [K_{k-1/2}^+(r_{k-1}^+) + K_{k-1/2}^-(r_k^-)](U_k^n - U_{k-1}^n)$$

where

$$K_{k+1/2}^+ = \begin{cases} \frac{\nu}{2}(1-\nu)[1-\phi(r_k^+)] & \text{if } a > 0, \\ 0 & \text{if } a \leqq 0, \end{cases}$$

(3.17)

$$K_{k+1/2}^- = \begin{cases} 0 & \text{if } a \geqq 0, \\ \frac{\nu}{2}(1+\nu)[\phi(r_{k+1}^-) - 1] & \text{if } a < 0. \end{cases}$$

This method still requires that we know which direction is upwind. The determination of upwind directions for hyperbolic systems is a complicated procedure. Therefore we redefine the K's as follows and attempt to construct a method without upstream weighting.

$$K_{k+1/2}^+ = \frac{|\nu|}{2}(1-|\nu|)[1-\phi(r_k^+)],$$
$$K_{k+1/2}^- = \frac{|\nu|}{2}(1-|\nu|)[1-\phi(r_{k+1}^-)]. \tag{3.18}$$

That is, we always include terms involving both sets of K's.

Next we need to consider how the added terms affect the accuracy in smooth regions. To do this we need to define what we mean by a smooth region. For the purposes of this paper, we define a smooth region as a region in which $\Delta U = O(h)$ and $(1-r) = O(h)$ where h is the discretization parameter. With some changes of notations, this definition is consistent with the work of Harten [3] and Roe [5]. In [3] Harten shows that a scheme is second-order accurate if its numerical flux differs from the Lax-Wendroff numerical flux by $O(h^2)$ in smooth regions. In the following the limiter ϕ is chosen so that $(1-\phi(r)) = O(h)$ when $(1-r) = O(h)$ and the Harten condition is satisfied.

Since the terms involving the K's are second difference terms with positive coefficients, it is obvious that they are dissipative. The question that needs to be addressed is how much more dissipative (3.18) is than (3.17). To obtain some idea about this we compute the solution to the equation

$$u_t + u_x = 0 \tag{3.19}$$

with square wave initial data. We use the Lax-Wendroff method and the limiter defined by

$$\phi(r) = \begin{cases} \min(2r, 1) & \text{if } r > 0, \\ 0 & \text{if } r \leqq 0. \end{cases} \tag{3.20}$$

Note that we are using the same limiter for both the upwind and nonupwind schemes and are thus not taking advantage of the wider choice of limiters allowed by upwind schemes. This is done to compare upwind and nonupwind versions of the same scheme and isolate the effect of upwinding. In a future paper we examine other choices of limiter functions for both methods.

Fig. 2a shows the result of this computation after 100 steps at a Courant number, $\nu = .9$ using the Lax-Wendroff method without additional dissipation. Note that the solution exhibits severe oscillations in those regions where r^+ is small as predicted by Fig. 1. Figure 2b is the result of the same computation using the Lax-Wendroff scheme with the upwind dissipation (3.17) and the flux limiter (3.20). Notice that the spurious oscillations have been removed. Finally, Fig. 2c shows the result of the same computation using the Lax-Wendroff scheme with the simplified dissipation (3.18) and the flux limiter (3.20). These results are almost indistinguishable from those of Fig. 2b. In addition we can prove the following theorem.

THEOREM 3.1. *The method* (3.16), (3.18) *with flux limiter* (3.20) *is* TVD *under the restriction that the Courant number* $|\nu| < 1$.

The proof appears in the Appendix.

It is trivial to extend these results to scalar nonlinear problems. We simply define a local wave speed by

$$a_{k+1/2} = \begin{cases} \dfrac{\Delta f_{k+1/2}}{\Delta U_{k+1/2}} & \text{if } \Delta U_{k+1/2} \neq 0, \\ \dfrac{df(U_k)}{dU} & \text{if } \Delta U_{k+1/2} = 0 \end{cases} \tag{3.21}$$

and apply the schemes (3.16), (3.17) or (3.16), (3.18) as before. The wave speed definition (3.33) makes the resulting scheme conservative.

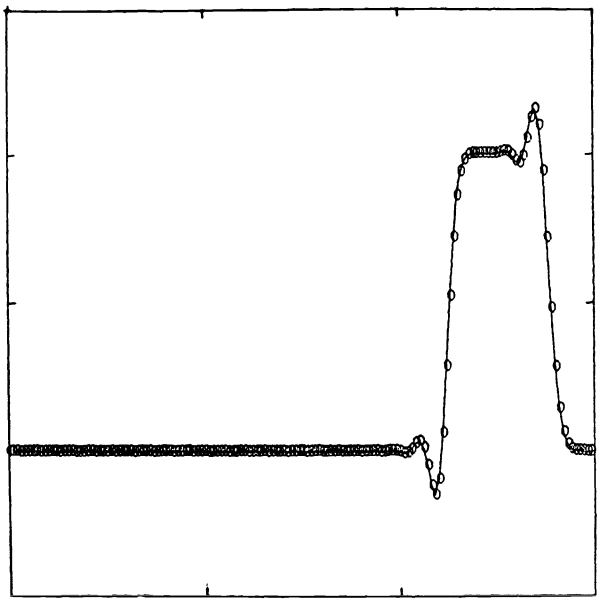


FIG. 2a. *Solution of* (3.19) *with square wave initial data after* 100 *steps at* $\nu = .9$ *using Lax-Wendroff method.*

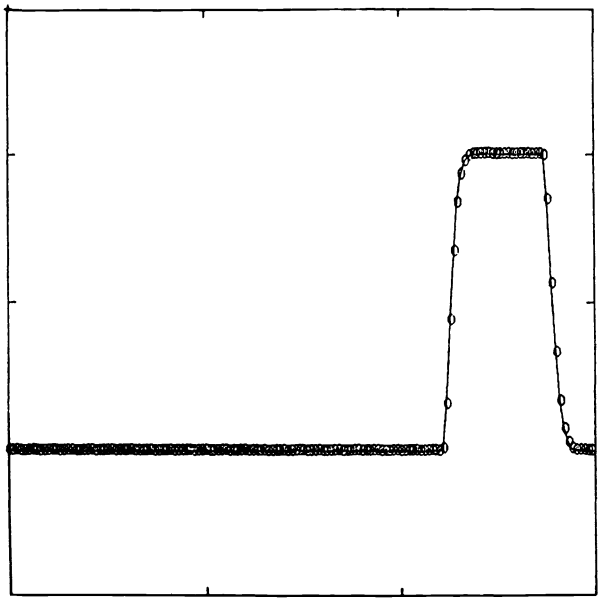


FIG. 2b. *Solution of* (3.19) *with square wave initial data after* 100 *steps at* $\nu = .9$ *using upwind scheme* (3.16), (3.17).

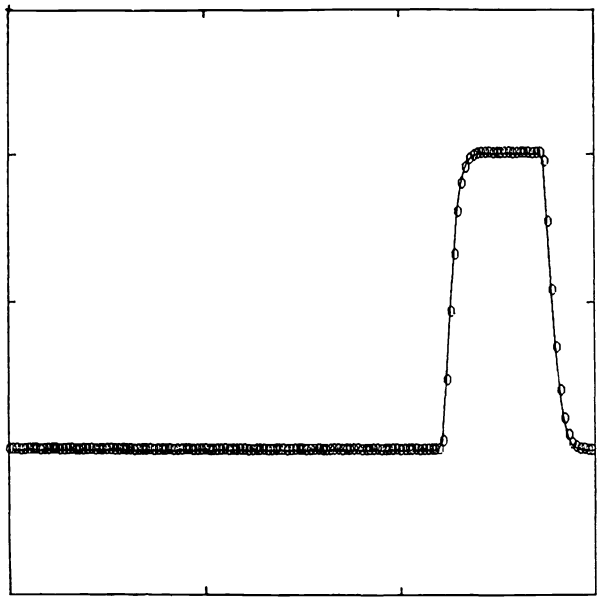


FIG. 2c. *Solution of* (3.19) *with square wave initial data after* 100 *time steps at* $\nu = .9$ *using* TVD *scheme* (3.16), (3.18).

Figures 3a, 3b and 3c show computed solutions of the inviscid Burgers' equation with square wave initial data and periodic boundary conditions. These results were obtained using the MacCormack two-step Lax-Wendroff scheme defined in the next section, the second order upwind scheme (3.16), (3.17) and the simplified scheme (3.16), (3.18), respectively. The MacCormack results exhibit severe oscillations in the vicinity of the shock and an entropy violating expansion shock. The upwind scheme

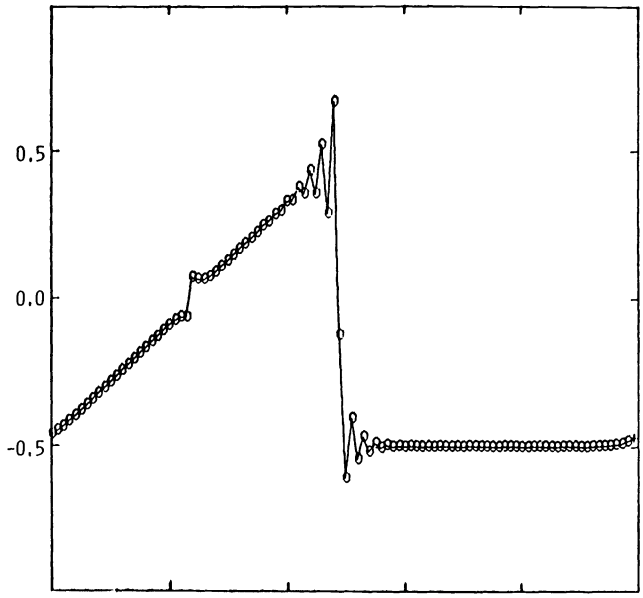


FIG. 3a. *Solution of Burgers' equation using MacCormack scheme.*

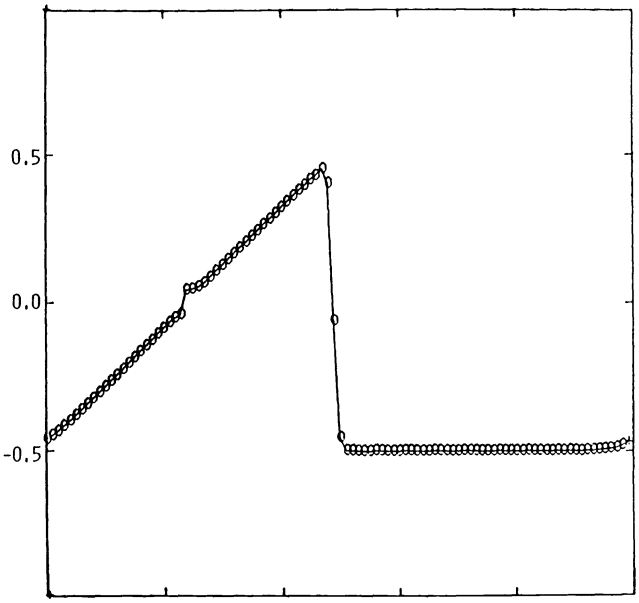


FIG. 3b. *Solution to Burgers' equation using upwind scheme* (3.16), (3.17).

eliminates the oscillations but not the expansion shock while the simplified scheme eliminates all but a small "entropy glitch" in the expansion region. There is no difference between the upwind method and the simplified method in their ability to resolve the shock.

The Burgers' equation results were highly dependent on the choice of initial conditions. Indeed, for certain initial conditions, the simplified method also computed solutions containing expansion shocks. To our knowledge, the only way to avoid

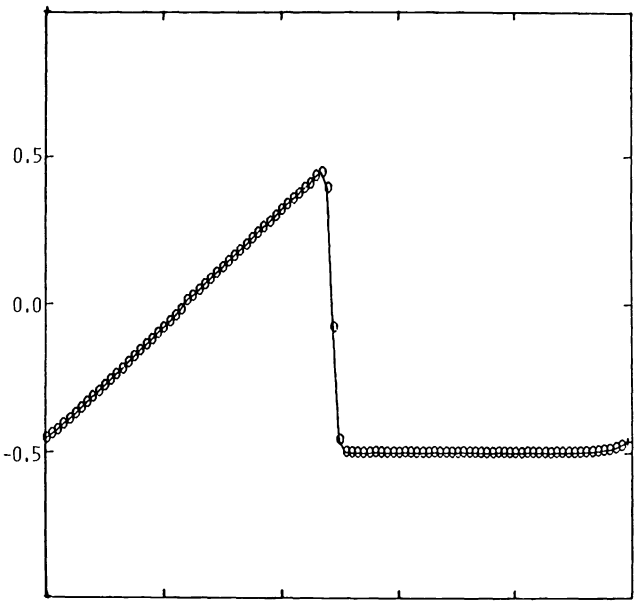


FIG. 3c. *Solution to Burgers' equation using* TVD *scheme* (3.16), (3.18).

expansion shocks in all cases is to explicitly add extra dissipation to the method when a sonic point occurs in an expansion region. Although we cannot guarantee that the simplified scheme will do this automatically, our computations indicate that this scheme is more robust than the unmodified upwind scheme.

4. Hyperbolic systems. In this section we extend the results of the previous section to hyperbolic systems. To that end, we consider first the linear, constant coefficient system

$$u_t + Au_x = 0, \qquad A = \text{const} \tag{4.1}$$

where u is an m vector and A is an $m \times m$ matrix.

If the system (4.1) is hyperbolic, the matrix A has real eigenvalues and a complete set of linearly independent right eigenvectors. If we let P denote the matrix whose columns are the right eigenvectors of A, then

$$P^{-1}AP = \Lambda \tag{4.2}$$

where

$$\Lambda = \begin{bmatrix} \lambda_1 & & & 0 \\ & \lambda_2 & & \\ & & \ddots & \\ 0 & & & \lambda_m \end{bmatrix} \tag{4.3}$$

and λ_j are the eigenvalues of A.

If we define a new set of dependent variables by the formula

$$v = P^{-1}u \tag{4.4}$$

and multiply (4.1) by P^{-1} we obtain

$$(P^{-1}u)_t + P^{-1}AP(P^{-1}u)_x = 0 \tag{4.5}$$

or

$$v_t + \Lambda v_x = 0. \tag{4.6}$$

This is an uncoupled set of scalar equations. We solve (4.6) using (3.16). That is,

$$\begin{aligned} V_k^{n+1} = V_k^n - \frac{\nu}{2}(V_{k+1}^n - V_{k-1}^n) + \frac{\nu^2}{2}(V_{k+1}^n - 2V_k^n + V_{k-1}^n) \\ + [K_{k+1/2}^+(r_k^+) + K_{k+1/2}^-(r_{k+1}^-)](V_{k+1}^n - V_k^n) \\ - [K_{k-1/2}^+(r_{k-1}^+) + K_{k-1/2}^-(r_k^-)](V_k^n - V_{k-1}^n) \end{aligned} \tag{4.7}$$

where the diagonal matrices $\nu = \Lambda \Delta t/\Delta x$ and $K^{\pm}$ and the vectors $r^{\pm}$ are defined below.

Multiply (4.7) by P to obtain an equation in terms of the original dependent variables. The result is

$$\begin{aligned} U_k^{n+1} = U_k^n - A\frac{\Delta t}{2\Delta x}(U_{k+1}^n - U_{k-1}^n) + A^2\frac{\Delta t}{2\Delta x}(U_{k+1}^n - 2U_k^n + U_{k-1}^n) \\ + P[K_{k+1/2}^+(r_k^+) + K_{k+1/2}^-(r_{k+1}^-)]P^{-1}(U_{k+1}^n - U_k^n) \\ - P[K_{k-1/2}^+(r_{k-1}^+) + K_{k-1/2}^-(r_k^-)]P^{-1}(U_k^n - U_{k-1}^n). \end{aligned} \tag{4.8}$$

The first three terms on the right of (4.8) comprise the well-known Lax-Wendroff scheme and need no further discussion. The last two terms require that we know the matrices P and P^{-1}. In the following we construct a simplified version of this scheme which removes this requirement.

We remove the requirement that P and P^{-1} be known by approximating the diagonal matrices $K^{\pm}$ by scalar matrices. That is, we let

$$K^{\pm}(r^{\pm}) \approx \bar{K}^{\pm}(r^{\pm})I \tag{4.9}$$

where the $\bar{K}^{\pm}(r^{\pm})$ are scalar functions of $r^{\pm}$.

We remove the local Courant number dependence by choosing

$$\bar{K}^{\pm}(r^{\pm}) = .5C(\nu)[1 - \phi(r^{\pm})] \tag{4.10}$$

where the Courant number ν is defined as

$$\nu = \max_j |\lambda_j| \frac{\Delta t}{\Delta x} \tag{4.11}$$

and $C(\nu)$ is chosen as follows:

$$C(\nu) = \begin{cases} \nu(1-\nu) & \text{if } \nu \leqq .5, \\ .25 & \text{if } \nu > .5. \end{cases} \tag{4.12}$$

This definition of $C(\nu)$ is an upper bound to the Courant number-dependent coefficient in (3.18).

Thus far we have not defined r^+ and r^-. In light of the fact that we do not wish to compute P or P^{-1}, we have chosen the following definitions for r^+ and r^-:

$$r_k^+ = \frac{(\Delta U_{k-1/2}^n, \Delta U_{k+1/2}^n)}{(\Delta U_{k+1/2}^n, \Delta U_{k+1/2}^n)}, \tag{4.13a}$$

$$r_k^- = \frac{(\Delta U^n_{k-1/2}, \Delta U^n_{k+1/2})}{(\Delta U^n_{k-1/2}, \Delta U^n_{k-1/2})}, \tag{4.13b}$$

where $(\cdot,\cdot)$ denotes the usual inner product on R^m.

If P does not vary significantly over adjacent mesh intervals, these definitions can be interpreted as averages of the scalar definitions. These were the most simple definitions that we could construct. They worked so well in our numerical experiments that we saw no reason to investigate more sophisticated $r^\pm$ definitions. It might be advantageous to look at other definitions since we have not been able to prove rigorously that (4.13) always assures that (4.7) is TVD. Such a proof seems to require that we base $r^\pm$ on the largest and smallest jumps in the characteristic fields. We note in passing that other definitions of $r^\pm$ have been proposed by Sweby [8] and Chakravarthy and Osher [2].

With these simplifications, our numerical method takes the form

$$\begin{aligned} U_k^{n+1} = U_k^n &- A\frac{\Delta t}{2\Delta x}(U^n_{k+1} - U^n_{k-1}) + A^2\frac{\Delta t}{2\Delta x}(U^n_{k+1} - 2U^n_k + U^n_{k-1}) \\ &+ [\bar{K}^+_{k+1/2}(r^+_k) + \bar{K}^-_{k+1/2}(r^-_{k+1})](U^n_{k+1} - U^n_k) \\ &- [\bar{K}^+_{k-1/2}(r^+_{k-1}) + \bar{K}^-_{k-1/2}(r^-_k)](U^n_k - U^n_{k-1}), \end{aligned} \tag{4.14}$$

where $\bar{K}^\pm$ and $r^\pm$ are defined by equations (4.10) and (4.13) respectively. Note that the resulting scheme does not depend explicitly on the transformation (4.4). Therefore, we can use the scheme without modification when the matrix A is not a constant matrix. In a future paper we hope to examine the stability and TVD properties of this scheme rigorously. Here we present only numerical results.

For the computations which follow, we replace the Lax–Wendroff scheme (the first three terms on the right of (4.14)) by the conservative MacCormack scheme. These schemes are equivalent for linear problems.

The resulting algorithm is as follows:

1. For each grid interval $j+\frac{1}{2}$ compute

$$D_{j+1/2} \leftarrow [\bar{K}^+_{j+1/2}(r^+_k) + \bar{K}^-_{j+1/2}(r^-_{k+1})](U^n_{j+1} - U^n_j)$$

and save.

2. Compute a provisional solution at each grid point using some variant of the MacCormack method, e.g.,

$$\begin{aligned} U_j^{(1)} &= U_j^n - \lambda[F(U_j^n) - F(U^n_{j-1})], \\ U_j^{(2)} &= .5*\{U_j^n + U_j^{(1)} - \lambda[F(U^{(1)}_{j+1}) - F(U_j^{(1)})]\} \end{aligned}$$

where $\lambda = \Delta t/\Delta x$. We note here that the computed results did not differ significantly when the other MacCormack variant was used.

3. Add the dissipative terms to the provisional solution, i.e.,

$$U_j^{n+1} = U_j^{(2)} + \{D_{j+1/2} - D_{j-1/2}\}.$$

As a first test, we demonstrate the performance of this method on the Riemann problem. That is, we solve the Euler equations

$$u_t + f(u)_x = 0, \quad -\infty < x < \infty, \quad t > 0 \tag{4.15a}$$

where

$$u=\begin{bmatrix}\rho\\ \rho u\\ E\end{bmatrix},\qquad f(u)=\begin{bmatrix}\rho u\\ \rho u^2+p\\ (E+p)u\end{bmatrix}, \tag{4.15b}$$

$$p=(\gamma-1)(E-\tfrac{1}{2}\rho u^2) \tag{4.15c}$$

and $\gamma=1.4$ with initial conditions

$$u(x,0)=\begin{cases}u_L, & x<0,\\ u_R, & x>0.\end{cases} \tag{4.15d}$$

In this case the initial conditions were

$$u_L=\begin{bmatrix}.445\\ .311\\ 8.928\end{bmatrix},\qquad u_R=\begin{bmatrix}.5\\ 0\\ 1.4275\end{bmatrix}. \tag{4.16}$$

Figure 4 shows the solution after 100 time steps computed on 140 grid points at a Courant number of .95. These conditions are the same as those used by Harten [3] and Chakravarthy and Osher [2] in their numerical experiments. The results shown in Fig. 4 cannot be distinguished from those shown in the cited references with one exception. Harten was able to obtain a dramatic improvement in the resolution of the contact discontinuity by selectively adding artificial compression to his second order upwind scheme. We intend to study this technique in the future.

Initially, we attempted to extend this algorithm to two dimensions in an obvious manner. That is:

1. For each x-direction grid interval compute

$$D_{i+1/2,j}=[\bar{K}^{+}_{i+1/2,j}(r^{+}_{xi,j})+\bar{K}^{-}_{i+1/2,j}(r^{-}_{xi+1,j})](U^{n}_{i+1,j}-U^{n}_{i,j})$$

and save.

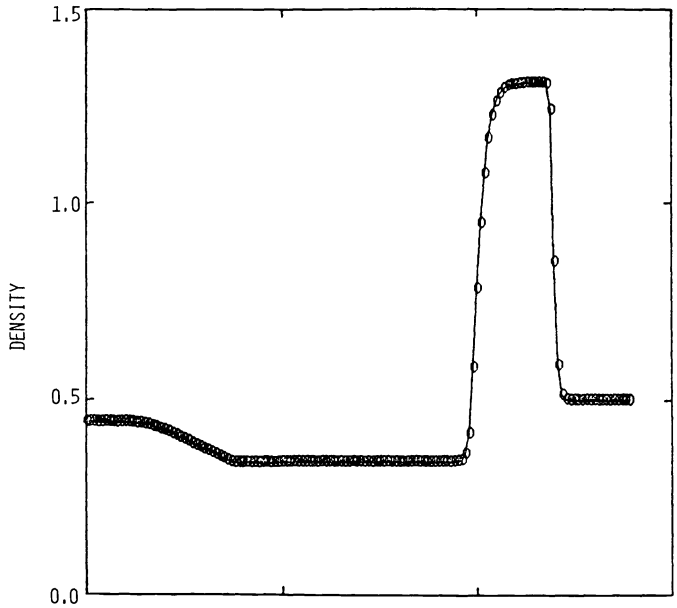


FIG. 4. *Solution to Riemann problem after* 100 *time steps at* $\nu=.95$ *using scheme* (4.14), (4.10).

2. For each y-direction grid interval compute

$$D_{i,j+1/2}=[\bar{K}^{+}_{i,j+1/2}(r^{+}_{yi,j})+\bar{K}^{-}_{i,j+1/2}(r^{-}_{yi,j+1})](U^{n}_{i,j+1}-U_{i,j})$$

and save.

3. Compute a provisional solution at each grid point using some variant of the MacCormack scheme, e.g.,

$$U^{(1)}_{i,j}=U^{n}_{i,j}-\lambda_x[F(U^{n}_{i,j})-F(U^{n}_{i-1,j})]-\lambda_y[G(U^{n}_{i,j+1})-G(U^{n}_{i,j})],$$

$$U^{(2)}_{i,j}=.5*\{U^{n}_{i,j}+U^{(1)}_{i,j}-\lambda_x[F(U^{(1)}_{i+1,j})-F(U^{(1)}_{i,j})]-\lambda_y[G(U^{(1)}_{i,j})-G(U^{(1)}_{i,j-1})]\}.$$

Results of computations made using different MacCormack variants did not differ significantly.

4. Add the dissipative terms to the provisional solutions, i.e.,

$$U^{n+1}_{i,j}=U^{(2)}_{i,j}+[D_{i+1/2,j}-D_{i-1/2,j}+D_{i,j+1/2}-D_{i,j-1/2}].$$

In the case that all of the r's are negative, each of the $\bar{K}$'s can be as large as .125. This would give a dissipative coefficient of .25 which can introduce into the method a weak checkerboard instability which is independent of the stability of the MacCormack method. To avoid this, we follow the suggestion of Roe [6] and combine the two coefficients $\bar{K}^{+}$ and $\bar{K}^{-}$ into a single coefficient $\bar{K}$ where

$$\bar{K}(r^{+}, r^{-})=.5C(\nu)[1-\phi(r^{+}, r^{-})],$$

where $C(\nu)$ is as before and

$$\phi(r^{+}, r^{-})=\max[0, \min(2r^{+}, r^{-}, 1), \min(2r^{-}, r^{+}, 1)].$$

We are currently studying other limiters. This reduces the maximum dissipation coefficient to .125, which would be acceptable in three dimensions as well as two.

Next we demonstrate the performance of our method on a two-dimensional problem. Figure 5 shows a comparison of the present method with the second order upwind scheme of van Leer [9]. To obtain these results, we solve the two-dimensional Euler equations

$$u_t+f(u)_x+g(u)_y=0 \tag{4.17a}$$

where

$$u=\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ E\end{bmatrix},\quad f(u)=\begin{bmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ (E+p)u\end{bmatrix},\quad g(u)=\begin{bmatrix}\rho v\\ \rho uv\\ \rho v^2+p\\ (E+p)v\end{bmatrix}, \tag{4.17b}$$

$$p=(\gamma-1)[E-\tfrac{1}{2}(\rho u^2+\rho v^2)] \tag{4.17c}$$

and $\gamma=1.4$ for the problem of the reflection of an oblique shock from a plane wall.

For the computations shown, we specify a uniform $M=2.9$ flow at the left boundary. At the top boundary, we specify the conditions behind a shock that would turn the flow 11° (cf. [1]). A flow tangency condition is specified at the wall and all variables are extrapolated at the right boundary. The computation is started with the upstream conditions specified everywhere except the top boundary. The results shown consist of a three-dimensional plot of the converged density solution and a longitudinal section of this plot taken at $y=.5$. Once again it is difficult to distinguish between the results computed using the two methods.

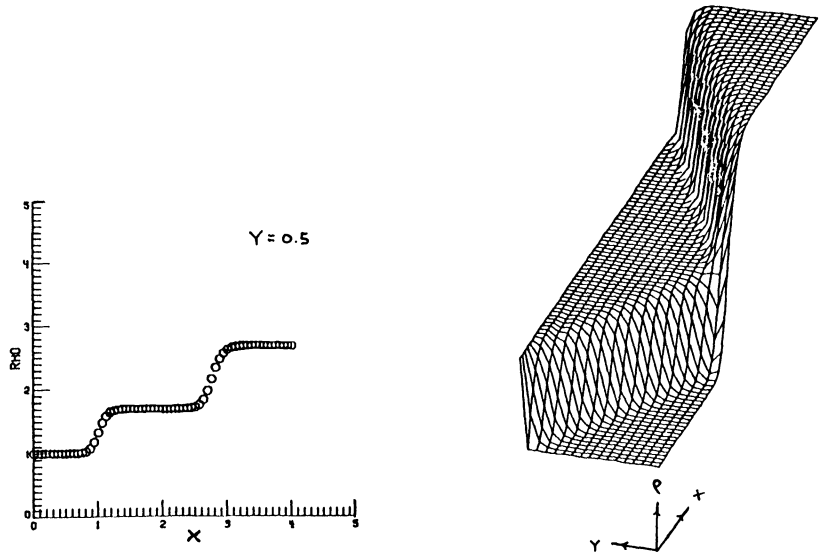


FIG. 5a. *Density profile for shock reflection problem using van Leer method.*

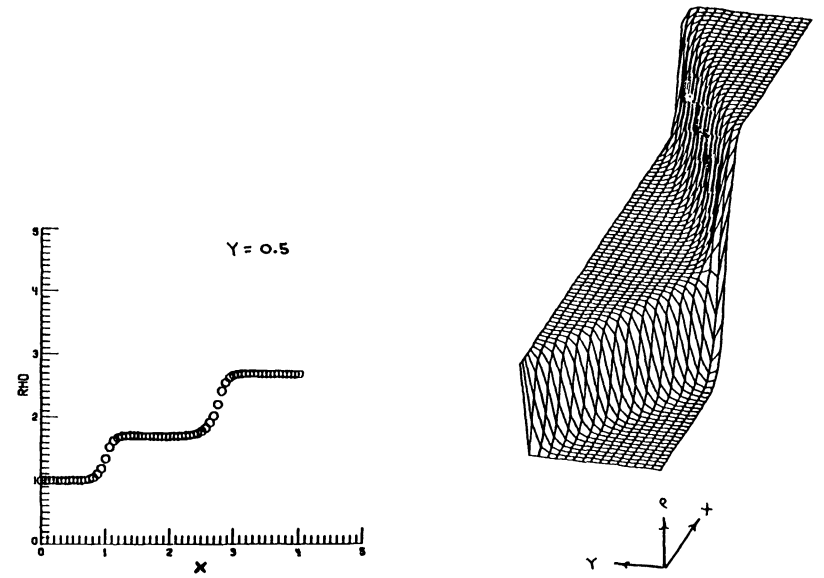


FIG. 5b. *Density profile for shock reflection problem using modified MacCormack scheme.*

Finally, we compare the present method to the second order upwind method of van Leer on a model transonic flow problem.

The two-dimensional Euler equations (4.17) are solved for the flow over a 10% thick parabolic arc bump in a channel. The flow is assumed to be uniform initially and then the condition that the flow be tangent to the bump is applied at the wall in the manner of small disturbance theory. Nonreflecting boundary conditions are applied at both upstream and downstream boundaries. Figures 6a and 6b show the converged pressure distribution on the wall for a flow with inlet Mach number .675 using the van Leer method and the present method, respectively.

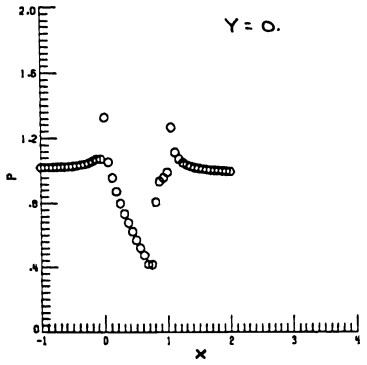


FIG. 6a. *Pressure distribution along wall for transonic problem using van Leer scheme.*

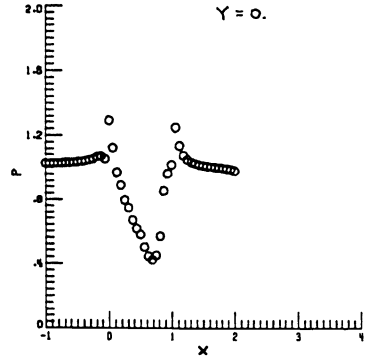


FIG. 6b. *Pressure distribution along wall for transonic problem using modified MacCormack scheme.*

The flow consists of an expansion region over the forward part of the bump, $0 \leqq x \leqq .5$, followed by a shock near $x = .75$. This is the type of problem that the van Leer method was designed to solve since the shock is steady and nearly aligned with the computing grid. Figure 6a shows that the van Leer scheme does an admirable job on this problem. In particular, the flow expands smoothly through the sonic point ($P \approx .717$) and there is only one grid point within the shock. By comparison, Fig. 6b shows that the flow computed using the present method is not quite as smooth in the expansion region and there are two points within the shock. Still, considering the fact that the current method is considerably easier to program and that it runs in $\frac{2}{3}$ the time of the van Leer scheme, the results shown are quite acceptable.

5. Concluding remarks. In this paper we construct a simple artificial viscosity term for Lax-Wendroff type methods which is based on the theory of Total Variation Diminishing upwind finite difference schemes. This method has advantages over both conventional artificial viscosity schemes and the TVD upwind schemes. In particular, the method does not contain the problem dependent parameters of conventional artificial viscosity schemes and it does not require the complex logic of upwind schemes.

The numerical experiments that have been performed thus far have been very encouraging but more numerical experimentation is needed. We intend to carry this out in the future and also to apply the ideas of this paper to other numerical schemes.

Appendix. Here we prove the following theorem.

THEOREM 3.1. *The method* (3.16), (3.18) *with flux limiter* (3.20) *is* TVD *under the restriction that the Courant number* $|\nu|<1$.

Proof. The proof is a direct application of Lemma 2.2. That is, we show that if the scheme is put into the form

$$U_k^{n+1} = U_k^n - C_{k-1/2}\Delta U_{k-1/2}^n + D_{k+1/2}\Delta U_{k+1/2}^n \tag{A.1}$$

then

$$C_{k+1/2} \geqq 0, \tag{A.2}$$

$$D_{k+1/2} \geqq 0, \tag{A.3}$$

$$0 \leqq C_{k+1/2} + D_{k+1/2} < 1. \tag{A.4}$$

Here we prove the result for $\nu > 0$. The computations for $\nu < 0$ are similar.

Rearrange (3.16) as follows

$$U_k^{n+1} = U_k^n - \left[\nu\left\{1+\frac{1}{2}(1-\nu)\left(\frac{1}{r_k^+}-1\right)\right\} - \left\{\frac{K_{k+1/2}^+}{r_k^+} - K_{k-1/2}^+ - K_{k-1/2}^-\right\}\right]\Delta U_{k-1/2}^n + K_{k+1/2}^-\Delta U_{k+1/2}^n. \tag{A.5}$$

That is,

$$C_{k+1/2} = \nu\left\{1+\frac{1}{2}(1-\nu)\left(\frac{1}{r_{k+1}^+}-1\right)\right\} - \left\{\frac{K_{k+3/2}^+}{r_{k+1}^+} - K_{K+1/2}^+ - K_{k+1/2}^-\right\}, \tag{A.6}$$

$$D_{k+1/2} = K_{k+1/2}^-, \tag{A.7}$$

$D_{K+1/2} > 0$ since $K_{k+1/2}^- > 0$.

To show that $C_{k+1/2} > 0$, substitute (3.18) into (A.6) and rearrange terms to obtain

$$C_{K+1/2} = \nu + \frac{\nu(1-\nu)}{2}\left[\frac{\phi(r_{k+1}^+)}{r_{k+1}^+} - \phi(r_k^+) + 1 - \phi(r_{k+1}^-)\right]. \tag{A.8}$$

Note that (3.20) implies

$$0 < \frac{\phi(r)}{r} < 2, \tag{A.9}$$

$$0 < \phi(r) < 1, \tag{A.10}$$

so

$$C_{k+1/2} \geqq \nu - \frac{\nu(1-\nu)}{2} = \frac{\nu}{2} + \frac{\nu^2}{2} > 0.$$

Since $D_{k+1/2} > 0$ and $C_{k+1/2} > 0$, $C_{k+1/2} + D_{k+1/2} > 0$. To prove that

$$C_{k+1/2} + D_{k+1/2} < 1,$$

substitute (3.18) into (A.7) and add to (A.8). The result is

$$C_{k+1/2} + D_{k+1/2} = \nu + \frac{\nu(1-\nu)}{2}\left[\frac{\phi(r_{k+1}^+)}{r_{k+1}^+} - \phi(r_k^+) + 2 - 2\phi(r_{k+1}^-)\right]. \tag{A.11}$$

(A.11) is $\leqq 1$, if the term in the square brackets is $\leqq 2$. To show this, we note that

$$r^-_{k+1} = \frac{1}{r^+_{k+1}} \tag{A.12}$$

and we consider four cases.

Case 1.

$$r^+_{k+1} < 0 \Rightarrow \frac{1}{r^+_{k+1}} < 0 \Rightarrow \phi(r^+_{k+1}) = \phi\left(\frac{1}{r^+_{k+1}}\right) = 0.$$

In this case the term in square brackets becomes

$$2 - \phi(r^+_k) \leqq 2.$$

Case 2.

$$2r^+_{k+1} < 1 \Rightarrow r^+_{k+1} < \frac{1}{2}, \qquad \frac{1}{r^+_{k+1}} > 2,$$

so

$$\phi(r^+_{k+1}) = 2r^+_{k+1}, \qquad \phi\left(\frac{1}{r^+_{k+1}}\right) = 1$$

and the term in square brackets is

$$2 - \phi(r^+_k) \leqq 2.$$

Case 3.

$$\frac{1}{2} < r^+_{k+1}, \qquad \frac{1}{r^+_{k+1}} < 2 \Rightarrow \phi(r^+_{k+1}) = \phi\left(\frac{1}{r^+_{k+1}}\right) = 1.$$

Then the term in square brackets becomes

$$\frac{1}{r^+_{k+1}} - \phi(r^+_k) \leqq 2 - \phi(r^+_k) \leqq 2.$$

Case 4.

$$2 \leqq r^+_{k+1} \Rightarrow \frac{2}{r^+_{k+1}} < 1 \Rightarrow \phi(r^+_{k+1}) = 1, \qquad \phi\left(\frac{1}{r^+_{k+1}}\right) = \frac{2}{r^+_{k+1}}.$$

Then the term in square brackets becomes

$$\frac{1}{r^+_{k+1}} - \phi(r^+_k) + 2 - \frac{4}{r^+_{k+1}} = 2 - \frac{3}{r^+_{k+1}} - \phi(r^+_k) \leqq 2.$$

This completes the proof.

REFERENCES

[1] AMES RESEARCH STAFF, *Equations, tables and charts for compressible flow*, NACA Report 1135, 1953.

[2] S. CHAKRAVARTHY AND S. OSHER, *High resolution applications of the Osher upwind scheme for the Euler equations*, Proc. AIAA Computational Fluid Dynamics Conference, Danvers, MA, 1983, pp. 362-372.

[3] A. HARTEN, *High resolution schemes for hyperbolic conservation laws*, J. Comput. Phys., 49 (1983), pp. 357-393.

[4] A. Y. Le Roux, *A numerical conception of entropy for quasi linear equations*, Math. Comput., 31 (1977), pp. 848-872.

[5] P. L. Roe, *Some contributions to the modelling of discontinuous flows*, Proc. AMS/SIAM Seminar on Large Scale Computations in Fluid Mechanics, San Diego, CA, June 26-July 8, 1983, to appear.

[6] ———, *Generalized formulations of* TVD *Lax-Wendroff schemes*, ICASE Report No. 84-53, NASA CR-172478, Hampton, VA, 1984.

[7] R. Sanders, *On convergence of monotone finite difference schemes with variable space differencing*, Math. Comput., 40 (1983), pp. 91-106.

[8] P. K. Sweby, *High resolution schemes using flux limiters for hyperbolic conservation laws*, SIAM J. Numer. Anal., 21 (1984), pp. 995-1011.

[9] B. van Leer, *Computational methods for ideal compressible flow*, ICASE Report No. 83-38, NASA CR 172180, Hampton, VA, 1983.

[10] R. F. Warming and R. M. Beam, *Upwind second order difference schemes and applications in aerodynamics*, AIAA J., 14 (1976), pp. 1241-1249.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 1, January 1987

© 1987 Society for Industrial and Applied Mathematics

CONFORMAL MAPPING OF CIRCULAR ARC POLYGONS*

PETTER BJØRSTAD† AND ERIC GROSSE‡

Abstract. An algorithm is described which computes the conformal mapping from the unit disk onto an arbitrary polygon having circular arcs as sides. This generalizes the Schwarz-Christoffel program of Trefethen (SIAM J. Sci. Stat. Comp., 1 (1980), pp. 82-102). Our algorithm must also determine certain parameters by solving a nonlinear least squares problem. Instead of using Gauss-Jacobi quadrature to evaluate the Schwarz-Christoffel integral, however, an ordinary differential equation solver is applied to a nonsingular formulation of the Schwarzian differential equation. The construction of a conformal mapping reduces simple elliptic partial differential equations on an irregular region to similar problems on a disk, for which existing programs can compute solutions very efficiently. Typical examples arise in the modeling of conductivity past an array of conducting cylinders and electrical fields inside a waveguide.

Key words. Schwarzian differential equation, Schwarz-Christoffel transformation

AMS(MOS) subject classifications. 65E05, 30C20, 30C30

1. Introduction. We seek the analytic function w mapping the interior of the unit disc D to the simply connected interior of a given polygon C having n circular arcs as sides. See Figure 1. Straight sides are regarded as special cases of circular arcs. We assume for simplicity that 0 is in the interior of C and that C is bounded. By the Riemann mapping theorem, there is a unique function w such that $w(0) = 0$ and $w(1) = w_n$.

One approach to this problem is to apply the Karman-Trefftz transformation repeatedly to remove corners and then apply a general conformal mapping program. Alternatively, it is possible to approximate the arcs by multiple line segments with ordinary Schwarz-Christoffel [1]. But to the best of our knowledge, no previous numerical method has dealt directly with curved polygons.

There exist fast techniques for solving Poisson's equation on regular regions such as the square or disk; an important application of conformal mapping is to convert a problem on an irregular region into an equivalent problem on a regular region. Sometimes the reduced problem is so simple that the solution can be found immediately; for example, the resistance of thin film resistors can be trivially calculated given a conformal mapping from the unit disk [18]. Our own interest in conformal mapping of circular arc polygons was stimulated by a chemistry problem just barely out of the reach of Trefethen's [17] program, which only allows polygons with straight sides; see the first example. Our work consists essentially of a careful numerical implementation of a generalization of the Schwarz-Christoffel mapping due to Schwarz [16] and further developed by others [6], [12]. For related work in conformal mapping see [3], [14] and for a survey of nonclassical applications ranging from ion optics to the drying of apples, see [9], [8]. Recently conformal mapping has been applied to the analysis of protective structures for oil platforms in the North Sea [10].

Let the given polygon C have vertices $w_1, \cdots, w_n$ in counterclockwise order and corresponding interior angles $\alpha_1\pi, \cdots, \alpha_n\pi$. As Figure 2 illustrates, this information

* Received by the editors June 14, 1983, and in revised form March 18, 1986.

† Department of Computer Science, University of Bergen, N-5000 Bergen, Norway. This work was largely done while the author was at the Courant Institute of Mathematical Sciences, New York University, New York, New York, under U. S. Department of Energy contract DE-AC02-76ER03077.

‡AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

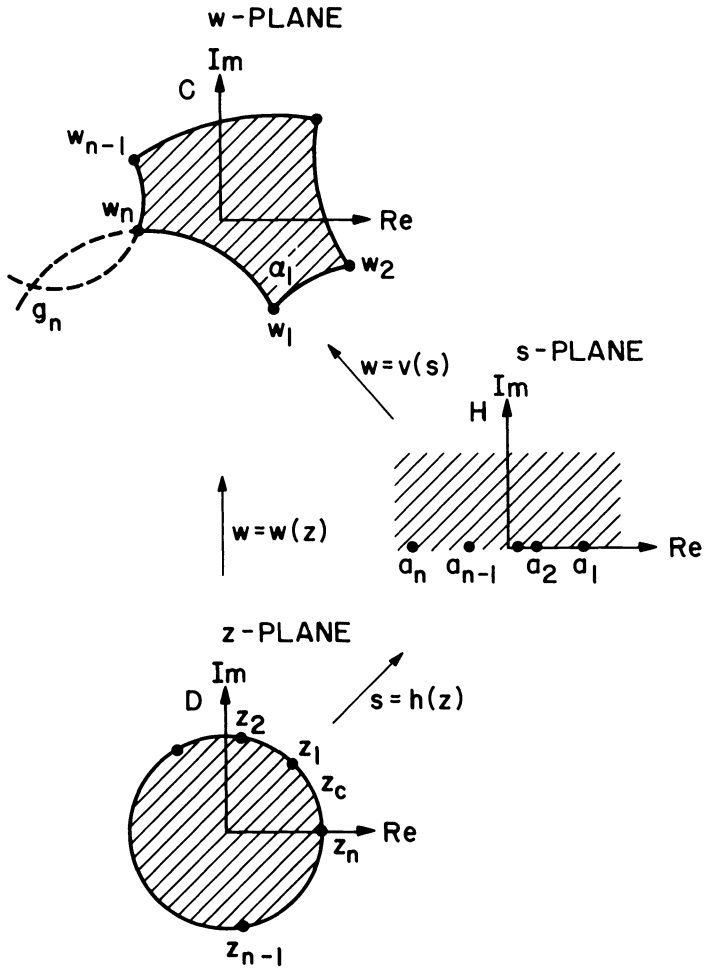


FIG 1. *$w(z)$ is the conformal mapping from the unit disk to a given polygon with circular arcs as sides. For mathematical purposes, it is convenient to decompose the map $w(z) = v(h(z))$ as the composition of a Möbius transformation from the unit disk D to the half plane H and a Schwarz-Christoffel type map from the half plane to the polygon C.*

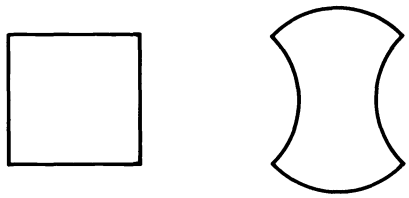


FIG 2. *This square and "squeezed square" both have the same vertices and right angles at the vertices, but are distinct polygons.*

alone does not always determine the polygon uniquely; Section 5 gives various ways in which the polygon can be completely specified.

The polygon is required to be simply connected but otherwise is general. Note that the assumption that the polygon C is bounded is actually no restriction, since a Möbius (or rational linear) transformation carrying infinity to a finite point and some finite exterior point to infinity maps an unbounded circular arc polygon to a bounded one. (The same trick does not apply in the Schwarz-Christoffel case, because such a mapping changes straight boundary segments into curved ones.) In section 6, we illustrate this reduction by mapping a notched quarter plane into a quarter circle. Obviously the assumption that 0 lies in the interior can be removed by a translation.

2. The Schwarzian differential equation. For any differentiable function v, let

$$\boldsymbol{D}v = \frac{v'''}{v'} - \frac{3}{2}\left(\frac{v''}{v'}\right)^2.$$

This is called the *Schwarzian differential operator.* If h is a Möbius transformation, $h(z) = (c_1 z + c_2)/(c_3 z + c_4)$, for complex coefficients $c_1, \cdots, c_4$, then [4, p. 649] for arbitrary v, $u(z) = h(v(z))$ implies $(\boldsymbol{D}u)(z) = (\boldsymbol{D}v)(z)$, and $w(z) = v(h(z))$ implies $(\boldsymbol{D}w)(z) = (h'(z))^2(\boldsymbol{D}v)(h(z))$. Möbius transformations arise naturally in our problem because the image of a circular arc polygon under such a transformation is again a circular arc polygon. In constructing w, it is easiest to start with the conformal mapping v from the upper half plane H to C.

THEOREM 1. *Let v map H conformally to C. Then there exist real numbers $a_1, a_2, \cdots, a_n, \beta_1, \beta_2, \cdots, \beta_n$ such that*

* *v satisfies the differential equation $\boldsymbol{D}v(s) = p(s)$ with right-hand side*

$$p(s) = \sum_{j=1}^{n}\left(\frac{1-\alpha_j^2}{2(s-a_j)^2} + \frac{\beta_j}{s-a_j}\right),$$

* *the a_j are preimages of the vertices of the polygon:*

$$a_1 < a_2 < \cdots < a_n,$$

$$v(a_j) = w_j, \qquad 1 \leqslant j \leqslant n,$$

* *and certain algebraic side-conditions*

$$\sum_{j=1}^{n}\beta_j = 0,$$

$$\sum_{j=1}^{n}[2\beta_j a_j + 1 - \alpha_j^2] = 0,$$

$$\sum_{j=1}^{n}[\beta_j a_j^2 + (1-\alpha_j^2)a_j] = 0$$

are also met.

Proof. See [4, Thm. D.2.1]. □

This theorem can also be stated using a linear differential equation to define functions r and s such that $v = r/s$, but we have not found a way to take computational advantage of this, and therefore have preferred to use the most direct formulation.

3. Computing the conformal mapping. If the parameters $a_1, a_2, \cdots, a_n$, $\beta_1, \beta_2, \cdots, \beta_n$ are chosen to satisfy the algebraic side-conditions of the theorem and the differential equation is integrated to compute a function v, then it will be found that v maps H conformally onto some circular arc polygon, with vertices in arbitrary places but always with correct interior angles $\alpha_j \pi$. The parameter a_j has a clear geometric meaning as a vertex preimage, but the parameter β_j is somewhat more obscure. Besides being the residue of $\boldsymbol{D}v$ at a_j, the parameters β_j can be shown [12, p. 200] to satisfy

$$\beta_j = \frac{1-\alpha_j^2}{\alpha_j} \frac{f'(a_j)}{f(a_j)}$$

for the function

$$f(s) = \frac{v(s)-w_j}{(s-a_j)^{\alpha_j}},$$

which is regular at a_j. For some parameter choices, the mapping may no longer be globally one-to-one; some of the sides of the polygon will intersect, and the polygon will overlap itself. There are theoretical results describing when such "nicht schlicht" polygons can arise [11], but in practice we have not encountered such difficulties.

To carry the theorem on the half plane back to the unit disk, compose v with a Möbius transformation,

$$h(z) = i\frac{z_c+z}{z_c-z},$$

of the unit disk D onto H to get the desired $w(z) = v(h(z))$. (The cut here is chosen arbitrarily with $|| z_c || = 1$.) By the special chain rule for $\boldsymbol{D}$, the Schwarzian differential equation for w is:

$$w'''(z) = \frac{3}{2}\frac{w''(z)^2}{w'(z)} + p(h(z))h'(z)^2 w'(z).$$

We next specialize this to a form suitable for numerical integration. Replace z here by a general point $r\zeta$ on a radial line from 0 to a point ζ on the boundary of the unit disk D, reduce to a first order system in the customary fashion and equate real and imaginary parts. This produces a six-component ordinary differential equation describing the image of a radial line under w. Let $a_j = h(z_j)$. For any particular choice of parameters $z_1, z_2, \cdots, z_n, \beta_1, \beta_2, \cdots, \beta_n$, integrate along n radial lines from the origin to a point $\hat{z}_j$ on the boundary of the unit disk midway between z_j and z_{j+1}. From the endpoint data, w, w', w'' at $\hat{z}_j$, and elementary tools of analytic geometry described in Section 5, the vertices and sides of the polygon can finally be found. We use Shampine and Gordon's ODE routine to perform the integration.

Our choice of ζ avoids the exact singularity that would occur if, as in Trefethen's program, the integration extended to the points z_j. It may be possible to use some special scheme for integrating the differential equation given the known form of the singularity, just as Trefethen uses a Gauss-Jacobi rule for his integration, but we have not investigated this in detail. A near singularity is inevitable when z_j and z_{j+1} are close, since then ζ is near a singular point.

Perhaps the trickiest part of this integration is setting up the initial conditions, which are intimately related to normalization of the polygon. Because of the invariance of the Schwarzian under a Möbius transformation, for a fixed choice of parame-

ters there is a family of polygons with six real degrees of freedom; each polygon is the conformal image of the unit disk according to the differential equation and each pair of polygons can be mapped to one another by a Möbius transformation. By analogy with Trefethen's program, we set $z_n = 1$ and insist that $w(0)=0$. We can now select a unique member from the family of polygons by requiring that

* $w(0) = 0$;
* $w(z_n) = w_n$; and
* side 1 lies in the correct circle.

This last condition says that the image of the arc between z_n and z_1 lies on the same circle as side 1 of the given polygon C. We impose these conditions as follows. First, arbitrarily choose initial conditions for the differential equation. Solve for the image polygon. Find the Möbius transformation which, composed with the mapping from the differential equation, satisfies the three conditions; that is, shift the polygon into its normalized position. Use the derivatives of order 0, 1, and 2 of the resulting composite conformal map to determine what the initial conditions ought to have been to produce that mapping without any postnormalization. These initial conditions are used for the next integration (which presumably has slightly perturbed parameters a_j and β_j) so that only a small normalization will be required. We have found that this decreases the cost of the later integrations by as much as a factor of 4.

Constructing the normalizing Möbius transformation can be done quickly by decomposing it into four parts. First, by a scaling and rotation, move vertex n of the computed polygon onto 1. Second, find a Möbius transformation mapping side 1 into the unit circle while holding 0 fixed. Next, determine the analogous two mappings for the original polygon; invert these two to get mappings three and four.

Thus the core problem can be formulated as: given a circular arc Γ passing through 1, determine coefficients a, b, c, and d in $u(w) = (aw+b)/(cw+d)$ such that

$$u(0) = 0;$$

$$u(1) = 1;$$

$$|u(\Gamma)| = 1.$$

(This last equation means that the image of Γ under u lies on the unit circle.) Clearly $b = 0$. If Γ already lies on the unit circle, then $c = 0$ and we are done. Otherwise set $c = 1$, so $d = a-1$. Let $a=x+iy$ and let $w=r+is$ be an arbitrary point on Γ. Substituting into $|u(w)|=1$ and squaring gives

$$(x^2+y^2)(r^2+s^2-1)-2x(r-1)-2ys-(r-1)^2-s^2 = 0.$$

This defines a circle in the (x,y) plane passing through $(1,0)$. By forcing two points w_1 and w_2 on Γ to map to the unit circle, it is possible to solve explicitly for a as the point (other than $(1,0)$) where the two circles intersect. For w_i let the circle have center x_i,y_i. If $|w_1|\neq 1$ and $|w_2|\neq 1$ then the intersection point is at

$$x = \frac{(x_1-x_2)^2+(2x_1-1)(y_1-y_2)^2-2y_1(x_2-x_1)(y_2-y_1)}{(x_1-x_2)^2+(y_1-y_2)^2},$$

$$y = \frac{2(x_1-x_2)[y_1(x_1-x_2)+(x_1-1)(y_2-y_1)]}{(x_1-x_2)^2+(y_1-y_2)^2}.$$

If instead $|w_2|=1$, then the "circle" defined by w_2 is actually a line $y=x_2(1-x)$

where $x_2=(r_2-1)/s_2$. The intersection point is

$$x = \frac{x_2(x_2-2y_1)+2x_1-1}{1+x_2^2},$$

$$y = 2x_2\frac{y_1x_2-x_1+1}{1+x_2^2}.$$

By hypothesis, Γ did not lie on the unit circle, so it is not possible for both w_1 and w_2 to both have absolute value 1.

Computationally, the difficult part of the overall problem is finding the parameters $z_1,z_2,\cdots,z_n,\beta_1,\beta_2,\cdots,\beta_n$. By our choice of normalization $z_n = 1$. In order to satisfy the side conditions relating the β_j, z_j, and α_j, we solve for $\beta_{n-2},\beta_{n-1},\beta_n$ in terms of the other parameters by solving a linear system. Also, we make the same transformation of variables as Trefethen to remove constraints on the z_j by defining

$$z_j = e^{i\theta_j}, \qquad 0\leqslant\theta_j<2\pi,$$

$$x_j = \log\frac{\theta_j-\theta_{j-1}}{\theta_{j+1}-\theta_j},$$

where $\theta_0=0$. This leaves us with $2n-4$ real unknowns $x_1,x_2,\cdots,x_{n-1},\beta_1,\beta_2,\cdots,\beta_{n-3}$. To determine these, we call Gay's nonlinear least squares routine N2F, formerly known as NL2SOL [2], applied to the $2n-2$ real equations

$$w_j-\tilde{w}_j = 0, \quad 1\leqslant j\leqslant n-1,$$

where w_j is a vertex in the given polygon, and $\tilde{w}_j$ is a vertex in the image polygon for the current values of the parameters. Earlier versions of our program used other optimization routines with somewhat less success. One helpful feature of N2F is that a particular function evaluation can fail without aborting the entire optimization.

4. Efficient evaluation using Taylor series. We have focused so far on determining the parameters of the conformal mapping. Once these are known, the mapping can be evaluated at any point at the cost of solving an ordinary differential equation. A much faster method is to use Taylor polynomials at the center of the disk and at the vertices, obtaining the coefficients from the recurrence formulas derived below. For points not within the radius of convergence of these expansions, the differential equation is used.

We focus on polygons with straight sides, since the resulting formulas are simpler. In this case the mapping is the standard Schwarz-Christoffel transformation:

$$w(z) = c\int_0^z \prod_{k=1}^{n}\left(1-\frac{u}{z_k}\right)^{\alpha_k-1} du.$$

Consider the functions, indexed by $0\leqslant j\leqslant n$,

$$D_j^s(z) = \sum_{\substack{1\leqslant k\leqslant n\\ k\neq j}} (\alpha_k-1)(z_k-z)^{-s}$$

and

$$G_j^{(1)}(z) = c \prod_{\substack{1 \le k \le n \\ k \ne j}} \left(1 - \frac{z}{z_k}\right)^{\alpha_k - 1}.$$

(The superscript on the D indicates a power of $z_k - z$, while the superscript on the G is chosen so that $G_0^{(m)}(z_0) = w^{(m)}(z_0)$ at regular points.) It is easy to prove by induction that successive derivatives of $G_j^{(1)}$ are given by

$$G_j^{(r)}(z) = \sum_{s=1}^{r-1} \frac{(r-2)!}{(s-1)!} D_j^{r-s}(z)\, G_j^{(s)}(z)$$

for $r = 2, 3, 4, \cdots$.

We seek an expansion for w about a point z_0. If z_0 is a regular point then $w(z) = \sum_{m=0}^{\infty} k_m (z - z_0)^m$ with coefficients

$$k_0 = w(z_0),$$

$$k_1 = c \prod_{k=1}^{n} \left(1 - \frac{z_0}{z_k}\right)^{\alpha_k - 1},$$

$$k_m = \sum_{s=1}^{m-1} \frac{s}{m(m-1)} D_0^{m-s}(z_0)\, k_s, \qquad m = 2, 3, 4, \cdots.$$

If instead $z_0 = z_j$, then in a neighborhood of z_j,

$$w(z) - w(z_j) = \int_{z_j}^{z} \left(1 - \frac{u}{z_j}\right)^{\alpha_j - 1} G_j^{(1)}(u)\, du$$

$$= (-z_j)^{1-\alpha_j} \sum_{m=1}^{\infty} \frac{G_j^{(m)}(z_j)}{(m-1)!(m-1+\alpha_j)} (z - z_j)^{m-1+\alpha_j}.$$

Thus

$$w(z) = w(z_j) + \sum_{m=1}^{\infty} k_m \left(\frac{z_j}{z_j - z}\right)^{1-\alpha_j} (z - z_j)^m$$

with coefficients

$$k_m = \frac{G_j^{(m)}(z_j)}{(m-1)!(m-1+\alpha_j)} \qquad m = 1, \cdots.$$

The recurrence for the derivative of G_j yields

$$k_m = \sum_{s=1}^{m-1} \frac{(s-1+\alpha_j)}{(m-1)(m-1+\alpha_j)} D_j^{m-s}(z_j)\, k_s, \qquad m = 2, \cdots.$$

The generalization to circular arc sides is more complicated. Solving the differential equation immediately gives the first few derivatives of w at z, and using a symbolic arithmetic package it is an easy exercise to calculate the higher derivatives. The resulting Fortran code for, say, the first ten derivatives is rather ugly, but not expensive to evaluate.

5. Geometric tools. Since the difficult numerical aspects of this problem (solving differential equations and nonlinear equations) are already treated by library software,

much of the programming effort in this project went into representing circular arc polygons and building low-level tools for handling arcs and Möbius transformations.

As in the special case of straight sides, mathematical presentations of the general Schwarz-Christoffel transformation typically give the locations of the vertices w_j, in counter-clockwise order around C as j ranges from 1 to n, and the sizes of the adjacent interior angles $\alpha_j\pi$, assumed to satisfy $0<\alpha_j<2$. But when n is even, this does not uniquely characterize a polygon. For example, Figure 2 shows different "squares" with identical vertices and interior angles. Let *side j* mean the arc between w_j and w_{j-1}. The extra information needed for a unique specification may be given as:

* the angle between the real axis and the tangent at w_1 to the side 2, or
* a third point on side 2, or
* the so-called *gegenecke*, g_n, the second intersection point (perhaps at infinity) between the (possibly extended) sides n and 1. (In Figure 1, the gegenecke g_n is at the intersection of the dashed arcs.)

To formulate the parameter optimization problem, we rely on the following. It is surely not the unique best characterization, but it is reasonably scaled and seems to work in practice at least as well as the half dozen other possibilities we tried.

CHARACTERIZATION 1. *C is uniquely determined by*

$$w_1, w_2, \cdots, w_n,$$

$$\alpha_1, \cdots, \alpha_n,$$

and the circular ray containing side 1. *(By circular ray, we mean a circle and an orientation.)*

Proof. Start from w_n and move along the given ray to w_1. For $j = 2, \cdots, n$, find the arc containing side j by using side $j-1$, vertex w_j, and the angle $\alpha_j\pi$ of intersection. □

We ask the user to initially provide a different set of information more compatible with engineering practice. Other items of information about the polygon needed for the computation, such as the interior angles α_j, are then derived automatically.

CHARACTERIZATION 2. *C is uniquely determined by*

$$w_1, \cdots, w_n,$$

$$r_1, \cdots, r_n,$$

$$d_1, \cdots, d_n,$$

where d_j is 1 *if side j is convex (bulging outward) and* -1 *if concave.*

Proof. An arc is completely determined by its endpoints, radius, and orientation, so the polygon is fully determined. □

Next, let us consider how to locate vertices of a circular arc polygon given its sides, that is, to locate the one or two points of intersection between two circles or lines. For each circular arc making up the side j between vertex w_{j-1} and w_j, the data supplied by integration to a midpoint consists of:

* m_j, a point on the circular arc making up the side;
* n_j, an outward-pointing unit vector; and
* r_j, the radius of curvature.

From this data it is possible to locate the center immediately:

$$c_j = m_j - r_j n_j.$$

(If r_j is infinite, we will not use c_j.) Actually, the integration gives only derivatives (of order 0, 1, and 2). But n_j can be computed directly from the first derivative and r_j from the second by standard formulas of analytic geometry [7, problem 8.5.15]. We assume that the sides intersect in exactly one or two points. This condition would be violated if the integration provided data so inaccurate that two sides did not intersect at all, or in certain degenerate cases involving interior angles of π. We also assume, without loss of generality, that $r_j \leq r_{j-1}$. The intersection point $w = u+iv$ of two lines can be found by solving the two-by-two real linear system in u and v

$$\mathrm{Re}(\bar{n}_{j-1}w) = \mathrm{Re}(\bar{n}_{j-1}m_{j-1}), \quad \mathrm{Re}(\bar{n}_j w) = \mathrm{Re}(\bar{n}_j m_j)$$

where a bar indicates complex conjugation.

Next consider the two intersections points of a line and a circle. Without loss of generality assume that the line is parallel to the imaginary axis. Let u and v be the real and the absolute value of the imaginary parts of the intersection points. Then

$$u = \mathrm{Re}(\bar{n}_{j-1}(m_{j-1}-c_{j-1})), \quad v = \sqrt{r_j^2-u^2}$$

and the intersection points are given by

$$w_1 = c_j+wn_{j-1}, \quad w_2 = c_j+\bar{w}n_{j-1}.$$

Finally, consider the intersection of two circles. Once again the equations are simplified by choosing u and v as real and imaginary parts, after rotating so that the circle centers are on the real axis. Let $e = |c_{j-1}-c_j|$. Then

$$u^2+v^2 = r_j^2, \quad (u-e)^2+v^2 = r_{j-1}^2.$$

Subtracting these equations and solving gives

$$u = \frac{e^2-r_{j-1}^2+r_j^2}{2e}, \quad v = \sqrt{r_j^2-u^2}.$$

The intersection points are

$$w_1 = c_j+w\frac{c_{j-1}-c_j}{e}, \quad w_2 = c_j+\bar{w}\frac{c_{j-1}-c_j}{e}.$$

Here we have implicitly assumed that the circles are arranged so that $e \geq r_{j-1}$. Otherwise replace e by $-e$ in the equations above.

Unfortunately this calculation is inherently inaccurate when r_{j-1} is finite but large, since the absolute error in c_{j-1} is then large. In such a case we use a local parameterization of an arbitrary point w on the side, expanding about the given point m_j in arclength s: $w(s) = m_j+n_j r_j(\exp(is/r_j)-1)$. A Taylor series expansion is used to accurately compute the parenthesized exponential expression for small s/r_j, using the EXPREL routine from Wayne Fullerton's FNLIB. Setting this formula for $w(s)$ equal to the corresponding one about m_{j-1} gives one nonlinear complex equation in two real unknowns, which may be solved by a simple Newton iteration using the inaccurate explicit method to provide an initial guess.

If circle C with center c_3 is the image of a circle with center c_0 and radius r under a Möbius transformation $w = (az+b)/(cz+d)$, then the center of C can be found by decomposing the transformation into translations, a rotation, and an inversion:

$$\frac{az+b}{cz+d} = \frac{a}{c}+\frac{bc-ad}{c^2}\frac{1}{z+d/c}.$$

Only inversion maps centers in a nontrivial way.

Let c_1 be the center and r the radius of the circle before the inversion and c_2 the center afterwards. Then

$$c_2 = \frac{1}{2}\left(\frac{1}{b_1}+\frac{1}{b_2}\right)$$

where

$$b_1 = \left(1-\frac{r}{|c_1|}\right)c_1, \quad b_2 = \left(1+\frac{r}{|c_1|}\right)c_1.$$

The center c_3 is given by:

$$c_1 = c_0+\frac{d}{c}, \qquad c_3 = \begin{cases} \text{if } c_1 = 0 \text{ then } a/c \\ \text{else if} |c_1| = r \text{ then } \infty \\ \text{else } \dfrac{a}{c}+\dfrac{bc-ad}{c^2}c_2. \end{cases}$$

6. Numerical examples and discussion. Figures 2, 3, 4, and 5 show regions onto which our program has mapped the unit disk, starting with an initial guess of 0 for all parameters.

A model problem representative of the intended applications of our program was introduced by Lord Rayleigh [13] and further studied by Keller and Sachs [5]: "Let an infinite medium with dc conductivity σ_0 contain a uniform square array of perfectly conducting parallel circular cylinders." By symmetry, it suffices to consider the two-dimensional region cross-hatched in Figure 3. After construction of the conformal mapping, the problem can be reduced to solving Laplace's equation on a rectangle, for which fast algorithms already exist.

Another, more challenging, region is pictured in Figure 4. The region to be conformally mapped is a symmetric quarter of a circular waveguide with rectangular fins.

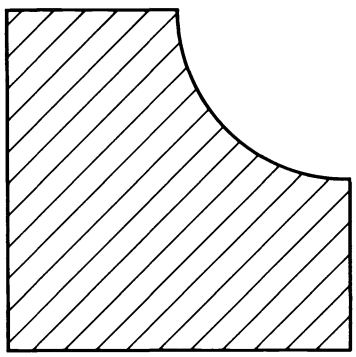


FIG 3. *This notched square is the region left in the analysis of conductivity around a uniform array of cylinders after removing symmetries.*

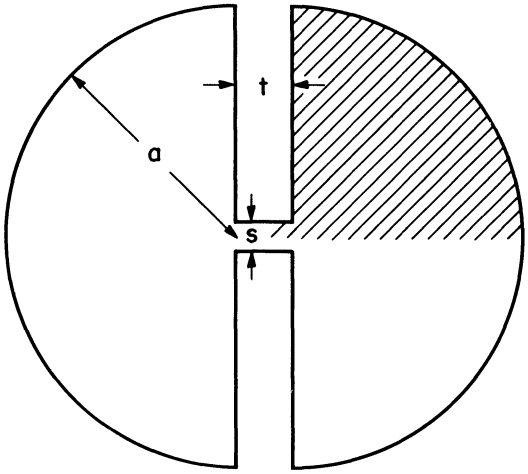


FIG 4. *The shaded region in this figure indicates the asymmetric part of a circular fin waveguide, with critical dimensions indicated. Typical values are* $a=1$, $s=1/64$, *and* $t=1/16$.

When Ryder and Sanderson [15] tried solving Laplace's equation on this region, they found the problem too ill-conditioned. This also caused problems for our program; we were able to converge from a 0 starting guess for $a=1$, $s=1$, $t=1$, but for the realistic values of $s=1/16$, $t=1/64$, we had to use a sequence of target polygons to converge to the final answer.

To illustrate the ability to deal with unbounded regions, we finally present the quarter plane with a notch cut out of the corner, as indicated in Figure 5. By performing an inversion with respect to the unit circle, this region is mapped to a quarter circle, which can be handled directly by our program. (Of course, this example can also be done analytically. It is included to illustrate how to deal with unbounded domains.)

We indicate the order of magnitude expense for such computations in Table 1. Runtimes (in seconds) include plotting and diagnostics; the code has not been optimized, so presumably the times can be improved. In all cases, the starting guess $x=0$ is used and error tolerances are as described later. All computations were done on (one processor) of a Cray XMP using the CFT1.14 compiler.

TABLE 1

problem	iterations	func+grad	runtime
square	11	69	5.7
squeezed square	12	74	5.6
notch	15	118	11.
quarter circle	4	19	1.
quarter guide 1,1	36	268	23.3

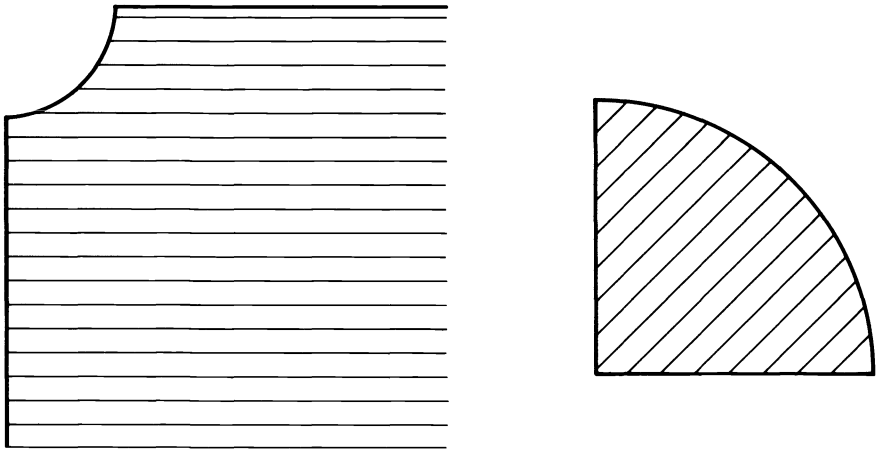


FIG 5. *This notched quarter plane illustrates the technique of dealing with unbounded polygons. The transformation $u=1/w$ maps the notched quarter plane into a (finite) quarter circle.*

For those who may be interested in the resistances (or conformal moduli) $R_{i,j}$ between sides i and j, or who wish to reproduce our computations, we now quote results of these examples. Here $\tau = \arg z$. We do not claim all digits printed are uncontaminated by rounding error.

square: $w_n=.5,-.5$, $\beta_1=3.543593E-1$, $\beta_2=1.226968E-1$, $\beta_3=-3.543593E-1$, $\beta_4=-1.226968E-1$, $\tau_1=1.570796$, $\tau_2=31.415927$, $\tau_3=47.123890$, $R_{1,3}=1.00000000$,

squeezed square: $w_n=.5,-.5$, $\beta_1=3.857471E-1$, $\beta_2=1.016990E-1$, $\beta_3=-3.624988E-1$, $\beta_4=-1.249473E-1$, $\tau_1=1.757134$, $\tau_2=31.415927$, $\tau_3=48.987266$, $R_{1,3}=1.08920317$,

notched square: $w_n=.5,0$, $\beta_1=1.343139$, $\beta_2=-6.599197E-1$, $\beta_3=-3.617259E-1$, $\beta_4=-1.635336E-1$, $\beta_5=-1.579598E-1$, $\tau_1=3.274522$, $\tau_2=35.887998$, $\tau_3=47.788539$, $\tau_4=59.689080$, $R_{1,3}=1.48892070$, $R_{1,4}=2.04311771$, $R_{2,4}=1.48892070$,

quarter circle: $w_n=1,0$, $\beta_1=4.215185E-1$, $\beta_2=-2.950210E-1$, $\beta_3=-1.264975E-1$, $\tau_1=2.477624$, $\tau_2=43.804044$,

waveguide with $s=1$, $t=1$: $w_n=1.5,0$, $\beta_1=2.489092E-2$, $\beta_2=-2.438678E-1$, $\beta_3=1.451395$, $\beta_4=-1.143379$, $\beta_5=-8.903888E-2$, $\tau_1=0.409942$, $\tau_2=10.472788$, $\tau_3=26.805252$, $\tau_4=30.298488$, $R_{1,3}=1.70930749$, $R_{1,4}=8.71027731E-1$, $R_{2,4}=6.34051648E-1$.

Figure 6 shows the dependence of the conformal mapping on the parameters by plotting the computed image polygon on top of the given polygon for each combination of five values of β_1 and of β_2. Since this region has five sides, β_3, β_4, and β_5 are determined by the algebraic side conditions. The influence of the four x_j parameters has been suppressed in the figure by leaving them set to their exact values. From the figure it can be seen that β_1 and β_2 are almost linearly dependent. Note that when $\beta_1+\beta_2$ is held constant, there is little change in the polygon. This dependence manifests itself in a large condition number of the Jacobian of the nonlinear equations. (Expanding the plot to cover more of parameter space demonstrates that β_1 and β_2 are not exactly linearly dependent.)

Figure 7 traces the solutions of the Schwarzian differential equation. Each line is the image of one of the radial midpoint rays in the unit disk.

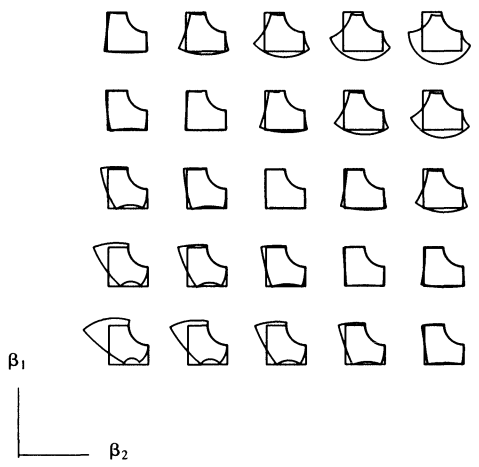


FIG 6. *This array of computed polygons superimposed on the given polygon in the notched square example illustrates the dependence of the conformal mapping on two of the parameters, as β_1 and β_2 are varied about their correct values.*

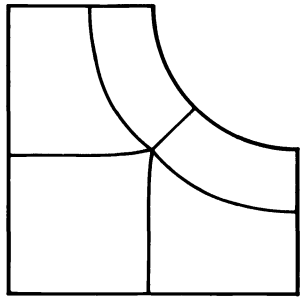


FIG 7. *This figure traces the solutions of the Schwarzian differential equation. Each line is the image of one of the radial midpoint rays in the unit disk.*

We regard our program as experimental. However, like SCPACK, it is available by electronic mail for others who wish to pursue circular arc polygons. Send the electronic mail message `send index for conformal` to `research!netlib` or `netlib@anl-mcs` or contact the authors. One of the defects of the current code is the restriction $\alpha_i \neq k\pi$. This is inessential, but removing it will require some change to the strategy of determining curvatures at the midpoints of sides and then computing vertices as intersections. For $\alpha_i = \pi$, a special case solution setting $\beta_i = 0$ and integrating directly to the vertex could be tried. Another rough edge in need of polishing is the choice of error tolerances. At present, computing on a machine with 14 decimal digits, we ask ODE for 11 digits, and get a squared residual from N2F of about 10^{-20}; this gave a maximum vertex error of 1.2E-9. Probably it would pay to use cruder tolerances for ODE until the final iterations.

Two areas we feel deserve future attention are: better normalization and better parameters. Sometimes we have seen that forcing side 1 of the computed polygon to lie on side 1 of the target leads to a rather distorted polygon, and another normalization would have given a much better fit. We were not able to produce any scheme for optimizing the normalization that did as well in general as the side 1 to side 1 criterion, however. Figure 6 shows that there are inherent conditioning problems in using β_i. Can some alternative, more independent, parameters be devised?

Acknowledgment. We thank Martin Gutknecht and Nick Trefethen for their thoughtful comments on the formulation of the nonlinear system and organization of the paper.

REFERENCES

[1] R. T. DAVIS, *Numerical methods for coordinate generation based on Schwarz-Christoffel transformation,* AIAA Paper 79-1463, 4th Computational Fluid Dynamics Conference, 1979.

[2] J. E. DENNIS, D. M. GAY, AND R. E. WELSCH, *An adaptive nonlinear least-squares algorithm,* ACM Trans. Math. Software 7 (1981), pp. 348-368.

[3] D. GAIER, *Konstruktive Methoden der konformen Abbildung,* Springer, Berlin 1964.

[4] E. HILLE, *Lectures on ordinary differential equations,* Addison-Wesley, Reading, MA 1969.

[5] H. B. KELLER AND D. SACHS, *Calculations of the conductivity of a medium containing cylindrical inclusions,* J. Appl. Phys. 35 (1964), pp. 537-538.

[6] W. KOPPENFELS AND F. STALLMANN, *Praxis der konformen Abbildung,* Springer, Berlin 1959.

[7] E. KREYSZIG, *Advanced Engineering Mathematics, 4th ed.,* John Wiley, New York 1979.

[8] J. R. KUTTLER AND V. G. SIGILLITO, *Eigenvalues of the Laplacian in two dimensions,* SIAM Rev. 26 (1984), pp. 163-193.

[9] P. A. A. LAURA, *A survey of modern applications of the method of conformal mapping,* Revista de la Union Matematica Argentina 27 (1975), pp. 167-179.

[10] T. MARTHINSEN AND T. VINJE, *Nonlinear hydrodynamic interaction in offshore loading systems,* Behaviour of Offshore Structures (1985), pp. 293-302, Elsevier, Amsterdam.

[11] Z. NEHARI, *The schwarzian derivative and schlicht functions,* Bull. Amer. Math. Soc. 55 (1949), pp. 545-551.

[12] Z. NEHARI, *Conformal mapping,* McGraw, Dover 1952, 1975.

[13] LORD RAYLEIGH, *On the influence of obstacles arranged in rectangular order upon the properties of a medium,* Philosophical Magazine 34 (1892), pp. 481-502.

[14] K. REPPE, *Berechnung von Magnetfeldern mit Hilfe der konformen Abbildung durch numerische Integration der Abbildungsfunktion von Schwarz-Christoffel,* Siemens Forsch.- u. Entwickl.-Ber. 8 (1979), pp. 190-195.

[15] B. G. RYDER AND J. G. SANDERSON, *A program for computing the eigenvalues of Laplace's equation,* Bell Labs Computing Science Technical Report 20, 1974.

[16] H. A. SCHWARZ, *Uber einige Abbildungsaufgaben,* J. reine u. angew. Math. 70 (1869), pp. 105-120.

[17] L. N. TREFETHEN, *Numerical computation of the Schwarz-Christoffel transformation,* this Journal 1 (1980), pp. 82-102.

[18] L. N. TREFETHEN, *Analysis and design of polygonal resistors by conformal mapping,* J. Appl. Math. Phys. (ZAMP) 35 (1984), pp. 692-704.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 1, January 1987

© 1987 Society for Industrial and Applied Mathematics
003

A NEW ALGORITHM FOR SPLINE SMOOTHING BASED ON SMOOTHING A STOCHASTIC PROCESS*

ROBERT KOHN† AND CRAIG F. ANSLEY‡

Abstract. We derive a new efficient algorithm for optimal spline smoothing as the conditional expectation of a stochastic process observed with noise, using the stochastic model of Wahba (J. Royal Statist. Soc. Ser. B, 40 (1978), pp. 364–372). The conditional expectation is computed by expressing the process in state space form and using the filtering and smoothing results in Ansley and Kohn (Annals Statist., 11 (1985), pp. 1286–1316). We show how to use our algorithms to estimate the smoothness parameter and how to obtain Bayesian confidence intervals for the unknown function and its derivatives. Algorithms based on other stochastic models are compared to ours, and a stochastic derivation is given for Reinsch's (Numer. Math. 10 (1967), pp. 177–183) algorithm for polynomial splines.

Key words. Bayesian confidence interval, cross validation, fixed interval smoothing, interpolating spline, Kalman filter, maximum likelihood, smoothing spline, stochastic process

AMS(MOS) subject classifications. 65D10, 62F15, 62G99

1. Introduction. It was shown by Wahba (see Wahba (1978) and the references therein for earlier joint work by this author) that a smoothing spline can be obtained by taking the conditional expectation of a stochastic process observed with noise. For a different stochastic process this result was also obtained by Weinert, Byrd and Sidhu (1980) (see also the references therein for earlier joint work by Weinert), who furthermore showed that the stochastic process could be written in state space form. Now it is well known in the time series literature that the conditional expectation of a stochastic process expressed in state space form can be efficiently computed in $O(N)$ operations using Kalman filtering and smoothing algorithms, where N is the sample size. This naturally leads to $O(N)$ spline smoothing algorithms, and is the basis of the computationally efficient $O(N)$ spline smoothing algorithms of Weinert, Byrd and Sidhu (1980) and Wecker and Ansley (1983).

In this paper we present a new computationally efficient $O(N)$ spline smoothing algorithm based on Wahba's (1978) stochastic model. Wahba's stochastic model assumes that the stochastic process and its derivatives at the initial time point are diffuse, so that the ordinary Kalman filter, which requires a proper prior distribution for the initial values, cannot be applied directly. We overcome this difficulty by using the modified Kalman filter and fixed interval smoothing algorithms of Kohn and Ansley (1986) and Ansley and Kohn (1985). As pointed out in § 7, our algorithm is computationally more efficient than that of Wecker and Ansley (1983) and is less restrictive than that of Weinert, Byrd and Sidhu (1980).

To evaluate the smoothing spline we must specify the smoothing parameter which controls the tradeoff between the fit and smoothness of the spline. Usually this parameter is estimated from the data. Craven and Wahba (1979) suggest a generalized cross validation estimate of the smoothing parameter, while Wecker and Ansley (1983) use their stochastic model to compute the likelihood of the observations and then estimate the smoothing parameter by maximum likelihood. In § 5 we show how we can use our

* Received by the editors March 12, 1984; accepted for publication (in revised form) December 5, 1985.
† Graduate School of Business, University of Chicago, Chicago, Illinois 60637.
‡ Australian Graduate School of Management, University of New South Wales, New South Wales, 2033 Australia.

algorithms to estimate the smoothness parameter efficiently either by generalized cross validation or by maximizing a marginal likelihood.

In § 6 we view spline smoothing as nonparametric regression, so that as in Wecker and Ansley (1983) and Wahba (1983) we can interpret the stochastic process as a prior distribution for the unknown regression function that we are estimating and its derivatives. The filtering and smoothing algorithms then give us the posterior distribution for the function and its derivatives, and hence also posterior confidence intervals.

Reinsch (1967) gives a $O(N)$ polynomial spline smoothing algorithm based on the special structure of two tridiagonal matrices. In § 8 we give a stochastic derivation of Reinsch's algorithm and so motivate statistically the special structure of the matrices mentioned above. We also point out that our algorithm is numerically more stable than Reinsch's and allows the computation of Bayesian confidence intervals for both the unknown function and its derivatives.

Because of frequent citation we refer to Ansley and Kohn (1985) as AK, Kohn and Ansley (1986) as KA, Wecker and Ansley (1983) as WA, and Weinert, Byrd and Sidhu (1980) as WBS.

The paper is structured as follows. In § 2 we define smoothing splines, and in § 3 we present the Wahba (1978) stochastic model and write it in state space form. We also state some results needed in the following sections. In § 4 we first give the modified Kalman filter and fixed interval smoothing algorithms as applied to the spline smoothing problem and then use them to obtain the new spline smoothing algorithm. Section 5 shows how to estimate the smoothness parameter, and § 6 views the stochastic process as a prior distribution for an unknown regression function. In § 7 we compare our algorithm to that of WBS and WA, and finally in § 8 we present a stochastic interpretation of Reinsch's (1967) algorithm.

2. Defining a smoothing spline. In this section we define the spline smoothing problem.

Assumption 2.1. Let L be the mth degree linear differential operator

$$L=\frac{d^m}{dt^m}-a_{m-1}(t)\frac{d^{m-1}}{dt^{m-1}}-\cdots-a_0(t)$$

with the coefficients $a_j(t)$ $(j=0,\cdots,m-1)$ having j continuous derivatives on the interval $[0,1]$.

DEFINITION 2.1. Let H be the space of real functions that are m times differentiable on $[0,1]$ with mth derivative that is square integrable on $[0,1]$. Let H_0 be the null space of L with respect to H; i.e. $H_0=\{g\in H\colon Lg=0\}$.

Assumption 2.2. (i) For $i=1,\cdots,N$, μ_i is an extended Hermite-Birkhoff linear functional on H of the form

$$\mu_i(f)=\sum_{j=0}^{m-1}h_{ij}f^{(j)}(t_i) \tag{2.1}$$

with $f\in H$ and $0\leqq t_1\leqq t_2\leqq\cdots\leqq t_N\leqq 1$. Here and below the notation $f^{(j)}(t)$ means $d^jf(t)/dt^j$.

(ii) Amongst the N linear functionals $\mu_1,\cdots,\mu_N$, there are m that are linearly independent over H_0.

The general functionals (2.1) are used in WBS.

We now define an Lg smoothing spline.

DEFINITION 2.2. For $i=1,\cdots,N$, suppose we are given the observation $y(i)$ at the point t_i, with $0\leqq t_1\leqq t_2\leqq\cdots\leqq t_N\leqq 1$, and the smoothing constant $\kappa>0$. We define

the optimal Lg smoothing spline for $y(1), \cdots, y(N)$ with respect to the differential operator L and the functionals $\mu_1, \cdots, \mu_N$ as that $\hat{f} \in H$ which minimizes

$$\sum_{i=1}^{N} \{y(i) - \mu_i(f)\}^2 + \left(\frac{1}{\kappa}\right) \int_0^1 \{Lf(t)\}^2 \, dt \tag{2.2}$$

over all $f \in H$. Assumption 2.2 ensures that a unique minimizing $\hat{f}$ exists; see for example WBS.

A particularly important example of Lg smoothing splines are polynomial splines.

Application to polynomial splines. Suppose $L = d^m/dt^m$ and $\mu_i(f) = f(t_i)$ for $i = 1, \cdots, N$. Then $\hat{f}(t)$ is the optimal polynomial smoothing spline of order $2m-1$, so that $\hat{f}(t)$ consists of piecewise polynomials of degree $2m-1$. In particular if $m=2$, then $\hat{f}(t)$ is a cubic spline.

Remark 2.1. For $t_1 < t_2 < \cdots < t_N$ we define the spline interpolation problem as finding that $\hat{f} \in H$ so that $\int \{Lf(t)\}^2 \, dt$ is minimized subject to $\mu_i(f) = y(i)$ for $i = 1, \cdots, N$. This can be regarded as the optimal smoothing spline in the limit as $\kappa \to \infty$.

3. A stochastic model for spline smoothing. We can interpret spline smoothing as follows. Suppose that for $i = 1, \cdots, N$

$$y(i) = \mu_i(f) + e(i) \tag{3.1}$$

where the $e(i)$ are independent random variables with equal variance, so that $y(i)$ is an observation on the functional $\mu_i(f)$ with noise. To estimate the unobserved function $f(t)$ we minimize (2.2), which is the sum of two terms. The first term is the sum of squares loss function $\sum_i \{y(i) - \mu_i(f)\}^2$, while the second term $\int (Lf)^2 \, dt$ is a smoothness penalty function. The smoothing constant κ determines the relative weight given to the two terms.

In particular for cubic splines $y(i) = f(t_i) + e(i)$, so that we observe $f(.)$ with noise, and the smoothness penalty function is just the integral of the curvature squared.

Wahba (1978) shows that the optimal smoothing spline $\hat{f}(t)$ is the limiting conditional expectation of $f(t)$ given $y(1), \cdots, y(N)$, with $f(t)$ generated by the stochastic process described below. A similar result is obtained by WBS for a different stochastic model.

The Wahba (1978) stochastic model is (3.1) with $f(t)$ generated by the stochastic differential equation

$$Lf(t) = \sigma \kappa^{1/2} \frac{dW(t)}{dt} \tag{3.2}$$

where the $e(i)$ and $W(t)$ satisfy the following assumptions.

Assumption 3.1. (i) $\{e(i), i \geqq 1\}$ is a sequence of independent normally distributed scalar random variables having zero mean and variance σ^2.

(ii) $W(t)$ is a zero mean Wiener process with $W(0) = 0$ and $\operatorname{var}\{W(1)\} = 1$. For all i and t, $W(t)$ is independent of the $e(i)$.

Assumption 3.2. The $m \times 1$ vector $\{f(0), f^{(1)}(0), \cdots, f^{(m-1)}(0)\}'$ has the multivariate $N(0, kI_m)$ distribution, with $k > 0$, and is independent of $W(t)$ and the $e(i)$.

In the Wahba (1978) model and below we let $k \to \infty$, so that from Assumption 3.2, $f(t)$ and its first $(m-1)$ derivatives at $t=0$ have a joint distribution which is diffuse and is independent of $W(t)$ and the $e(i)$.

The following lemma obtained by Wahba (1978) leads to an efficient spline smoothing algorithm.

LEMMA 3.1. *The optimal smoothing spline* $\hat{f}(t)$ *is given by*

$$\hat{f}(t) = \lim_{k\to\infty} E\{f(t)|y(1), \cdots, y(N)\}.$$

Proof. Wahba (1978) obtains the result for Chebyshevian splines. For the general Lg smoothing spline case the result follows the equivalence between the WBS and Wahba stochastic models established in § 7 of Ansley and Kohn (1986), and the fact that the WBS stochastic model gives rise to all Lg smoothing splines.

In order to obtain an efficient spline smoothing algorithm we follow WBS and WA and write (3.1) and (3.2) in state space form. From (3.2) and a standard representation of linear differential equations (see, for example, Hochstadt (1964, p. 51) or WBS), it follows that we can write

$$\frac{dx(t)}{dt} = A(t)x(t) + \kappa^{1/2}\sigma g\frac{dW(t)}{dt}, \tag{3.3}$$

$$f(t) = \iota' x(t), \tag{3.4}$$

where $x(t)$ is the $m\times 1$ state vector $x(t) = \{f(t), f^{(1)}(t), \cdots, f^{(m-1)}(t)\}'$, $g = (0, 0, \cdots, 1)'$ and $\iota = (1, 0, \cdots, 0)'$. The state transition matrix $A(t)$ is given by

$$A(t) = \left[\begin{array}{c|c} 0 & I_{m-1} \\ \hline a_{m-1}(t) & a_{m-2}(t)\cdots a_0(t) \end{array}\right].$$

Now let $\chi(t)$ be the $m\times m$ matrix solution of the matrix differential equation $d\chi(t)/dt = A(t)\chi(t)$ for $0\leqq t\leqq 1$, with initial condition $\chi(0) = I_m$. Then, by Hochstadt (1964, Chap. 2), $\chi(t)$ exists and is unique. Let $\phi(t, s) = \chi(t)\chi(s)^{-1}$. Then, by Hochstadt (1964, Chap. 2)

$$x(t) = \phi(t, s)x(s) + \kappa^{1/2}\sigma w(t, s) \tag{3.5}$$

where

$$w(t, s) = \int_s^t \phi(t, \nu)gW(d\nu) \tag{3.6}$$

with $s, t\in[0, 1]$ and $s\leqq t$.

From (2.1), $\mu_i(f) = H(i)x(t_i)$, where $H(i) = (h_{i,0}, \cdots, h_{i,m-1})$. Therefore from (3.1) and (3.5) we can write the stochastic model in state space form as

$$y(i) = H(i)x(t_i) + e(i), \qquad i = 1, \cdots, N, \tag{3.7}$$

$$x(t_{i+1}) = \phi(t_{i+1}, t_i)x(t_i) + \sigma\kappa^{1/2}w(t_{i+1}, t_i), \tag{3.8}$$

with $t_0 = 0$. Let

$$S_w(t, s) = \text{var}\{w(t, s)\}/\sigma^2 \tag{3.9}$$

for $s\leqq t$ and $s, t\in[0, 1]$.

Polynomial splines. For the polynomial spline of degree $2m-1$, we have from (3.3) of WA that the ijth element of $\chi(t)$ is

$$\chi(t) = \begin{cases} t^{j-i}/(j-i)!, & j\geqq i, \\ 0, & j<i, \end{cases}$$

and from (3.9) of WA that the ijth element of $S_w(t, s)$ is

$$S_w(t, s)_{ij} = \frac{(t-s)^{2m-i-j+1}}{\{(m-i)!\,(m-j)!\,(2m-i-j+1)\}}$$

for $s \leqq t$. In particular for cubic splines

$$\chi(t) = \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix} \quad \text{and} \quad S_w(t, s) = \begin{bmatrix} (t-s)^3/3 & (t-s)^2/2 \\ (t-s)^2/2 & (t-s) \end{bmatrix}.$$

Remark 3.1. The stochastic model for spline interpolation is $y(i) = \mu_i(f)$ for $i = 1, \cdots, N$ together with (3.3) and (3.4), and with $\kappa = \sigma = 1$ in (3.3). Lemma 3.1 then holds for the optimal interpolating spline.

We now adopt the following notation. For $i = 1, \cdots, N$ and $t \in [0, 1]$, let

$$x(t|i; k) = E\{x(t)|y(1), \cdots, y(i)\}$$

and

$$S_x(t|i; k) = \text{var}\{x(t) - x(t|i; k)\}.$$

We define $f(t|i; k)$ and $S_f(t|i; k)$ similarly.

The following lemma gives a decomposition of $x(t|i; k)$ and $S_x(t|i; k)$ in terms of powers of k.

LEMMA 3.2. (i) *For $i = 1, \cdots, N$ and $t \in [0, 1]$, we can write*

$$x(t|i; k) = x^{(0)}(t|i) + O(1/k), \tag{3.10}$$

$$S_x(t|i; k) = kS_x^{(1)}(t|i) + \sigma^2 S_x^{(0)}(t|i) + O(1/k), \tag{3.11}$$

where $x^{(0)}(t|i)$, $S_x^{(1)}(t|i)$ and $S_x^{(0)}(t|i)$ do not depend on k or σ^2.

(ii) *Let i_* be the smallest value of i for which $\mu_1, \cdots, \mu_i$ contain m linearly independent functionals over H_0. Then,*

$$S_x^{(1)}(t|i) = 0 \quad \textit{for } i \geqq i_* \textit{ and } t \in [0, 1]. \tag{3.12}$$

Proof. Because $x(0) \sim N(0, kI_m)$ by Assumption 3.2, (3.10) and (3.11) follow from Lemma 6.1 of AK. An alternative constructive proof can be obtained from § 5 and Appendix A of KA.

Let

$$B_i = H(i)\phi(t_i, 0), \quad B^{(i)} = (B_1', \cdots, B_i')', \quad \omega(i) = H(i)w(t_i, 0) + e(i),$$
$$\omega^{(i)} = \{\omega(1), \cdots, \omega(i)\}' \quad \text{and} \quad y^{(i)} = \{y(1), \cdots, y(i)\}'.$$

Then it is not difficult to check that

$$y^{(i)} = B^{(i)}x(0) + \omega^{(i)} \tag{3.13}$$

and we have from (3.5) that $x(t) = \phi(t, 0)x(0) + \kappa^{1/2}\sigma w(t, 0)$. It follows that we can write the joint covariance matrix of $y^{(i)}$ and $x(t)$ in the form $k\Omega^{(1)} + \Omega^{(0)}$, where $\Omega^{(1)}$ and $\Omega^{(0)}$ do not depend on k and rank $[\Omega^{(1)}] \leqq m$.

By Assumption 2.2(ii), $i_* \leqq N$, and so by Weinert and Sidhu (1977), $B^{(i)}$ is of full column rank for $i \geqq i_*$. Now $\text{var}\{y^{(i)}\} = kB^{(i)}B^{(i)\prime} + \text{var}\{\omega^{(i)}\}$ and rank $\{B^{(i)}B^{(i)\prime}\} = m$ for $i \geqq i_*$. It therefore follows from Theorem 10.2, part (iii) of AK that (3.12) holds. □

Finally we obtain the following result which is used in the next section.

LEMMA 3.3. *If $t_i < t_{i+1}$, then $S_x^{(0)}(t_{i+1}|i)$ is nonsingular.*

Proof. From the proof of Theorem 6.1 of AK,

$$S_x(t_{i+1}|i; k) = \phi(t_{i+1}, t_i)S_x(t_i|i; k)\phi(t_{i+1}, t_i)' + \kappa\sigma^2 S_w(t_{i+1}, t_i)$$

and so it is sufficient to show that $S_w(t, s)$ is nonsingular for $s < t$. Fixing s and t, suppose there is an $m \times 1$ vector α so that $\alpha' S_w(t, s)\alpha = 0$. We will show that $\alpha = 0$. It

follows from (3.6) that $\alpha'\phi(t, \zeta)g = 0$ for $s \leqq \zeta \leqq t$, and so

$$\alpha'\left\{\frac{\partial^j \phi(t, \zeta)}{\partial \zeta^j}\right\} g = 0 \tag{3.14}$$

for $j = 0, \cdots, m-1$. Now put $\eta_0(t) = g$ and define $\eta_j(t)$, $j = 1, \cdots, m-1$, recursively by

$$\eta_{j+1}(t) = \frac{\partial \eta_j(t)}{\partial t} - A(t)\eta_j(t). \tag{3.15}$$

It is not difficult to check from (3.15) that for $j = 0, \cdots, m-1$

$$\eta_j(t) = (0, 0, \cdots, 0, 1, x, \cdots, x)' \tag{3.16}$$

where the 1 is in the $(m-j)$th position and the x's indicate elements that are not identically zero.

As in Kohn and Ansley (1983) we can check by induction that

$$\alpha'\left\{\frac{\partial^j \phi(t, \zeta)}{\partial \zeta^j}\right\} \eta_0 = \alpha'\phi(t, \zeta)\eta_j(\zeta)$$

so that from (3.14), $\alpha'\phi(t, \zeta)\eta_j(\zeta) = 0$ for $j = 0, \cdots, m-1$. From (3.16), $\alpha'\phi(t, \zeta) = 0$ and so $\alpha = 0$. □

4. The spline smoothing algorithm. The new spline smoothing algorithm presented below is based on the Wahba stochastic model described in § 3 and its state space representation. The algorithm consists of two parts, the first based on the modified Kalman filter given in § 5 of KA (with the multivariate generalization in § 6 of AK), and the second part is based on the modified fixed interval smoothing algorithm given in § 8 of AK. For clarity of presentation we first present the modified Kalman filter as Algorithm 4.1 below, then the modified fixed interval smoothing algorithm as Algorithm 4.2 below, and finally we present the spline smoothing algorithm which combines the above two algorithms. Throughout this section we take the smoothness parameter κ to be given.

For $i = 1, \cdots, N$, let

$$\varepsilon(i; k) = y(i) - E\{y(i)|y(1), \cdots, y(i-1)\}, \qquad R(i; k) = \text{var}\{\varepsilon(i; k)\},$$

so that $\varepsilon(i; k)$ is the ith innovation and $R(i; k)$ is its variance. Then, as in Lemma 3.1 of KA,

$$\varepsilon(i; k) = \varepsilon^{(0)}(i) + O(1/k)$$

and

$$R(i; k) = kR^{(1)}(i) + \sigma^2 R^{(0)}(i) + O(1/k),$$

where $\varepsilon^{(0)}(i)$, $R^{(1)}(i)$ and $R^{(0)}(i)$ are independent of k and σ^2.

ALGORITHM 4.1 (Modified Kalman Filter).

To make the algorithm easier to follow, we break it up into four steps. The variable i_* is defined in Lemma 3.2(ii) above, and $S_x^{(1)}(t|i) = 0$ for $i \geqq i_*$ and all t, and $R^{(1)}(i+1) = 0$ for $i \geqq i_*$.

Step 0 (Initialization):

Set $x^{(0)}(0|0) = 0$, $S_x^{(0)}(0|0) = 0$ and $S_x^{(1)}(0|0) = I_m$.

Let $t_0 = 0$, and for convenience write $F(i) = \phi(t_{i+1}, t_i)$.

For $i=0,\cdots,N-1$:

Step 1:

$$x^{(0)}(t_{i+1}|i)=F(i)x^{(0)}(t_i|i),$$

$$S_x^{(0)}(t_{i+1}|i)=F(i)S_x^{(0)}(t_i|i)F(i)'+\kappa S_w(t_{i+1},t_i).$$

If $i<i_*$, then

$$S_x^{(1)}(t_{i+1}|i)=F(i)S_x^{(1)}(t_i|i)F(i)'.$$

Step 2:

$$\varepsilon^{(0)}(i+1)=y(i+1)-H(i+1)x^{(0)}(t_{i+1}|i),$$

$$R^{(0)}(i+1)=\iota'S_x^{(0)}(t_{i+1}|i)\iota+1.$$

If $i<i_*$, then

$$R^{(1)}(i+1)=\iota'S_x^{(1)}(t_{i+1}|i)\iota.$$

Step 3: We divide this step into two parts, depending on whether $R^{(1)}(i+1)>0$ or $R^{(1)}(i+1)=0$, noting that $R^{(1)}(i+1)$ is identically zero for $i\geqq i_*$. For notational simplicity we write $S_x^{(1)}(t_{i+1}|i)$ as $S^{(1)}$, $S_x^{(0)}(t_{i+1}|i)$ as $S^{(0)}$, $R^{(1)}(i+1)$ as $R^{(1)}$, $R^{(0)}(i+1)$ as $R^{(0)}$, and $\varepsilon^{(0)}(i+1)$ as $\varepsilon^{(0)}$.

Part (a). $R^{(1)}>0$. (This can occur only if $i<i_*$.)

$$x^{(0)}(t_{i+1}|i+1)=x^{(0)}(t_{i+1}|i)+S^{(1)}\iota\varepsilon^{(0)}/R^{(1)},$$

$$S_x^{(0)}(t_{i+1}|i+1)=S^{(0)}-S^{(1)}\iota\iota'S^{(0)}/R^{(1)}-S^{(0)}\iota\iota'S^{(1)}/R^{(1)}$$
$$+S^{(1)}\iota\iota'S^{(1)}R^{(0)}/\{R^{(1)}\}^2,$$

$$S_x^{(1)}(t_{i+1}|i+1)=S^{(1)}-S^{(1)}\iota\iota'S^{(1)}/R^{(1)}.$$

Part (b). $R^{(1)}=0$. (This is always true if $i\geqq i_*$.)

$$x^{(0)}(t_{i+1}|i+1)=x^{(0)}(t_{i+1}|i)+S^{(0)}\iota\varepsilon^{(0)}/R^{(0)},$$

$$S_x^{(0)}(t_{i+1}|i+1)=S^{(0)}-S^{(0)}\iota\iota'S^{(0)}/R^{(0)}.$$

If $i+1<i_*$, then

$$S_x^{(1)}(t_{i+1}|i+1)=S^{(1)}.$$

Proof. Given the state space equations (3.7) and (3.8), the derivation is very similar to that of Theorem 5.1 of KA. □

We next use the modified fixed interval smoothing algorithm to obtain recursively $x^{(0)}(t|N)$ and $S_x^{(0)}(t|N)$ for $0\leqq t\leqq 1$. It is convenient to first state the following lemmas.

LEMMA 4.1. *Suppose that* $t_i<t_{i+1}$ *and let* $t_i\leqq t<t_{i+1}$. Let $\Omega(t_{i+1},t|i;k)$ *be the* $2m\times 2m$ *conditional covariance matrix of the* $2m\times 1$ *vector* $[x(t_{i+1})',x(t)']'$ *given* $y(1),\cdots,y(i)$. *Then*:

$$\text{(4.1)}\qquad \text{(i)}\qquad \Omega(t_{i+1},t|i;k)=\begin{bmatrix} S_x(t_{i+1}|i;k) & \Psi(t|i;k)\\ \Psi(t|i;k)' & S_x(t|i;k)\end{bmatrix}$$

with

$$\text{(4.2)}\qquad \Psi(t|i;k)=\phi(t_{i+1},t_i)S_x(t_i|i;k)\phi(t,t_i)'+\kappa S_w(t,t_i)$$

and

$$\text{(4.3)}\qquad S_x(t|i;k)=\phi(t,t_i)S_x(t_i|i;k)\phi(t,t_i)'+\kappa\sigma^2S_w(t,t_i).$$

(ii) *We can write*

$$\Omega(t_{i+1}, t|i; k) = k\Omega^{(1)}(t_{i+1}, t|i) + \sigma^2\Omega^{(0)}(t_{i+1}, t|i) + O(1/k) \tag{4.4}$$

with $\Omega^{(1)}(t_{i+1}, t|i)$ *and* $\Omega^{(0)}(t_{i+1}, t|i)$ *independent of* k *and* σ^2.

(iii) *We can factorize* $\Omega(t_{i+1}, t|i; k)$ *as* $L(k)\Lambda(k)L(k)'$, *where* $L(k)$ *is a lower triangular matrix with* 1*'s on the diagonal and* $\Lambda(k)$ *is a diagonal matrix. For notational convenience, we do not indicate dependence of* $L(k)$ *and* $\Lambda(K)$ *on* t_{i+1}, t *and* i. *Furthermore, we can write,*

$$L(k) = L^{(0)} + \sigma^2(1/k)L^{(-1,1)} + (1/k)L^{(-1,0)}(k),$$

$$\Lambda(k) = k\Lambda^{(1)} + \sigma^2\Lambda^{(0)} + O(1/k)$$

where the matrices $L^{(0)}$, $L^{(-1,1)}$, $\Lambda^{(1)}$ *and* $\Lambda^{(0)}$ *do not depend on* k *and* σ^2, *and* $L^{(-1,0)}(k)$ *has elements that are bounded in* k. *Furthermore,* $L^{(-1,0)}(k)\Lambda^{(1)} = O(1/k)$.

Proof. The derivation of parts (i) and (ii) is similar to that of parts (i) and (ii), respectively, of Lemma 8.1 of AK. Equation (4.4) follows from (4.1)-(4.3) and Lemma 3.2. □

An algorithm for constructing the factors in part (iii) of Lemma 4.1 is given in Corollary 10.2 of AK. For completeness it is reproduced in the Appendix.

LEMMA 4.2. *Let the conditions of Lemma* 4.1 *hold, and let* $L(k)$ *and* $\Lambda(k)$ *be as in Lemma* 4.1. *Partition* $L(k)$ *and* $\Lambda(k)$ *as*

$$L(k) = \begin{bmatrix} L_{11}(k) & 0 \\ L_{21}(k) & L_{22}(k) \end{bmatrix}, \qquad \Lambda(k) = \begin{bmatrix} \Lambda_{11}(k) & 0 \\ 0 & \Lambda_{22}(k) \end{bmatrix}$$

with $L_{11}(k)$ *and* $\Lambda_{11}(k)$ $m \times m$ *matrices and the rest of the submatrices in* $L(k)$ *and* $\Lambda(k)$ *partitioned conformally. Partition* $L^{(0)}$, $L^{(-1,1)}$, $\Lambda^{(0)}$ *and* $\Lambda^{(1)}$ *correspondingly.*

Let $C(k) = L_{21}(k)L_{11}^{-1}(k)$. *Then we can write*

$$C(k) = C^{(0)} + \sigma^2(1/k)C^{(-1,1)} + (1/k)C^{(-1,0)}(k)$$

where $C^{(0)}$ *and* $C^{(-1,1)}$ *are independent of* k *and* σ^2 *and* $C^{(-1,0)}(k)$ *has elements that are bounded in absolute value. Furthermore,*

$$C^{(0)} = L_{21}^{(0)}\{L_{11}^{(0)}\}^{-1},$$

$$C^{(-1,1)} = L_{21}^{(-1,1)}\{L_{11}^{(0)}\}^{-1} - L_{21}^{(0)}\{L_{11}^{(0)}\}^{-1}L_{11}^{(-1,1)}\{L_{11}^{(0)}\}^{-1}.$$

Proof. Let $\tilde{\Lambda}_{11}$ be a diagonal matrix with ith diagonal element 1 if the ith diagonal element of $\Lambda_{11}(k)$ is positive and zero otherwise. If $t_i < t_{i+1}$, then by Lemma 3.2, $S_x(t_{i+1}|i)$ is nonsingular and hence $\tilde{\Lambda}_{11} = I_m$, so that $C(k) = L_{21}(k)\tilde{\Lambda}_{11}L^{-1}(k)$. The rest of the lemma follows from part (iii) of Lemma 8.1 of AK. □

We now present the modified fixed interval smoothing algorithm.

ALGORITHM 4.2.

(i) For $i = N-1, \cdots, 1$:

If $t_i < t_{i+1}$, then

Construct $\Omega^{(1)}(t_{i+1}, t_i|i)$ and $\Omega^{(0)}(t_{i+1}, t_i|i)$ as in part (i) of Lemma 4.1. Obtain $L^{(0)}$, $L^{(-1,1)}$, $\Lambda^{(1)}$ and $\Lambda^{(0)}$ as in the Appendix and hence obtain $C^{(0)}$ and $C^{(-1,1)}$ as in Lemma 4.2. If $i \geqq i_*$, then obtain $L^{(0)}$, $\Lambda^{(0)}$ and $C^{(0)}$ only.

$$x^{(0)}(t_i|N) = x^{(0)}(t_i|i) + C^{(0)}\{x^{(0)}(t_{i+1}|N) - x^{(0)}(t_{i+1}|i)\},$$

$$S_x^{(0)}(t_i|N) = S_x^{(0)}(t_i|i) - C^{(0)}\{S_x^{(0)}(t_{i+1}|i) - S_x^{(0)}(t_{i+1}|N)\}C^{(0)\prime}.$$

If $i < i_*$, then modify $S_x^{(0)}(t_i|N)$ by

$$S_x^{(0)}(t_i|N) = S_x^{(0)}(t_i|N) - C^{(0)}\{S_x^{(1)}(t_{i+1}|i) - S_x^{(1)}(t_{i+1}|N)\}C^{(-1,1)\prime},$$
$$- C^{(-1,1)}\{S_x^{(1)}(t_{i+1}|i) - S_x^{(1)}(t_{i+1}|N)\}C^{(0)\prime}.$$

(ii) We now obtain $x^{(0)}(t|N)$ and $S_x^{(0)}(t|N)$ for $0 \leqq t \leqq 1$ and $t \neq t_i$, $i = 1, \cdots, N$.

Case 1: $t_i < t < t_{i+1}$ for some i.

Construct $\Omega^{(1)}(t_{i+1}, t|i)$ and $\Omega^{(0)}(t_{i+1}, t|i)$ as in part (i) of Lemma 4.1. Obtain $L^{(0)}$, $L^{(-1,1)}$, $\Lambda^{(1)}$ and $\Lambda^{(0)}$ as in the Appendix, and hence obtain $C^{(0)}$ and $C^{(-1,1)}$ as in Lemma 4.2. If $i \geqq i_*$, then obtain $L^{(0)}$, $\Lambda^{(0)}$ and $C^{(0)}$ only.

$$x^{(0)}(t|i) = \phi(t, t_i)x^{(0)}(t_i|i),$$
$$x^{(0)}(t|N) = x^{(0)}(t|i) + C^{(0)}\{x^{(0)}(t_{i+1}|N) - x^{(0)}(t_{i+1}|i)\},$$
$$S_x^{(0)}(t|N) = S_x^{(0)}(t|i) - C^{(0)}\{S_x^{(0)}(t_{i+1}|i) - S_x^{(0)}(t_{i+1}|N)\}C^{(0)\prime}.$$

If $i < i_*$ then modify $S_x^{(0)}(t|N)$ by

$$S_x^{(0)}(t|N) = S_x^{(0)}(t|N) - C^{(0)}\{S_x^{(1)}(t_{i+1}|i) - S_x^{(1)}(t_{i+1}|N)\}C^{(-1,1)\prime}$$
$$- C^{(-1,1)}\{S_x^{(1)}(t_{i+1}|i) - S_x^{(1)}(t_{i+1}|N)\}C^{(0)\prime}.$$

Case 2: $t > t_N$.

$$x^{(0)}(t|N) = \phi(t, t_N)x^{(0)}(t_N|N),$$
$$S_x^{(0)}(t|N) = \phi(t, t_N)S_x^{(0)}(t_N|N)\phi(t, t_N)' + \kappa S_w(t, t_N).$$

Case 3: $t < t_1$.

$$x^{(0)}(t|N) = \phi(t, t_1)x^{(0)}(t_1|N),$$
$$S_x^{(0)}(t|N) = \phi(t, t_1)S_x^{(0)}(t_1|N)\phi(t, t_1)' + \kappa S_w(t_1, t).$$

Proof. Part (i) follows from Theorem 8.2 of AK and part (ii) can be proved similarly to Theorem 8.2 of AK. □

We now present our new spline smoothing algorithm which combines Algorithms 4.1 and 4.2.

ALGORITHM 4.3 (Spline Smoothing Algorithm).

Step 1: For $i = 1, \cdots, N-1$, obtain and store $x^{(0)}(t_i|i)$, $x^{(0)}(t_{i+1}|i)$, $S_x^{(0)}(t_i|i)$ and $S_x^{(0)}(t_{i+1}|i)$; if $i < i_*$ also obtain and store $S_x^{(1)}(t_i|i)$. In addition, obtain and store $x^{(0)}(t_N|N)$ and $S_x^{(0)}(t_N|N)$. All this is done using Algorithm 4.1.

Step 2: For $i = N-1, \cdots, 1$, if $t_i < t_{i+1}$ obtain $x^{(0)}(t_i|N)$ as in part (i) of Algorithm 4.2 and set $\hat{f}(t_i) = x_1^{(0)}(t_i|N)$. Store $x^{(0)}(t_1|N)$.

Step 3: For $0 \leqq t \leqq 1$, and $t \neq t_i$, $i = 1, \cdots, N$, obtain $x^{(0)}(t|N)$ as in part (ii) of Algorithm 4.2 and set $\hat{f}(t) = x_1^{(0)}(t|N)$.

Remark 4.1. It is clear that our algorithm is $O(N)$. For small m, the storage requirements are also modest. The output from Step 1 of Algorithm 4.3 requires

$$m(m+3)N + m(m+1)(i_* - 1)/2 + O(m^2)$$

locations. It will also be useful to store the matrices $\phi(t_{i+1}, t_i)$ and $S_w(t_{i+1}, t_i)$ for all i for which $t_i < t_{i+1}$, especially if we compute the spline for different values of κ. This requires $(3m^2 + m)(N' - 1)/2$ locations in the general case and $(m^2 + 3m - 2)(N' - 1)/2$ locations for polynomial splines, where N' is the number of distinct t_i.

If our data is equispaced so that $t_{i+1}-t_i$ is either zero or a constant, we need only store one ϕ and one S_w.

If we require $\hat{f}(t)$ for many points in the interval (t_i, t_{i+1}), then the method outlined below may be more efficient when $A(t)$ is independent of t than evaluating $\hat{f}(t)$ for each t using Step 3 of Algorithm 4.3.

First we note the following.

LEMMA 4.3. *Let $x_j^{(0)}(t|N)$ be the jth element of $x^{(0)}(t|N)$. Then for $j=0,\cdots,m-1$, $d^j\hat{f}(t)/dt^j = x_{j+1}^{(0)}(t|N)$.*

Proof. For the WBS stochastic model $\hat{f}(t)=x_1(t|N)$ exactly, and so from the definition of $x(t)$ we have that for the WBS model $d^j\hat{f}(t)/dt^j = x_{j+1}(t|N)$ for $j=0,\cdots,m-1$. But it is shown in Ansley and Kohn (1986) that $x(t|N)$ for the WBS model is equal to $x^{(0)}(t|N)$, thus completing the proof. □

Consider now the interval $[t_i, t_{i+1}]$ with $t_i < t_{i+1}$. Then from Weinert, Desai and Sidhu (1979), or more generally from Kohn and Ansley (1983),

$$L^*L\hat{f}(t)=0, \qquad t_i<t<t_{i+1} \tag{4.5}$$

where L^* is the adjoint of L. We can now solve for $\hat{f}(t)$ in the interval $[t_i, t_{i+1}]$ using the $2m$ boundary conditions:

$$\frac{d^j\hat{f}(t_i)}{dt^j} = x_{j+1}^{(0)}(t_i|N), \qquad \frac{d^j\hat{f}(t_{i+1})}{dt^j} = x_{j+1}^{(0)}(t_{i+1}|N) \tag{4.6}$$

for $j=0,\cdots,m-1$.

For example, for polynomial splines of degree $2m-1$, $L=d^m/dt^m=L^*$ so that from (4.5), $\hat{f}(t)$ is a polynomial of degree $2m-1$ in the interval $[t_i, t_{i+1}]$ and we can solve for its $2m$ coefficients using (4.6).

Remark 4.2. If we make the following modifications to Algorithms 4.1 and 4.2, then Algorithm 4.3 will give us an optimal interpolating spline. In Algorithms 4.1 and 4.2 set $\kappa=1$, and in Step 2 of Algorithm 4.1, $R^{(0)}(i+1)=\iota' S_x^{(1)}(t_{i+1}|i)\iota$.

5. Estimating the smoothing parameter. The optimality of our smoothing spline depends on the choice of the smoothing parameter κ, which controls the trade-off between the fit and smoothness of the spline. To obtain a suitable value of κ, we usually have to estimate it from the data, and there are several ways of doing so satisfactorily. Craven and Wahba (1979) suggest using generalized cross validation (GCV) to estimate κ. WA use the properties of $f(t)$ in their stochastic model to estimate κ by maximum likelihood, and Wahba (1985) also discusses an estimate of κ obtained by maximizing what she calls a generalized likelihood.

We now show how to estimate κ by both the above methods using Algorithms 4.1–4.3. First we discuss the marginal likelihood estimate of κ, taking $f(t)$ to be generated by the stochastic model of § 3. Then, from (3.13) we can write

$$y=Bx(0)+\omega \tag{5.1}$$

where B is the $N\times m$ matrix with ith row $H(i)\phi(t_i,0)$ and ω is the $N\times 1$ vector with ith element $H(i)w(t_i,0)+e(i)$; $y=\{y(1),\cdots,y(N)\}'$. From the proof of Lemma 3.2, B has full column rank.

Following § 3 of AK and § 4 of Kalbfleisch and Sprott (1970), we define the marginal likelihood of κ and σ^2 as follows. Let J be a $N\times N$ nonsingular matrix, independent of κ and σ^2, such that JB has exactly $N-m$ zero rows, and let J_2 be the $(N-m)\times N$ submatrix of J corresponding to the zero rows of JB. Such a J always

exists. Let

$$z = J_2 y = J_2 \omega$$

so the density of z depends only on κ and σ^2. We call the density of z the marginal likelihood of κ and σ^2.

The following key result is obtained in § 5 of AK.

LEMMA 5.1. *Let $g(y; \kappa, \sigma^2; k)$ and $g(z; \kappa, \sigma^2)$ be, respectively, the densities of y and z under the stochastic model of* § 3. *Then*

$$g(z; \kappa, \sigma^2) = c \lim_{k \to \infty} k^{m/2} g(y; \kappa, \sigma^2; k) \tag{5.2}$$

where c is a constant independent of κ and σ^2.

Define $L(\kappa, \sigma^2) = g(z; \kappa, \sigma^2)$ as the marginal likelihood. It follows from (5.2) that the marginal likelihood is independent of the transformation J. As in § 3 of KA, we can show that

$$-2 \log L(\kappa, \sigma^2) = (N-m) \log \sigma^2 + \sum_i{}' \log R^{(0)}(i) + \sum_i{}' \{\varepsilon^{(0)}(i)\}^2 / \sigma^2 R^{(0)}(i) \tag{5.3}$$

omitting additive constants that do not depend on κ and σ^2, where $\sum_i{}'$ is the sum over all $i = 1, \cdots, N$, for which $R^{(1)}(i) = 0$.

Concentrating σ^2 out of (5.3) we obtain

$$\hat{\sigma}^2(\kappa) = \left\{ \sum_i{}' \varepsilon^{(0)}(i)^2 / R^{(0)}(i) \right\} / (N-m)$$

and we therefore minimize

$$\sum_i{}' \log R^{(0)}(i) + (N-m) \log \left\{ \sum_i{}' \varepsilon^{(0)}(i)^2 / R^{(0)}(i) \right\} \tag{5.4}$$

to obtain $\hat{\kappa}$, the marginal maximum likelihood estimate of κ. $\hat{\sigma}^2(\hat{\kappa})$ is then the estimate of σ^2. For each κ, we can compute (5.4) efficiently by Algorithm 4.1. It follows from Lemma 5.1 that the generalized likelihood discussed by Wahba (1985) is $L(\kappa, \sigma^2)$.

WA take $x(0)$, κ and σ^2 as unknown and estimate them all by maximum likelihood. Thus their likelihood is different from the marginal likelihood $L(\kappa, \sigma^2)$ and hence so is their estimate of κ and the smoothing spline they obtain.

It seems likely that the marginal likelihood estimates of κ and σ^2 will be superior to the full maximum likelihood estimates of κ and σ^2 in WA as the former will be less biased. See Kalbfleisch and Sprott (1970, § 4), for further details.

We now discuss how to estimate κ by GCV. Let $\hat{f} = \{\hat{f}(t_1), \cdots, \hat{f}(t_N)\}'$. Then by Lemma 3.1 and Algorithms 4.1–4.3, for given κ we can write $\hat{f}(\kappa) = T(\kappa) y$. Craven and Wahba (1979) define the GCV estimate of κ as the value minimizing

$$V(\kappa) = \frac{\hat{e}(\kappa)' \hat{e}(\kappa) / N}{[\mathrm{Tr}\{I - T(\kappa)\}/N]^2}$$

where $\hat{e}(\kappa) = y - \hat{f}(\kappa)$.

Let ι_j be the $N \times 1$ vector having 1 in the jth position and zeros elsewhere, and let $\xi^{(j)} = T(\kappa) \iota_j$ for $j = 1, \cdots, N$. Then $T_{jj} = \xi_j^{(j)}$, the jth element of $\xi^{(j)}$, so that

$$\mathrm{Tr}(I - T) = N - \sum_{j=1}^{N} \xi_j^{(j)},$$

and

$$V(\kappa)=\frac{\hat{e}(\kappa)'\hat{e}(\kappa)/N}{\left[1-\left\{\sum_{j=1}^{N}\xi_j^{(j)}/N\right\}\right]^2}.$$

For $j=1,\cdots,N$, we obtain $\xi^{(j)}$ from ι_j using Algorithm 4.3, and so for given κ we obtain $V(\kappa)$ using $O(N^2)$ operations. Therefore we can minimize $V(\kappa)$ using $O(N^2)$ operations.

Wahba (1985) shows that under certain conditions the GCV estimate of κ is asymptotically superior to that obtained from the marginal likelihood, and provides some simulation evidence that this conclusion also holds in moderate sized samples. From the limited evidence we have, both marginal maximum likelihood and GCV seem to perform reasonably well.

In terms of computing time marginal maximum likelihood is more attractive than GCV as it is $O(N)$, whereas GCV generally requires $O(N^2)$ operations.

Silverman (1984) obtains an approximate GCV procedure for the cubic spline case that can be combined with any $O(N)$ spline smoothing algorithm to give an $O(N)$ GCV spline smoother. Hutchinson and de Hoog (1985) give an exact $O(N)$ method for the same case based on Reinsch's algorithm. In both papers the results also hold for general polynomial splines, but do not carry over to more general splines with functionals of the form (2.1).

6. A Bayesian approach to spline smoothing and posterior confidence intervals. If we look at spline smoothing as a way of estimating an unknown function by nonparametric regression, then we can follow WA and Wahba (1983) and view the stochastic model for $f(t)$ in § 3 as a prior distribution for the unknown function $f(t)$. Then our prior tells us that $f(t)$ has $(m-1)$ continuous derivatives, and because the variance of $dW(t)/dt$ is infinite, we are placing a diffuse prior on the mth derivative of $f(t)$. Also, letting $k\to\infty$ in Assumption 3.2 means that we are placing a diffuse prior on $f(t)$ and its first $(m-1)$ derivatives at $t=0$.

The prior distribution for $f(t)$ just described is not implausible. A diffuse prior on $f(t)$ and its first $(m-1)$ derivatives at $t=0$ indicates our lack of knowledge about the starting values for $f(t)$ and its derivatives. Similarly, a diffuse prior on the mth derivative of $f(t)$ indicates a lack of knowledge about the mth derivative.

Following WA we can use the prior distribution for $f(t)$ to obtain the joint posterior distribution of $f(t)$ and its first $(m-1)$ derivatives, and hence posterior confidence intervals for $f(t)$ and its first $(m-1)$ derivatives. From (3.10) and (3.11), the limiting posterior distribution of $x(t)$ is $N[x^{(0)}(t|N),\sigma^2 S_x^{(0)}(t|N)]$ where $x^{(0)}(t|N)$ and $S_x^{(0)}(t|N)$ are obtained by Algorithm 4.3. Because $x(t)=\{f(t),f^{(1)}(t),\cdots,f^{(m-1)}(t)\}'$, it follows that (6.1) gives the joint posterior distribution of $f(t)$ and its first $(m-1)$ derivatives. It is now straightforward to obtain 95% confidence intervals for $f(t)$ and its first $(m-1)$ derivatives.

An alternative method for obtaining confidence intervals for $f(t)$ but not its derivatives is also suggested by Wahba (1983).

7. Comparison with the WBS and WA spline smoothing algorithms. Both WBS and WA present $O(N)$ spline smoothing algorithms based on smoothing a stochastic process, and are equivalent, in a sense defined below, to our algorithm. Thus the choice between the various algorithms depends on computational efficiency and lack of restrictive assumptions. We now show why our algorithm is superior to the other two if judged by these two criteria.

WBS (and the papers referenced therein) were the first to recognize that the stochastic process generating the spline could be written in state space form, and so filtering and smoothing algorithms could be used to obtain the smoothing spline. The WBS stochastic process is such that the signal $f(t)$ is correlated with the noise $e(i)$ for $i=1,\cdots,m$. For $t\geqq t_m$, the WBS algorithm is carried out using the Kalman filter and the fixed interval smoothing algorithm. Because of the correlation between signal and noise mentioned above, special treatment is required for $t\leqq t_m$ to initialize and end the algorithm.

A major drawback of the WBS algorithm is that it requires the first m functionals $\mu_1,\cdots,\mu_m$ to be linearly independent over H_0. This condition is removed by Ansley and Kohn (1984) at the expense of a more complicated algorithm.

For given $\kappa>0$, let $x(t|N)_{\text{WBS}}=E\{x(t)|y\}$ and $S_x(t|N)_{\text{WBS}}=\text{var}\{x(t)|y\}/\sigma^2$ for the WBS stochastic model. Then Ansley and Kohn (1986) show that

$$x^{(0)}(t|N)=x(t|N)_{\text{WBS}},\qquad S_x^{(0)}(t|N)=S_x(t|N)_{\text{WBS}} \tag{7.1}$$

so that the output from our algorithm is equivalent to that of WBS. Equation (7.1) is somewhat surprising because the stochastic process of § 3 is different to that of WBS.

Next we look at the WA algorithm. WA use the same stochastic model as in § 3, except that instead of Assumption 3.2, $x(0)$ is regarded as a vector of unknown constants. Let $D(t)=\text{cov}\{w(t,0),\omega\}\,\text{var}(\omega)^{-1}$, $G(t)=\text{var}\{w(t,0)|\omega\}$ and in (5.1), let $\hat{x}(0)$ be the generalized least squares estimate of $x(0)$ and $\hat{\Sigma}=\text{var}\{\hat{x}(0)\}/\sigma^2$. Then WA obtain

$$\hat{x}(t|N)=D(t)y+\{\phi(t,0)-D(t)B\}\hat{x}(0) \tag{7.2}$$

and

$$\hat{S}_x(t|N)=\{\phi(t,0)-D(t)B\}\hat{\Sigma}\{\phi(t,0)-D(t)B\}'+G(t) \tag{7.3}$$

as the estimates of $x(t)$ and its conditional variance under their stochastic model.

LEMMA 7.1. *Algorithm* 4.3 *is equivalent to that of* WA *in the sense that*

$$\hat{x}(t|N)=x^{(0)}(t|N)\quad \textit{and}\quad \hat{S}_x(t|N)=S_x^{(0)}(t|N). \tag{7.4}$$

Proof. It is not difficult to check that for the stochastic model of § 3,

$$x(t|N;k)=D(t)y+\{\phi(t,0)-D(t)B\}x(0|N;k) \tag{7.5}$$

and

$$S_x(t|N;k)=\{\phi(t,0)-D(t)B\}S_x(0|N;k)\{\phi(t,0)-D(t)B\}'+G(t). \tag{7.6}$$

Now it is well known (see, for example Box and Tiao (1973, Chap. 2)), that

$$\hat{x}(0)=x^{(0)}(0|N),\qquad \hat{\Sigma}=S_x^{(0)}(0|N). \tag{7.7}$$

By Lemma 3.1, $x(t|N;k)$ and $S_x(t|N;k)$ tend to $x^{(0)}(t|N)$ and $S_x^{(0)}(t|N)$ respectively as $k\to\infty$, and in particular this is true for $t=0$, so that in the limit as $k\to\infty$ substituting (7.7) in (7.2) and (7.3) and using (7.5) and (7.6) gives (7.4). □

The computation of the initial conditions $\hat{x}(0)$ and $\hat{\Sigma}$ by generalized least squares in the WA algorithm is substantially more costly than our solution through the modified filtering and smoothing algorithms.

8. A stochastic derivation of Reinsch's algorithm. Reinsch (1967) gives an $O(N)$ cubic spline smoothing algorithm by exploiting the special structure of two tridiagonal matrices. Reinsch's derivation of his algorithm is algebraic. We give an alternative

derivation of Reinsch's algorithm based on our stochastic model. In particular we show that after taking second quasi-divided differences of our stochastic process we obtain a nonstationary moving average process of order one. This explains the origins of the two tridiagonal matrices mentioned above.

Let the $(N-m)\times N$ matrix J_2, and the $(N-m)\times 1$ and $N\times 1$ vectors z and y be defined as in § 5, and let $e=\{e(1),\cdots,e(N)\}'$, $\hat{f}=\{\hat{f}(t_1),\cdots,\hat{f}(t_N)\}'$ and $f=\{f(t_1),\cdots,f(t_N)\}'$. Then for polynomial splines $f-y=-e$, and because e is independent of $x(0)$ it follows from Theorem 5.2 of AK that

$$\hat{f}=y-E(e|z)=y-\operatorname{cov}(e,z)\operatorname{var}(z)^{-1}z \tag{8.1}$$

so that the right side of (8.1) gives the polynomial spline evaluated at the points $t_1,\cdots,t_N$.

From now on we deal only with the cubic spline case and obtain an explicit expression for $\hat{f}$ in (8.1) by constructing an appropriate J_2 and z. From § 3 it is clear that for this case the state vector has dimension 2. As in Reinsch (1967), we will also assume that $t_i<t_{i+1}$ for $i=1,\cdots,N-1$.

For $i=2,\cdots,N$, let $h(i)=t_i-t_{i-1}$, and let $w_1(t,s)$ and $w_2(t,s)$ be the first and second elements of $w(t,s)$. For $i=1,\cdots,N-2$, define the following second quasi-divided differences:

$$\Delta_2 y(i)=\{y(i+2)-y(i+1)\}/h(i+2)-\{y(i+1)-y(i)\}/h(i+1),$$

$$\Delta_2 f(t_i)=\{f(t_{i+2})-f(t_{i+1})\}/h(i+2)-\{f(t_{i+1})-f(t_i)\}/h(i+1),$$

and define $\Delta_2 e(i)$ similarly to $\Delta_2 y(i)$. Then

$$\Delta_2 y(i)=\Delta_2 f(t_i)+\Delta_2 e(i) \tag{8.2}$$

and it is not difficult to show that

$$\Delta_2 f(t_i)=w_1(t_{i+2},t_{i+1})/h(i+2)-w_1(t_{i+1},t_i)/h(i+1)+w_2(t_{i+1},t_i). \tag{8.3}$$

From (8.3), $\Delta_2 f(t_i)$ is a nonstationary moving average process of order one in the sense that $\operatorname{cov}\{\Delta_2 f(t_i),\Delta_2 f(t_{i+j})\}=0$ for $|j|\geqq 2$. Similarly, we can check that $\Delta_2 e(i)$ is a nonstationary moving average process of order two. For $i=1,\cdots,N-2$, let $z(i)=\Delta_2 y(i)$ and let $\eta(i)=\Delta_2 f(t_i)$, and put $z=\{z(1),\cdots,z(N-2)\}'$ and $\eta=\{\eta(1),\cdots,\eta(N-2)\}'$. Then we can write $z=J_2 y$, where J_2 is the $(N-2)\times N$ tridiagonal matrix described by

$$J_2(i,i)=1/h(i+1),\quad J_2(i,i+1)=-1/h(i+1)-1/h(i+2),\quad J_2(i,i+2)=1/h(i+2),$$

$(i=1,\cdots,N-2)$ with the rest of the elements of J_2 zero. It can be checked that J_2 has the properties described in § 5.

From (8.2), $z=\eta+J_2 e$, and because η is independent of e it follows that $\operatorname{cov}(e,z)=\sigma^2 J_2'$, and so from (8.1)

$$\hat{f}=y-\sigma^2 J_2'\operatorname{var}(z)^{-1}J_2 y. \tag{8.4}$$

From above, $z(i)$ is a nonstationary moving average process of order two and so var (z) has just its five central diagonals nonzero. Since J_2 is tridiagonal, the right side of (8.4) gives an $O(N)$ algorithm for $\hat{f}$.

We now show that (8.4) is in fact exactly Reinsch's (1967) expression for $\hat{f}$.

Let $V_\eta=2\operatorname{var}(\eta)/\kappa\sigma^2$, so that V_η is the tridiagonal symmetric matrix with

$$V_\eta(i,i)=2\{h(i+1)+h(i+2)\}/3,\qquad V_\eta(i,i+1)=h(i+2)/3.$$

Putting $\tilde{\kappa} = \kappa\sigma^2/2$, it follows that

$$\hat{f} = y - \sigma^2 J_2'(\sigma^2 J_2 J_2' + \tilde{\kappa} V_\eta)^{-1} J_2 y.$$

If we identify J_2, V_η and $\tilde{\kappa}$ with Q', T and ρ, respectively, in Reinsch (1967), we obtain exactly Reinsch's expression for $\hat{f}$.

We note the following:

(i) We can obtain a similar algorithm for polynomial splines of order $2m-1$ because the mth quasi-divided difference of $y(i)$ yields a nonstationary moving average of order m.

(ii) If the points t_i are equally spaced, so that $t_{i+1} - t_i$ is constant for all i, then $\Delta_2 f(t_i)$ and $\Delta_2 e(i)$ are stationary moving average processes, and hence $z(i)$ is a stationary moving average process or order 2.

(iii) As noted by Reinsch (1971, p. 452), the Reinsch (1967) algorithm may be numerically unstable if the $h(i)$ are small because the matrix J_2 depends on the reciprocals of the $h(i)$.

(iv) Our algorithm is much more general than that of Reinsch because it computes the smoothing spline for a general differential operator L and for general functionals of the form (2.1). Also Reinsch's algorithm does not give Bayesian confidence intervals for either the unobserved $f(t)$ or its derivatives, whereas our algorithm can do so routinely.

(v) Our Algorithm 4.3 using the Kalman filter and the fixed interval smoothing algorithm, implicitly computes a J_2 matrix which is different from above and numerically more stable.

Appendix.

ALGORITHM FOR THE CHOLESKY FACTORS IN LEMMA 4.1. The following procedure is given in Corollary 10.2 of AK. Let $L_{ij}^{(0)}$, $L_{ij}^{(-1,1)}$, $\Omega_{ij}^{(1)}$ and $\Omega_{ij}^{(0)}$ be the ijth elements of $L^{(0)}$, $L^{(-1,1)}$, $\Omega^{(1)}(t_{i+1}, t|i)$ and $\Omega^{(0)}(t_{i+1}, t|i)$, respectively; let $\lambda_j^{(1)}$ and $\lambda_j^{(0)}$ be the jth diagonal elements of $\Lambda^{(1)}$ and $\Lambda^{(0)}$, respectively. We construct $L^{(0)}$, $L^{(-1,1)}$, $\Lambda^{(1)}$ and $\Lambda^{(0)}$ as follows:

$$\lambda_1^{(1)} = \Omega_{11}^{(1)} \text{ and } \lambda_1^{(0)} = \Omega_{11}^{(0)},$$

for $i = 2, \cdots, 2m$.

For $j = 1, \cdots, i-1$:

Case 1: $\lambda_j^{(1)} > 0$.

$$L_{ij}^{(0)} = \left\{ \Omega_{ij}^{(1)} - \sum_{h=1}^{j-1} L_{ih}^{(0)} L_{jh}^{(0)} \lambda_h^{(1)} \right\} \Big/ \lambda_j^{(1)},$$

$$L_{ij}^{(-1,1)} = \left[\Omega_{ij}^{(0)} - \sum_{h=1}^{j-1} \{ (L_{ih}^{(0)} L_{jh}^{(-1,1)} + L_{ih}^{(-1,1)} L_{jh}^{(0)}) \lambda_h^{(1)} + L_{ih}^{(0)} L_{jh}^{(0)} \lambda_h^{(0)} \} - L_{ij}^{(0)} \lambda_j^{(0)} \right] \Big/ \lambda_j^{(1)}.$$

Case 2: $\lambda_j^{(1)} = 0$, $\lambda_j^{(0)} > 0$.

$$L_{ij}^{(0)} = \left[\Omega_{ij}^{(0)} - \sum_{h=1}^{j-1} \{ (L_{ih}^{(0)} L_{jh}^{(-1,1)} + L_{ih}^{(-1,1)} L_{jh}^{(0)}) \lambda_h^{(1)} + L_{ih}^{(0)} L_{jh}^{(0)} \lambda_h^{(0)} \} \right] \Big/ \lambda_j^{(0)},$$

$$L_{ij}^{(-1,1)} = 0.$$

Case 3: $\lambda_j^{(1)} = 0$, $\lambda_j^{(0)} = 0$: $L_{ij}^{(0)} = L_{ij}^{(-1,1)} = 0$.

$$L_{ii}^{(0)} = 1,\ L_{ii}^{(-1,1)} = 0,$$

$$\lambda_i^{(1)} = \Omega_{ii}^{(1)} - \sum_{h=1}^{i-1} \{L_{ih}^{(0)}\}^2 \lambda_h^{(1)},$$

$$\lambda_i^{(0)} = \Omega_{ii}^{(0)} - \sum_{h=1}^{i-1} [2L_{ih}^{(0)} L_{ih}^{(-1,1)} \lambda_h^{(1)} + \{L_{ih}^{(0)}\}^2 \lambda_h^{(0)}].$$

If $i \geqq i_*$, then $\Omega^{(1)}(t_{i+1}, t|i) = 0$, and $L^{(0)}$ and $\Lambda^{(0)}$ as obtained above are simply the Cholesky factors of $\Omega^{(0)}(t_{i+1}, t|i)$.

Acknowledgment. We would like to thank the two referees for helping to improve the presentation of the paper, and also for encouraging us to relate Reinsch's algorithm to the stochastic model of § 3.

REFERENCES

C. F. ANSLEY AND R. KOHN, *Estimation, filtering and smoothing in state space models with incompletely specified initial conditions*, Annals Statist., 11 (1985), pp. 1286–1316.

———, *The equivalence of two stochastic approaches to spline smoothing*, Technical Report 19, Statist. Res. Ctr., Grad. School of Business, Univ. Chicago, Chicago, IL, 1984.

———, *The equivalence of two stochastic approaches to spline smoothing*, J. Appl. Probab., 23A (1986), pp. 391–405.

G. E. P. BOX AND G. C. TIAO, *Bayesian Inference in Statistical Analysis*, Addison-Wesley, Reading, MA, 1973.

P. CRAVEN AND G. WAHBA, *Smoothing noisy data with spline functions*, Numer. Math., 31 (1979), pp. 377–403.

H. HOCHSTADT, *Differential Equations: A Modern Approach*, Holt, Rinehart and Winston, New York, 1964.

M. F. HUTCHINSON AND F. R. DE HOOG, *Smoothing noisy data with spline functions*, Numer. Math., 47 (1985), pp. 99–106.

J. D. KALBFLEISCH AND D. A. SPROTT, *Application of likelihood methods to models involving large numbers of parameters*, J. Roy. Statist. Soc., Ser. B, 32 (1970), pp. 175–194.

R. KOHN AND C. F. ANSLEY, *On the smoothness properties of the best linear unbiased estimate of a stochastic process observed with noise*, Annals Statist., 11 (1983), pp. 1011–1017.

———, *Estimation, prediction and interpolation for ARIMA models with missing data*, J. Amer. Statist. Assoc., 81 (1986), to appear.

C. H. REINSCH, *Smoothing by spline functions*, Numer. Math., 10 (1967), pp. 177–183.

———, *Smoothing by spline functions* II, Numer. Math., 16 (1971), pp. 451–454.

B. W. SILVERMAN, *A fast and efficient cross-validation method for smoothing parameter choice in spline regression*, J. Amer. Statist. Assoc., 79 (1984), pp. 584–589.

G. WAHBA, *Improper priors, spline smoothing and the problem of guarding against model errors in regression*, J. Roy. Statist. Soc., Ser. B, 40 (1978), pp. 364–372.

———, *Bayesian confidence intervals for the cross-validated smoothing spline*, J. Roy. Statist. Soc., Ser. B, 45 (1983), pp. 133–150.

———, *A comparison of GCV and GML for choosing the smoothing parameter in the generalized spline smoothing problem*, Annals Statist., 11 (1985), pp. 1378–1402.

W. WECKER AND C. F. ANSLEY, *The signal extraction approach to nonlinear regression and spline smoothing*, J. Amer. Statist. Assoc., 78 (1983), pp. 81–89.

H. L. WEINERT AND G. S. SIDHU, *On uniqueness conditions for optimal curve fitting*, J. Optim. Theory Appl., 23 (1977), pp. 211–216.

H. L. WEINERT, V. B. DESAI AND G. S. SIDHU, *ARMA splines, system inverses and least squares estimates*, SIAM J. Control Optim., 17 (1979), pp. 525–536.

H. L. WEINERT, R. H. BYRD AND G. S. SIDHU, *A stochastic framework for recursive computation of spline functions: Part* II, *Smoothing splines*, J. Optim. Theory Appl., 30 (1980), pp. 255–268.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 1, January 1987
© 1987 Society for Industrial and Applied Mathematics
004

A GENERALIZED GAUSS–NEWTON PROCEDURE FOR MULTI-RESPONSE PARAMETER ESTIMATION*

DOUGLAS M. BATES† AND DONALD G. WATTS‡

Abstract. A method is presented for calculating the gradient and an approximate Hessian of the Box–Draper multi-response parameter estimation criterion using only the first-order derivatives of the model functions. This is an analogue of the Gauss–Newton iterative procedure for nonlinear least squares. We also describe an implementation based on a *QR* decomposition of the residual matrix which allows incorporation of a regularization procedure similar to the Levenburg–Marquardt method in nonlinear least squares.

The method incorporates a convergence criterion based on a comparison of the increment size to the statistical variability of the estimates.

Key words. nonlinear optimization, determinant, convergence criterion

AMS(MOS) subject classifications. 65U05, 62J02, 62F10

1. Introduction. An important part of the statistical analysis of experimental data is the estimation of parameters in mathematical models. While linear regression models are the easiest to specify and to fit to data, many statistical packages offer the ability to fit other types of models such as nonlinear regression models (e.g. SAS [20], BMDP [11]) and generalized linear models (GLIM [1]). The statistical models implicit in these packages are based on a single response and, generally, several inputs or regressor variables. Sometimes, however, more than one measured variable could be considered as a response: for example, one could measure the concentration of a number of different chemical species in a chemical reaction at each of several different times during the reaction (Byrne et al. [8]).

Box and Draper [5] considered the problem of parameter estimation for such multi-response data and, through a Bayesian argument, derived an estimation criterion. If there are N observations on each of K different responses, the $N \times K$ matrix of observed responses is written $\mathbf{Y}$. The kth expected response at the nth experimental setting $\mathbf{X}_n$ is assumed to depend on a P-dimensional vector of parameters, $\boldsymbol{\theta}$, and is written

$$\eta_{nk} = \eta_k(\mathbf{X}_n, \boldsymbol{\theta}).$$

For fixed experiment settings, these can be collected into an $N \times K$ matrix $\mathbf{H}(\boldsymbol{\theta}) = \{\eta_{nk}\}$. The parameter estimates, $\hat{\boldsymbol{\theta}}$ are chosen to minimize the $K \times K$ determinant

$$|\mathbf{Z}'\mathbf{Z}| = |(\mathbf{Y} - \mathbf{H}(\boldsymbol{\theta}))'(\mathbf{Y} - \mathbf{H}(\boldsymbol{\theta}))| \tag{1.1}$$

where the vertical bars indicate the determinant, the prime denotes the transpose, and $\mathbf{Z}(\boldsymbol{\theta})$ is the residual matrix $\mathbf{Y} - \mathbf{H}(\boldsymbol{\theta})$. The posterior density for the parameters is then $f(\boldsymbol{\theta}) \propto |\mathbf{Z}'\mathbf{Z}|^{-N/2}$.

This type of objective function is also used in experimental design for nonlinear models using the D-optimal design criterion (Box and Lucas [4]).

* Received by the editors February 21, 1984, and in revised form August 13, 1985. This work was supported by the Natural Sciences and Engineering Research Council of Canada and the National Science Foundation under research grant MCS-8102732.

† University of Wisconsin, Madison, Wisconsin 53706.

‡ Queen's University, Kingston, Ontario, Canada K7L 3N6.

One advantage of single-response, relative to multi-response, parameter estimation is that easily-used software for nonlinear least squares optimization is available (e.g. the Gauss–Newton and Levenberg–Marquardt algorithms, Kennedy and Gentle [16], Moré [18], and the code given in Dennis and Schnabel [10] and in Moré et al. [19]). In this paper we present a procedure for multi-response parameter estimation which generalizes the Gauss–Newton method for single-response nonlinear least squares so that, given values for the expectation functions η_{nk}, $k = 1, \cdots, K$, and the derivatives of the expectation function with respect to the parameters

$$\frac{\partial \eta_{nk}}{\partial \theta_p}, \qquad k = 1, \cdots, K, \quad p = 1, \cdots, P,$$

the algorithm approximates both the gradient and the Hessian of the objective function so that a step size and direction can be calculated. A modified version of the procedure, analogous to the Levenberg–Marquardt comprise, is easy to implement if required.

2. Determining the gradient and the Hessian. We require the gradient and Hessian of $|\mathbf{Z}'\mathbf{Z}|$, which involves partial derivatives, and so we simplify notation by using a subscript to indicate the partial derivative of a quantity with respect to a component of $\boldsymbol{\theta}$, and omit explicit reference to dependencies on $\boldsymbol{\theta}$. Thus, for example, we write

$$\frac{\partial \mathbf{Z}(\boldsymbol{\theta})}{\partial \theta_p} = \mathbf{Z}_p = -\mathbf{H}_p.$$

Now, the gradient of

$$\tfrac{1}{2} \log |\mathbf{Z}'\mathbf{Z}| \tag{2.1}$$

was derived in [3], with elements calculated as

$$g_p = \operatorname{tr}(\mathbf{Z}^+\mathbf{Z}_p) = -\operatorname{tr}(\mathbf{Z}^+\mathbf{H}_p) \tag{2.2}$$

where the + superscript denotes the pseudo-inverse.

Using expressions for the derivative of the pseudo-inverse [15], we can write the (p, q)th component of the Hessian as

$$\begin{aligned} &-\operatorname{tr}[\mathbf{Z}^+\mathbf{Z}_q\mathbf{Z}^+\mathbf{Z}_p] + \operatorname{tr}[\mathbf{Z}^+\mathbf{Z}^{+\prime}\mathbf{Z}_q'(\mathbf{I}-\mathbf{Z}\mathbf{Z}^+)\mathbf{Z}_p] + \operatorname{tr}[\mathbf{Z}^+\mathbf{Z}_{p,q}] \\ &\quad = -\operatorname{tr}[\mathbf{Z}^+\mathbf{H}_q\mathbf{Z}^+\mathbf{H}_p] + \operatorname{tr}[\mathbf{Z}^+\mathbf{Z}^{+\prime}\mathbf{H}_q'(\mathbf{I}-\mathbf{Z}\mathbf{Z}^+)\mathbf{H}_p] - \operatorname{tr}[\mathbf{Z}^+\mathbf{H}_{p,q}]. \end{aligned} \tag{2.3}$$

As with the Gauss–Newton algorithm for nonlinear least squares, we approximate the Hessian by the expression in (2.3) with the second-order derivatives of $\mathbf{H}$ replaced by zero. This gives an approximate Hessian matrix $\mathbf{G}$ with (p, q)th component

$$g_{p,q} = -\operatorname{tr}[\mathbf{Z}^+\mathbf{H}_p\mathbf{Z}^+\mathbf{H}_q] + \operatorname{tr}[\mathbf{Z}^+\mathbf{Z}^{+\prime}\mathbf{H}_p'(\mathbf{I}-\mathbf{Z}\mathbf{Z}^+)\mathbf{H}_q]. \tag{2.4}$$

$\mathbf{G}$ will be exact when all the expected responses $\mathbf{H}$ depend linearly on $\boldsymbol{\theta}$.

Since

$$|\mathbf{Z}'\mathbf{Z}| = \exp(\log |\mathbf{Z}'\mathbf{Z}|), \tag{2.5}$$

the components of the gradient $\boldsymbol{\gamma}$, and of the approximate Hessian $\boldsymbol{\Gamma}$, of $|\mathbf{Z}'\mathbf{Z}|$ are

$$\boldsymbol{\gamma}_p = 2|\mathbf{Z}'\mathbf{Z}|g_p, \tag{2.6}$$

and

$$\boldsymbol{\Gamma}_{p,q} = 2|\mathbf{Z}'\mathbf{Z}|[g_{p,q} + 2g_pg_q], \tag{2.7}$$

where g_p and $g_{p,q}$ are given in (2.2) and (2.4). The increment is then $\boldsymbol{\delta} = -\boldsymbol{\Gamma}^{-1}\boldsymbol{\gamma}$. Note that the scalar factor $2|\mathbf{Z}'\mathbf{Z}|$ cancels and so it is not necessary to include it in $\boldsymbol{\gamma}$ and $\boldsymbol{\Gamma}$.

3. Implementation. A numerically stable way of implementing the calculations is to form a QR decomposition of the matrix Z at each iteration,

$$\mathbf{Z} = \tilde{\mathbf{Q}}\tilde{\mathbf{R}}, \tag{3.1}$$

where $\tilde{\mathbf{Q}}$ is $N \times N$ and orthogonal and $\tilde{\mathbf{R}}$ is $N \times K$ and zero below the main diagonal. We assume that $\tilde{\mathbf{R}}$ has full rank, since a rank-deficient $\mathbf{Z}$ matrix would indicate either an absolute minimum for the objective function, or, more likely, an unstable estimation situation where deterministic relationships in the data are reflected in the expected responses as described in [17]. With this decomposition, the determinant becomes

$$|\mathbf{Z}'\mathbf{Z}| = \prod_{k=1}^{K} r_{k,k}^2, \tag{3.2}$$

and the pseudo-inverse $\mathbf{Z}^+$

$$\mathbf{Z}^+ = \mathbf{R}^{-1}\mathbf{Q}', \tag{3.3}$$

where $\mathbf{Q}$ is the first K columns of $\tilde{\mathbf{Q}}$ and $\mathbf{R}$ is the first K rows of $\tilde{\mathbf{R}}$.

Even better economy in the calculation of the gradient and approximate Hessian can be obtained by first calculating

$$\tilde{W}_p = \tilde{\mathbf{Q}}'\mathbf{Z}_p\mathbf{R}^{-1} \tag{3.4}$$

since

$$g_p = \operatorname{tr}(\mathbf{Z}^+\mathbf{Z}_p) = \operatorname{tr}(\mathbf{R}^{-1}\mathbf{Q}'\mathbf{Z}_p) = \operatorname{tr}(\mathbf{Q}'\mathbf{Z}_p\mathbf{R}^{-1}) = \sum_{k=1}^{K} \{\tilde{W}_p\}_{k,k} \tag{3.5}$$

and, letting $\mathbf{Q}^*$ be the last $N-K$ columns of $\tilde{\mathbf{Q}}$,

$$\begin{aligned} g_{p,q} &= -\operatorname{tr}(\mathbf{Z}^+\mathbf{Z}_p\mathbf{Z}^+\mathbf{Z}_q) + \operatorname{tr}(\mathbf{Z}^+\mathbf{Z}^{+\prime}\mathbf{Z}_q'\mathbf{Q}^*\mathbf{Q}^{*\prime}\mathbf{Z}_p) \\ &= -\operatorname{tr}(\mathbf{Q}'\mathbf{Z}_p\mathbf{R}^{-1}\mathbf{Q}'\mathbf{Z}_q\mathbf{R}^{-1}) + \operatorname{tr}(\mathbf{Q}\mathbf{R}'^{-1}\mathbf{Z}_q'\mathbf{Q}^*\mathbf{Q}^{*\prime}\mathbf{Z}_p\mathbf{R}^{-1}\mathbf{Q}') \\ &= -\sum_{j=1}^{K}\sum_{k=1}^{K} \{\tilde{W}_p\}_{j,k}\{\tilde{W}_q\}_{k,j} + \operatorname{tr}[(\mathbf{Q}^{*\prime}\mathbf{Z}_q\mathbf{R}^{-1})'(\mathbf{Q}^{*\prime}\mathbf{Z}_p\mathbf{R}^{-1})] \\ &= -\sum_{j=1}^{K}\sum_{k=1}^{K} \{\tilde{W}_p\}_{j,k}\{\tilde{W}_q\}_{k,j} + \sum_{k=1}^{K}\sum_{j=K+1}^{N} \{\tilde{W}_p\}_{j,k}\{\tilde{W}_q\}_{j,k}. \end{aligned} \tag{3.6}$$

The values for g_p and $g_{p,q}$ may now be substituted into expressions (2.6) and (2.7) to obtain the gradient γ_p and the approximate Hessian Γ, from which the increment $\boldsymbol{\delta} = -\boldsymbol{\Gamma}^{-1}\boldsymbol{\gamma}$ can be calculated.

4. Convergence. An important part of any nonlinear optimization algorithm is the choice of convergence criterion. In [2] a statistically meaningful convergence criterion for nonlinear least squares was derived through a geometrical argument considering the inaccuracy of the current parameter estimates relative to the statistical variability in the estimates.

To determine a convergence criterion in the multiresponse case, we expand the posterior density for the parameters about $\hat{\boldsymbol{\theta}}$,

$$\begin{aligned} f(\boldsymbol{\theta}) &\propto |\mathbf{Z}'\mathbf{Z}|^{-N/2} \\ &\cong |\hat{\mathbf{Z}}'\hat{\mathbf{Z}}|^{-N/2}\left\{1 + (\boldsymbol{\theta}-\hat{\boldsymbol{\theta}})'\frac{\boldsymbol{\Gamma}}{2|\mathbf{Z}'\mathbf{Z}|}(\boldsymbol{\theta}-\hat{\boldsymbol{\theta}})\right\}^{-N/2}. \end{aligned} \tag{4.1}$$

This is in the form of a P-variate Student's T density with location parameter $\hat{\boldsymbol{\theta}}$, degrees of freedom $N-P$, scale factor $s^2=|\mathbf{Z}'\mathbf{Z}|/(N-P)$, and covariance matrix $2s^2\boldsymbol{\Gamma}^{-1}$, where $\boldsymbol{\Gamma}$ and $|\mathbf{Z}'\mathbf{Z}|$ are evaluated at $\hat{\boldsymbol{\theta}}$, [7]. An approximate $(1-\alpha)$ posterior density region for the parameters is given by

$$(\boldsymbol{\theta}-\hat{\boldsymbol{\theta}})'(\boldsymbol{\Gamma}/2)(\hat{\boldsymbol{\theta}}-\boldsymbol{\theta}) \leqq Ps^2F(P, N-P; \alpha), \tag{4.2}$$

where $F(P, N-P; \alpha)$ is the upper 100α percentage point of Fisher's F distribution with P and $N-P$ degrees of freedom.

If we declare convergence at a particular parameter vector $\boldsymbol{\theta}^0$, the statistical variability will be represented by elliptical regions based on the approximate variance-covariance matrix evaluated at $\boldsymbol{\theta}^0$. The magnitude of the increment relative to such a region is best calculated by taking a linear transformation of the parameters to make the region spherical. When this transformation is made, the length of the increment from $\boldsymbol{\theta}^0$ is $\|\mathbf{C}\boldsymbol{\delta}\|$, where $\mathbf{C}$ is the Cholesky factor of $\boldsymbol{\Gamma}$ and $\boldsymbol{\delta}$ is the increment in the original parameters. This results in the convergence criterion

$$\frac{\|\mathbf{C}\boldsymbol{\delta}\|^2/P}{2|\mathbf{Z}'\mathbf{Z}|/(N-P)}=\frac{(N-P)\|\mathbf{C}\boldsymbol{\delta}\|^2}{2P|\mathbf{Z}'\mathbf{Z}|}<\varepsilon^2, \tag{4.3}$$

where ε^2 is the tolerance level. When the criterion is small, it indicates that the requested increment is negligible relative to the statistical variability. The value of ε can be set at .001, say, using similar reasoning to that in [2].

Note that the determinant criterion implies two constraints on the number of observations. First, $N \geqq K$ since, otherwise, the determinant is identically zero. Second, $N > P$ since, otherwise, the determinant can be made equal to zero by fitting any one response perfectly and this can generate up to K distinct minima with equal values of the criterion. Furthermore, the multi-response nature of the data is not being used.

5. Practical considerations. Multiresponse estimation, like other nonlinear iterative procedures, is subject to practical difficulties [6], [17]. One common computing difficulty encountered in multiresponse estimation, but not in single-response nonlinear least squares, is a Hessian which is not positive definite. This is detected when the Cholesky decomposition of $\boldsymbol{\Gamma}$ cannot be calculated and indicates the need for a regularization procedure like the Levenberg-Marquardt procedure in nonlinear least squares [16]. One simple regularization procedure is to calculate the eigenvalues of Γ and add twice the magnitude of the eigenvalue which is most negative to the diagonal of Γ. This may result in a modified Hessian which is nearly singular. An alternative, safer and probably faster, choice of inflation factor is described in Gay [13] with additional discussion on other modified Cholesky decompositions in Gill et al. [14], Dennis and Schnabel [10], and Fletcher [12].

The situation of having K distinct minima, described in the previous section, would necessarily give rise to indefinite Hessians at some points in the parameter space. While multiple minima are not always encountered with this objective function when $N > P$, they are not uncommon. One strategy to determine if there are multiple minima is to use several starting points which span the region of reasonable parameter values. If convergence to the same minimum occurs, then one has some assurance that there is only one minimum. There is, of course, no guarantee that all local minima will be found in this way. We must determine from the context of the data if the minimum which is determined by the algorithm seems reasonable.

A second important practical consideration is detecting and removing singularities in the data by means of eigenvalue-eigenvector analyses at the start of an analysis, as

explained by Box et al. [6]. This is vital since spurious multiple local minima of the determinant can occur with consequent convergence to unsatisfactory parameter values.

A third practical consideration is deciding what summary information should be printed out after convergence. Clearly, one should print the converged values, $\hat{\boldsymbol{\theta}}$, their approximate standard errors given by the square root of the diagonal values of $2|\mathbf{Z}'\mathbf{Z}|\boldsymbol{\Gamma}^{-1}/(N-P)$, and the approximate parameter correlation matrix derived from $\boldsymbol{\Gamma}^{-1}$. It is also useful to print the eigenvalues and eigenvectors of the matrix $\boldsymbol{\Gamma}^{-1}$ to provide insight into the multidimensional distributional behavior of the parameters. Another useful quantity is the matrix $\mathbf{Z}'\mathbf{Z}/(N-P)$, which provides information on the variances and covariances of the disturbances: it is helpful to present this in the form of the disturbance standard deviations (square root of the diagonal terms of $\mathbf{Z}'\mathbf{Z}/(N-P)$), the correlations of the disturbances ($\mathbf{Z}'\mathbf{Z}$ scaled to unit diagonal terms), and an eigenvalue-eigenvector decomposition, as discussed in [6]. Plots of residuals versus expected and control variables should be made, as well as normal probability plots.

An example. To illustrate some of the difficulties which can occur in multiresponse estimation, we consider a simple example in which the expected responses are linear in the parameters,

$$\boldsymbol{\eta}_t = (\beta_0+\beta_1 x_{t1},\ \beta_0+\beta_1 x_{t2},\ \beta_0+\beta_1 x_{t3})'.$$

Eight observation triplets were simulated with $\boldsymbol{\beta} = (0.5, 2.0)'$ and are recorded in Appendix 1. The disturbances were normally distributed with 0 means, independence between rows, variances 1, 2 and 5, and contemporaneous correlations $\rho_{12} = 0.71$, $\rho_{13} = 0.89$ and $\rho_{23} = 0.63$.

Three sets of starting values for the estimation procedure were obtained by using linear least squares on each individual response. Even for this simple model and data

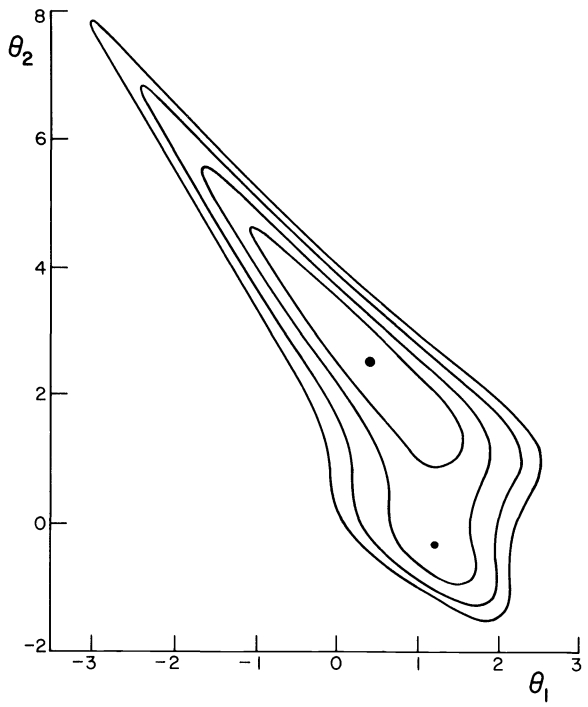


FIG. 1. *Determinant contours for a linear model.*

set, an indefinite Hessian was obtained at a number of points, necessitating use of the regularization procedure discussed above. Also, two points of convergence were found (1.23, −0.31) with a determinant of 677, and (0.41, 2.55) with a determinant of 569.

In Fig. 1 we show a contour plot of the determinant as a function of the parameters, with contours corresponding to approximately 50, 75, 90 and 95% confidence regions. The two minima are evident, as is a saddle point where an indefinite Hessian would occur.

Appendix 1: Data for linear model.

x_{t1}	x_{t2}	x_{t3}
0.4076	0.5505	0.2129
0.8408	0.4085	0.5591
0.5017	0.7217	0.4305
0.9129	0.4769	0.0228
0.4436	0.6393	0.5761
0.5984	0.9964	0.7149
0.7742	0.1575	0.9322
0.7922	0.5351	0.1233
y_{t1}	y_{t2}	y_{t3}
−0.2875	0.4633	0.0423
3.5302	3.5414	2.8750
0.5428	1.5101	−1.7595
3.7360	3.4240	2.1991
0.8253	0.2383	1.0350
2.0930	1.0528	2.2779
3.2956	1.5196	4.4792
2.6799	3.3145	−0.7375

Acknowledgments. The authors wish to acknowledge helpful comments from the Associate Editor and a reviewer.

REFERENCES

[1] R. J. BAKER AND J. A. NELDER, *The* GLIM *System, Release* 3, Numerical Algorithms Group, Oxford, 1978.

[2] D. M. BATES AND D. G. WATTS, *A relative offset orthogonality convergence criterion for nonlinear least squares*, Technometrics, 23 (1981), pp. 179-183.

[3] D. M. BATES, *The derivative of $|X'X|$ and its uses*, Technometrics, 25 (1983), pp. 373-376.

[4] G. E. P. BOX AND H. L. LUCAS, *Design of experiments in non-linear situations*, Biometrika, 46 (1959), pp. 77-90.

[5] G. E. P. BOX AND N. R. DRAPER, *The Bayesian estimation of common parameters from several responses*, Biometrika, 52 (1965), pp. 355-365.

[6] G. E. P. BOX, W. G. HUNTER, J. F. MACGREGOR AND J. ERJAVEC, *Some problems associated with the analysis of multiresponse data*, Technometrics, 15 (1973), pp. 33-51.

[7] G. E. P. BOX AND G. C. TIAO, *Bayesian Inference in Statistical Analysis*, Addison-Wesley, Reading, MA, 1973, p. 117.

[8] G. D. BYRNE, A. J. DEGREGORIA AND D. E. SALANE, *A program for fitting rate constants in gas phase chemical kinetics models*, this Journal, 5 (1984), pp. 642-657.

[9] J. E. DENNIS, D. M. GAY AND R. E. WELSCH, *Algorithm* 573. NL2SOL, *An adaptive nonlinear least-squares algorithm*, ACM Trans. Math. Software, 7 (1981), pp. 348-368 and 369-383.

[10] J. E. DENNIS, JR. AND R. B. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[11] W. J. DIXON, M. B. BROWN, L. ENGLEMAN, J. W. FRANE, M. A. HILL, R. I. JENNRICH AND J. D. TOPOREK, BMDP *Statistical Software*, University of California Press, Berkeley, CA, 1983.

[12] R. FLETCHER, *Practical Methods of Optimization*, Vol. 1, John Wiley, New York, 1981.

[13] D. M. GAY, *Computing optimal locally constrained steps*, this Journal, 2 (1981), pp. 186-197.

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

[15] G. H. GOLUB AND V. PEREYRA, *The differentiation of pseudo-inverses and nonlinear least squares problems whose variables separate*, SIAM J. Numer. Anal., 10 (1973), pp. 413-432.

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

[17] D. D. MCLEAN, J. PRITCHARD, D. W. BACON AND J. DOWNIE, *Singularities in multiresponse modelling*, Technometrics, 21 (1979), pp. 291-298.

[18] J. J. MORÉ, *The Levenberg-Marquardt algorithm: implementation and theory*, in Lecture Notes in Mathematics 630, Numerical Analysis, G. Watson, ed., Springer-Verlag, New York, 1978, pp. 105-116.

[19] J. J. MORÉ, B. S. GARBOW AND K. E. HILLSTROM, User Guide for MINPACK-1, Report ANL-80-74, Argonne National Laboratory, Argonne, IL, 1980.

[20] A. A. RAY, SAS *User's Guide: Basics*, 1982 ed., SAS Institute Inc., Cary, NC, 1982.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 1, January 1987
© 1987 Society for Industrial and Applied Mathematics

MAXIMUM LIKELIHOOD ESTIMATION OF PROBABILISTIC CHOICE MODELS*

DAVID S. BUNCH†

Abstract. Probabilistic choice models are used by social scientists in the analysis of individual choice behavior involving discrete choice alternatives. The development of new complex choice models has given rise to a need for algorithms and software for parameter estimation of choice models. One important estimator is the maximum likelihood estimator, which historically has been avoided due to the necessity of solving a nonlinear optimization problem. New algorithms for maximum likelihood estimation of a general probabilistic choice model that exploit special structure are developed; the MLE problem for choice models can be written as a problem in generalized regression. The algorithms are implemented using state-of-the-art nonlinear optimization techniques including model/trust regions, least-change secant updates, and model switching. The algorithms perform well in tests on the multinomial logit model, the elimination-by-aspects model, and a new model of market share by Batsell and Polking.

Key words. Multinomial logit, maximum likelihood estimation, nonlinear optimization, generalized regression, model switching

AMS(MOS) subject classifications. 65K05, 65U05

1. Introduction.

The analysis of choice behavior is a major concern in such disciplines as psychology, transportation science, marketing research, and especially economics. Traditional microeconomic analysis deals with consumer demand for goods under the assumption that the quantities demanded can be measured in terms of continuous variables (e.g., pounds of meat, gallons of gasoline). In recent years, however, economists and others have become increasingly concerned with the behavior of individual economic units faced with choice decisions involving discrete responses. Examples are: choice of an automobile, transportation mode for commuting to work, family size, occupation, and destination for a shopping trip.

The need to analyze these types of problems has generated a significant amount of work on probabilistic choice models. The most familiar and widely-used of these is the multinomial logit (MNL) model, which is related to a model discussed by Luce [17] in his seminal work on individual choice behavior. The MNL and Luce models share some bothersome limitations due to an assumption of independence of irrelevant alternatives (IIA), and this has given rise to the development of choice models involving much more complex functional forms and larger numbers of parameters.

In order to use these models in actual practical applications, it is necessary to be able to estimate their parameters. Estimators with the most desirable statistical properties invariably require that a nonlinear optimization problem be solved; the estimator discussed here is the maximum likelihood estimator (MLE). Calculating MLE's has historically been a nontrivial computational task at best, and at worst has been intractable. In this paper we develop and test new special-purpose algorithms for calculating MLE's for the case of a general probabilistic choice model.

Because of the interdisciplinary nature of this topic, a certain amount of background material is required to facilitate the discussion. The next section includes a brief introduction to the subject of probabilistic choice models, and develops useful

* Received by the editors May 13, 1985; accepted for publication (in revised form) April 8, 1986.

† Graduate School of Administration, University of California, Davis, California 95616. This work appears as part of the author's Ph.D. thesis, Rice University, September 1984. This research was supported by National Science Foundation grant MCS81-16779. Additional summer support was provided by Jesse H. Jones Graduate School of Administration, Rice University, Houston, Texas 77251.

notation. Section 3 defines the maximum likelihood estimator for probabilistic choice models and reviews algorithms proposed in the choice model literature for finding MLE's. In § 4 the MLE problem for choice models is formulated as a problem in generalized regression, and relationships are derived which are useful in development of new algorithms. In § 5, relevant nonlinear optimization techniques are reviewed, new algorithms are proposed, and their implementation is described. Section 6 includes numerical results for three different choice models. A summary of conclusions and a discussion of the algorithms' performance are included in § 7.

2. Probabilistic choice models.

2.1. Notation. It is useful to discuss choice models in the context of a quantal choice experiment, in which individuals are asked to make selections from sets of alternatives. The experiment consists of N trials, indexed by $n = 1, \cdots, N$. At the nth trial an individual I_n is drawn at random from a population of individuals, and is presented with a choice set C_n. The individual is asked to make a selection from the choice set a total of R_n times, thus producing a vector of selection frequencies denoted by $(S_{1n}, S_{2n}, \cdots, S_{J_n n})$, where J_n is the cardinality of the choice set C_n. Note that in many real-world experimental settings, $R_n = 1$ for $n = 1, \cdots, N$.

The eventual goal of the experimenter is to fit a model to the data obtained in the quantal choice experiment. The assumption is that there is an underlying vector of probabilities $(P_{1n}, P_{2n}, \cdots, P_{J_n n})$, associated with the elements of the choice set, and that the probability vector is a function of the characteristics of the individual and the attributes of the alternatives. Let the matrix Z_n denote the explanatory data for the nth trial of the experiment. Then the probability that individual I_n selects alternative j from choice set C_n is denoted by $P(j|\theta, Z_n)$, where θ is a parameter vector defining the model. In other words, we denote a general probabilistic choice model by $P(j|\theta, Z), j \in C$.

Once the data are collected in the choice experiment, there are two important tasks to be performed as part of an analysis of the choice behavior: (i) selection of a specific functional form for $P(j|\theta, Z)$, and (ii) estimation of the parameter vector θ. Functional forms are discussed in the next section.

2.2. Functional forms. A simple and elegant functional form is afforded by the following model, in which each alternative k in the choice set C is assigned a scale value v_k. The probability for alternative j is given by

$$P_j = \frac{e^{v_j}}{\sum_{k \in C} e^{v_k}}. \tag{1}$$

This model is due to Luce [17], and its simplicity derives from an underlying assumption of independence of irrelevant alternatives (IIA).

Independence of Irrelevant Alternatives Axiom
The relative odds of two choice alternatives will not be altered by the addition or removal of other alternatives from the choice set.

Although this axiom yields a model with a simple functional form, it places severe restrictions on the ability of the model to capture many common types of choice behavior, such as the effect of similarity among alternatives. In particular, it cannot explain choices in which alternatives are substitutes for one another (the well-known "red bus-blue bus" problem); see, e.g., McFadden [20, p. 222]. The Luce model also does not explain situations in which one alternative dominates another ("dominance"),

or in which the addition of an alternative to a choice set increases the choice probability of one of the original members ("irregularity"). For a discussion of violations of the IIA axiom, see Batsell and Polking [2].

For the purposes of modeling, it is desirable to develop functional forms based on explanatory variables rather than scale values. A general extension to the model in (1) is to replace the scale values (v_k's) by general nonlinear functions $v(\theta, z_k)$, $k \in C$, where z_k is a vector of explanatory data associated with alternative k:

$$P(j|\theta, Z) = \frac{e^{v(\theta, z_j)}}{\sum_{k \in C} e^{v(\theta, z_k)}}. \tag{2}$$

In the particular case where $v(\theta, z_k)$ is assumed to be linear, (2) reduces to the well-known multinomial logit (MNL) model

$$P(j|\theta, Z) = \frac{e^{z_j^T \theta}}{\sum_{k \in C} e^{z_k^T \theta}}. \tag{3}$$

This model has been widely used in the literature, but suffers from the same problems associated with the IIA axiom described above for the Luce model—as does any model which takes the form of (2). This need for choice models which can capture choice behavior inconsistent with the IIA axiom has stimulated work to develop new choice models. Examples of these are the elimination-by-aspects (EBA) model (Tversky [23], [24]; Tversky and Sattah [25]), a class of generalized extreme value (GEV) models (McFadden [20], [21]), the multinomial probit model (Daganzo [6]), and a new model of market share by Batsell and Polking [2].

A complete discussion of these is beyond the scope of this paper; however, a few important points stand out. These models are much more complex than the MNL/Luce models, and typically involve much larger numbers of parameters. In the case of the multinomial probit model, determination of choice probabilities involves expensive evaluation of integrals in higher dimensional space. These facts have presented difficulties to those analysts wishing to estimate parameters for these models, and have led to a need for good estimation procedures for probabilistic choice models.

3. Maximum likelihood estimation. Recall the quantal choice experiment described in § 2, and consider a typical trial n. The assumption is that the vector of observed frequencies $(S_{1n}, S_{2n}, \cdots, S_{J_n n})$ is a random vector with a multinomial distribution $M(R_n, P_{1n}, P_{2n}, \cdots, P_{J_n n})$. The P_{jn}'s are assumed to be given by a particular model with a parameter vector θ, i.e., one of those discussed in § 2.2. The maximum likelihood estimator is that $\hat{\theta}$, which is "most likely" to have produced the observed data and is defined by the problem:

$$\operatorname*{Max}_{\theta \in R^p} l(\theta) \tag{4}$$

where

$$l(\theta) = \prod_{n=1}^{N} \left[\frac{R_n!}{\prod_{j \in C_n} S_{jn}!} \prod_{j \in C_n} P_{jn}(\theta)^{S_{jn}} \right]. \tag{5}$$

Note that $l(\theta)$ is the joint probability density associated with observation of the complete data set collected in the quantal choice experiment. An equivalent problem is

$$\operatorname*{Min}_{\theta \in R^p} L(\theta) \tag{6}$$

where

$$L(\theta) = -\sum_{n=1}^{N} \sum_{j \in C_n} S_{jn} \log P_{jn}(\theta) + \text{const.} \tag{7}$$

This is a nonlinear optimization problem that must be solved by some iterative technique; the minimization form of the problem is used here to be consistent with the standard convention in numerical optimization.

Maximum likelihood estimators are known to have desirable statistical properties: under "general conditions" they are consistent, asymptotically efficient, and asymptotically normally distributed with asymptotic variance-covariance matrix equal to $E[\nabla^2 L(\theta)]$, the Fisher information matrix. (This matrix is commonly denoted by "$I(\theta)$".) For a discussion, see McFadden [20], [21].

Two standard methods used by econometricians for finding the maximum likelihood estimator are Newton's method (or, "the Newton-Raphson method") and the method of scoring, defined by the following iterations:

Newton's Method:

$$\theta_{k+1} = \theta_k - [\nabla^2 L(\theta_k)]^{-1} \nabla L(\theta_k); \tag{8}$$

Method of Scoring:

$$\theta_{k+1} = \theta_k - I(\theta_k)^{-1} \nabla L(\theta_k). \tag{9}$$

More recently, econometricians have used quasi-Newton methods with linesearches, in which the iteration is defined by

$$\theta_{k+1} = \theta_k - \alpha_k H_k^{-1} \nabla L(\theta_k) \tag{10}$$

where H_k is an approximation to the Hessian, and α_k is the length of the step taken in the search direction, determined so as to guarantee a decrease in L.

Two choices for H_k in (10) have been suggested in the literature. McFadden [18]-[21] and Daganzo [6] recommended the use of the DFP update (see, e.g., Dennis and Schnabel [11]) for solving the problem for the multinomial logit and multinomial probit models, respectively. The second choice for H_k is the sample variance-covariance matrix for the gradient of the negative log-likelihood function. This approach based on the work of Berndt, Hall, Hall and Hausman [3]—the so-called BHHH approach—has been recommended by McFadden [20], [21] for more complex choice models.

We now discuss the features of the three methods given above. The well-known advantage of (8) is that it converges q-quadratically to the solution if a good starting point is provided. However, the methods defined by (8) and (9) are not guaranteed to converge from arbitrary starting points, i.e., the methods are not globally convergent. (Although these methods can be globalized via linesearches, the econometric literature often includes only unglobalized versions.) McFadden [18] reports that (8) may display very slow convergence for the multinomial logit model.

There are other disadvantages to (8) and (9). Newton's method requires calculation of the Hessian at each iteration, which can be quite expensive for complex choice models. The method of scoring does not have fast local convergence properties.

The DFP with cubic linesearch is a global method and is in theory locally q-superlinearly convergent. A minor criticism is that, although the DFP has gained wide acceptance in econometrics, the BFGS update is preferred (see, e.g., Dennis and Schnabel [11]). The major point of concern here is that in practice a significant number of iterations may be required before the secant approximation to the Hessian is "good

enough" for the algorithm to converge. The BHHH approach, although globalized via a linesearch, has no guarantee of fast local convergence.

The remainder of this paper concerns the development, implementation, and testing of a new class of model switching algorithms for finding MLE's of probabilistic choice models.

4. Special structure in the MLE problem.

4.1. Generalized regression. When solving an optimization problem with special structure, it is often worthwhile to design an algorithm specifically tailored to exploit the structure. (The alternative is a straightforward application of general-purpose techniques.) In this section we derive formulations that lead to special-purpose algorithms for maximum likelihood estimation of probabilistic choice models.

It will be especially useful to recognize that the MLE problem can be written as a problem in generalized regression. Consider the class of data-fitting problems in which the objective function $f(\theta)$ has the special composite structure

$$f(\theta) = \rho[R(\theta)], \quad \rho: R^N \to R^1, \quad R: R^p \to R^N. \tag{11}$$

In the MLE problem, the derivatives of ρ are easy to compute, but those of R are expensive. In the context of generalized regression (see Gay [14]), the N components of R are the generalized residuals for the N data points, and $R(\theta) = (r_1(\theta), r_2(\theta), \cdots, r_N(\theta))$, where r_n is the nth generalized residual. The function ρ is a sum of N criterion functions, $\phi_n(v)$, so that f has the form

$$f(\theta) = \sum_{n=1}^{N} \phi_n(r_n(\theta)). \tag{12}$$

Selection of specific functional forms for r_n and $\phi_n(v)$ corresponds to various regression problems. For example, if $\phi_n(v) = v^2/2$ for all n, and $r_n = y_n - m(\theta, z_n)$, where $m(\theta, z)$ is a nonlinear model, then the resulting regression is the familiar nonlinear least-squares problem. The selection of $\phi_n(v)$ and r_n which yields the MLE problem for choice models is described below.

Recall that standard nonlinear optimization algorithms based on Newton's method require expressions (exact or approximate) for the gradient and Hessian of the objective function. For the function f defined in (11), the gradient and Hessian are given by

$$\nabla f(\theta) = R'(\theta)^T \nabla \rho(R(\theta)) \tag{13}$$

and

$$\nabla^2 f(\theta) = R'(\theta)^T \nabla^2 \rho(R(\theta)) R'(\theta) + \sum_{n=1}^{N} \partial_n \rho(R(\theta)) \nabla^2 r_n(\theta). \tag{14}$$

Examination of (14) motivates the derivation of specialized algorithms that do not require exact Hessian information. Note that the second term of (14) requires the calculation of N (expensive) Hessian matrices.

An alternative is to approximate the Hessian via a least-change secant update (e.g., DFP or BFGS); however, examination of (14) reveals that such an approach is tantamount to discarding useful second-order information at each step. In optimization algorithms based on gradients, R' must be obtained either analytically or via finite differences. Once R' is available, it is a simple matter to calculate the first part of the Hessian in (14).

This motivates an "intermediate" approach, namely, to calculate the first part of the Hessian at each iteration, and to approximate the remaining part via a secant update

technique. General results for algorithms of this type are presented in Dennis and Schnabel [10] and Dennis and Walker [12]. For an application to the nonlinear least-squares problem, see Dennis, Gay and Welsch [8], [9]. Details of implementing such an approach are considered in § 5. We now develop some useful formulations for the MLE problem.

4.2. MLE as a generalized regression problem. Let the generalized residual vector $R(\theta)$ consist of terms of the form

$$r_n(\theta) \equiv \prod_{j \in C_n} P_{jn}(\theta)^{S_{jn}} \tag{15}$$

and let the nth criterion function be defined by

$$\phi_n(v) \equiv -\log(v). \tag{16}$$

Then substitution of these expressions into (12) yields the negative of the log-likelihood $L(\theta)$ defined by (6). It is instructive, however, to consider the expressions in the context of (12)-(14). The function ρ is given by

$$\rho(x) = -\sum_{n=1}^{N} \log(x_n) \tag{17}$$

where $x \in R^N$, and the gradient and Hessian of ρ are as follows:

$$\nabla\rho(x) = \left[-\frac{1}{x_1}, \cdots, -\frac{1}{x_N}\right], \tag{18}$$

and

$$\nabla^2\rho(x) = \operatorname{diag}\left(\frac{1}{x_1^2}, \cdots, \frac{1}{x_N^2}\right). \tag{19}$$

In the following discussion we denote $r_n(\theta)$ by r_n for notational convenience, and define the matrix M by

$$M = \begin{bmatrix} \frac{1}{r_1}\frac{\partial r_1}{\partial \theta_1} & \cdots & \frac{1}{r_1}\frac{\partial r_1}{\partial \theta_p} \\ \vdots & & \vdots \\ \frac{1}{r_N}\frac{\partial r_N}{\partial \theta_1} & \cdots & \frac{1}{r_N}\frac{\partial r_N}{\partial \theta_p} \end{bmatrix}. \tag{20}$$

It is straightforward to obtain the following expressions for $\nabla L(\theta)$ and $\nabla^2 L(\theta)$:

$$\nabla L(\theta) = -M^T \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \tag{21}$$

and

$$\nabla^2 L(\theta) = M^T M - \sum_{n=1}^{N} \frac{1}{r_n} \nabla^2 r_n \tag{22}$$

$$\equiv H_{\mathrm{GNL}} + S_{\mathrm{GNL}}. \tag{23}$$

The above equations have been written in a form analogous to those used in the derivation of the Gauss-Newton method for nonlinear least-squares (see Dennis and

Schnabel [11, Chap. 10]). Due to this similarity, we identify this formulation as "Gauss-Newton-like," and will refer to (22)-(23) as the GNL formulation.

A second, alternative (Alt) formulation is readily derived by direct substitution of (15) for r_n in (22), yielding

$$\nabla^2 L(\theta) = H_{\text{Alt}} + S_{\text{Alt}} \tag{24}$$

where

$$[H_{\text{Alt}}]_{kl} = = \sum_{n=1}^{N} \sum_{j \in C_n} \frac{S_{jn}}{P_{jn}^2} \frac{dP_{jn}}{d\theta_k} \frac{dP_{jn}}{d\theta_l} \tag{25}$$

and

$$[S_{\text{Alt}}]_{kl} = - \sum_{n=1}^{N} \sum_{j \in C_n} \frac{S_{jn}}{P_{jn}} \frac{d^2 P_{jn}}{d\theta_k \, d\theta_l}. \tag{26}$$

These two formulations provide the basis for the new MLE algorithms developed in the next section.

5. Implementation of MLE algorithms. Our goal is to implement new algorithms for MLE that exploit the special structure identified in the previous section while utilizing modern optimization techniques. The motivation for such algorithms arises from the "$C(x)+A$" framework, following the terminology of Dennis and Walker [12]. With both formulations (22) and (24), the Hessian matrix is the sum of a matrix that is easy to calculate (C) and an expensive matrix. As mentioned in § 4.1, this structure suggests maintaining and updating a matrix A that *approximates* the expensive part. The $C(x)+A$ framework also presents the possibility of switching between two different quadratic models at each iteration. (See Dennis, Gay and Welsch [8], [9] for an application of these ideas to nonlinear least-squares.)

We now describe specific new MLE algorithms for probabilistic choice models. (The implementation of these algorithms was greatly facilitated by the availability of subroutines developed by David M. Gay.)

5.1. Step determination. The step taken at each iteration will be determined using the model/trust region approach. Algorithms using this approach exhibit good global convergence, while preserving local convergence properties. The specific approach to be used is the locally constrained optimal step. (For a detailed discussion of model/trust region methods, see Dennis and Schnabel [11, § 6.7].)

5.2. Approximating part of the Hessian. Least-change secant update techniques for approximating the full Hessian are in wide use (for example, the DFP and BFGS updates mentioned in § 3). At iteration k, a matrix M_k is available that approximates the Hessian. The new approximation is obtained by specifying an appropriate secant condition, which takes the form

$$M_{k+1} g_k = y_k^* \tag{27}$$

where M_{k+1} is the new matrix approximation, $s_k \equiv \theta_{k+1} - \theta_k$ is the current step, and y_k^* is a vector that typically involves gradient information from both the current and previous iterates. The general principles discussed in Dennis and Schnabel [10] and [11, Chap. 11] are used below to construct secant updates for the "A" matrix in the $C(x)+A$ framework, i.e., for S_{GNL} and S_{Alt} of equations (23) and (26), respectively.

First consider S_{GNL}, and note that

$$\nabla^2 r_n(\theta_{k+1})(\theta_{k+1} - \theta_k) \approx \nabla r_n(\theta_{k+1}) - \nabla r_n(\theta_k). \tag{28}$$

Substitution into (22) gives

$$S_{\mathrm{GNL}}(\theta_{k+1})s_k = [R'(\theta_{k+1}) - R'(\theta_k)]^T \nabla L(\theta_{k+1}). \tag{29}$$

Since a secant condition takes the form of (27), where y_k^* is some function of gradient information, we can define a particular condition by specifying the form of y_k^*. The secant condition (29) is

$$y_{\mathrm{GNL}_1} \equiv [R'(\theta_{k+1}) - R'(\theta_k)]^T \nabla L(\theta_{k+1}). \tag{30}$$

It is possible to define yet another secant condition for S_{GNL} by imposing the standard secant equation for the full Hessian:

$$\nabla^2 L(\theta_{k+1})s_k = (H_{\mathrm{GNL}}(\theta_{k+1}) + S_{\mathrm{GNL}}(\theta_{k+1}))s_k = y_k \tag{31}$$

where $y_k = \nabla L(\theta_{k+1}) - \nabla L(\theta_k)$. From this we get a second secant condition—called the "default" secant condition in the terminology of Dennis and Schnabel [10]:

$$y_{\mathrm{GNL}_*} = y_k - H_{\mathrm{GNL}}(\theta_{k+1})s_k. \tag{32}$$

Now consider the Alt formulation. Recall the expression for S_{Alt} from (26) and consider the approximation

$$\nabla^2 P_{jn}(\theta_{k+1})s_k \approx \nabla P_{jn}(\theta_{k+1}) - \nabla P_{jn}(\theta_k). \tag{33}$$

Substitution as for (28) yields the definition

$$y_{\mathrm{Alt}_1} \equiv \nabla L(\theta_{k+1}) + \sum_{n=1}^{N} \sum_{j \in C_n} \frac{S_{jn}}{P_{jn}(\theta_{k+1})} \frac{dP_{jn}(\theta_k)}{d\theta}. \tag{34}$$

Finally, a default secant condition analogous to (32) can be defined for the Alt formulation:

$$y_{\mathrm{Alt}_*} \equiv y_k - H_{\mathrm{Alt}}(\theta_{k+1})s_k. \tag{35}$$

Reviewing, we have two formulations, GNL and Alt, and two secant conditions for each formulation, yielding a possible four combinations. Using the model-switching procedure described in the next section, four new algorithms can be defined as follows:

(1) GNL(1): $H = H_{\mathrm{GNL}}$, and $S \approx U(s_k, y_{\mathrm{GNL}_1})$—see (23) and (30),
(2) GNL(#): $H = H_{\mathrm{GNL}}$, and $S \approx U(s_k, y_{\mathrm{GNL}_*})$—see (23) and (32),
(3) Alt(1): $H = H_{\mathrm{Alt}}$, and $S \approx U(s_k, y_{\mathrm{Alt}_1})$—see (25) and (34),
(4) Alt(#): $H = H_{\mathrm{Alt}}$, and $S \approx U(s_k, y_{\mathrm{Alt}_*})$—see (25) and (35),

where $U(s, y)$ denotes that S is determined by a secant update.

In addition, we can attempt to solve the problem by using a model/trust region approach and an "unaugmented" quadratic model based on the calculated part of the Hessian only, in analogy to the Levenberg-Marquardt method for nonlinear least-squares:

(5) H_{Alt}-only,
(6) H_{GNL}-only.

The next section briefly describes the model-switching aspect of the algorithms.

5.3. Model-switching. In this section we consider minimization of a general function $f(\theta)$ in the special instance where more than one quadratic model is available for approximating $f(\theta)$ at each iteration.

For the purposes of this discussion assume that there are two quadratic models, although these ideas can readily be extended to situations with more than two models. Let m denote a model index, and let B_k^m denote the Hessian of the quadratic model q_k^m at iteration k:

$$q_k^m(s) = f(\theta_k) + \nabla f(\theta_k)^T s + \tfrac{1}{2} s^T B_k^m s. \tag{36}$$

Suppose that at the end of iteration $(k-1)$, one model is somehow deemed to be better than the other. The preferred model (denoted by q_k^c) is designated the current model, and is used to start iteration k. The remaining model is denoted by q_k^a (the alternate model).

For the MLE problem the two quadratic models available at each iteration are of the form

$$q_k^1(s) = f(\theta_k) + \nabla f(\theta_k)^T s + \tfrac{1}{2} s^T C(\theta_k) s \tag{37}$$

and

$$q_k^2(s) = f(\theta_k) + \nabla f(\theta_k)^T s + \tfrac{1}{2} s^T (C(\theta_k) + A_k) s. \tag{38}$$

For our purposes, $H_{\mathrm{GNL}}(\theta_k)$ or $H_{\mathrm{Alt}}(\theta_k)$ can be substituted for $C(\theta_k)$, with A_k approximating $[\nabla^2 L(\theta_k) - H_{\mathrm{GNL}}(\theta_k)] = S_{\mathrm{GNL}}(\theta_k)$ or $[\nabla^2 L(\theta_k) - H_{\mathrm{Alt}}(\theta_k)] = S_{\mathrm{Alt}}(\theta_k)$ using the least-change secant updates discussed above.

The typical behavior of a model switching algorithm is as follows. At the first iteration $A_k = 0$, $q_k^1 = q_k^2$, and the initial current model is q_k^1. As the algorithm proceeds, the secant approximation A_k is accumulated. In instances where the $S(\theta_{\mathrm{MLE}})$ is small and a good starting guess is provided, the algorithm may never switch to the augmented model. In other instances the algorithm may take a few iterations to build up A_k and then switch to the augmented model for the remainder of the procedure.

5.4. Remarks. Sections 5.1 through 5.3 represent "modules" that can be integrated to produce the final algorithms. The missing pieces include a driver, and user-provided subroutines for calculating the probabilities and the derivatives of the probabilities. A general code has been developed for finding maximum likelihood estimates for choice models using reliable, state-of-the-art subroutines discussed above. In addition, code has been implemented and tested for various choice models, as described in the next section.

6. Experimental results and comparison of MLE algorithms. A summary of experimental results for the six MLE algorithms appears below. Parameters were estimated for the multinomial logit (MNL), elimination-by-aspects (EBA), and Batsell–Polking (BP) models using both actual data obtained in choice experiments, and simulated data.

6.1. MNL results. This discussion is a logical starting point due to the wide use of the MNL model in the literature. Calculation of the Hessian of $L(\theta)$ is straightforward, thus allowing a comparison between the proposed algorithms and HUMSOL (Gay [15]), a FORTRAN subroutine for unconstrained minimization that requires an exact Hessian and uses a model/trust region approach. Preliminary results using actual data are summarized first, followed by more comprehensive results obtained via a Monte Carlo simulation.

Data for the calculations were obtained in an experiment involving choice among wagers by Batsell and Lodish [1]. The design of the experiment included a repetition factor $R_n = 6$; choice data were collected for thirteen subjects. Figures 1 and 2 summarize the numerical results obtained from data sets for two individuals, using various model

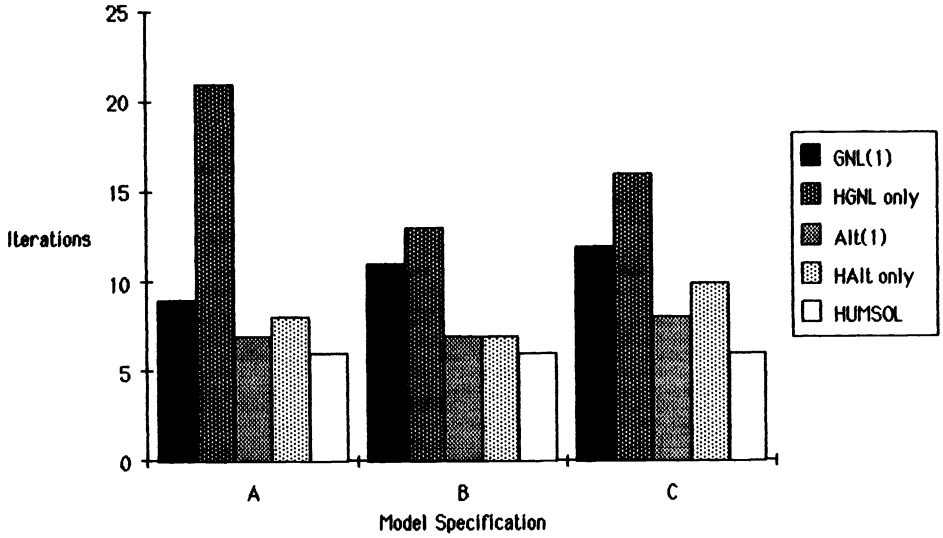


FIG. 1. MNL *model results using Subject* 1 *data.*

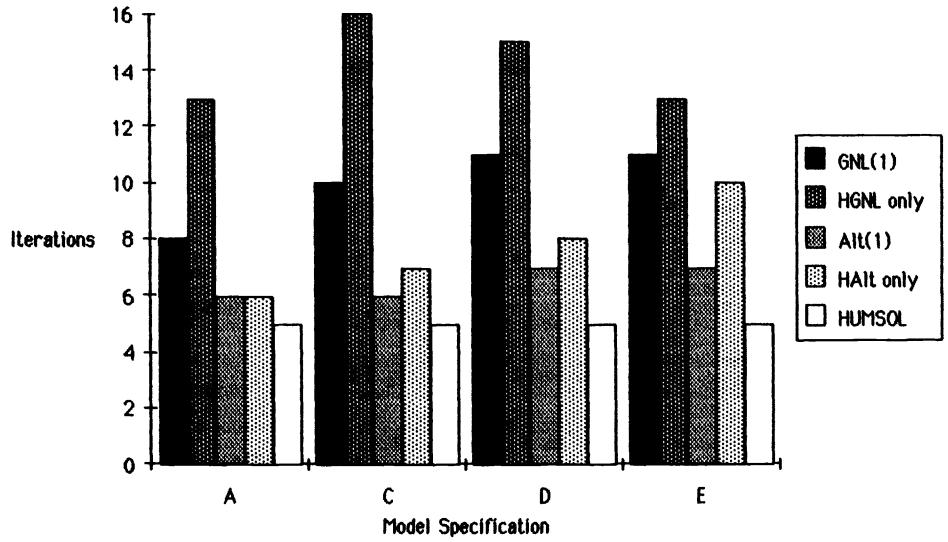


FIG 2. MNL *model results using Subject* 13 *data.*

specifications. In the following discussion we focus on the number of iterations required for convergence. (Our conclusions are also consistent with the numbers of function and gradient evaluations, which are therefore not included. Results for the (#) algorithms are comparable to those for the (1) algorithms, and have also been omitted to reduce clutter.) For more detailed numerical results, see Bunch [5].

First, we note that there were no failures, and that all methods converged to the same solutions. Second, although this was a small test on a simple model, the results indicated a clear dominance of the algorithms based on the Alt formulation over those based on the GNL formulation. Other preliminary findings were corroborated by the results of Monte Carlo simulation, which we now discuss.

The purpose of the simulation was to compare the average performance of the algorithms over both a larger number of data sets and a wider range of data set types.

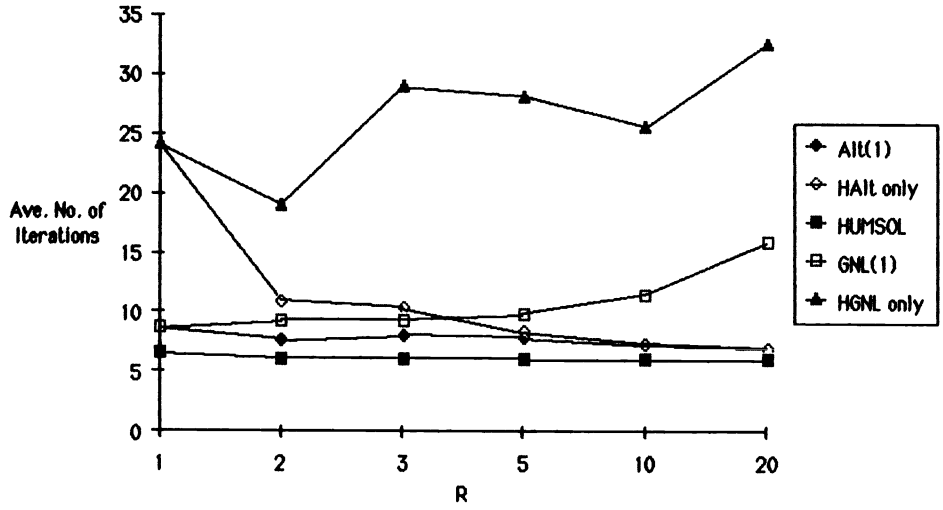


FIG. 3. MNL *model results using simulated data (iterations).*

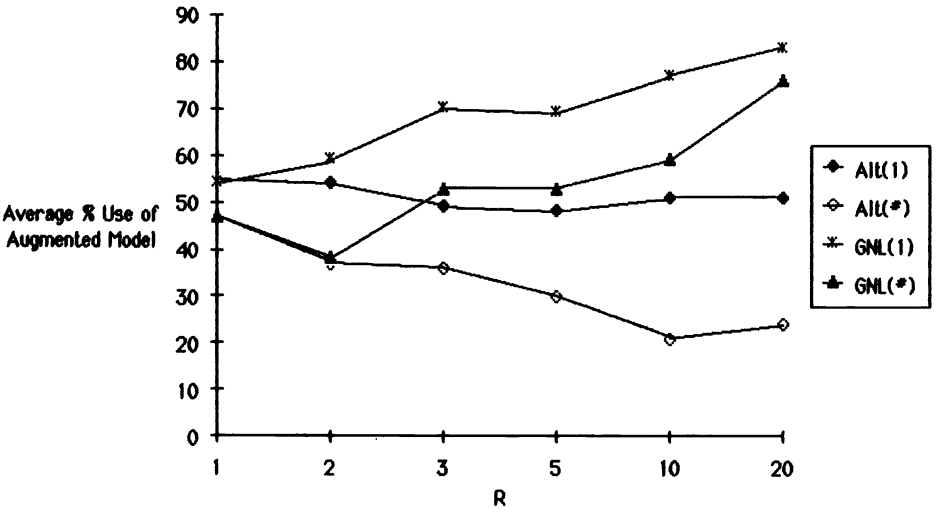


FIG. 4. MNL *model results using simulated data (augmented model usage).*

The wager data for the two individuals discussed above provided a convenient framework for the simulation. Runs were obtained using 20 Monte Carlo repetitions, with $R = 1, 2, 3, 5, 10$, and 20. At each trial, simulated frequencies were produced by making random multinomial drawings from the choice sets. For a detailed description of the simulation, see Bunch [5].

The important trends are illustrated in Figs. 3 and 4, which include some of the results obtained using wager data from Subject 1. Consider Fig. 3. As R increases, the average number of iterations required for the Alt-based algorithms decreases, while the opposite trend holds for the GNL-based algorithms. In fact, the method using H_{Alt}-only quickly becomes competitive with the other methods as R increases.

Consider Fig. 4. The GNL algorithms use the augmented model a greater percentage of the time than the Alt algorithms. These observations are consistent with the interpretation that H_{Alt} becomes a better and better approximation to the true Hessian

as R increases, whereas H_{GNL} does not. Equivalently, the S_{Alt} matrix converges to zero as R increases. This interpretation is further corroborated by relative norm data for the true S matrices, which are not included here. See Bunch [5].

6.2. Other results. The results of the previous section, although interesting, do not in themselves demonstrate the usefulness of model switching algorithms. Results for the MNL model might just as well be obtained using HUMSOL, since the special structure of the model makes calculation of the full Hessian no more burdensome than calculating either H_{Alt} or H_{GNL}. However, this is not the case for more general choice models, as we now discuss.

Numerical results have been obtained for both Tversky's EBA model and the model of Batsell and Polking. In contrast to the MNL model, both models exhibit complicated structure and require many parameters in order to capture the effect of similarity among choice alternatives. In fact, the number of parameters required by these models grows so rapidly as a function of the number of alternatives, that calculating second derivatives quickly becomes intractable. It was not even practical to use HUMSOL for the examples contained in this section.

Once again, parameters were estimated for both experimental data sets and simulated data sets. A summary of results obtained using two experimental data sets—"snack data" (S) and "political data" (P)—is included in Figs. 5 and 6. Various model specifications were used for each data set: results are identified by the type of data set and the number of parameters used in the model specification. For detailed tables of results, and more discussion about the data sets and models, see Bunch [5].

These results establish that complex choice models can be estimated in a reasonable number of iterations using these methods, whereas such estimation is impractical using, for example, HUMSOL. In addition, the trends identified in the previous section are also exhibited in Figs. 5 and 6: the Alt-based algorithms dominate the GNL-based algorithms.

An important question remains. Based on these results we would conclude that model switching is not necessary, and that an algorithm using H_{Alt}-only would suffice. As we shall see, the excellent performance of the H_{Alt}-only algorithm can be explained

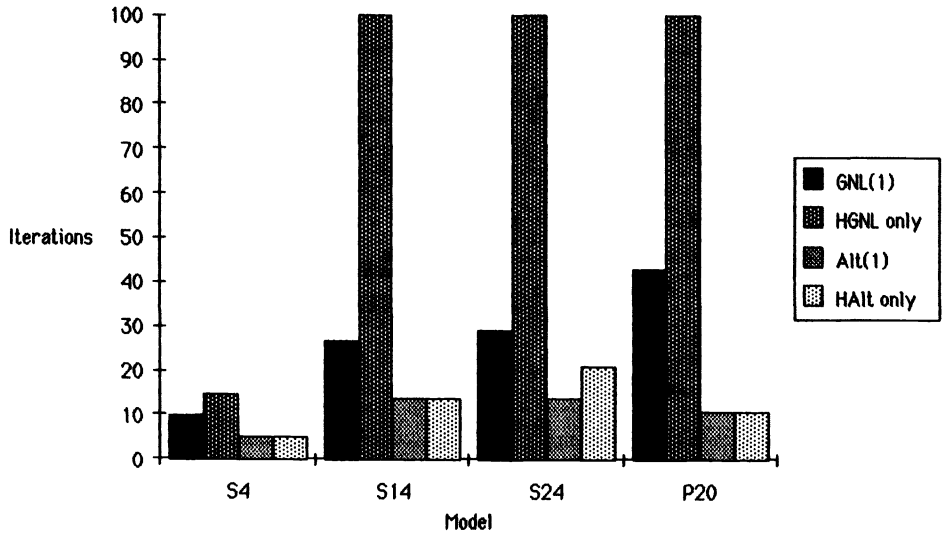


FIG. 5. EBA *model results.*

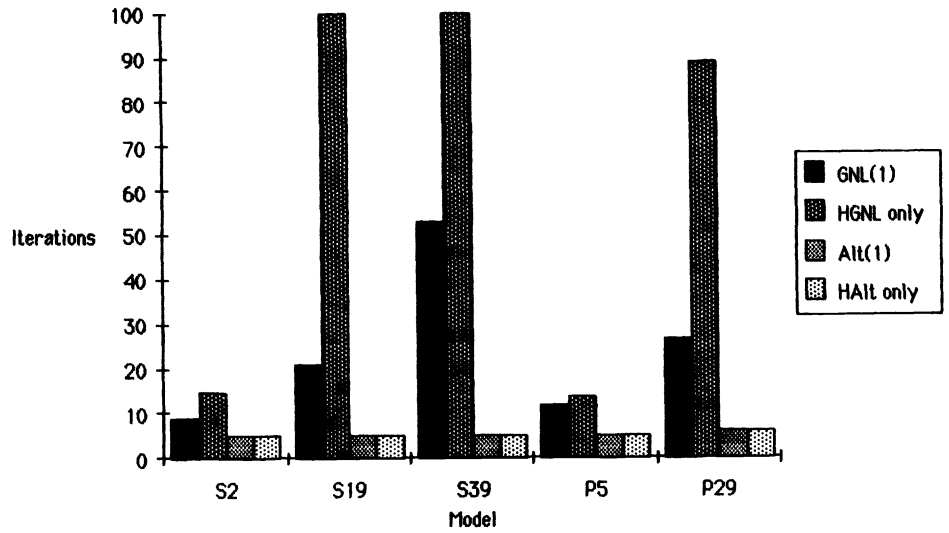


FIG. 6. *Batsell-Polking model results (empirical data).*

by the fact that both of the data sets have a large repetition factor: $R = 12$ for the snack data, and $R = 40$ for the political data. This point is verified via simulated data sets with $R = 3, 5$, and 10 (see Fig. 7). The number of iterations can be cut approximately in half by the use of model-switching.

7. Discussion. The results of § 6 illustrate the utility of the model-switching algorithms defined in § 5. In summary:

(1) The four model-switching algorithms reliably converge to solutions of the MLE problem for the three models tested.
(2) The two Alt algorithms outperform the two GNL algorithms based on iterations and function/gradient evaluations, which implies that the matrix H_{Alt} is a closer approximation to the true Hessian than is H_{GNL}.
(3) For the MNL model, the Alt algorithms' performance is comparable to an algorithm that employs exact Hessian information at each iteration (HUMSOL).

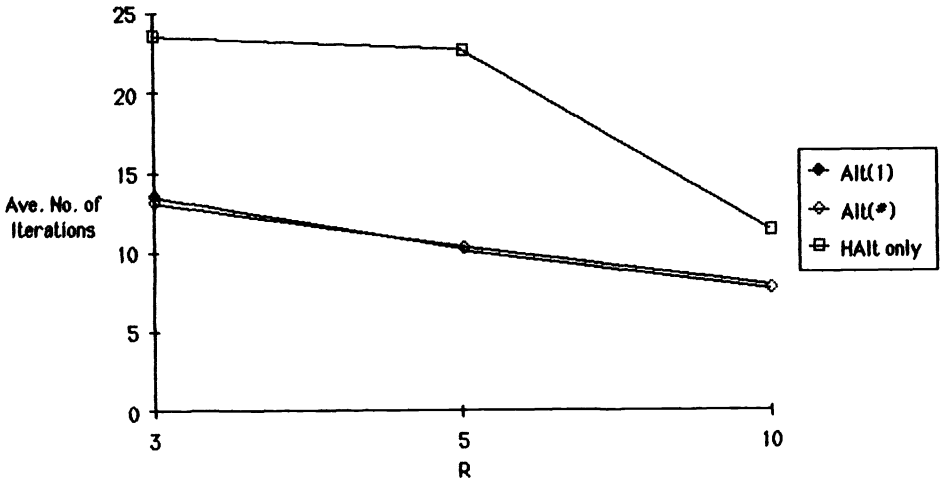


FIG. 7. *Batsell-Polking model results (simulated data for model S19).*

(4) For models in which it is impractical to use HUMSOL, e.g., EBA and Batsell-Polking, the algorithms make the maximum likelihood estimation problem tractable.

The remainder of this discussion examines explanations for the efficacy of these algorithms. Recall that the original successful application of the model switching approach was for the nonlinear least-squares (NLLS) problem. It is instructive to review what was learned from the NLLS experience.

Recall that the NLLS problem is defined in the context of generalized regression by choosing $\phi_n(v) = v^2/2$, and $r_n = y_n - m(\theta, z_n)$ in (12). The resulting expression for the Hessian is

$$\begin{aligned}\nabla^2 f(\theta) &= J(\theta)^T J(\theta) + \sum_{n=1}^{N} r_n(\theta)\nabla^2 r_n(\theta) \\ &= J(\theta)^T J(\theta) + S(\theta)\end{aligned} \tag{39}$$

where $J(\theta)$ is the Jacobian matrix for the residual vector $R(\theta)$. Note that each term of the "expensive" part of the Hessian involves an r_n. If the model does a good job of fitting the data at the solution, then each r_n is close to zero and the matrix S is very small relative to $J^T J$. It is in this instance that the well-known Gauss-Newton method converges quickly.

Based on these observations, the implications for an NLLS model switching algorithm are obvious. Given the success of the MLE algorithms as demonstrated in § 6, it seems likely that the GNL and Alt Hessian formulations exhibit properties analogous to those described above.

Ideally, the S matrices would be zero at the true solution, by analogy to the zero-residual case for nonlinear least-squares described above. Failing this, it would be desirable for the *expectation* of the S matrices to be zero at the solution. Bunch [5] proves the following theorems, which show that the latter is true for both formulations:

THEOREM 1. *Assume that the choice model P_{jn} has been correctly specified. Then the expectation of the matrix S_{Alt}, evaluated at the true parameter vector θ, is zero.*

THEOREM 2. *Assume that the choice model P_{jn} has been correctly specified. Then the expectation of the matrix S_{GNL}, evaluated at the true parameter vector θ, is zero.*

Theorems 1 and 2 can be interpreted as explanations for why all four model switching algorithms might work. On the average, the size of the S matrix at the solution is expected to be small. In instances where it is not small, the augmented model helps speed the convergence to the solution. See Dennis and Walker [12].

The more interesting question concerns the superiority of the Alt-based algorithms and the trends exhibited in the Monte Carlo experiments. The following theorem (Bunch [5]) verifies the interpretation of the results from § 6 regarding the Alt algorithms.

THEOREM 3. *Assume that the choice model P_{jn} has been correctly specified. Then the value of the matrix S_{Alt} at the true parameter vector θ approaches zero as the repetition factor R_n becomes large for all n.*

As the number of repetitions increases, it is indeed the case that (at the solution) the matrix H_{Alt} becomes a better and better approximation to the true Hessian. No such result exists for the matrix H_{GNL}.

Finally, we note the following:

THEOREM 4. *Assume that $R_n = 1$ for all n in the quantal choice experiment. Then $H_{\text{GNL}} = H_{\text{Alt}}$, and furthermore,* GNL(1) = Alt(1) *and* GNL(#) = Alt(#).

In many experimental settings there will only be one repetition for each trial, and in this instance the corresponding GNL and Alt algorithms are equivalent.

The results presented here demonstrate the utility of algorithms that exploit the special structure of the MLE problem and utilize modern optimization methods. The new algorithms are reliable, efficient, easy to use, and can routinely solve problems previously regarded as intractable. These capabilities should enhance the development of new and better models.

Acknowledgments. The author is indebted to John Dennis for serving as thesis director, and to Randy Batsell for providing the original stimulus (and data sets) for this work. The editor provided many helpful suggestions for improving the manuscript.

REFERENCES

[1] R. BATSELL AND L. LODISH, *A model and measurement methodology for predicting individual consumer choice*, J. of Marketing Research, 18(1) (1981), pp. 1–12.

[2] R. BATSELL AND J. POLKING, *A new class of market share models*, Marketing Sci., 4 (1985), pp. 177–198.

[3] E. BERNDT, B. HALL, R. HALL AND J. HAUSMAN, *Estimation and inference in non-linear structural models*, Annals of Economic and Social Measurement, 3 (1979), pp. 653–655.

[4] C. G. BROYDEN, *The convergence of a class of double-rank minimization algorithms, Parts* I *and* II, J.I.M.A., 6 (1970), pp. 76–90, 222–236.

[5] D. BUNCH, *Maximum likelihood estimation of probabilistic choice models*, Working Paper 85-02, Graduate School of Administration, University of California, Davis, 1985.

[6] C. DAGANZO, *Multinomial Probit*, Academic Press, New York, 1979.

[7] W. C. DAVIDON, *Variable metric methods for minimization*, Argonne National Labs Report ANL-5990, Argonne, IL, 1959.

[8] J. DENNIS, D. GAY AND R. WELSCH, *An adaptive nonlinear least-squares algorithm*, ACM Trans. Math. Software, 7 (1981a), pp. 348–368.

[9] ———, *Algorithm* 573 NL2SOL—*An adaptive nonlinear least-squares algorithm*, ACM Trans. Math. Software, 7 (1981b), pp. 369–383.

[10] J. DENNIS AND R. SCHNABEL, *Least change secant updates for quasi-Newton methods*, SIAM Rev., 21 (1979), pp. 443–459.

[11] ———, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[12] J. DENNIS AND H. WALKER, *Convergence theorems for least-change secant update methods*, SIAM J. Numer. Anal., 18 (1981), pp. 949–987, 19 (1981), p. 443.

[13] R. FLETCHER AND M. J. D. POWELL, *A rapidly convergent descent method for minimization*, Comput. J., 6 (1963), pp. 163–168.

[14] D. GAY, *On solving robust and generalized linear regression problems*, MRC Technical Summary Report # 2000, 1979.

[15] ———, *Algorithm* 611. *Subroutines for unconstrained minimization using a model/trust-region approach*, ACM Trans. Math. Software, 8 (1983), pp. 503–524.

[16] D. GOLFARB, *A family of variable metric methods derived by variational means*, Math. Comp., 24 (1970), pp. 23–26.

[17] R. D. LUCE, *Individual Choice Behavior; A Theoretical Analysis*, John Wiley, New York, 1959.

[18] D. MCFADDEN, *Conditional logit analysis of qualitative choice behavior*, in Frontiers of Econometrics, P. Zarembka, ed., Academic Press, New York, 1973.

[19] ———, *Quantal choice analysis: A survey*, Annals of Economic and Social Measurement, 5 (1976), pp. 363–390.

[20] ———, *Econometric models of probabilistic choice*, in Structural Analysis of Discrete Data, C. Manski and D. McFadden, eds., MIT Press, Cambridge, MA, 1980a.

[21] ———, *Econometric models for probabilistic choice among products*, J. of Business, 53 (1980b), pp. 513–529.

[22] D. F. SHANNO, *Conditioning of quasi-Newton methods for function minimization*, Math. Comp., 24 (1970), pp. 647–657.

[23] A. TVERSKY, *Elimination by aspects: A theory of choice*, Psychological Rev., 79 (1972a), pp. 281–299.

[24] ———, *Choice by elimination*, J. Math. Psych., 9 (1972b), pp. 341–367.

[25] A. TVERSKY AND S. SATTAH, *Preference trees*, Psychological Rev., 86 (1979), pp. 542–573.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 1, January 1987

© 1987 Society for Industrial and Applied Mathematics

CONTINUOUS TIME SERIES MODELS FOR UNEQUALLY SPACED DATA APPLIED TO MODELING ATOMIC CLOCKS*

RICHARD H. JONES† AND PETER V. TRYON‡

Abstract. State space representations and Kalman filters used to calculate likelihoods have increased the ease and flexibility of fitting time series models to data. When data are unequally spaced with no underlying basic sampling interval, continuous time series models are more natural than discrete time series models. State space representations still provide the flexibility needed to include a large class of models. This paper gives a survey of state space methods for continuous time processes, discusses extensions to multivariate data at unequally spaced time points with missing data within the observation vector, and gives an example of estimating time and model parameters from an ensemble of atomic clocks.

Key words. unequally spaced data, Kalman filters, state space representations, time series analysis, missing observations, atomic clocks.

1. Introduction. State space representations and the use of Kalman (1960) filters for the recursive calculation of exact or conditional likelihoods (Schweppe (1965)) are becoming the method of choice for many time series modeling problems. Used with a nonlinear optimization program, maximum likelihood estimates of parameters which occur anywhere in the model can be obtained. This includes estimates of unknown variances of random terms. The use of maximum likelihood methods allows the use of asymptotic hypothesis tests and confidence intervals on parameters. One of the main advantages of state space methods is their flexibility. For example, Harvey and Phillips (1979) used these methods for regression models with autoregressive-moving average errors. Missing observations become an almost trivial problem (Jones (1980)). Gelb (1974) gives a very good introduction to the subject of optimal estimation methods.

The case of unequally spaced data will be distinguished from missing observations in that the observations can occur at any time points rather than at multiples of some sampling interval. Data will consist of an observation time together with the observation or observations collected at that time. Whether the observation time is determined by some random or deterministic mechanism is immaterial so long as the observation times are known. It will only be assumed that the times of the observations are not affected by the levels of the variables being measured. Since unequally spaced data implies that there is an underlying continuous time process being sampled, it is natural to consider fitting continuous time models to the data.

From the continuous time state space representation it is possible to derive the corresponding discrete time state space representation at the observation times. Unequally spaced data are then handled the same as equally spaced data except that certain parameters of the discrete state representation depend on the length of the step.

2. Continuous time state space models. Doob (1953) gives much of the mathematics for continuous time stationary processes, and Kalman and Bucy (1961) provide the state space approach for a wide class of stationary and nonstationary processes. The problem of fitting a first order continuous time autoregression to unequally spaced data by maximum likelihood was independently solved by Robinson (1977) and Jones

* Received by the editors November 16, 1984, and in revised form July 12, 1985.

† Department of Preventive Medicine and Biometrics, Box B-119, University of Colorado Health Sciences Center, Denver, Colorado 80262.

‡ Statistical Engineering Division, National Bureau of Standards, Boulder, Colorado 80303, deceased.

(1977). For continuous time autoregressions, the usual difference equation formulation used for discrete time processes is replaced by a linear differential equation. Using a state space representation, Jones (1981) developed a method for fitting a pth order continuous time autoregression with observational errors to unequally spaced data. This model is

$$d\begin{bmatrix} x(t) \\ x^{(1)}(t) \\ \cdot \\ \cdot \\ \cdot \\ x^{(p-1)}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & & 0 \\ \cdot & & \cdot & & \\ \cdot & & \cdot & & \\ 0 & 0 & 0 & & 1 \\ -\alpha_0 & -\alpha_1 & -\alpha_2 & \cdots & -\alpha_{p-1} \end{bmatrix} \begin{bmatrix} x(t) \\ x^{(1)}(t) \\ \cdot \\ \cdot \\ \cdot \\ x^{(p-1)}(t) \end{bmatrix} dt + \begin{bmatrix} 0 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 1 \end{bmatrix} dW(t), \tag{2.1}$$

where $x^{(j)}(t)$ denotes the jth derivative of $x(t)$, and $W(t)$ is a Wiener process or continuous time random walk. $dW(t)$ represents a continuous time "white noise" process. Let the parameter σ^2 be the variance of this Wiener process over a unit time interval. The process $x(t)$ will be stationary if the roots of

$$\alpha_0 + \alpha_1 z + \alpha_2 z^2 + \cdots + \alpha_{p-1} z^{p-1} + z^p = 0 \tag{2.2}$$

have negative real parts. If $x(t)$ is observed at time t_i and there is observational error, the observation equation is

$$y(t_i) = [1\ 0 \cdots 0] \begin{bmatrix} x(t_i) \\ x^{(1)}(t_i) \\ \vdots \\ x^{(p-1)}(t_i) \end{bmatrix} + v(t_i), \tag{2.3}$$

where $v(t_i)$ is the observational error with zero mean and variance R. Assuming normally distributed random inputs and errors, from observations at times $t_1 < t_2 < \cdots < t_n$, Jones (1981) obtains maximum likelihood estimates of the parameters $\alpha_0, \cdots, \alpha_{p-1}, \sigma^2$, and R, where the alphas are constrained to produce a stationary process.

The direct modeling of observational errors is an important feature of state space methods. It may be more parsimonious than allowing observational error to be absorbed by the other parameters. It is well known (Bartlett (1946)) that if a continuous time pth order autoregression is sampled at equally spaced time points, the resulting discrete time process is ARMA $(p, p-1)$. The $2p-1$ parameter are functions of the p parameters of the original continuous time process. If observational error is also included, the resulting process becomes ARMA (p, p) (Box and Jenkins (1976, p. 122)). It follows that fitting a continuous time autoregression with observational error modeled directly may be a more parsimonious representation even if the data are equally spaced, since this model contains only $p+1$ parameters, the p autoregressive parameters and the observational error variance.

Wecker and Ansley (1983) consider integrated Wiener processes for modeling nonstationary processes such as growth curves, and show their relationship to smoothing polynomial splines (see also Wahba (1978), Weinert et al. (1979)). Although Wecker and Ansley (1983) formulate their model at discrete unequally spaced time points, the

underlying continuous time model is

$$(2.4)\qquad d\begin{bmatrix} x(t) \\ x^{(1)}(t) \\ \cdot \\ \cdot \\ \cdot \\ x^{(m-1)}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & & 0 \\ \cdot & & & & \\ \cdot & & & & \\ 0 & 0 & 0 & & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} x(t) \\ x^{(1)}(t) \\ \cdot \\ \cdot \\ \cdot \\ x^{(m-1)}(t) \end{bmatrix} dt + \begin{bmatrix} 0 \\ 0 \\ \cdot \\ \cdot \\ 0 \\ 1 \end{bmatrix} dW(t).$$

These processes are closely related to models used here for atomic clocks.

Continuous time multivariate models are the most interesting in applied problems since they allow the investigation of interactions among series when observations are unequally spaced. Here, the term multivariate means that at each observation time, more than one variable is observed. Some medical applications of continuous time multivariate models are given in Jones (1984).

The above processes are special cases of a more general continuous time state space model (Gelb (1974, Chap. 3)).

$$(2.5)\qquad dX(t) = FX(t)\,dt + D\,dt + G\,dW(t)$$

where $X(t)$ is the m by 1 state vector, F is an m by m system dynamics matrix, D is an m by 1 vector of constants representing deterministic inputs to the system, $dW(t)$ is a $s \leqq m$ vector of random inputs assumed to be continuous time Gaussian "white noise" with instantaneous covariance matrix Q,

$$(2.6)\qquad E\{dW(t)\,dW'(t)\} = Q\,dt,$$

and G is an m by s matrix of constants. The observation equation at an observation time t_i is

$$(2.7)\qquad y(t_i) = HX(t_i) + v(t_i)$$

where $y(t_i)$ is a vector of the observations at time t_i, H is a matrix of constants that determine the linear combinations of the elements of the state vector that are observed, and $v(t_i)$ is a random vector of observational errors assumed to be normally distributed with zero mean and covariance matrix R, independent of the input random noise $dW(t)$.

The homogeneous portion of the state space model (2.5) is

$$(2.8)\qquad \frac{d}{dt}X(t) = FX(t),$$

a matrix differential equation. The solution of this equation over a finite time step from t to $t+\delta$ is

$$(2.9)\qquad X(t+\delta) = \exp(F\delta)X(t),$$

where the exponential is a matrix exponential defined in terms of its power series,

$$(2.10)\qquad \exp(F\delta) = I + \sum_{k=1}^{\infty} \frac{(F\delta)^k}{k!},$$

where I is the identity matrix (Gelb, (1974, p. 60)). The transition matrix for a time step of length δ is therefore

$$(2.11)\qquad \Phi(\delta) = \exp(F\delta).$$

In the case of a continuous time AR process (2.1), this matrix exponential can be evaluated using an eigenvalue-eigenvector rotation (Jones (1981)). For the integrated Wiener process (2.4), the matrix exponential can be evaluated directly since as F is raised to powers, the ones on the superdiagonal shift up one diagonal position for each power, so for this case

$$F^m = 0 \tag{2.12}$$

and

$$\Phi(\delta) = \begin{bmatrix} 1 & \delta & \frac{\delta^2}{2!} & \cdots & \frac{\delta^{m-1}}{(m-1)!} \\ 0 & 1 & \delta & & \frac{\delta^{m-2}}{(m-2)!} \\ 0 & 0 & 1 & & \frac{\delta^{m-3}}{(m-3)!} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & & \delta \\ 0 & 0 & 0 & & 1 \end{bmatrix}. \tag{2.13}$$

Adding the vector of deterministic inputs to (2.8),

$$\frac{d}{dt} X(t) = FX(t) + D \tag{2.14}$$

requires a particular solution of the matrix differential equation in order to predict for the time interval of length δ. The solution is given by the matrix superposition integral (Gelb (1974, p. 64)),

$$X(t+\delta) = \Phi(\delta)X(t) + \int_t^{t+\delta} \Phi(t+\delta-\tau)D\, d\tau. \tag{2.15}$$

The random input over a step of length δ is calculated in a similar way,

$$U(t+\delta, t) = \int_t^{t+\delta} \Phi(t+\delta-\tau)G\, dW(\tau). \tag{2.16}$$

Since this random input is unpredictable, it is only necessary to calculate the covariance matrix of this random input,

$$Q(\delta) = E\{U(t+\delta, t)U'(t+\delta, t)\} = \int_t^{t+\delta} \Phi(t+\delta-\tau)GQG'\Phi'(t+\delta-\tau)\, d\tau. \tag{2.17}$$

The discrete time Kalman filter derived from the continuous time state space model starts by specifying the initial condition on the state vector denoted $X(t_1|0)$, and its covariance matrix $P(t_1|0)$. The notation $X(t_i|t_{i-1})$ denotes the estimate of the state vector at observation time t_i given observations up until time t_{i-1}, and $P(t_i|t_{i-1})$ denotes the covariance matrix of the estimate. Similarly, $X(t_i|t_i)$ is the estimate of the state at time t_i given observations up to and including t_i, and $P(t_i|t_i)$ is its covariance matrix. The recursion for a general step is:

1) Calculate the length of the time step and a one step prediction

$$\delta_i = t_i - t_{i-1},$$

$$X(t_i|t_{i-1}) = \Phi(\delta_i)X(t_{i-1}|t_{i-1}) + \int_0^{\delta_i} \Phi(\delta_i-\tau)D\, d\tau.$$

2) Calculate the covariance matrix of this prediction

$$P(t_i|t_{i-1}) = \Phi(\delta_i)P(t_{i-1}|t_{i-1})\Phi'(\delta_i) + Q(\delta_i).$$

3) Predict the next observation vector

$$y(t_i|t_{i-1}) = HX(t_i|t_{i-1}).$$

4) The innovation vector is the difference between the observation and the predicted observation

$$I(t_i) = y(t_i) - y(t_i|t_{i-1}).$$

5) The innovation covariance matrix is

$$V(t_i) = HP(t_i|t_{i-1})H' + R.$$

6) The contribution to $-2\ln$ likelihood for this step is

$$I'(t_i)V^{-1}(t_i)I(t_i) + \ln|V(t_i)|$$

where $|\ |$ denotes the determinant of the matrix. The contribution for each step is summed over all steps.

7) The Kalman gain matrix is

$$K(t_i) = P(t_i|t_{i-1})H'V^{-1}(t_i).$$

8) The updated estimate of the state vector is

$$X(t_i|t_i) = X(t_i|t_{i-1}) + K(t_i)I(t_i).$$

9) Its covariance matrix is

$$P(t_i|t_i) = P(t_i|t_{i-1}) - K(t_i)HP(t_i|t_{i-1}).$$

3. Continuous time model for cesium beam atomic clock. One of the most accurate standards that exists today is the time standard, and this standard is calculated from an ensemble of cesium beam atomic clocks. The accuracy is on the order of one part in 10^{13}. In fact, the second is defined as 9,192,631,770 cycles of the fundamental resonance of the cesium atom. Cesium beam atomic clocks are very stable; however, they are not deterministic, and undergo stochastic variations in both time and frequency. An example of a realization of the time difference between a commercial atomic clock and a time scale generated from a group of atomic clocks is shown in Fig. 1. The top graph shows how a clock has a frequency drift so that it appears to drift linearly away from "true time," or in this case the time generated by a good time standard. This linear drift does not affect the clock's ability to keep accurate time as long as the rate of drift is known or estimated very well. Note that the drift of the clock is on the order of 50 microseconds per year. The middle graph are the residuals after fitting a straight line to the top graph by simple linear regression. The scale is expanded and the variation is on the order of 5 microseconds per year. The quadratic appearance indicates a frequency drift in the clock, i.e. the frequency of the clock would be changing linearly with time. The life of a cesium beam atomic clock is on the order of five or six years, and this frequency drift characteristic is sometimes apparent near the beginning and end of the clock's life. The bottom graph shows the residuals after removing a quadratic function, and shows the random walk characteristics. The scale on this graph has again been expanded.

It is a characteristic of atomic clocks that only the time differences between clocks or clocks and time scales can be observed, and these differences are measured very

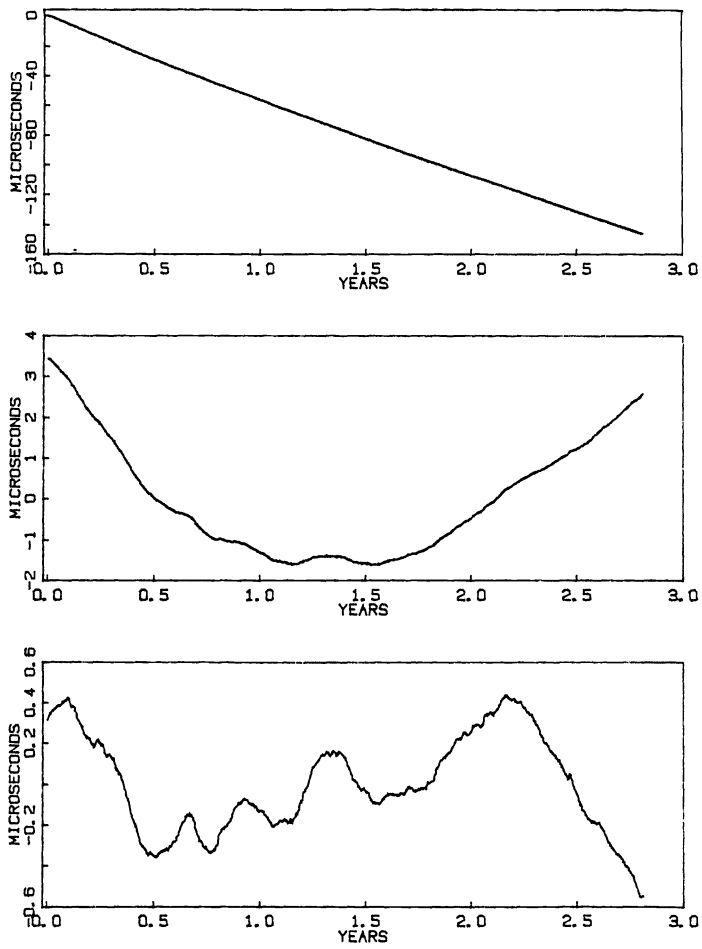


FIG. 1. *A realization of the time difference between a commercial atomic clock and a United States Naval Observatory time scale (top). The middle graph has a linear trend removed, and the bottom graph has a quadratic removed.*

accurately. All clocks have a frequency offset from perfect frequency which produces a linear time trend. This frequency offset has some random variation and is continuously estimated and adjusted when producing a time scale from an ensemble of atomic clocks. The residuals then tend to have random walk characteristics which are actual changes in the frequency behavior of the clocks. A study of processes of this type is given in Rutman (1978). There are many applications which can be modeled as time varying linear trends.

From the physics of cesium beam atomic clocks and from empirical verification (Tryon and Jones (1983)), the sum of a random walk plus an integrated random walk models the data very well. This was determined by examining residuals (normalized innovations from the Kalman filter) after fitting the model to approximately a year of daily observations on an ensemble of seven clocks. This model can be interpreted as a random walk in frequency which gets integrated into time, plus an independent random walk in time. The variances of these two random walks can be associated with clock characteristics, i.e. frequency stability which affects long term behavior, and time stability which affects short term behavior. A continuous time model may be more

appropriate than the discrete time model used by Tryon and Jones (1983) and Jones and Tryon (1983) since observations are taken every two hours of every day, while the random walk behavior is essentially continuous. This means that at the sampling times there is correlation between the random input to time and the random input to frequency caused by the continuous integration of the random walk in frequency. This model generalizes the Wecker and Ansley (1983) model to the case of more than one random input.

The continuous time model for a cesium beam atomic clock with frequency drift is

$$\frac{d}{dt}\begin{bmatrix} x(t) \\ w(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} x(t) \\ w(t) \end{bmatrix} + \begin{bmatrix} 0 \\ d \end{bmatrix} + \begin{bmatrix} \varepsilon(t) \\ \eta(t) \end{bmatrix}, \tag{3.1}$$

where $x(t)$ denotes the time state of the clock, i.e. the difference between the time of the clock and "true" time, and $w(t)$ denotes the frequency random walk component which gets integrated into the time component. The deterministic frequency drift is denoted by d, and $\varepsilon(t)$ and $\eta(t)$ are independent continuous time "white noise" processes with spectral densities σ_ε^2 and σ_η^2. Letting

$$X(t) = \begin{bmatrix} x(t) \\ w(t) \end{bmatrix}, \quad F = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 0 \\ d \end{bmatrix}, \quad dW(t) = \begin{bmatrix} \varepsilon(t) \\ \eta(t) \end{bmatrix}, \tag{3.2}$$

(3.1) can be written

$$dX(t) = FX(t)\,dt + D\,dt + dW(t). \tag{3.3}$$

For the particular F matrix in this problem,

$$F^2 = 0;$$

therefore,

$$\exp(F\delta) = I + F\delta = \begin{bmatrix} 1 & \delta \\ 0 & 1 \end{bmatrix} = \Phi(\delta). \tag{3.4}$$

For a discrete time step of length δ, since the random input is unpredictable, the prediction of the state vector is

$$\begin{aligned} X(t+\delta|t) &= \Phi(\delta)X(t|t) + \int_0^\delta \begin{bmatrix} 1 & \delta - t \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 0 \\ d \end{bmatrix} dt \\ &= \Phi(\delta)X(t|t) + D(\delta), \end{aligned} \tag{3.5}$$

where

$$D(\delta) = d\begin{bmatrix} \dfrac{\delta^2}{2} \\ \delta \end{bmatrix}.$$

The random input to this prediction is given by

$$\int_t^{t+\delta} \begin{bmatrix} 1 & t+\delta-\tau \\ 0 & 1 \end{bmatrix}\begin{bmatrix} \varepsilon(\tau) \\ \eta(\tau) \end{bmatrix} d\tau, \tag{3.6}$$

which has covariance matrix

$$Q(\delta) = \int_0^\delta \begin{bmatrix} 1 & \delta - t \\ 0 & 1 \end{bmatrix}\begin{bmatrix} \sigma_\varepsilon^2 & 0 \\ 0 & \sigma_\eta^2 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ \delta - t & 1 \end{bmatrix} dt = \begin{bmatrix} \delta\sigma_\varepsilon^2 + \delta^3\sigma_\eta^2/3 & \delta^2\sigma_\eta^2/2 \\ \delta^2\sigma_\eta^2/2 & \delta\sigma_\eta^2 \end{bmatrix}. \tag{3.7}$$

Although the white noise processes are independent, predicting over a finite time step introduces correlation in the Q matrix between time and frequency. This was pointed out to the first author by Chaffee (1982).

We now have an equation (3.5) for predicting a continuous time process which is a random walk plus an integrated random walk and a possible frequency drift over an arbitrary finite time interval, and equation (3.7) for the covariance matrix of the error introduced into this prediction by the continuous time random input to the state equation. It is now possible to obtain estimates of the states of an ensemble of atomic clocks from observations on clock time differences obtained at equal or unequally spaced time points.

The state equation for an ensemble of m clocks is obtained by concatenating the two state elements of each clock into a column vector of length $2m$. The state transition matrix $\Phi(t)$ is a $2m$ by $2m$ matrix consisting of 2 by 2 blocks on the main diagonal corresponding to the state transition matrix in (3.4), and zero blocks off the main diagonal. This state equation for m clocks will be written

$$X(t+\delta)=\Phi(\delta)X(t)+D(\delta)+U(t+\delta). \tag{3.8}$$

The frequency drift vector, $D(\delta)$, and the random input vector, $U(t+\delta)$, are of length $2m$ and represent the d, $\varepsilon(t)$ and $\eta(t)$ for each clock over the time interval $(t, t+\delta)$. The covariance matrix of $U(t+\delta)$, $Q(\delta)$ is a $2m$ by $2m$ block diagonal matrix with 2 by 2 blocks along the main diagonal as in equation (2.9), where the σ_ε^2 and σ_η^2 may be different for each clock. While reasonable guesses are available for the σ_ε^2 and σ_η^2, maximum likelihood estimates of these parameters, along with approximate confidence intervals, can be obtained from a sequence of observations.

When observations are taken, one of the clocks is used as a reference clock, and time differences between this clock and each of the other clocks are recorded. This gives an observation vector $Z(t)$ of length $m-1$, and an observation equation of the form

$$Z(t)=H(t)X(t)+V(t) \tag{3.9}$$

where $H(t)$ is a matrix indicating which clock differences are observed. If the first clock is the reference clock, $H(t)$ is of the form

$$H(t)=\begin{bmatrix} 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & \\ 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & \\ \vdots & & & & & & & & \end{bmatrix} \tag{3.10}$$

and has dimension $(m-1)$ by $2m$. If some of the observations are missing or deleted because of error detection, rows of $H(t)$ are eliminated. $V(t)$ is a vector of observational errors. These errors are very small when reading clock differences, and the standard deviations can be assumed known from the electronics of the measuring instruments. The covariance matrix of $V(t)$, R, is assumed to be diagonal with constant diagonal elements representing the variance of an observational error. While these variances are small, the inclusion of R in the recursion does add numerical stability.

4. Recursive calculation of −2 ln likelihood with error detection. Usually the initial time of each clock is determined by comparison with a time standard, and the initial frequency of each clock is determined from its value in a currently running time scale. $P(0|0)$ is set as a diagonal matrix with the diagonal elements corresponding to time set equal to zero (or a small number), and the diagonal elements corresponding to

frequency set equal to zero if the initial frequency is considered known, or a large number if the frequency is considered unknown and is to be estimated by the recursion. Since only time differences are observed, absolute time and absolute frequency can not be determined by the algorithm, so these must be tied to some standard during initialization. The initial frequency of at least one clock must be assumed to be known fairly accurately. There are frequency standards for this purpose.

While observations are nominally taken at equally spaced time intervals, for reasons usually related to hardware or power problems, all the observations are not equally spaced. Also, clocks are taken out of the ensemble for repair, and then returned. Unless the cesium beam tube has been replaced, it is assumed that the clock parameters have not changed. Replacing the cesium beam tube essentially makes it a new clock, and new parameters must be estimated.

The innovation vector and its covariance matrix are the keys to diagnostic checking of the model, and error detection. If the model is correct and the parameters properly estimated, the innovations should be normally distributed with possibly different covariance matrices $C(t)$, and be uncorrelated at different time points. It is at this point that error detection and correction can be carried out, details of which are given in Jones and Tryon (1983).

5. Results. The results presented here represent the final analysis of the National Bureau of Standards, Boulder, Colorado, daily clock data, and update the results given in Jones and Tryon (1983). The Kalman filter has been converted to a continuous time model. Daily observations are available on twelve clocks from April 1, 1981, to June 30, 1982. During the summer of 1982 conversion took place for automatic data collection every two hours using a PDP 11/03 system for data acquisition, and a PDP 11/70 for data storage and analysis.

Numerical optimization was carried out using a version of the quasi-Newton algorithms given in Dennis and Schnabel (1983), and coded by Schnabel et al. (1986). Table 1 shows the parameter estimates for each clock with standard errors estimated by calculating the Hessian of -2 ln likelihood numerically at the solution and inverting it. This also gives the correlation matrix of the estimated parameters. The correlations are surprisingly low. The highest correlations (in absolute value) were between the

TABLE 1

Estimated standard deviation of the random walk in time σ_ε and standard deviation of the random walk in frequency σ_η and their standard errors.

		σ_ε		σ_η	
Clock	n	Estimate	s.e.	Estimate	s.e.
1316	450	4.15	0.21	0.61	0.23
167	446	13.43	0.60	0.96	0.44
137	446	11.38	0.54	2.16	0.49
61	67	6.83	0.95	2.95	1.02
352	439	8.90	0.50	3.09	0.56
323	420	2.88	0.17	0.86	0.16
1375	443	10.68	0.49	1.48	0.34
NBS4	162	1.73	0.26	0.86	0.32
113	439	9.25	0.47	3.71	0.48
8	448	9.14	0.44	2.45	0.41
601	383	2.38	0.15	0.36	0.15
PHM4	286	1.25	0.17	0.64	0.16

estimates of σ_ε and σ_η for each clock, and ranged from -0.48 to -0.13 with a median value of -0.28. The only other correlations larger than 0.13 in absolute value occurred between the estimate of σ_η for the reference clock (1316) and σ_η for clock 167 ($\rho = 0.34$) and between σ_η for the reference clock and σ_η for clock 601 ($\rho = -0.27$). The median correlation between σ_η for the reference clock and σ_η for the other clocks was 0.06.

6. Conclusion. Many processes in the real world occur in continuous time, and continuous time models are therefore appropriate. When observations are taken at unequally spaced time points, continuous time models are necessary. The state space formulation provides great flexibility for modeling and generating the likelihood function for the purpose of estimating parameters. From the continuous time model, a discrete time recursion can be generated at the observation times, and this recursion can be used to generate the likelihood function for a given value of unknown parameters. These unknown parameters can occur anywhere in the model; for example, they may be variances of random inputs. The use of the recursion to generate a function evaluation for a nonlinear optimization routine gives maximum likelihood estimates of the parameters.

This paper has shown that two continuous time models which have appeared in the statistical literature are special cases of a more general continuous time model. These special cases are a continuous time autoregression observed with error, and polynomials with integrated random walk error structure. The application of these methods to modeling cesium beam atomic clocks developed because existing algorithms for estimating time from an ensemble of clocks were empirical and did not incorporate the physics of the clocks. The project, which extended over a time period of three years, started by fitting these state space models to actual data and examining residuals or innovations for whiteness. When it was determined that the model gave a good fit to the data, the project progressed to develop a production algorithm for actually producing a time scale from the raw data without human intervention. This meant that all error checking and correction had to be automated. On September 1, 1982, this algorithm became the algorithm for one of the two official time scales of the National Bureau of Standards, TA(NBS).

Acknowledgments. David W. Allan and James A. Barnes of the Time and Frequency Division, National Bureau of Standards, Boulder, Colorado, suggested this approach for a time scale algorithm and worked closely with the authors during its development. This paper was written while the first author was on sabbatical during the 1982–83 academic year at the Department of Biostatistics, School of Public Health and Community Medicine, University of Washington, Seattle, WA 98195, and represents joint research with Peter V. Tryon who died August 15, 1982, at the age of 41 after a six year bout with cancer. The first author would also like to thank Dr. D. B. Percival, who read the manuscript, supplied the data for the figure, and made helpful suggestions.

REFERENCES

M. S. BARTLETT (1946), *On theoretical specification of sampling properties of autocorrelated time series*, J. Roy. Statist. Soc. Ser. B, 8, pp. 27–41.

G. E. P. BOX AND G. M. JENKINS (1976), *Time Series Analysis: Forecasting and Control*, revised edition, Holden-Day, San Francisco.

J. CHAFFEE (1982), OAO Corporation internal memo, 2 March 1982, P.O. Box 1897, Santa Maria, CA 93454.

J. E. DENNIS AND R. B. SCHNABEL (1983), *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ.

J. L. DOOB (1953), *Stochastic Processes*, John Wiley, New York.

A. GELB (1974), *Applied Optimal Estimation*, MIT Press, Cambridge, MA.

A. C. HARVEY AND G. D. A. PHILLIPS (1979), *Maximum likelihood estimation of regression models with autoregressive-moving average disturbances*, Biometrika, 66, pp. 49–58.

R. H. JONES (1977), *Spectrum estimation with unequally spaced data*, Preprints: Fifth Conference on Probability and Statistics in Atmospheric Sciences, American Meteorological Society, Boston, pp. 277–282.

R. H. JONES (1980), *Maximum likelihood fitting of* ARMA *models to time series with missing observations*, Technometrics, 22, pp. 389–395.

R. H. JONES (1981), *Fitting a continuous time autoregression to discrete data*, Applied Time Series Analysis II, D. F. Findley, ed., Academic Press, New York, pp. 651–682.

R. H. JONES (1984), *Fitting multivariate models to unequally spaced data*, Time Series Analysis of Irregularly Observed Data, E. Parzen, ed., Lecture Notes in Statistics 25, Springer-Verlag, New York, pp. 158–188.

R. H. JONES AND P. V. TRYON (1983), *Estimating time from atomic clocks*, J. Res. National Bureau of Standards, 88, pp. 17–24.

R. E. KALMAN (1960), *A new approach to linear filtering and prediction problems*, ASME Trans., Part D (J. Basic Engineering), 82, pp. 35–45.

R. E. KALMAN AND R. S. BUCY (1961), *New results in linear filtering and prediction theory*, ASME Trans., Part D (J. Basic Engineering), 83, pp. 95–108.

P. M. ROBINSON (1977), *Estimation of a time series model from unequally spaced data*, Stochastic Processes and Their Applications, 6, pp. 9–24.

J. RUTMAN (1978), *Characterization of phase and frequency instabilities in precision frequency sources: fifteen years of progress*, Proc. IEEE, 66, pp. 1048–1075.

R. B. SCHNABEL, J. E. KOONTZ AND B. E. WEISS (1986), *A modular system of algorithms for unconstrained minimization*, ACM Trans. Math. Software, to appear.

F. C. SCHWEPPE (1965), *Evaluation of likelihood functions for gaussian signals*, IEEE Trans. Inform. Theory, IT-11, pp. 61–70.

P. V. TRYON AND R. H. JONES (1983), *Estimation of parameters in models for cesium beam atomic clocks*, J. Res. National Bureau of Standards, 88, pp. 3–16.

G. WAHBA (1978), *Improper priors, spline smoothing and the problem of guarding against model errors in regression*, J. Roy. Statist. Soc. Ser. B, 40, pp. 364–372.

W. E. WECKER AND C. F. ANSLEY (1983), *The signal extraction approach to nonlinear regression and spline smoothing*, J. Amer. Statist. Assoc., 78, pp. 81–89.

H. L. WEINERT, U. B. DESAI AND G. S. SIDHU (1979), ARMA *splines, system inverses, and least-squares estimation*, SIAM J. Control Optim., 17, pp. 525–536.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 1, January 1987
© 1987 Society for Industrial and Applied Mathematics
007

A MODIFIED VERSION OF HANDSCOMB'S ANTITHETIC VARIATES THEOREM*

Y. R. RUBINSTEIN† AND G. SAMORODNITSKY†

Abstract. We consider a modified version of Handscomb's Antithetic Variates theorem and prove the existence of exact minimum rather than infinimum and that this minimum is achieved by using only one random number. Practical procedures for finding the optimal cumulative distribution function are given, for estimating the expected value of the response difference of a pair of arbitrary functions of scalar arguments.

Key words. antithetic variates, common random numbers, Monte Carlo method simulation, variance reduction

1. Introduction. The problem of obtaining accurate estimators while simulating complex stochastic systems is of importance in Monte Carlo simulation. Recently there has been a rapid growth of interest in use of variance reduction techniques in simulation analysis for the purpose of improving accuracy when estimating parameters of complex systems.

One of the widely used variance reduction techniques is the *antithetic random variables method* first introduced by Hammersley and Morton (1956) and further extended by Handscomb (1958) and Wilson (1979). For more references on antithetic random variables see Page (1967), Granovsky (1982), Mitchel (1973), Kleijnen (1974), Schruben and Margolin (1978), Schruben (1979), Wright and Ramsay (1979), Gal, Rubinstein and Ziv (1984), Rubinstein (1981), Grant and Solberg (1982), Rubinstein and Samorodnitsky (1986), Sullivan, Hayya and Schaul (1982), Fishman and Huang (1981), Fishman (1983), Rubinstein, Samorodnitsky and Shaked (1985), and Wilson (1984).

Use of antithetic (common) random variables leads to the following two advantages: the first one is associated with generating only one random number, and the second one with variance reduction.

The present work presents new results related to the theory of antithetic random variables and extends, in some sense, Handscomb's (1958) antithetic variables method.

Suppose that we need to estimate

$$\theta = E\left[\sum_{i=1}^{n} \psi_i(X_i)\right] \tag{1}$$

with a minimal variance, where ψ_i are given functions, and X_i are independent random variables (r.v.'s) with known cumulative distribution functions (cdf's) $F_{X_i}(x)$, $i = 1, 2, \cdots, n$. The antithetic variable method consists of inducing an appropriate dependence between X_i's in such a way that the variance of the estimator

$$\hat{\theta} = \sum_{i=1}^{n} \psi_i(X_i) \tag{2}$$

is reduced while preserving the marginal distributions of X_i's. In other words for given marginal cdf's $F_{X_i}(x)$, $i = 1, 2, \cdots, n$ the use of the antithetic variable method generates

* Received by the editors August 16, 1984, and in final revised form June 15, 1985. This work was supported by a fund for research promotion at the Technion, under contract 190-515.

† Technion Israel Institute of Technology, Haifa, Israel.

a joint cdf $F_{X_1,\cdots,X_n}(\cdot)$ for which the variance of (2) is less than the variance in the case when the random variables $X_1, \cdots, X_n$ are independent.

Assume that each random variable X_i can be generated by using the inverse transform method, i.e.

$$X_i \overset{D}{=} F_{X_i}^{-1}(U_i) = \inf\{x: F_{X_i}(x) \geqq U_i\}, \tag{3}$$

where $\overset{D}{=}$ means equal in distribution, and U_i, $i = 1, 2, \cdots, n$ are random variables uniformly distributed on (0, 1). Then (1) can be rewritten as

$$\theta = E\left\{\sum_{i=1}^{n} \psi_i[F_{X_i}^{-1}(U_i)]\right\} = E\left[\sum_{i=1}^{n} g_i(U_i)\right] \tag{4}$$

where $g_i(U_i) = \psi_i[F_{X_i}^{-1}(U_i)]$.

Now we can restate the variance reduction problem as: given $g_1, \cdots, g_n$ choose a joint cdf of $U_1, \cdots, U_n$, denoted $F_{U_1,\cdots,U_n}(\cdot)$, to minimize $\operatorname{Var} \hat{\theta} = \operatorname{Var}[\sum_{i=1}^{n} g_i(U_i)]$.

Handscomb's (1958) antithetic random variables theorem reads as follows.

THEOREM. *If I denotes the infimum of* $\operatorname{Var}(\sum_{j=1}^{n} g_j(U_j))$ *when all possible stochastic or functional dependences between the U_j are considered, then, provided that the g_j are bounded for all $j = 1, \cdots, n$*

$$\inf_{\substack{x_j \in \mathfrak{x}, \\ j=1,\cdots,n}} \operatorname{Var} \sum_{j=1}^{n} g_j[x_j(U)] = I, \tag{5}$$

where $\mathfrak{x}$ denotes the class of all functions $x(U)$ such that

(i) *$x(U)$ is a (one-to-one) mapping of the unit interval on itself.*

(ii) *dx/dU exists and equals* 1, *except perhaps at a finite set of points U.*

Wilson (1979) has extended this theorem to unbounded functions $g_1, \cdots, g_n$ with finite variances. Unfortunately, the theorems of Handscomb and Wilson say nothing about (a) existence of the minimal value of $\operatorname{Var} \hat{\theta}$ exactly, and (b) whether or not the minimal value of $\operatorname{Var} \hat{\theta}$ can be achieved while using *only a simple random number U*, in spite of the fact that the U appears in each $x_j(U)$ in (5).

In § 2 we present a modified version (see Theorem 1 below) of Handscomb's theorem, giving positive answers on issues (a) and (b) while removing condition (ii).

In § 3 we show that Theorem 1 can be readily extended to a rather general case (see Theorem 3 below). We chose to prove Theorem 1 rather than its generalized version (Theorem 3), only in order to provide an analogy with the well-known Handscomb theorem. In § 4, we show how our theory can be used to construct practical procedures for generating an *optimal* two-dimensional cdf $F_{U_1,U_2}(\cdot)$ which minimizes

$$\operatorname{Var}[g_1(U_1) + g_2(U_2)]. \tag{6}$$

2. A modified version of Handscomb's antithetic variates theorem.

THEOREM 1. *Let $g_1(U_1), g_2(U_2), \cdots, g_n(U_n)$ be bounded measurable functions on $[0, 1]$ and Ξ_n be the set of all possible joint* cdf*'s on $[0, 1]^n$ with uniform marginals. Then there exist transformations $h_1(U_1), \cdots, h_n(U_n)$ such that*

(i) $\min_{F \in \Xi_n} \operatorname{Var}[g_1(U_1) + g_2(U_2) + \cdots + g_n(U_n)] = \operatorname{Var}[g_1(h_1(U)) + \cdots + g_n(h_n(U))]$ *and*

(ii) $h_1(U) \sim U(0, 1), h_2(U) \sim U(0, 1), \cdots, h_n(U) \sim U(0, 1)$.

Note that on the right-hand side of (i) *the same U appears in all $h_k(\cdot)$, $k = 1, \cdots, n$.*

We denote by F^* the optimal cdf generated by the random variables $h_1(U), \cdots, h_n(U)$.

The proof of Theorem 1 is based on Theorem 2 (see below). However, first we prove the following.

LEMMA 1. *Let Ξ_n be defined as in Theorem 1 and $F \in \Xi_n$. Then there exist transformations $(t_1(U), t_2(U), \cdots, t_n(U))$ such that*

(i) $t_1(U) \sim U(0,1)$, $t_2(U) \sim U(0,1)$, $\cdots$, $t_n(U) \sim U(0,1)$,

(ii) *the joint distribution of $(t_1(U), t_2(U), \cdots, t_n(U))$ is F.*

Proof. Let $U \sim U(0,1)$. Write it as

$$U = 0.\ a_1 a_2 a_3 \cdots$$

in some base (say, base 10). Set

$$\begin{aligned} U_1 &= 0.\ a_1 a_{n+1} a_{2n+1} \cdots \\ U_2 &= 0.\ a_2 a_{n+2} a_{2n+2} \cdots \\ &\vdots \\ U_n &= 0.\ a_n a_{2n} a_{3n} \cdots ; \end{aligned}$$

then it is clear that $U_1, U_2, \cdots, U_n$ are independent random variables each distributed $U(0,1)$.

Set

$$\begin{aligned} & t_1(U) = U_1 \\ & \vdots \\ & t_i(U) = F^{-1}(U_i \mid t_1(U), \cdots, t_{i-1}(U)), \qquad i = 2, 3, \cdots, n. \end{aligned} \tag{7}$$

Here $F(x_i \mid x_1, \cdots, x_{i-1})$ is the conditional distribution of x_i given $x_1, \cdots, x_{i-1}$.

Then, (1) follows from the fact that $F \in \Xi_n$ and (2) follows from the fact that (7) is essentially the inverse transform method (Rubinstein (1981)). Q.E.D.

As an immediate consequence of this lemma we obtain the following analogue of Handscomb's (1958) antithetic variables theorem.

COROLLARY. *Let $g_1(U_1), \cdots, g_n(U_n)$ be measurable functions on $[0,1]$, with finite variance. Then*

$$\inf_{F \in \Xi_n} \operatorname{Var}[g_1(U_1) + \cdots + g_n(U_n)] = \inf_{\{h_1(U), \cdots, h_n(U)\}} \operatorname{Var}[g_1(h_1(U)) + \cdots + g_n(h_n(U))], \tag{8}$$

where $h_i(U) \sim U(0,1)$, $i = 1, \cdots, n$ and the same U appears in each h_i.

Note that the conditions in the corollary are weaker than in Handscomb's (1958) theorem since the restriction $dh(u)/du = 1$ is removed.

THEOREM 2. *Let $g(\mathbf{U})$ be any continuous function on $[0,1]^n$, and Ξ_n be the set of all distributions on $\mathbb{R}^n$ with uniform $[0,1]$ marginals. Then there exists $F^* \in \Xi_n$ such that*

$$\min_{F \in \Xi_n} E_F(g(\mathbf{U})) = E_{F^*}(g(\mathbf{U})). \tag{9}$$

Proof. Let

$$\inf_{F \in \Xi_n} E_F(g(\mathbf{U})) = a. \tag{10}$$

Then there exists a sequence $\{F_m\}_{m=1}^{\infty}$ in Ξ_n such that

$$\lim_{m \to \infty} E_{F_m}(g(\mathbf{U})) = a. \tag{11}$$

Note that the sequence $\{F_m\}$ is tight, because it is restricted to $[0,1]^n$.

Consequently, by Helley's theorem (Billingsley (1979, Thm. 29.3)), there exists a subsequence $\{F_{m_i}\}$ and a distribution function F^* such that

$$F_{m_i} \Rightarrow F^* \text{ in distribution.} \tag{12}$$

The function F^* has uniform marginals because $\{F_{m_i}\}$ has. Now by weak convergence (Billingsley (1979, Thm. 29.1)) we have

$$E_{F^*}(g(\mathbf{U})) = \lim_{i\to\infty} E_{F_{m_i}}(g(\mathbf{U})) = a \tag{13}$$

and this completes the proof. Q.E.D.

Proof of Theorem 1. Let us rewrite

$$\text{Var}_F\,[g_1(U_1)+g_2(U_2)+\cdots+g_n(U_n)] = E_F(g_1(U_1)+g_2(U_2)+\cdots+g_n(U_n)-\theta)^2. \tag{14}$$

By Lusin's theorem (see Natanson (1974)) we have that for every $m>0$ there exist *continuous* functions $\hat{g}_{m1}(U), \cdots, \hat{g}_{mn}(U)$ such that

$$\begin{aligned} &\text{(a)}\quad P(\hat{g}_{mi}(U) \neq g_i(U)) \leqq 2^{-m} \quad \forall i = 1, \cdots, n, \\ &\text{(b)}\quad |\hat{g}_{mi}(U)| \leqq K \quad \forall m, i, u. \end{aligned} \tag{15}$$

Denote $\hat{g}_m = (\hat{g}_{mi}(U_1)+\cdots+\hat{g}_{mn}(U_n)-\theta)^2$ and $\hat{g} = (g_1(U_1)+\cdots+g_n(U_n)-\theta)^2$. Then

$$\begin{aligned} &\text{(a)}\quad P_F(\hat{g}_m \neq \hat{g}) \leqq n2^{-m} \quad \text{for every } F \in \Xi_n, \\ &\text{(b)}\quad |\hat{g}_m| \leqq (nK+|\theta|)^2 = \hat{K} \quad \forall m. \end{aligned} \tag{16}$$

Now, by Theorem 2 for every $m>0$ there exists a $F_m \in \Xi_n$ such that

$$\inf_{F\in\Xi_n} E_F\hat{g}_m = E_{F_m}\hat{g}_m. \tag{17}$$

Let us show that the following statement holds

$$\lim_{m\to\infty} E_{F_m}\hat{g}_m \leqq \inf_{F\in\Xi_n} E_F\hat{g}. \tag{18}$$

Suppose that (18) does not hold and

$$\lim_{m\to\infty} E_{F_m}\hat{g}_m > L, \tag{19}$$

where $L = \inf_{F\in\Xi_n} E_F\hat{g}$.

By (16) we have for any $F \in \Xi_n$

$$E_F\hat{g} > E_F\hat{g}_m - \hat{K}2^{-m+1}. \tag{20}$$

Consequently,

$$E_{F_m}\hat{g}_m = \inf_{F\in\Xi_n} E_F\hat{g}_m < \inf_{F\in\Xi_n} (E_F\hat{g} + \hat{K}2^{-m+1}) = L + \hat{K}2^{-m+1}.$$

So

$$\lim_{m\to\infty} E_{F_m}\hat{g}_m \leqq \lim_{m\to\infty} (L + K2^{-m+1}) = L$$

and this proves (18).

Consequently, there exists a subsequence of $\{F_m\}$ such that $\lim_{m_K\to\infty} E_{F_{m_K}}\hat{g}_{m_K}$ exists and

$$\lim_{m_K\to\infty} E_{F_{m_K}}\hat{g}_{m_K} \leqq L.$$

Without loss of generality we may assume that this is the original sequence, i.e.

$$\lim_{m\to\infty} E_{F_m}\hat{g}_m = a \leqq L. \tag{21}$$

Now, by Helley's theorem (Billingsley (1979, Thms. 25.9, 29.3)) there exists such a subsequence of $\{F_m\}$ and $\hat{F}\in\Xi_n$ that

$$F_{m_K} \overset{D}{\Rightarrow} \hat{F}.$$

We suppose again, without loss of generality, that

$$F_m \overset{D}{\Rightarrow} \hat{F}. \tag{22}$$

Let us show that

$$E_{\hat{F}}\hat{g} \leqq L. \tag{23}$$

By (16) and by the first Borell-Cantelli lemma (see, for example, Billingsley (1979))

$$\hat{g}_{mi} \to \hat{g}_i \quad \text{a.s. } \forall i = 1, \cdots, n. \tag{24}$$

By Egorov's theorem (Natanson (1974, Thm. 4.3.5)) for every $\delta > 0$ there exist sets $E_\delta^1, \cdots, E_\delta^n$ such that

$$P(E_\delta^i) > 1-\delta \quad \forall i \tag{25}$$

and on E_δ^i the convergence in (24) is uniform.

Thus, for any $\varepsilon > 0$ there exist $N_1(\varepsilon), \cdots, N_n(\varepsilon)$ such that $\forall m > N_i = N_i(\varepsilon)$, for almost every u in E_δ^i:

$$|\hat{g}_{mi}(u) - \hat{g}_i(u)| \leqq \varepsilon. \tag{26}$$

Thus on the set $\hat{E}_\delta = \prod_{i=1}^n E_\delta^i$ for every $m > N = \max(N_1(\varepsilon), \cdots, N_n(\varepsilon))$

$$|\hat{g}_m - \hat{g}| \leqq 2K_1 n\varepsilon \tag{27}$$

and

$$P_F(\hat{E}_\delta) \geqq 1 - n\delta \quad \forall F \in \Xi_n. \tag{28}$$

Now, when $m > N$ and $l > N$

$$\begin{aligned}
&\left|\int_{[0,1]^n} \hat{g}\, d\hat{F} - \int_{[0,1]^n} \hat{g}_m\, dF_m\right| \\
&\quad \leqq \left|\int_{\hat{E}_\delta} \hat{g}\, d\hat{F} - \int_{\hat{E}_\delta} \hat{g}_m\, dF_m\right| + \left|\int_{\hat{E}_\delta^c} \hat{g}\, d\hat{F}\right| + \left|\int_{\hat{E}_\delta^c} \hat{g}_m\, dF_m\right| \\
&\quad \leqq \left|\int_{\hat{E}_\delta} \hat{g}\, d\hat{F} - \int_{\hat{E}_\delta} \hat{g}\, dF_m\right| + \left|\int_{\hat{E}_\delta} \hat{g}\, dF_m - \int_{\hat{E}_\delta} \hat{g}_m\, dF_m\right| + 2\hat{K}n\delta \\
&\quad \leqq \left|\int_{\hat{E}_\delta} \hat{g}\, d\hat{F} - \int_{\hat{E}_\delta} \hat{g}_l\, dF_m\right| + \left|\int_{\hat{E}_\delta} \hat{g}_l\, dF_m - \int_{\hat{E}_\delta} \hat{g}\, dF_m\right| \\
&\qquad + \int_{\hat{E}_\delta} |\hat{g} - \hat{g}_m|\, dF_m + 2\hat{K}n\delta
\end{aligned} \tag{29}$$

$$\leqq \left| \int_{\hat{E}_\delta} \hat{g}\, d\hat{F} - \int_{\hat{E}_\delta} \hat{g}_l\, d\hat{F} \right| + \left| \int_{\hat{E}_\delta} \hat{g}_l\, d\hat{F} - \int_{\hat{E}_\delta} \hat{g}_l\, dF_m \right| + \int_{\hat{E}_\delta} |\hat{g} - \hat{g}_l|\, dF_m + 2K_1 n\varepsilon + 2\hat{K} n\delta$$

$$\leqq \int_{\hat{E}_\delta} |\hat{g} - \hat{g}_l|\, d\hat{F} + \left| \int_{\hat{E}_\delta} \hat{g}_l\, d\hat{F} - \int_{\hat{E}_\delta} \hat{g}_l\, dF_m \right| + 4K_1 n\varepsilon + 2\hat{K} n\delta$$

$$\leqq \left| \int_{\hat{E}_\delta} \hat{g}_l\, d\hat{F} - \int_{\hat{E}_\delta} \hat{g}_l\, dF_m \right| + 6K_1 n\varepsilon + 2\hat{K} n\delta.$$

Further, fix $l > N$ and let $m \to \infty$.

Since $\hat{g}_l$ is continuous, bounded, and $F_m \overset{D}{\Rightarrow} \hat{F}$, the first term in (29) goes to zero (Billingsley (1979, Thm. 28.1)). Because ε and δ are arbitrary, (29) means that

$$\int_{[0,1]^n} \hat{g}_m\, dF_m \to \int_{[0,1]^n} \hat{g}\, d\hat{F}.$$

Consequently,

$$a = \lim_{m\to\infty} \int_{[0,1]^n} \hat{g}\, d\hat{F} \leqq L$$

by (21). This establishes (23). But (23) means simply $E_{\hat{F}}\hat{g} = L$, and the proof of Theorem 1 is completed. Q.E.D.

3. Generalization. Theorem 1 can be readily generalized as follows.

THEOREM 3. *Let $\varphi[g_1(\mathbf{U}_1), \cdots, g_n(\mathbf{U}_n)]: R^n \to R^1$ be a continuous function defined on a closed bounded set, where $\mathbf{U}_i = (U_i^1, \cdots, U_i^m)$, $i = 1, \cdots, n$ is an m-dimensional vector with independent components each distributed $U(0,1)$ and $g_1(\cdot), \cdots, g_n(\cdot)$ be bounded measurable functions on $[0,1]^m$. Let Ξ_n^m be the set of all possible joint distributions on $[0,1]^{n\times m}$ with uniform m-dimensional marginals. Then there exist transformations $h_i^j(U)$, $i = 1, \cdots, n$, $j = 1, \cdots, m$ such that*

$$\min_{F\in\Xi_n^m} E\varphi[g_1(U_1^1, \cdots, U_1^m), \cdots, g_n(U_n^1, \cdots, U_n^m)] = E\varphi[g_1(h_1^1(U^1), \cdots, h_1^m(U^m)) + \cdots + g_n(h_n^1(U^1), \cdots, h_n^m(U^m))], \tag{30}$$

where

$$h_i^j(U^j) \sim U(0,1), \qquad i = 1, \cdots, n, \quad j = 1, \cdots, m, \tag{31}$$

$$h_i^j(U^j) \text{ does not depend on } h_i^k(U^k), \qquad k \neq j, \quad k = 1, \cdots, m, \quad i = 1, \cdots, n. \tag{32}$$

The proof of this theorem is omitted since it coincides almost verbatim with the proof of Theorem 1.

COROLLARY. *Suppose that we want to estimate*

$$\theta = E\left[\sum_{i=1}^n g_i(\mathbf{U}_i)\right]$$

minimal variance. Denote

$$\varphi[g_1(\mathbf{U}_1), \cdots, g_n(\mathbf{U}_n)] = \sum_{i=1}^n [g_i(\mathbf{U}_i) - \theta]^2.$$

Then, it follows directly from Theorem 3 *that there exist transformations* $h_i(U)$ *satisfying* (31) *and* (32) *such that*

$$\min_{F\in\Xi_n^m} \operatorname{Var}[g_1(\mathbf{U}_1)+\cdots+g_n(\mathbf{U}_n)] = \operatorname{Var}[g_1(\mathbf{h}_1(\mathbf{U}))+\cdots+g_n(\mathbf{h}_n(\mathbf{U}))], \tag{33}$$

where $\mathbf{h}_i(U^1)=(h_i^1(U^1),\cdots,h_i^m(U^m))$, $i=1,\cdots,n$.

This corollary presents the multidimensional version of Theorem 1. Obviously, in the particular case where $\mathbf{U}_i$ is a one-dimensional random variable we obtain Theorem 1.

4. The optimal procedures for $n=2$. Theorems 1 and 3 are pure existence theorems. They say nothing about constructing the optimal transformation $h_1(\mathbf{U}),\cdots,h_n(\mathbf{U})$ which minimizes $\operatorname{Var}\sum_{i=1}^n g_i(\mathbf{U}_i)$.

In this section we present two *practical* procedures (see Procedures 1 and 2 below) for constructing $h_1(U)$ and $h_2(U)$ for the particular case $n=2$ and therefore for generating the optimal cdf F_{U_1,U_2} which minimizes

$$\operatorname{Var}[g_1(U_1)+g_2(U_2)].$$

It is interesting to note that the application of antithetic variables to the case $n=2$ has deserved a lot of attention in the recent literature on simulation (see Page (1967), Mitchel (1973), Kleijnen (1974), Schruben and Margulin (1978), Schruben (1979), Sullivan, Hayya and Schaul (1982), Rubinstein and Samorodnitsky (1986), Rubinstein, Samorodnitsky and Shaked (1985)). Most of these applications are related, however, to the case where both g_1 and g_2 are monotone functions. To the best of our knowledge there are only two papers (see Rubinstein and Samorodnitsky (1986) and Rubinstein, Samorodnitsky and Shaked (1985)) in which a special class of nonmonotonic pairs of functions g_1, g_2 (called corresponding functions) are defined for which the antithetic (common) random variables are optimal. In this work we shall find the optimal cdf $F^*_{U_1,U_2}$ which minimizes $\operatorname{Var}[g_1(U_1)+g_2(U_2)]$, without making any specific assumption about g_1 and g_2.

(a) *Theoretical results.* Let $g(u)$ be any measurable function on $[0,1]$. Let $F_g(x)=P[g(U)\leqq x]$ be the cdf of the random variable $g(U)$. By the inverse transform method $F_g^{-1}(U)$ is distributed as $g(U)$. It is clear that $F_g^{-1}(1-U)$ is also distributed as $g(U)$. $F_g^{-1}(U)$ and $F_g^{-1}(1-U)$ are, respectively, nondecreasing and nonincreasing functions of U. We denote $g^*(u)=F_g^{-1}(u)$ and $g_*(u)=F_g^{-1}(1-u)$ and shall call them accordingly nondecreasing and nonincreasing functions of U distributed as $g(U)$.

PROPOSITION 1. *For every measurable function* $g(u)$ *on* $[0,1]$ *there exists a pair of transformations* $\{\bar{h}_g(u), \underline{h}_g(u)\}$ *such that*

(1) $\bar{h}_g(U)\sim U(0,1)$ *and* $\underline{h}_g(U)\sim U(0,1)$,

(2) $g^*[\bar{h}_g(U)]=g(U)$ *a.s. and* $g_*[\underline{h}_g(U)]=g(U)$ *a.s.*

Proof. We prove the proposition for $\bar{h}_g$. The proof for $\underline{h}_g$ is essentially the same. Denote for simplicity $\bar{h}_g$ by h and set

$$h(u)=P(g(U)<g(u))+P(g(U)=g(u),\ U\leqq u). \tag{34}$$

It is rather straightforward to see that $h(U)\sim U(0,1)$.

Let us show that $g^*[h(U)]=g(U)$ a.s. By definition

$$g^*[h(u)]=F_g^{-1}[h(u)]=\inf\{y\colon F_g(y)\geqq h(u)\}.$$

Clearly, $F_g(g(u))=P(g(U)\leqq g(u))\geqq h(u)$. Consequently, $g^*[h(u)]\leqq g(u)$. Let us

find the cases where

$$g^*[h(u)] < g(u). \tag{35}$$

Denote by $F^-(x)$ the value of F just to the left of point x, and denote $y^* = g^*[h(u)]$. It follows from (35) that $F_g(y^*) \leqq F_g^-(g(u))$. On the other hand $F_g(y^*) \geqq h(u) \geqq F_g^-(g(u))$, so it must be

$$F_g(y^*) = F_g^-(g(u)). \tag{36}$$

Fix u that satisfies (35).

Suppose first that $P(g(U) = g(u)) > 0$.

Denote $C_u = \{v: g(v) = g(u)\}$ and $v_u = \sup\{v \in C_u: P((0, v) \cap C_u) = 0\}$. Then $P((0, v_u) \cap C_u) = 0$, while $\forall v \in ((v_u, 1) \cap C_u)$, $P((0, v) \cap C_u) > 0$. So

$$P(g(U) = g(v),\ U \leqq v) > 0.$$

This means that $h(v) > F_g^-(g(v)) = F_g(y^*)$ and therefore v does not satisfy (35). Consequently, the measure of the points belonging to C_u and satisfying (35) is zero. Since the number of different sets C_u with positive measure is at most countable, the set of all points u satisfying (35) and $P(g(U) = g(u)) > 0$ has measure zero.

Suppose now that $P(g(U) = g(u)) = 0$. Since (36) implies

$$P\{y^* < g(U) < g(u)\} = 0,$$

we conclude that $g(u)$ belongs to some interval of positive length on which F_g is constant. The function F_g can have at most countable number of such different intervals. But $g(U)$ cannot fall with positive probability within any of these intervals. Consequently, the measure of the set of all points u satisfying (35) and $P(g(U) = g(u)) = 0$ is also zero. This completes the proof. Q.E.D.

Remark. There are two cases of particular interest.

(i) $g(u)$ takes only two values, 0 and 1 (say), and $E = \{u: g(u) = 0\}$, then the transformation (34) has the form

$$h(u) = \begin{cases} P\{E \cap [0, u]\}, & u \in E, \\ P(E) + P\{E^c \cap [0, u]\}, & u \in E^c. \end{cases} \tag{37}$$

(ii) $P\{g(U) = g(u)\} = 0$ for every $u \in [0, 1]$, which means that the random variable $g(U)$ has a continuous cdf. In this case the transformation (34) is just

$$\bar{h}_g(u) = h(u) = P[g(U) \leqq g(u)] = F_g(g(u)) \tag{38}$$

and similarly

$$\underline{h}_g(u) = P[g(U) > g(u)] = 1 - F_g(g(u)) = 1 - \bar{h}_g(u). \tag{39}$$

Note, however, that when the cdf of the random variable $g(U)$ is not continuous the transformations (38) and (39) will generally not be uniformly distributed on [0, 1].

Example 1. Let (see Fig. 1)

$$g(u) = \begin{cases} 4u, & 0 < u \leqq \frac{1}{4}, \\ 0.5, & \frac{1}{4} < u \leqq \frac{1}{2}, \\ 4u - 2, & \frac{1}{2} < u \leqq \frac{3}{4}, \\ 0.5, & \frac{3}{4} < u \leqq 1. \end{cases}$$

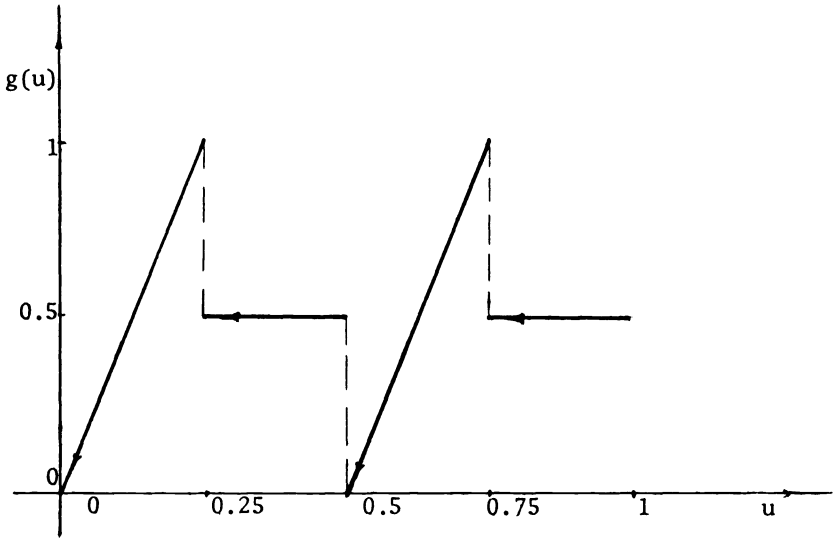


FIG. 1

Then (see Fig. 2)

$$F_g(x)=\begin{cases}\dfrac{x}{2}, & 0<x<0.5,\\ \dfrac{x}{2}+0.5, & 0.5\leqq x\leqq 1.\end{cases}$$

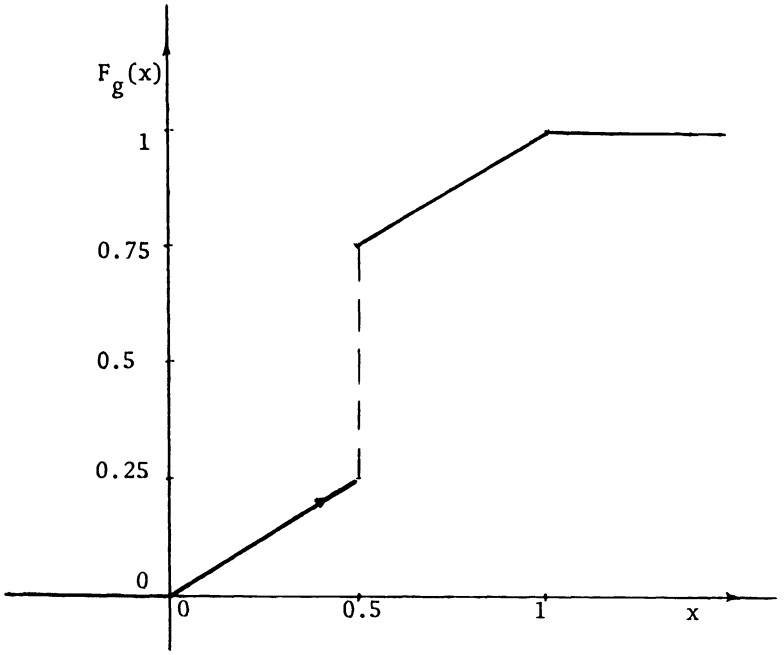


FIG. 2

Hence

$$g^*(u) = F_g^{-1}(u) = \begin{cases} 2u, & 0 < u < \frac{1}{4}, \\ 0.5, & \frac{1}{4} \leqq u \leqq \frac{3}{4}, \\ 2u-1, & \frac{3}{4} < u \leqq 1, \end{cases}$$

and consequently (see Fig. 3)

$$g_*(u) = F_g^{-1}(1-u) = \begin{cases} 1-2u, & 0 < u < \frac{1}{4}, \\ 0.5, & \frac{1}{4} \leqq u \leqq \frac{3}{4}, \\ 2-2u, & \frac{3}{4} < u \leqq 1. \end{cases}$$

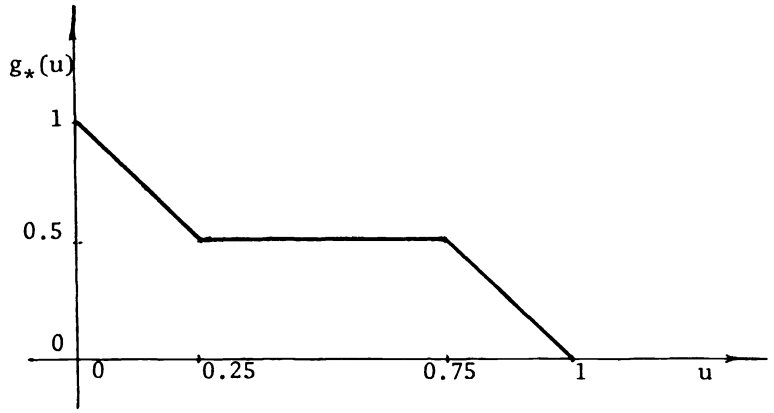


FIG. 3

By Proposition 1 there exists a pair of transformations $\{\bar{h}_g(u), \underline{h}_g(u)\}$ (regardless of the continuity of $g(u)$) which is a function of one random variable U only.

For example (see Fig. 4) one of the possible functions $h_g(u)$ is the following:

$$\underline{h}_g(u) = \begin{cases} 1-2u, & 0 < u < \frac{1}{8}, \\ 0.5-2u, & \frac{1}{8} \leqq u < \frac{1}{4}, \\ 1-u, & \frac{1}{4} \leqq u < \frac{1}{2}, \\ 2-2u, & \frac{1}{2} \leqq u < \frac{5}{8}, \\ 1.5-2u, & \frac{5}{8} \leqq u < \frac{3}{4}, \\ \frac{5}{4}-u, & \frac{3}{4} \leqq u \leqq 1. \end{cases}$$

(b) *Optimal procedures and examples.* Now, we shall use Proposition 1 to construct the pair of transformations $\{h_1(u), h_2(u)\}$ which generates the optimal cdf F^* and therefore minimizes

$$\text{Var}\,[g_1(U_1) + g_2(U_2)]. \tag{40}$$

We start with the case where one of the functions, say function g_2, is monotone.

LEMMA 2. *If $g_1(u_1)$ and $g_2(u_2)$ are two measurable functions on $[0, 1]$ and $g_2(u_2)$ is monotone then there exists a transformation $h(u)$ such that:*

(1) $h(U) \sim U(0, 1)$,

(2) $\min_{F(u_1,u_2)\in\Xi_2} \{\text{Var}\,[g_1(U_1) + g_2(U_2)]\} = \text{Var}\,\{g_1(U) + g_2[h(U)]\}$.

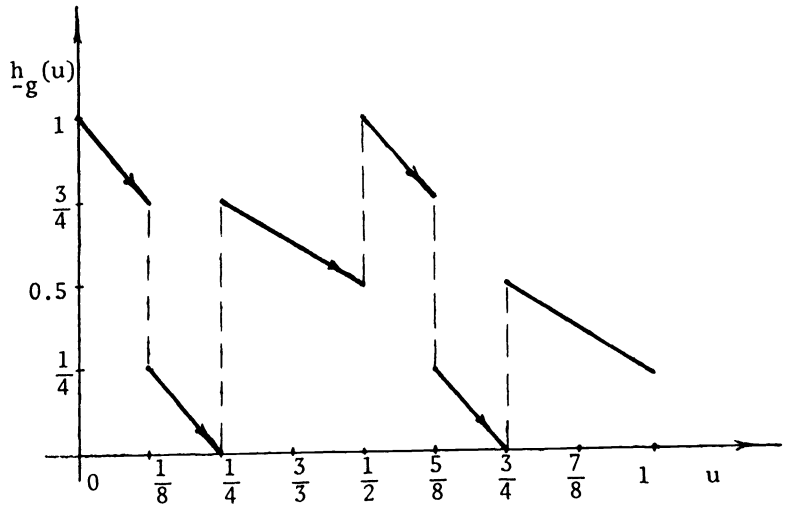


FIG. 4

Proof. Assume that $g_2(u)$ is nonincreasing in u. By Proposition 1 there exists a measure preserving transformation $\bar{h}_{g_1}(u)$ such that $g_1^*[\bar{h}_{g_1}(u)] = g_1(u)$ a.s. Set $h(u) = \bar{h}_{g_1}(u)$. Then (1) holds. To show that (2) holds let $F(u_1, u_2)$ be any cdf from the set Ξ_2 of all possible cdf's. Then

$$\min_{F_{u_1,u_2}\in\Xi_2} \operatorname{Var}[g_1(U_1)+g_2(U_2)] = \min_{F_{u_1,u_2}\in\Xi_2} \operatorname{Var}[g_1^*(\bar{h}_{g_1}(U_1))+g_2(U_2)]$$

$$\geqq \operatorname{Var}[g_1^*(U)+g_2(U)] = \operatorname{Var}[g_1^*(\bar{h}_{g_1}(U))+g_2(\bar{h}_{g_1}(U))] \tag{41}$$

$$= \operatorname{Var}[g_1(U)+g_2(\bar{h}_{g_1}(U))] = \operatorname{Var}[g_1(U)+g_2(h(U))].$$

The inequality in (41) holds since both $g_1^*(u) = F_{g_1}^{-1}(u)$ and $g_2(u)$ are monotone in opposite directions (see, for example, Gal, Rubinstein and Ziv (1984)).

The first equality after the inequality holds since $\bar{h}_{g_1}(U) \sim U(0, 1)$. The last two equalities in (41) are obvious. This completes the proof. Q.E.D.

We present now two examples demonstrating how Lemma 2 can be applied to minimize the variance (40) when g_2 is monotone.

Example 2. Let $g_2(u_2)$ be any monotone increasing function on $[0, 1]$ and $g_1(u_1)$ be a symmetric unimodal function, i.e. $g_1(0.5-\theta) = g_1(0.5+\theta)$, $\forall\theta\in[0, 0.5]$ and $g_1(u)$ is monotone on $[0, 0.5]$, for example nonincreasing. Then it is readily obtainable that

$$\bar{h}_{g_1}(u) = \begin{cases} 1-2u, & 0<u\leqq 0.5, \\ 2u-1, & 0.5<u\leqq 1, \end{cases}$$

and it follows from Lemma 2 (property (2)) that

$$\min_{F_{u_1,u_2}\in\Xi_2} \operatorname{Var}[g_1(U_1)+g_2(U_2)] = \operatorname{Var}\{g_1(U)+g_2[\bar{h}_{g_1}(U)]\}. \tag{42}$$

As a numerical example let $g_1(u_1) = \sin \pi u_1$ and $g_2(u_2) = u_2^2$. Denote $\theta_1 = g_1(U_1)+g_2(U_2)$ and $\theta_2 = g_1(U)+g_2[\bar{h}_{g_1}(U)]$. Then, for the crude Monte Carlo method and for our optimal procedure, we obtain Var $\theta_1 = 0{,}1836$ and Var $\theta_2 = 0{,}0189$, respectively. The gain in variance reduction is

$$\varepsilon = \frac{\operatorname{Var}\theta_2}{\operatorname{Var}\theta_1} = 9.71.$$

Example 3. Let $g_1(u_1)$ and $g_2(u_2)$ be a piecewise continuous function and a monotone function, respectively, and let the cdf of the random variable $g_1(U)$ be continuous. Then the following procedure can be used for minimizing the variance in (40).

Procedure 1.

1. Generate $U_1 \sim U(0, 1)$.
2. Find the set $\{u \in [0, 1]: g_1(u) \leqq g_1(u_1)\}$ which presents a union of some intervals.
3. Compute $l =$ sum of the lengths of these intervals.
4. Deliver $\theta_2 = g_1(u_1) + g_2(l)$ as a desired estimator.

To see that Procedure 1 minimizes (40) it is sufficient to write $g_2(l)$ as

$$g_2(l) = g_2\{P[g_1(U)] \leqq g_1(u_1)]\} = g_2[F_{g_1}(g_1(u_1))] = g_2[\bar{h}_{g_1}(u_1)].$$

As a numerical example take

$$g_1(u_1) = \begin{cases} e^{10u_1^2}, & 0 < u_1 \leqq 0.3, \\ 3u_1 - 0.9, & 0.3 < u_1 \leqq 1, \end{cases}$$

$$g_2(u_2) = 1 - u_2, \qquad 0 < u_2 \leqq 1.$$

For example, if $0 < u_1 \leqq 0.272$ then (see Fig. 5) $l = u_1 + e^{10u_1^2}/3$. Proceeding further we obtain

$$l = \begin{cases} u_1 + e^{10u_1^2}/3, & 0 < u_1 \leqq 0.272, \\ 0.7 + u_1, & 0.272 < u_1 \leqq 0.3, \\ u_1 - 0.3, & 0.3 < u_1 \leqq 0.633, \\ u_1 - 0.3 + \sqrt{\ln(3u_1 - 0.9)/10}, & 0.633 < u_1 \leqq 1. \end{cases}$$

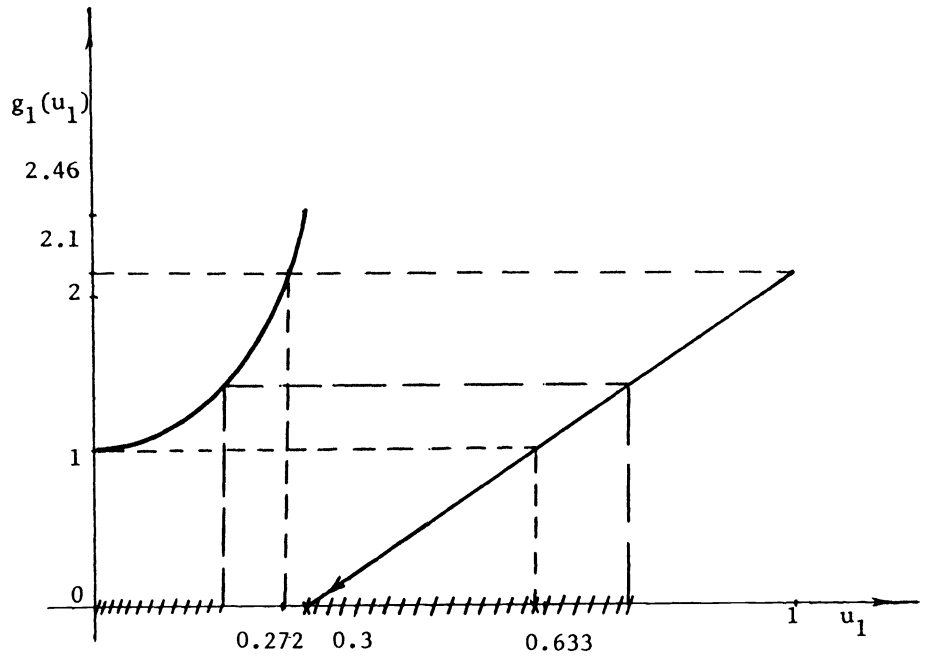


FIG. 5

By using the crude Monte Carlo method and by the optimal procedure we obtain accordingly

$$\text{Var}\,\theta_1 = 0.717 \quad \text{and} \quad \text{Var}\,\theta_2 = 0.533.$$

The gain is 25.66%.

Consider now the case where g_1 and g_2 are arbitrary measurable functions. Let us show how Proposition 1 can be used to construct a pair of transformations $\{h_1(U), h_2(U)\}$ (to generate an optimal cdf) which minimizes (40). The simplest way to do this would be to set

$$h_1(U) = \bar{h}_{g_1}^{-1}(U), \qquad h_2(U) = \underline{h}_{g_2}^{-1}(U). \tag{43}$$

However, in general the transformations $\bar{h}_g$ and $\underline{h}_g$ are not one-to-one mappings, so the inverse transformations (43) cannot be defined. To overcome this difficulty some randomization is needed.

Consider first a pair of random vectors $\{u, \bar{h}_{g_1}(u)\}$ and $\{u, \underline{h}_{g_2}(u)\}$ which defines a pair of distributions on $[0,1]\times[0,1]$, with uniform marginals. (Denote it $G_1(U^{(1)}, U^{(2)})$ and $G_2(U^{(1)}, U^{(2)})$ accordingly.) Then the procedure for minimizing (40) is as follows.

Procedure 2.

1. Generate $U \sim U(0,1)$.
2. Find the conditional distributions $G_1(U^{(1)} \mid U^{(2)} = U)$ and $G_2(U^{(1)} \mid U^{(2)} = U)$.
3. Generate $\tilde{U}_1$ and $\tilde{U}_2$ from $G_1(\cdot \mid U^{(1)} = U)$ and $G_2(\cdot \mid U^{(2)} = U)$, respectively.
4. Deliver $g_1(\tilde{U}_1) + g_2(\tilde{U}_2)$ as a desired estimator.

To see that Procedure 2 is optimal (minimizes (40)), note first that the cdf $F_{\tilde{U}_1,\tilde{U}_2}$ has uniform marginals since the cdf's G_1, G_2 do. Further from construction of the procedure it follows that $\bar{h}_{g_1}(\tilde{U}_1) = U$ and $\underline{h}_{g_2}(\tilde{U}_2) = U$. Hence, $g_1(\tilde{U}_1) = g_1^*(U)$ and $g_2(\tilde{U}_2) = g_{2_*}(U)$. Now arguing as we did in (41) we obtain:

$$\begin{aligned}
\min_{F_{u_1,u_2}\in\Xi_2} \text{Var}\,[g_1(U_1)+g_2(U_2)] &= \min_{F_{u_1,u_2}\in\Xi_2} \text{Var}\,[g_1^*[\bar{h}_{g_1}(U_1)]+g_{2*}[\underline{h}_{g_2}(U_2)]] \\
&\geqq \text{Var}\,[g_1^*(U)+g_{2_*}(U)] \\
&= \text{Var}\,\{g_1^*[\bar{h}_{g_1}(\tilde{U}_1)]+g_{2_*}[\underline{h}_{g_2}(\tilde{U}_2)]\} \\
&= \text{Var}\,[g_1(\tilde{U}_1)+g_2(\tilde{U}_2)].
\end{aligned}$$

Thus, within the set Ξ_2 of all cdf's the cdf $F_{\tilde{U}_1,\tilde{U}_2}$ minimizes (40). Q.E.D.

Remark. If the cdf's of g_1 and g_2 are continuous then it is readily seen that Procedure 2 provides a one-to-one correspondence between the values of g_1 and g_2.

Example 4. Assume that g_1 and g_2 are given as (see Fig. 6 and Fig. 7, respectively)

$$g_1(u_1) = \sin \pi u_1, \qquad 0 < u_1 \leqq 1,$$

$$g_2(u_2) = \begin{cases} 1 - 3u_2, & 0 < u_2 \leqq \frac{1}{3}, \\ \frac{3}{4}u_2 - \frac{1}{4}, & \frac{1}{3} < u_2 \leqq 1. \end{cases}$$

Let us show how to construct h_1 and h_2 in order to minimize $\text{Var}\,[g_1(U_1)+g_2(U_2)]$.

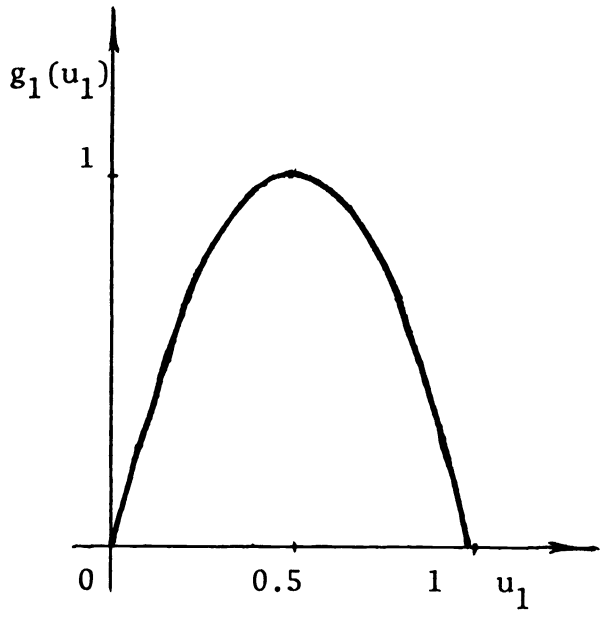


FIG. 6

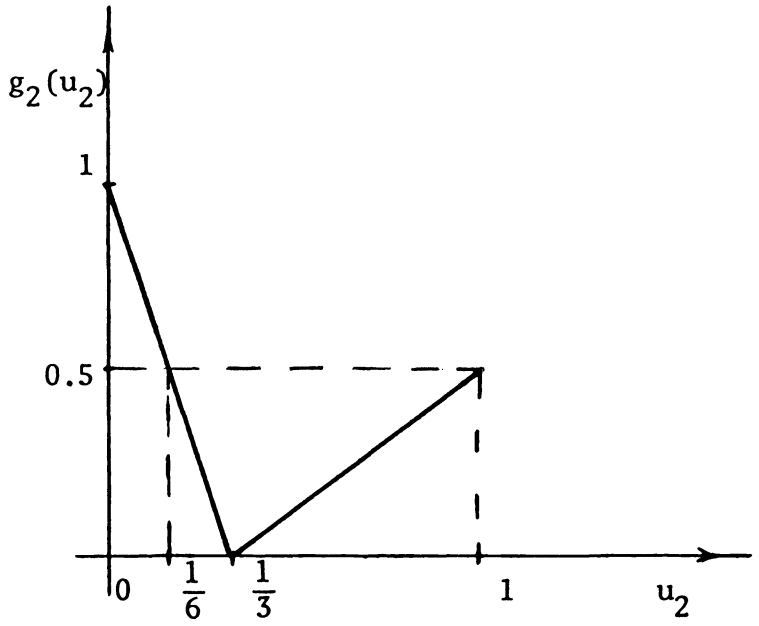


FIG. 7

We have (see Fig. 8 and Fig. 9, respectively)

$$g_1^*(u_1) = \sin\frac{\pi u_1}{2},$$

$$g_2^*(u_2) = \begin{cases} 3u_2/5, & u_2 \leqq \frac{5}{6}, \\ 3u_2 - 2, & \frac{5}{6} < u_2 \leqq 1. \end{cases}$$

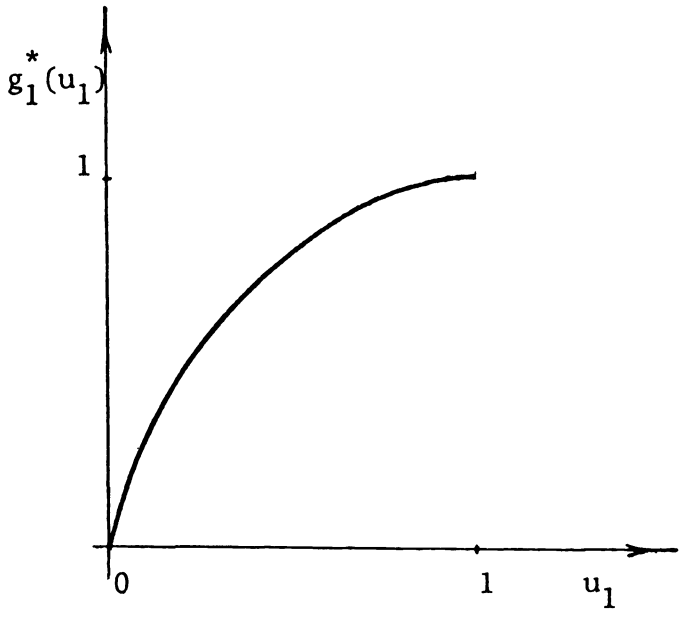


FIG. 8

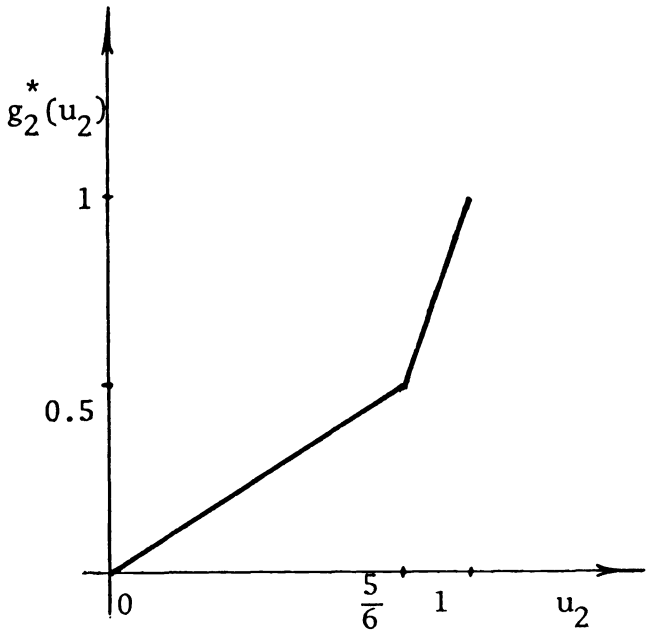


FIG. 9

The optimal transformations $h_1(u)$ and $h_2(u)$ can be readily constructed according to Procedure 2 and by using Lemma 2. They are, respectively,

$$h_1(u)=\begin{cases} 5u, & 0<u\leqq 0.1,\\ \frac{5}{4}u-\frac{1}{8}, & 0.1<u\leqq 0.5,\\ 3.5-5u, & 0.5<u\leqq 0.6,\\ \frac{7}{4}-\frac{5}{4}u, & 0.6<u\leqq 1, \end{cases}$$

and

$$h_2(u)=\begin{cases}\frac{1}{3}-2u, & 0<u\leqq\frac{1}{12},\\ 1-10u, & \frac{1}{12}<u\leqq\frac{1}{10},\\ 2u+\frac{2}{15}, & \frac{1}{10}<u\leqq\frac{13}{30},\\ \frac{5}{4}-5u/2, & \frac{13}{30}<u\leqq\frac{1}{2},\\ \frac{4}{3}-2u, & \frac{1}{2}<u\leqq\frac{7}{12},\\ 6-10u, & \frac{7}{12}<u\leqq 0.6,\\ 2u-\frac{13}{15}, & 0.6<u\leqq\frac{14}{15},\\ \frac{5}{2}-\frac{5}{2}u, & \frac{14}{15}<u\leqq 1.\end{cases}$$

By using the crude Monte Carlo method and by our optimal procedure we obtain Var $\theta_1=(1/2-4/\pi^2)+7/72=0.1913$ and Var $\theta_2=0.0991$, respectively. The gain in variance reduction is 48.36%.

Acknowledgments. We would like to express our indebtedness to the referees for several valuable suggestions on the earlier version of our paper, and especially to Andreas Buja from University of Washington, Seattle, for finding a flaw in the earlier version of the proof of Proposition 1, and his constructive idea for an alternative proof.

REFERENCES

P. BILLINGSLEY (1979), *Probability and Measurement*, John Wiley, New York.

G. S. FISHMAN (1983), *Variance reduction in the simulation of stochastic activity networks*, Technical Report UNC/ORSA/TR-82/7.

G. S. FISHMAN AND B. D. HUANG (1981), *Antithetic variates revised*, Technical Report 80-4, Curriculum in Operations Research and System Analysis, University of North Carolina, Chapel Hill.

S. GAL, Y. R. RUBINSTEIN AND A. ZIV (1984), *On the optimality and efficiency of common random numbers*, Math. Comp. Simulation, pp. 502-512.

B. GRANOVSKY (1982), *Optimal variance reduction theorem in simulation by the Monte Carlo method*, Technion Report series No. MT-571.

F. H. GRANT AND J. J. SOLBERG (1982), *Variance reduction techniques in stochastic shortest route analysis*, Pritsker & Associates, Inc., West Lafayette, Indiana.

J. M. HAMMERSLEY AND D. C. HANDSCOMB (1964), *Monte Carlo Methods*, Methuen, London.

J. M. HAMMERSLEY AND J. G. MAULDON (1956), *General principles of antithetic variates*, Math. Proc. Cambridge Philos. Soc., 52, pp. 476-481.

J. M. HAMMERSLEY AND K. W. MORTON (1956), *A new Monte Carlo technique: antithetic variates*, Math. Proc. Cambridge Philos. Soc., 52, pp. 449-475.

D. C. HANDSCOMB (1958), *Proof of the antithetic variates theorem for $n>2$*, Math. Proc. Cambridge Philos. Soc., pp. 300-301.

J. P. C. KLEIJNEN (1974), *Statistical Techniques in Simulation, Part* II, Marcel Dekker, New York.

D. G. LUENBERGER (1969), *Optimization by Vector Space Methods*, John Wiley, New York, 1969.

B. MITCHEL (1973), *Variance reduction by antithetic variates in* GI/G/1 *queueing simulation*, Oper. Res., 21, pp. 988-997.

I. P. NATANSON (1974), *Theory of Functions of Real Variables*, Nauka Moscow. (In Russian.)

E. S. PAGE (1967), *On Monte Carlo methods in congestion problems*, II: *simulation of queueing systems*, Oper. Res., 13, pp. 300-305.

Y. R. RUBINSTEIN (1981), *Simulation and the Monte Carlo Methods*, John Wiley, New York.

Y. R. RUBINSTEIN AND G. SAMORODNITSKY (1986), *Antithetic and common random numbers in simulation stochastic systems*, J. Statist. Comput. Simulation, 2, pp. 161-184..

Y. R. RUBINSTEIN, G. SAMORODNITSKY AND M. SHAKED (1985), *Antithetic variates, multivariate dependence and simulation of complex stochastic systems*, Management Sci., 31, pp. 66-77.

L. W. SCHRUBEN (1979), *Designing correlation induction strategies for simulation experiments*, in Current Issues in Computer Simulation, Academic Press, New York, pp. 235-356.

L. W. SCHRUBEN AND B. MARGOLIN (1978), *Pseudo-random number assignment in statistically designed simulation and distribution sampling experiments*, J. Amer. Statist. Assoc., 73, pp. 504-525.

R. S. SULLIVAN, J. C. HAYYA AND R. SCHAUL (1982), *The antithetic variate method for simulating stochastic networks*, Management Sci., 28, pp. 563–572.

W. WHITT (1976), *Bivariate distributions with given marginals*, Ann. Statist., 4, pp. 1280–1289.

J. R. WILSON (1979), *Proof of the antithetic variates theorem for unbounded functions*, Math. Proc. Cambridge Philos. Soc., 86, pp. 477–479.

——— (1984), *Variance reduction techniques for digital simulation*, Amer. J. Math. Management Sci., 4, pp. 277–313.

R. D. WRIGHT AND T. E. RAMSAY, JR. (1979), *On the effectiveness of common random numbers*, Management Sci., 25, pp. 649–656.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 2, March 1987
© 1987 Society for Industrial and Applied Mathematics

A FULLY PARALLEL ALGORITHM FOR THE SYMMETRIC EIGENVALUE PROBLEM*

J. J. DONGARRA† AND D. C. SORENSEN†

Abstract. In this paper we present a parallel algorithm for the symmetric algebraic eigenvalue problem. The algorithm is based upon a divide and conquer scheme suggested by Cuppen for computing the eigensystem of a symmetric tridiagonal matrix. We extend this idea to obtain a parallel algorithm that retains a number of active parallel processes that is greater than or equal to the initial number throughout the course of the computation. We give a new deflation technique which together with a robust root finding technique will assure computation of an eigensystem to full accuracy in the residuals and in the orthogonality of eigenvectors. A brief analysis of the numerical properties and sensitivity to round off error is presented to indicate where numerical difficulties may occur. The algorithm is able to exploit parallelism at all levels of the computation and is well suited to a variety of architectures.

Computational results are presented for several machines. These results are very encouraging with respect to both accuracy and speedup. A surprising result is that the parallel algorithm, even when run in serial mode, can be significantly faster than the previously best sequential algorithm on large problems, and is effective on moderate size problems when run in serial mode.

Key words. parallel algorithm, symmetric eigenvalue problem, divide and conquer, tridiagonal

AMS (MOS) subject classification. 65F15

1. Introduction.

The symmetric eigenvalue problem is one of the most fundamental problems of computational mathematics. It arises in many applications and therefore represents an important area for algorithmic research. The problem has received considerable attention in the literature and was probably the first algebraic eigenvalue problem for which reliable methods were obtained. It would be surprising, therefore, if a new method were to be found that would offer a significant improvement in execution time over the fundamental algorithms available in standard software packages such as EISPACK [12]. However, it is reasonable to expect that eigenvalue calculations might be accelerated through the use of parallel algorithms for parallel computers that are becoming available. We shall present such an algorithm in this paper. The algorithm is able to exploit parallelism at all levels of the computation and is well suited to a variety of architectures. However, the surprising result is that the parallel algorithm, even when run in serial mode, is significantly faster than the previously best sequential algorithm on large problems, and is effective on moderate size (order $\geqq 30$) problems when run in serial mode.

The problem we consider is the following: Given a real $n \times n$ symmetric matrix A, find all of the eigenvalues and corresponding eigenvectors of A. It is well known [14] that under these assumptions

$$A = QDQ^T \quad \text{with } Q^TQ = I, \tag{1.1}$$

so that the columns of the matrix Q are the orthonormal eigenvectors of A and $D = \operatorname{diag}(\delta_1, \delta_2, \cdots, \delta_n)$ is the diagonal matrix of eigenvalues. The standard algorithm

* Received by the editors December 23, 1985; accepted for publication (in revised form) May 27, 1986. This article was presented at the Second SIAM Conference on Parallel Processing for Scientific Computing, Norfolk, Virginia, November 20, 1985. This work was supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy under contracts W-31-109-Eng-38, DE-AC05-840R21400 and DE-FG02-85ER25001.

† Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439.

for computing this decomposition is first to use a finite algorithm to reduce A to tridiagonal form using a sequence of Householder transformations, and then to apply a version of the QR-algorithm to obtain all the eigenvalues and eigenvectors of the tridiagonal matrix [14]. The primary purpose of this paper is to describe a method for parallelizing the computation of the eigensystem of the tridiagonal matrix. However, the method we present is intended to be used in conjunction with the initial reduction to tridiagonal form to compute the complete eigensystem of the original matrix A. We, therefore, briefly touch upon the issues involved in parallelizing this initial reduction and suggest a way to combine the parallel initial reduction with the tridiagonal algorithm to obtain a fully parallel algorithm for the symmetric eigenvalue problem.

The method is based upon a divide and conquer algorithm suggested by Cuppen [3]. A fundamental tool used to implement this algorithm is a method that was developed by Bunch, Nielsen and Sorensen [2] for updating the eigensystem of a symmetric matrix after modification by a rank-one change. This rank-one updating method was inspired by some earlier work of Golub [4] on modified eigenvalue problems. The basic idea of the new method is to use rank-one modifications to tear out selected off-diagonal elements of the tridiagonal problem in order to introduce a number of independent subproblems of smaller size. The subproblems are solved at the lowest level using the subroutine TQL2 from EISPACK and then results of these problems are successively glued together using the rank-one modification routine SESUPD that we have developed based upon the ideas presented in [2].

In the following discussion we describe the partitioning of the tridiagonal problem into smaller problems by rank-one tearing. Then we describe the numerical algorithm for gluing the results back together. The organization of the parallel algorithm is laid out, and finally some computational results are presented. Throughout this paper we adhere to the convention that capital Roman letters represent matrices, lower case Roman letters represent column vectors, and lower case Greek letters represent scalars. A superscript T denotes transpose. All matrices and vectors are real, but the results are easily extended to matrices over the complex field.

2. Partitioning by rank-one tearing. The crux of the algorithm is to divide a given problem into two smaller subproblems. To do this, we consider the symmetric tridiagonal matrix

$$T = \begin{pmatrix} T_1 & \beta e_k e_1^T \\ \beta e_1 e_k^T & T_2 \end{pmatrix} = \begin{pmatrix} \hat{T}_1 & 0 \\ 0 & \hat{T}_2 \end{pmatrix} + \theta\beta \begin{pmatrix} e_k \\ \theta^{-1} e_1 \end{pmatrix} (e_k^T, \theta^{-1} e_1^T) \tag{2.1}$$

where $1 \leqq k \leqq n$ and e_j represents the jth unit vector of appropriate dimension. The kth diagonal element of T_1 has been modified to give $\hat{T}_1$ and the first diagonal element of T_2 has been modified to give $\hat{T}_2$. Potential numerical difficulties associated with cancellation may be avoided through the appropriate choice of θ. If the diagonal entries to be modified are of the same sign, then $\theta = \pm 1$ is chosen so that $-\theta\beta$ has this sign and cancellation is avoided. If the two diagonal entries are of opposite sign, then the sign of θ is chosen so that $-\theta\beta$ has the same sign as one of the elements and the magnitude of θ is chosen to avoid severe loss of significant digits when $\theta^{-1}\beta$ is subtracted from the other. This is perhaps a minor detail, but it does allow the partitioning to be selected solely on the basis of position and without regard to numerical considerations.

Now we have two smaller tridiagonal eigenvalue problems to solve. According to (1.1) we compute the two eigensystems

$$\hat{T}_1 = Q_1 D_1 Q_1^T, \qquad \hat{T}_2 = Q_2 D_2 Q_2^T.$$

This gives

$$
\begin{aligned}
T &= \begin{pmatrix} Q_1 D_1 Q_1^T & 0 \\ 0 & Q_2 D_2 Q_2^T \end{pmatrix} + \theta\beta \begin{pmatrix} e_k \\ \theta^{-1} e_1 \end{pmatrix} (e_k^T, \theta^{-1} e_1^T) \\
&= \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \left(\begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} + \theta\beta \begin{pmatrix} q_1 \\ \theta^{-1} q_2 \end{pmatrix} (q_1^T, \theta^{-1} q_2^T) \right) \begin{pmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{pmatrix}
\end{aligned} \tag{2.2}
$$

where $q_1 = Q_1^T e_k$ and $q_2 = Q_2^T e_1$. The problem at hand now is to compute the eigensystem of the interior matrix in (2.2). A numerical method for solving this problem has been provided in [2] and we shall discuss this method in the next section.

It should be fairly obvious how to proceed from here to exploit parallelism. One simply repeats the tearing on each of the two halves recursively until the original problem has been divided into the desired number of subproblems and then the rank-one modification routine may be applied from bottom up to glue the results together again.

3. The updating problem. The general problem we are required to solve is that of computing the eigensystem of a matrix of the form

$$\hat{Q}\hat{D}\hat{Q}^T = D + \rho z z^T \tag{3.1}$$

where D is a real $n \times n$ diagonal matrix, ρ is a nonzero scalar, and z is a real vector of order n. It is assumed without loss of generality that z has Euclidean norm 1.

We seek a formula or an eigenpair for the matrix on the right-hand side of (3.1). Let us assume for the moment that $D = \operatorname{diag}(\delta_1, \delta_2, \cdots, \delta_n)$ with $\delta_1 < \delta_2 < \cdots < \delta_n$ and that no component ζ_i of the vector z is zero. In § 4 we discuss how this may always be arranged. If q, λ is such an eigenpair, then

$$(D + \rho z z^T) q = \lambda q,$$

and a simple rearrangement of terms gives

$$(D - \lambda I) q = -\rho (z^T q) z.$$

If $\lambda = \delta_i$ for some i then our assumption that the ith component of z is nonzero implies $z^T q = 0$. This together with the assumption of distinct eigenvalues implies that q is the eigenvector e_i of D, but then $0 = z^T e_i$ would contradict the original assumption. Thus, multiplying on the left by $z^T (D - \lambda I)^{-1}$ is valid and gives

$$z^T q = -\rho (z^T q) z^T (D - \lambda I)^{-1} z.$$

Our assumptions again imply that $z^T q \neq 0$; hence

$$1 + \rho z^T (D - \lambda I)^{-1} z = 0$$

must be satisfied by λ. Starting with a λ that is a root to this last equation and putting

$$q = \theta (D - \lambda I)^{-1} z$$

for some scalar θ, one may easily verify that q, λ is an eigenpair for $D + \rho z z^T$.

If we write this equation in terms of the components ζ_i of z, then λ must be a root of the equation

$$f(\lambda) \equiv 1 + \rho \sum_{j=1}^{n} \frac{\zeta_j^2}{\delta_j - \lambda} = 0. \tag{3.2}$$

Golub [4] refers to this as the secular equation and the behavior of its roots is pictorially described by the graph in Fig. 1.

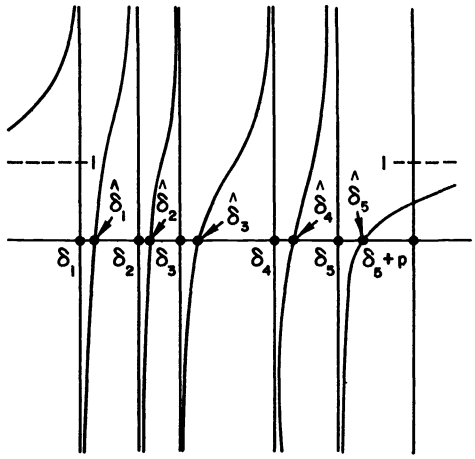


FIG. 1. *The secular equation.*

Moreover, as shown in [2] the eigenvectors (i.e., the columns of $\hat{Q}$ in (3.1)) are given by the formula

$$\hat{q}_i = \gamma_i \Delta_i^{-1} z \tag{3.3}$$

with γ_i chosen to make $\|\hat{q}_i\| = 1$, and with $\Delta_i = \text{diag}\,(\delta_1 - \hat{\delta}_i, \delta_2 - \hat{\delta}_i, \cdots, \delta_n - \hat{\delta}_i)$. Due to this structure, an excellent numerical method may be devised to find the roots of the secular equation and as a by-product to compute the eigenvectors to full accuracy. It must be stressed, however, that great care must be exercised in the numerical method used to solve the secular equation and to construct the eigenvectors from formula (3.3).

In the following discussion we assume that $\rho > 0$ in (3.2). A simple change of variables may always be used to achieve this, so there is no loss of generality. The method we shall describe was inspired by the work of Moré [9] and Reinsch [10], [11], and relies on the use of simple rational approximations to construct an iterative method for the solution of equation (3.2). Given that we wish to find the ith root $\hat{\delta}_i$ of the function f in (3.2), we may write this function as

$$f(\lambda) = 1 + \phi(\lambda) + \psi(\lambda)$$

where

$$\psi(\lambda) = \rho \sum_{j=1}^{i} \frac{\zeta_j^2}{\delta_j - \lambda}$$

and

$$\phi(\lambda) \equiv \rho \sum_{j=i+1}^{n} \frac{\zeta_j^2}{\delta_j - \lambda}.$$

From the graph in Fig. 1 it is seen that the root $\hat{\delta}_i$ lies in the open interval (δ_i, δ_{i+1}) and for λ in this interval all of the terms of ψ are negative and all of the terms of ϕ are positive. We may derive an iterative method for solving the equation

$$-\psi(\lambda) = 1 + \phi(\lambda)$$

by starting with an initial guess λ_0 in the appropriate interval and then constructing simple rational interpolants of the form

$$\frac{p}{q - \lambda}, \quad r + \frac{s}{\delta - \lambda}$$

where δ is fixed at the current value of $\hat{\delta}_{i+1}$, and the parameters p, q, r, s are defined by the interpolation conditions

$$\frac{p}{q-\lambda_0}=\psi(\lambda_0), \qquad r+\frac{s}{\delta-\lambda_0}=\phi(\lambda_0),$$
$$\frac{p}{(q-\lambda_0)^2}=\psi'(\lambda_0), \qquad \frac{s}{(\delta-\lambda_0)^2}=\phi'(\lambda_0). \tag{3.4}$$

The new approximate λ_1 to the root $\hat{\delta}_i$ is then found by solving

$$\frac{-p}{q-\lambda}=1+r+\frac{s}{\delta-\lambda}. \tag{3.5}$$

It is possible to construct an initial guess which lies in the open interval $(\delta_i, \hat{\delta}_i)$. A sequence of iterates $\{\lambda_k\}$ may then be constructed as we have just described with λ_{k+1} being derived from λ_k as λ_1 was derived from λ_0 above. The following theorem, proved in [2] then shows that this iteration converges quadratically from one side of the root and does not need any safeguarding.

THEOREM 3.6. *Let $\rho>0$ in* (3.2). *If the initial iterate λ_0 lies in the open interval $(\delta_i, \hat{\delta}_i)$ then the sequence of iterates $\{\lambda_k\}$ as constructed in* (3.4)-(3.5) *are well defined and satisfy $\lambda_k<\lambda_{k+1}<\hat{\delta}_i$ for all $k\geqq 0$. Moreover, the sequence converges quadratically to the root $\hat{\delta}_i$.*

In our implementation of this scheme (3.5) is cast in such a way that we solve for the iterative correction $\tau=\lambda_1-\lambda_0$. The quantities $\delta_j-\lambda_k$ which are used later in the eigenvector calculations are maintained and these iterative corrections are applied directly to them as well as to the eigenvalue approximation. Cancellation in the computation of these differences is thus avoided because the corrections become smaller and smaller and are eventually applied to the lowest order bits. These values are then used directly in the calculation of the updated eigenvectors to obtain the highest possible accuracy. The rapid convergence of the iterative method allows the specification of very stringent convergence criteria that will ensure a relative residual and orthogonality of eigenvectors to full machine accuracy. The stopping criteria used are

(i) $|f(\lambda)|\leqq\eta\max(|\delta_1|, |\delta_n|)$, and
(ii) $|\tau|\leqq\eta\min(|\delta_i-\lambda|, |\delta_{i+1}-\lambda|)$,

where λ is the current iterate and τ is the last iterative correction that was computed. The condition on τ is very stringent. The purpose of such stringent stopping criteria will be clarified in the next section when we discuss orthogonality of computed eigenvectors. Let it suffice at this point to say that condition (i) assures a small residual and that (ii) assures orthogonal eigenvectors.

In most problems these criteria are easily satisfied. However, there are pathological situations involving nearly equal eigenvalues that make (ii) very difficult to satisfy. It is here that the basic root finding method described above must be modified. The problem stems from the fact that the iterative corrections cease to modify the value of f due to round off error. When working in single precision, this situation can be rectified through the use of extended precision accumulation of inner products in the evaluation of f and its derivative. However, this becomes less attractive when working in 64-bit arithmetic as is done in most scientific calculations. Whether or not we shall be able to provide a root finder that will satisfy these stringent requirements in 64-bit arithmetic for very pathological cases remains to be seen.

This has been a brief description of the rank-one updating scheme. Full theoretical details are available in [2]. This calculation represents the workhorse of the parallel algorithm. The method seems to be very robust in practice and exhibits high accuracy. It does not seem to suffer the effects of nearly equal roots as Cuppen suggests [3] but instead was able to solve such ill-conditioned problems as the Wilkinson matrices W_{2k+1}^{+} [15, p. 308] to full machine precision and with slightly better residual and orthogonality properties than the standard algorithm TQL2 from EISPACK. In the next section we offer some reasons for this behavior.

4. Deflation and orthogonality of eigenvectors. At the outset of this discussion we made the assumption that the diagonal elements of D were distinct and that no component of the vector z was zero. These conditions are not satisfied in general, so deflation techniques must be employed to ensure their satisfaction. A deflation technique was suggested in [2] to provide distinct eigenvalues which amounts to rotating the basis for the eigenspace corresponding to a multiple eigenvalue so that only one component of the vector z corresponding to this space is nonzero when represented in the new basis. Those terms in (3.2) corresponding to zero components of z may simply be dropped. The eigenvalues and eigenvectors corresponding to these zero components remain static. In finite precision arithmetic the situation becomes more interesting. Terms corresponding to small components of z may be dropped. This can have a very dramatic effect upon the amount of work required in our parallel method. As first observed by Cuppen [3], there can be significant deflation in the updating process as the original matrix is rebuilt from the subproblems.

This deflation can occur in two ways in exact arithmetic, either through zero components of z or through multiple eigenvalues. In order to obtain an algorithm suitable for finite precision arithmetic, we must refine these notions to include "nearly zero" components of z and "nearly equal" eigenvalues. Analysis of the first situation is straightforward. The second situation can be quite delicate in certain pathological cases, however, so we shall discuss it here in some detail. It turns out that the case of nearly equal eigenvalues may be reduced to the case of small components of z.

It is straightforward to see that when $z^T e_i = 0$ the ith eigenvalue and corresponding eigenvector of D will be an eigenpair for $D+\rho zz^T$. Let us ask the question "when is an eigenpair of D a good approximation to an eigenpair for the modified matrix?" This question is easily answered. Recall that $\|z\| = 1$, so

$$\|(D+\rho zz^T)e_i - \delta_i e_i\| = |\rho\zeta_i|\,\|z\| = |\rho\zeta_i|.$$

Thus we may accept this eigenpair as an eigenpair for the modified matrix whenever

$$|\rho\zeta_i| \leqq \text{tol}$$

where $\text{tol} > 0$ is our error tolerance. Typically $\text{tol} = \text{macheps}\,(\eta\|A\|)$, where macheps is machine precision and η is a constant of order unity. However, at each stage of the updating process we may use

$$\text{tol} = \text{macheps}\;\eta(\max(|\delta_1|, |\delta_n|) + |\rho|)$$

since at every stage this will represent a bound on the spectral radius of a principal submatrix of A. Let us now suppose there are two eigenvalues of D separated by ε so that $\varepsilon = \delta_{i+1} - \delta_i$. Consider the 2×2 submatrix

$$\begin{pmatrix} \delta_i & \\ & \delta_{i+1} \end{pmatrix} + \rho \begin{pmatrix} \zeta_i \\ \zeta_{i+1} \end{pmatrix} (\zeta_i, \zeta_{i+1})$$

of $D+\rho zz^T$ and let us construct a Givens transformation to introduce a zero in one of the two corresponding components in z. Then

$$\begin{pmatrix} \gamma & \sigma \\ -\sigma & \gamma \end{pmatrix}\left(\begin{pmatrix} \delta_i & \\ & \delta_{i+1} \end{pmatrix}+\rho\begin{pmatrix} \zeta_i \\ \zeta_{i+1} \end{pmatrix}(\zeta_i, \zeta_{i+1})\right)\begin{pmatrix} \gamma & -\sigma \\ \sigma & \gamma \end{pmatrix}$$
$$=\begin{pmatrix} \hat{\delta}_i & \\ & \hat{\delta}_{i+1} \end{pmatrix}+\rho\begin{pmatrix} \tau \\ 0 \end{pmatrix}(\tau, 0)+\varepsilon\sigma\gamma\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

where $\gamma^2+\sigma^2=1$, $\hat{\delta}_i=\delta_i\gamma^2+\delta_{i+1}\sigma^2$, $\hat{\delta}_{i+1}=\delta_i\sigma^2+\delta_{i+1}\gamma^2$ and $\tau^2=\zeta_i^2+\zeta_{i+1}^2$. Now, if we put G_i equal to the $n\times n$ Givens transformation constructed from the identity by replacing the appropriate diagonal block with the 2×2 rotation just constructed, we have

$$G_i(D+\rho zz^T)G_i^T=\hat{D}+\rho\hat{z}\hat{z}^T+E_i \tag{4.1}$$

with $e_i^T\hat{z}=\tau$ and $e_{i+1}^T\hat{z}=0$, $e_i^T\hat{D}e_i=\hat{\delta}_i$, $e_{i+1}^T\hat{D}e_{i+1}=\hat{\delta}_{i+1}$, and

$$\|E_i\|=|\varepsilon\sigma\gamma|.$$

If we choose, instead, to zero out the ith component of z, then we simply apply G_i on the right and its transpose on the left in equation (4.1) to obtain the desired similar result. The only exception to the previous result is that the sign of the matrix E_i is reversed. Of course this deflation is only done when

$$|\varepsilon\gamma\sigma|\leqq \text{tol}$$

is satisfied.

The result of applying all of the deflations is to replace the updating problem (3.1) with one of smaller size. When appropriate, this is accomplished by applying similarity transformations consisting of several Givens transformations. If G represents the product of these transformations the result is

$$G(D+\rho zz^T)G^T=\begin{pmatrix} D_1-\rho z_1z_1^T & 0 \\ 0 & D_2 \end{pmatrix}+E$$

where

$$\|E\|\leqq \text{tol}\,\eta_2$$

with η_2 of order unity. The cumulative effect of such errors is additive, and thus the final computed eigensystem $\hat{Q}\hat{D}\hat{Q}^T$, which satisfies

$$\|A-\hat{Q}\hat{D}\hat{Q}^T\|\leqq \eta_3\text{tol}$$

where η_3 is again of order 1 in magnitude. The reduction in size of $D_1-\rho z_1z_1^T$ over the original rank-one modification can be spectacular in certain cases. The effects of such deflation can be dramatic, for the amount of computation required to perform the updating is greatly reduced.

Let us now consider the possible limitations on orthogonality of eigenvectors due to nearly equal roots. Our first result is a perturbation lemma that will indicate the inherent difficulty associated with nearly equal roots.

LEMMA 4.2. *Let*

$$q_\lambda^T\equiv\left(\frac{\zeta_1}{\delta_1-\lambda}, \frac{\zeta_2}{\delta_2-\lambda}, \cdots \frac{\zeta_n}{\delta_n-\lambda}\right)\left[\frac{\rho}{f'(\lambda)}\right]^{1/2} \tag{4.3}$$

where f is defined by formula (3.2). *Then for any* λ, $\mu\notin\{\delta_i\colon i=1, \cdots, n\}$

$$|q_\lambda^Tq_\mu|=\frac{1}{|\lambda-\mu|}\frac{|f(\lambda)-f(\mu)|}{[f'(\lambda)f'(\mu)]^{1/2}}. \tag{4.4}$$

Proof. Note that

$$q_\lambda^T q_\mu = \left(\sum_{j=1}^{n} \frac{\zeta_j^2}{(\delta_j - \lambda)(\delta_j - \mu)} \right) \frac{|\rho|}{[f'(\lambda) f'(\mu)]^{1/2}}. \tag{4.5}$$

But

$$\frac{\lambda - \mu}{(\delta_j - \lambda)(\delta_j - \mu)} = \frac{1}{(\delta_j - \lambda)} - \frac{1}{(\delta_j - \mu)}.$$

Thus

$$(\lambda - \mu)\rho \sum_{j=1}^{n} \frac{\zeta_j^2}{(\delta_j - \lambda)(\delta_j - \mu)} = \rho \sum_{j=1}^{n} \frac{\zeta_j^2}{\delta_j - \lambda} - \rho \sum_{j=1}^{n} \frac{\zeta_j^2}{\delta_j - \mu} = f(\lambda) - f(\mu)$$

and the result follows. □

Note that in (4.3) q_λ is always a vector of unit length, and the set of n vectors selected by setting λ equal to the roots of the secular equation is the set of eigenvectors for $D + \rho z z^T$. Moreover, (4.4) shows that those eigenvectors are mutually orthogonal whenever λ and μ are set to distinct roots of f. Finally, the term $|\lambda - \mu|$ appearing in the denominator of (4.4) sends up a warning that it may be difficult to attain orthogonal eigenvectors when the roots λ and μ are close. We wish to examine this situation now. We will show that, as a result of the deflation process, the $\{\delta_i\}$ are sufficiently separated, and the weights ζ_i are uniformly large enough that the roots of f are bounded away from each other. This statement is made explicit in the following.

LEMMA 4.6. *Let λ be the root of f in the ith subinterval (δ_i, δ_{i+1}). If the deflation test is satisfied then either*

(i) $|\delta_{i+1} - \lambda| \geqq \frac{1}{2}|\delta_{i+1} - \delta_i|$ *and* $|\delta_i - \lambda| \geqq \dfrac{\text{tol}^2}{|\rho(\delta_{i+1} - \delta_i)| + 2\rho^2}$, *or*

(ii) $|\delta_i - \lambda| \geqq \frac{1}{2}|\delta_{i+1} - \delta_i|$ *and* $|\delta_{i+1} - \lambda| \geqq \dfrac{\text{tol}^2}{2\rho^2}(\delta_{i+1} - \delta_i)$.

Proof. In case (i) the fact that $f(\lambda) = 0$ provides

$$\rho \sum_{j=1}^{i} \frac{\zeta_j^2}{\lambda - \delta_j} = 1 + \rho \sum_{j=i+1}^{n} \frac{\zeta_j^2}{\delta_j - \lambda}.$$

From this equation we find that

$$|\rho| \frac{\zeta_i^2}{\lambda - \delta_i} \leqq 1 + \frac{|\rho|}{\delta_{i+1} - \lambda} \leqq 1 + \frac{2|\rho|}{\delta_{i+1} - \delta_i},$$

and it follows readily that

$$|\delta_{i+1} - \delta_i| \left(\frac{|\rho| \zeta_i^2}{|\delta_{i+1} - \delta_i| + 2|\rho|} \right) \leqq |\delta_i - \lambda|.$$

The deflation rules assure us that $|\rho| \zeta_i^2 \geqq \text{tol}^2 / |\rho|$ and the result follows. Case (ii) is similar. □

The form of the result given in Lemma 4.6 will have importance in the following lemma which gives better insight to the quality of numerical orthogonality attainable with this scheme. The bounds obtained are certainly less than one would hope for. Moreover, it is unfortunate that only the magnitude of the weights enter into the estimate. This obviously does not take into account the deflation due to nearly equal eigenvalues. Unfortunately, we have not been able to improve these estimates by other means and are left with this crude bound.

LEMMA 4.7. *Suppose that* $\hat{\lambda}$ *and* $\hat{\mu}$ *are numerical approximations to exact roots* λ *and* μ *of* f. *Assume that these roots are distinct and let the relative errors for the quantities* $\delta_i - \lambda$ *and* $\delta_i - \mu$ *be denoted by* θ_i *and* η_i, *respectively. That is, the computed quantities*

$$\delta_i - \hat{\lambda} = (\delta_i - \lambda)(1+\theta_i) \quad \text{and} \quad \delta_i - \hat{\mu} = (\delta_i - \mu)(1+\eta_i) \tag{4.8}$$

for $i = 1, 2, \cdots, n$. *Let* $q_{\hat{\lambda}}$ *and* $q_{\hat{\mu}}$ *be defined according to formula* (4.3) *using the computed quantities given in* (4.8). *If* $|\theta_i|, |\eta_i| \leqq \varepsilon \ll 1$, *then*

$$|q_{\hat{\lambda}}^T q_{\hat{\mu}}| = |q_{\lambda}^T E q_{\mu}| \leqq \varepsilon(2+\varepsilon)\left(\frac{1+\varepsilon}{1-\varepsilon}\right)^2$$

with E *a diagonal matrix whose* i*th diagonal element is*

$$E_{ii} = \frac{\theta_i + \eta_i + \theta_i\eta_i}{(1+\theta_i)(1+\eta_i)}\left[\frac{f'(\lambda)f'(\mu)}{f'(\hat{\lambda})f'(\hat{\mu})}\right]^{1/2}.$$

Proof. From formula (4.3) we have

$$\begin{aligned} -q_{\hat{\lambda}}^T q_{\hat{\mu}} &= -\left(\sum_{j=1}^{n} \frac{\zeta_j^2}{(\delta_j - \lambda)(\delta_j - \mu)(1+\theta_j)(1+\eta_j)}\right)\frac{|\rho|}{[f'(\hat{\lambda})f'(\hat{\mu})]^{1/2}} \\ &= \left(\sum_{j=1}^{n} \frac{\zeta_j^2}{(\delta_j - \lambda)(\delta_j - \mu)} - \sum_{j=1}^{n} \frac{\zeta_j^2}{(\delta_j - \lambda)(\delta_j - \mu)(1+\theta_j)(1+\eta_j)}\right)\frac{|\rho|}{[f'(\hat{\lambda})f'(\hat{\mu})]^{1/2}} \end{aligned}$$

due to the orthogonality of the exact vectors. Thus,

$$|q_{\hat{\lambda}}^T q_{\hat{\mu}m}| = \left|\sum_{j=1}^{n}\left(\frac{\zeta_j^2}{(\delta_j - \lambda)(\delta_j - \mu)}\right)\left(1 - \frac{1}{(1+\theta_i)(1+\eta_j)}\right)\frac{\rho}{[f'(\hat{\lambda})f'(\hat{\mu})]^{1/2}}\right|.$$

Since the quantity

$$\frac{f'(\lambda)}{f'(\hat{\lambda})} = \frac{\sum_{j=1}^{n} \frac{\zeta_j^2}{(\delta_j - \lambda)^2}}{\sum_{j=1}^{n} \frac{\zeta_j^2}{(\delta_j - \lambda)^2(1+\theta_j)^2}} \leqq (1+\varepsilon)^2$$

and since

$$\max E_{ii} \leqq \frac{\varepsilon + \varepsilon + \varepsilon^2}{(1-\varepsilon)(1-\varepsilon)}$$

the result follows. □

This shows that orthogonality can be assured whenever it is possible to provide small relative errors when computing the differences $\delta_i - \lambda$. Since, as we mentioned in § 3, these quantities are updated with the iterative corrections to λ and since deflation has guaranteed that the quantities in Lemma 4.6 are bounded away from zero, it will be possible in theory to provide small relative errors. Obviously, the analysis given here, simple as it may be, could form the core of a rigorous error analysis of the method. As yet we lack a root finder that could be proven to satisfy numerically the relative error requirements. The one described above will do so in finite precision but may require extended precision accumulation of inner products in pathological cases. However, in practice we have not experienced difficulty with the exception of contrived examples. Cuppen [3] has shown that the computed eigenvectors may be safely reorthogonalized when needed. We hope to avoid this alternative, though.

5. Reduction to tridiagonal form and related issues. This algorithm is designed to work in conjunction with Householder's reduction of a symmetric matrix to tridiagonal form. In this standard technique a sequence of $n-1$ Householder transformations is applied to form

$$A_{k+1} = (I - \alpha_k w_k w_k^T) A_k (I - \alpha_k w_k w_k^T),$$

with

$$A_k = \begin{pmatrix} T_k & 0 \\ 0 & \hat{A}_k \end{pmatrix}$$

where T_k is a tridiagonal matrix of order $k-1$. See [12] for details. We note here that we can form $(I - \alpha w w^T) A (I - \alpha w w^T)$ using the following algorithm.

ALGORITHM 5.1.

1. $v = Aw$
2. $y^T = v^T - \alpha (w^T v) w^T$
3. Replace A by $A - \alpha v w^T - \alpha w y^T$.

Steps 1 and 3 may all be parallelized because A may be partitioned into blocks of columns $A = (A_1, A_2, \cdots, A_j)$ and each of the contributions corresponding to these columns may be carried out independently in Steps 1 and 3. For example, in Step 3,

$$A_i \leftarrow A_i - \alpha v w_i^T - \alpha w y_i^T$$

where the vectors w and y have been partitioned in a corresponding manner. In Step 2 the partial results will have to be stored in temporary locations until all are completed and then they may be added together to obtain the final result. Note also that when the calculations are arranged this way advantage may be taken of vector operations when they are available.

Algorithm 5.1 has some disadvantages when incorporated into the reduction of a symmetric matrix to tridiagonal form. At the kth stage of the reduction we have

$$\begin{pmatrix} T^{(k)} & \beta e_k e_1^T \\ \beta e_1 e_k^T & A^{(k)} \end{pmatrix}.$$

The reduction is advanced one step through the application of Algorithm 5.1 to the submatrix $A^{(k)}$. First, a fork-join synchronization construct is imposed since the matrix-vector product requires the entire matrix $A^{(k)}$ to be in place before this product can be completed, so that no portion of Step 2 may begin until Step 1 is finished. Second, due to symmetry, only the lower triangle of $A^{(k)}$ need be computed and this implies that vector lengths shorten during the computation.

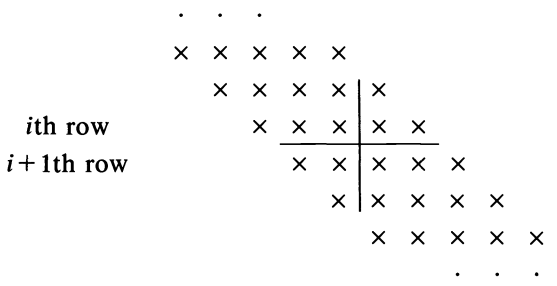
When used in conjunction with the rank-one tearing scheme, these drawbacks may be overcome. Suppose the final result of this decomposition is T and that this matrix would be partitioned into $(T_1, T_2, \cdots, T_m)$ by the rank-one tearing if it were known. It is not necessary to wait until the entire reduction is completed, for $T^{(k)}$ represents a leading principal submatrix of T. Thus, as soon as the first subtridiagonal matrix T_1 is exposed, the process of computing its eigensystem may be initiated. Similarly, as soon as T_2 is exposed its eigensystem may be computed. Then a rank-one update may occur, and so on. In this way a number of independent processes may be spawned early on, and the number of such processes ready to execute will remain above a reasonable level throughout the course of the computation. An efficient implementation of this scheme is difficult and will be the subject of a subsequent paper. Issues such as proper level of partitioning will have added importance due to the desire to have parallel processes set in motion as soon as possible.

When we do not wish to find eigenvectors there is no reason to store the product Q of these Householder transformations. Nor is it necessary to accumulate the product of the successive eigenvector transformations resulting from the updating problem. That is, we do not need to overwrite Q with

$$Q \leftarrow Q \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \hat{Q}$$

where Q_1, Q_2 and $\hat{Q}$ are the matrices appearing in (2.2) and (3.1) above. Instead, we may simply discard Q. Then the vector q_1 may be formed as $\hat{T}_1$ is transformed to D_1 in (2.2) by accumulating the products of the transformations constructed in TQL2 that make up Q_1 against the vector e_k. If there is more than one division, then Q_1 will have been calculated with the updating scheme. In this case we do not calculate all of Q_1, but instead use the component-wise formula for the eigenvectors to pick off the appropriate components needed to form q_1.

This algorithm can be generalized to handle band matrices as well. Instead of performing a rank-one tearing to split the matrix into two independent subproblems, we proceed by making $(m+1)m/2$ rank-one changes designed to split the matrix, (m is the half bandwidth, $m=1$ for tridiagonal matrices).



The three rank-one changes for this band matrix, ($m=2$), involve the following elements:

$$a_{i-1,i-1}, \quad a_{i-1,i+1},$$
$$a_{i+1,i-1}, \quad a_{i+1,i+1},$$
$$a_{i,i}, \quad a_{i,i+1},$$
$$a_{i+1,i}, \quad a_{i+1,i+1},$$

and

$$a_{i,i}, \quad a_{i,i+2},$$
$$a_{i+2,i}, \quad a_{i+2,i+2}.$$

The order in which they are applied does not matter in exact arithmetic. We have not studied the numerical properties of this scheme, however.

6. The parallel algorithm. Although it is fairly straightforward from § 2 to see how to obtain a parallel algorithm, certain details are worth discussing further. We shall begin by describing the partitioning phase. This phase amounts to constructing a binary tree with each node representing a rank-one tear and hence a partition into two subproblems. A tree of height 3 therefore represents a splitting of the original problem into 8 smaller eigenvalue problems. Thus, there are two standard symmetric tridiagonal eigenvalue problems to be solved at each leaf of the tree. Each of these problems may be spawned independently without fear of data conflicts. The tree is

then traversed in reverse order with the eigenvalue updating routine SESUPD applied at each node joining the results from the left son and right son calculations. The leaves each define independent rank-one updating problems and again there are no data conflicts between them. The only data dependency at a node is that the left and right son calculations must have been completed. When this condition is satisfied, the results of two adjacent eigenvalue subproblems are ready to be joined through the rank-one updating process and this node may spawn the updating process immediately. Information required at a node to define the problem consists of the index of the element torn out, together with the dimension of the left and right son problems. For example, if $n = 50$ with a tree of height 3 we have

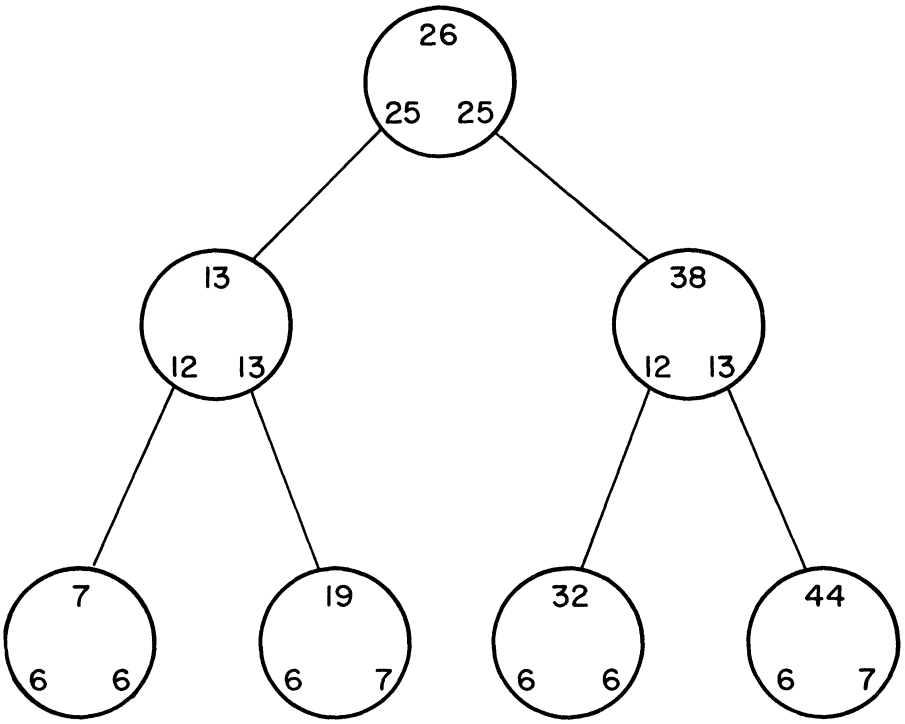


FIG. 2. *The computational tree.*

This tree defines 8 subproblems at the lowest level. There are two calls to TQL2 required at each leaf of the tree in Fig. 2 to solve tridiagonal eigenvalue problems. The beginning indices of these problems are 1, 7, 13, 19, 26, 32, 38, 44 and the dimension of each of them may be read off from left to right at the lowest level as 6, 6, 6, 7, 6, 6, 6, 7 respectively. As soon as the calculation for the problems beginning at indices 1 and 7 have been completed a rank-one update may proceed on the problem beginning at index 1 with dimension 12. The remaining updating problems at this level begin at indices 13, 26, 38. There are then two updating problems at indices 1 and 26 each of dimension 25 and a final updating problem at index 1 of dimension 50.

Evidently, we lose a degree of large grain parallelism as we move up a level on the tree. However, there is more parallelism to be found at the root finding level and the amount of this increases as we travel up the tree so there is ample opportunity for load balancing in this scheme. The parallelism at the root finding level stems from the fact that each of the root calculations is independent and requires only read access to

all but one array. That is the array that contains the diagonal entries of the matrix Δ_i described in § 3. For computational efficiency we may decide on an advantageous number of processes to create at the outset. In the example above that number was 8. Then as we travel up the tree the root-finding procedure is split into 2,4, and finally 8 parallel parts in each node at levels 3, 2, 1, respectively. As these computations are roughly equivalent in complexity on a given level it is reasonable to expect to keep all processors devoted to this computation busy throughout.

7. Implementation and library issues. The implementation described here is for computers with a shared memory architecture. The algorithm itself is not limited to this memory model; however, Jessop [5] has an implementation of a similar algorithm for a hypercube that is also based upon Cuppen's divide and conquer scheme.

Obviously, the implementation of this scheme is not so straightforward as simply parallelizing a loop using a fork-join construct. The synchronization mechanism requires more sophistication since it must be able to spawn processes at the root finding level based upon computation that has taken place. This process allocation problem is dynamic rather than static since, due to the possibility of deflation, it will not be known in advance how many roots will have to be calculated at a given level. If there are only a few roots to be found, then the desire for reasonable granularity will dictate fewer processes to be spawned.

In addition to requiring a certain level of sophistication in the synchronization scheme, we would like to adhere as much as possible to the principles of transportability. This algorithm is obviously a candidate for a library subroutine and potentially will find use on a wide variety of machines. When designing library subroutines, one wishes to conceal machine dependencies as much as possible from the user. These important considerations seem to be difficult to accommodate if we are to invoke parallelism at the level described above. It would appear that the user must be conscious of the number of parallel processes required by the library subroutines throughout his program. Should the library routines be called from multiple branches of a user's parallel program, then he could inadvertently attempt to create many more processes than allowed due to physical limitations.

We believe there is hope for implementing this algorithm while adhering to the goals set out in the previous two paragraphs. It may be accomplished by adopting a programming technique that is inspired by the work of Lusk and Overbeek [7], [8] and by Babb [1] on methodologies for implementing transportable parallel codes. We use a package called SCHEDULE [13] that we have been developing while this algorithm was being devised and tested. SCHEDULE is a package of Fortran subroutines designed to aid in programming explicitly parallel algorithms for numerical calculations. The design goal of SCHEDULE is to aid a programmer familiar with a Fortran programming environment to implement a parallel algorithm in a style of Fortran that will lend itself to transporting the resulting program across a wide variety of parallel machines. An important part of this package is the provision of a mechanism for dynamically spawning processes even when such a capability is not present within the parallel language extensions provided for a given machine. The Alliant computer is an example. A detailed discussion of this package is beyond the scope of this paper and will be presented elsewhere. A description of the package and its capabilities is available in [13].

8. Performance. In this section we present and analyze the results of this algorithm on a number of machines. The same algorithm has been run on a VAX 11/785, Denelcor HEP, Alliant FX/8, and a CRAY X-MP-4.

TABLE 8.1
A comparison on the CRAY X-MP *for* TQL2 *vs. the parallel algorithm run sequentially.*

n	Ratio of time TQL2/SESUPD	TQL2 $\|Ax-\lambda x\|$	SESUPD $\|Ax-\lambda x\|$	TQL2 $\|X^TX-I\|$	SESUPD $\|X^TX-I\|$
50	.78	1.6×10^{-12}	4.8×10^{-13}	1.9×10^{-12}	1.2×10^{-13}
100	1.26	2.6×10^{-12}	7.6×10^{-13}	3.9×10^{-12}	2.5×10^{-12}
200	1.93	6.1×10^{-12}	1.8×10^{-12}	7.0×10^{-12}	5.8×10^{-11}
300	3.62	1.1×10^{-11}	3.3×10^{-12}	1.1×10^{-11}	2.8×10^{-12}
400	4.70	1.2×10^{-11}	3.8×10^{-12}	1.4×10^{-11}	3.8×10^{-12}

TABLE 8.2
Ratio of execution time TQL2 *time/parallel time.*
order = 150

	VAX 785/FPA	Denelcor HEP	Alliant FX/8	CRAY X-MP-1	CRAY X-MP-4
random matrix	2.6	12	15.2	1.8	4.5

We have compared our implementation of the algorithm described in this paper to TQL2 from the EISPACK collection. Table 8.1 gives the ratio of execution time for TQL2 from EISPACK run sequentially and the algorithm presented here run in parallel on the same machine. In all cases, the TQL2 code used to test against and the TQL2 code used at lowest level in our algorithm were exactly the same code. Both the parallel algorithm and the serial version of TQL2 were called by a common driver test routine during the same computer run. This assures that both routines were compiled under the same compiler options and ran in the same computing environment to produce this timing data. The timings for TQL2 were obtained by executing it as a single process. By this we mean that the code for TQL2 executed on a single processor of the Alliant and CRAY machines and as the only created process executing on the HEP machine. In all cases the computations were carried out as though the tridiagonal matrix had come from Householder's reduction of a dense symmetric matrix to tridiagonal form. The identity was passed in place of the orthogonal basis that would have been provided by this reduction, but the arithmetic operations performed were the same as those that would have been required to transform that basis into the eigenvectors of the original symmetric matrix.

As can be seen in Table 8.2, the performance of the parallel algorithm as implemented to run on a sequential machine is quite impressive. The surprising result here is the observed speedup even in serial mode of execution. This is unusual in a parallel algorithm. Often more work is associated with synchronization and computational overhead required to split the problem into parallel parts.

These results are remarkable because in all cases speedups greater than the number of physical processors were obtained. The gain is due to the numerical properties of the deflation portion of the parallel algorithm. We do not fully understand why the algorithm performs as much deflation as is apparent by the comparisons. In all cases the word length was 64 bits and the same level of accuracy was achieved by both methods. The measurements of accuracy used was the maximum 2-norm of the residuals $Tq-\lambda q$ and of the columns of Q^TQ-I. The results in Table 8.1 are typical of the performance of this algorithm on random problems with speedups becoming more dramatic as the matrix order increases. In problems of order 500 speedups of 15 have

TABLE 8.3
A comparison on the Alliant FX/8 *for* TQL2 *vs. the parallel algorithm on* (1, 2, 1) *matrix.*

n	Ratio TQL2/SESUPD	TQL2 $\|Ax-\lambda x\|$	SESUPD $\|Ax-\lambda x\|$	TQL2 $\|X^TX-I\|$	SESUPD $\|X^TX-I\|$
100	9.4	9.6×10^{-15}	1.9×10^{-15}	7.8×10^{-15}	5.5×10^{-16}
200	15.4	1.7×10^{-14}	2.7×10^{-15}	1.1×10^{-14}	2.2×10^{-15}
300	17.7	1.9×10^{-14}	3.2×10^{-15}	1.7×10^{-14}	2.6×10^{-15}
400	20.0	2.9×10^{-14}	4.0×10^{-15}	2.0×10^{-14}	9.2×10^{-15}

TABLE 8.4
A comparison on the Alliant FX/8 *for* TQL2 *vs. the parallel algorithm on a random matrix.*

n	Ratio of time TQL2/SESUPD	TQL2 $\|Ax-\lambda x\|$	SESUPD $\|Ax-\lambda x\|$	TQL2 $\|X^TX-I\|$	SESUPD $\|X^TX-I\|$
100	12.1	1.6×10^{-14}	1.9×10^{-13}	1.6×10^{-14}	2.4×10^{-15}
200	19.5	2.1×10^{-14}	2.2×10^{-13}	2.8×10^{-14}	2.3×10^{-15}
300	38.8	4.2×10^{-14}	8.8×10^{-13}	3.7×10^{-14}	5.2×10^{-15}
400	60.7	3.5×10^{-14}	8.2×10^{-13}	3.7×10^{-14}	4.6×10^{-14}

been observed on the CRAY-XMP-4 and speedups over 50 have been observed on the Alliant FX/8, which are 4 and 8 processor machines respectively. The CRAY results can actually be improved because parallelism at the root finding level was not exploited in the implementation run on the CRAY but was fully exploited on the Alliant.

In Tables 8.3 and 8.4 we show a more complete set of runs on the Alliant/FX8. These test problems show two types of behavior. In the (1, 2, 1) matrix not very much deflation takes place. On the random matrix considerable deflation takes place. The observation of Cuppen that this deflation occurs in many cases was very fortunate. It brought our attention to this algorithm, but we did not really expect the remarkable performance observed here. In the case where many eigenvectors are sought along with the eigenvalues this algorithm seems to be very promising. However, as we mentioned in § 5 it is not necessary to compute the eigenvectors if they are not needed. We cannot recommend this algorithm in the case where only a few eigenvalues and eigenvectors are sought. A multisectioning algorithm such as the one developed by Lo, Philippe and Sameh [6] is to be preferred in this case.

REFERENCES

[1] R. G. BABB II, *Parallel processing with large grain data flow techniques*, IEEE Computer, 17 (1984), pp. 55-61.

[2] J. R. BUNCH, C. P. NIELSEN AND D. C. SORENSEN, *Rank-one modification of the symmetric eigenproblem*, Numer. Math., 31 (1978), pp. 31-48.

[3] J. J. M. CUPPEN, *A divide and conquer method for the symmetric tridiagonal eigenproblem*, Numer. Math., 36 (1981), pp. 177-195.

[4] G. H. GOLUB, *Some modified matrix eigenvalue problems*, SIAM Rev., 15 (1973), pp. 318-334.

[5] L. JESSOP, *Solving the symmetric tridiagonal eigenvalue problem on the hypercube*, Yale Computer Science Research Report, in preparation.

[6] S. LO, B. PHILIPPE AND A. SAMEH, *A parallel algorithm for the real symmetric tridiagonal eigenvalue problem*, Center for Supercomputing Research and Development Report, Univ. Illinois, Urbana, IL, November 1985.

[7] E. LUSK AND R. OVERBEEK, *Implementation of monitors with macros: A programming aid for the* HEP *and other parallel processors*, ANL-83-97, 1983.
[8] ———, *An approach to programming multiprocessing algorithms on the Denelcor* HEP, ANL-83-96, 1983.
[9] J. J. MORÉ, *The Levenberg-Marquardt algorithm: implementation and theory*, Proc. Dundee Conference on Numerical Analysis, G. A. Watson, ed., Springer-Verlag, New York, 1978.
[10] C. H. REINSCH, *Smoothing by spline functions*, Numer. Math., 10 (1967), pp. 177-183.
[11] ———, *Smoothing by spline functions* II, Numer. Math., 16 (1971), pp. 451-454.
[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 Science, 6, 2nd ed., Springer-Verlag, Berlin, 1976.
[13] D. C. SORENSEN, SCHEDULE *Users Guide*, ANL-MCS-TM-76, Argonne National Laboratory, Argonne, IL, May 1986.
[14] G. W. STEWART, *Introduction to Matrix Computations*, Academic Press, New York, 1973.
[15] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 2, March 1987
© 1987 Society for Industrial and Applied Mathematics

A MULTIPROCESSOR ALGORITHM FOR THE SYMMETRIC TRIDIAGONAL EIGENVALUE PROBLEM*

SY-SHIN LO†, BERNARD PHILIPPE† AND AHMED SAMEH†

Abstract. A multiprocessor algorithm for finding few or all eigenvalues and the corresponding eigenvectors of a symmetric tridiagonal matrix is presented. It is a pipelined variation of EISPACK routines—BISECT and TINVIT which consists of the three steps: isolation, extraction-inverse iteration, and partial orthogonalization. Multisections are performed for isolating the eigenvalues in a given interval, while bisection or the Zeroin method is used to extract these isolated eigenvalues. After the corresponding eigenvectors have been computed by inverse iteration, the modified Gram-Schmidt method is used to orthogonalize certain groups of these vectors. Experiments on the Alliant FX/8 and CRAY X-MP/48 multiprocessors show that this algorithm achieves high speed-up over BISECT and TINVIT; in fact it is much faster than TQL2 when all the eigenvalues and eigenvectors are required.

Key words. eigenvalues, multiprocessors, tridiagonal matrices

AMS(MOS) subject classification. 65F15

1. Introduction. This paper deals with solving the real symmetric tridiagonal eigenvalue problem on a multiprocessor. The main purpose of this study is solving for a few of the eigenvalues and the corresponding eigenvectors of large tridiagonal symmetric matrices such as those resulting from the Lanczos tridiagonalization of a sparse symmetric matrix. In fact, we show that our scheme, the origins of which date back to the Illiac IV [8] and [7], is equally or more effective than other multiprocessor schemes for obtaining either all the eigenvalues, or all the eigenvalues and eigenvectors of a symmetric tridiagonal matrix.

Two kinds of methods are usually used for solving this problem on a uniprocessor. When only a part of the spectrum is desired, the combination of bisection and inverse iteration is the method of choice. For the whole eigenvalue problem, the QR (or QL) method is more effective [2]. These methods are implemented in EISPACK: BISECT and TINVIT for the partial eigenvalue problem, and TQL1 (eigenvalues only) and TQL2 (eigenvalues and eigenvectors) for the whole problem. A multiprocessor version of TQL2 has already been designed [4] using a divide and conquer technique.

In § 2 we give a brief description of the method, in § 3 we analyze the various steps of the computation with respect to efficient use of the parallelism offered by the multiprocessor. In § 4, we compare our scheme with all of the above schemes on the Alliant FX/8 and the CRAY X-MP/48 multiprocessors.

2. Description of the method. Let $\mathbf{T}$ be a symmetric tridiagonal matrix of order n with d_i and e_i as the diagonal and subdiagonal elements, respectively,

$$\mathbf{T} = [e_i, d_i, e_{i+1}].$$

Let $p_n(\lambda)$ be its characteristic polynomial:

$$p_n(\lambda) = \det(\mathbf{T} - \lambda \mathbf{I}).$$

* Received by the editors December 23, 1985; accepted for publication (in revised form) May 12, 1986. This work was supported in part by the National Science Foundation under Grants US NSF DCR84-10110 and US NSF DCR85-09970, the U.S. Department of Energy under Grant US DOE DE-FG02-85ER25001, the IBM Donation, and the French Ministry of Defense under Grant DRET No. 84-823. This paper was submitted to the Second SIAM Conference on Parallel Processing for Scientific Computing that was held in Norfolk, Virginia during November 18-21, 1985.

† Center for Supercomputing Research and Development, University of Illinois, Urbana, Illinois 61801.

The sequence of the principal minors of the matrix can be built using the following recursion:

$$
\begin{aligned}
& p_0(\lambda) = 1, \\
& p_1(\lambda) = d_1 - \lambda, \\
& p_i(\lambda) = (d_i - \lambda) p_{i-1}(\lambda) - e_i^2 p_{i-2}(\lambda), \qquad i = 2, \cdots, n.
\end{aligned} \tag{2.1}
$$

We assume that no subdiagonal element is zero, since if some e_i is equal to 0, the problem can be partitioned into two smaller problems. The sequence $\{p_i(\lambda)\}$ is called the Sturm sequence of $\mathbf{T}$ in λ. It is well known [11] that the number of eigenvalues smaller than a given λ is equal to the number of sign variations in the Sturm sequence (2.1). Hence one can find the number of eigenvalues lying in a given interval [a, b] by computing the Sturm sequences at a and b. The linear recurrence (2.1), however, suffers from the possibility of over- or underflow. This is remedied by replacing the Sturm sequence $p_i(\lambda)$ by the sequence

$$
q_i(\lambda) = \frac{p_i(\lambda)}{p_{i-1}(\lambda)}, \qquad i = 1, n.
$$

The second order linear recurrence (2.1) is then replaced by the nonlinear recurrence

$$
q_1(\lambda) = d_1 - \lambda, \quad q_i(\lambda) = d_i - \lambda - \frac{e_i^2}{q_{i-1}(\lambda)}, \quad i = 2, \cdots, n. \tag{2.2}
$$

Here, the number of eigenvalues that are smaller than λ is equal to the number of negative terms in the sequence $\{q_i(\lambda)\}$.

Therefore, given an initial interval, we can find the eigenvalues lying in it by repeated bisection or multisection of the interval. This partitioning process can be performed until we obtain each eigenvalue to a given accuracy. On the other hand, we can stop the process once we have isolated each eigenvalue. In the latter case the eigenvalues may be extracted using a faster method. Several methods are available for extracting an isolated eigenvalue:

- -Bisection (linear convergence);
- -Newton's method (quadratic convergence);
- -The Zeroin scheme, which is based on the secant and bisection methods (convergence of order $(\sqrt{5}+1)/2$).

Ostrowski [10] defines an efficiency index which links the amount of computation to be done at each step and the order of the convergence. The respective indices of the three methods are 1, 1.414 and 1.618. This index, however, is not the only aspect to be considered here. Both Zeroin and Newton methods require the use of the linear recurrence (2.1) in order to obtain the value of $\det(\mathbf{T} - \lambda \mathbf{I})$, or its derivative as well, for a given λ. Hence, if the possibility of over- or underflow is small we select the Zeroin method; otherwise we select the bisection method. After the computation of an eigenvalue, the corresponding eigenvector can be found by inverse iteration [6]. This is a very fast process where one iteration is often sufficient to achieve convergence.

It is possible that some eigenvalues are computationally coincident; hence, the isolation process actually performs "isolation of clusters," where a cluster is defined as a single eigenvalue or a number of computationally coincident eigenvalues. If such a cluster of coincident eigenvalues is isolated the extraction step is skipped, since convergence has been reached. Observing that, there can be loss of orthogonality for those eigenvectors corresponding to close eigenvalues; orthonormalization of such eigenvectors via the Modified Gram-Schmidt method is necessary.

In summary, the whole computation consists of the following five steps:

1) Isolation by partitioning;
2) Extraction of a cluster by bisection or by the Zeroin method;
3) Computation of the eigenvectors of the cluster by inverse iteration;
4) Grouping of close eigenvalues;
5) Orthogonalization of the corresponding groups of vectors by the Modified Gram-Schmidt process.

3. The parallel algorithm.

3.1. The partitioning process. Obviously, the parallelism in this process is achieved by performing simultaneously the computation of several Sturm sequences. However, there are several ways for achieving this; two options are

-Performing bisection on several intervals;

-Performing a partition of one interval into several subintervals.

A multisection of order k splits the interval $I=[a, b]$ into $k+1$ subintervals $I_i=[x_i, x_{i+1}]$, where $x_i=a+i((b-a)/(k+1))$ for $i=0, \cdots, k+1$. If there exists only one eigenvalue in the interval I and if we wish to compute it with an absolute error ε, then it is necessary to perform

$$n_k=\log_2\left(\frac{(b-a)}{(2\varepsilon)}\right)\Big/\log_2(k+1)$$

multisections of order k. Thus, the efficiency of the multisection of order k compared to bisection (multisection of order 1) is

$$E_f=n_1/(k\,n_k)=(\log_2(k+1))/k.$$

Hence, for extraction of eigenvalues, we prefer to perform parallel bisections rather than one multisection of high order. On the other hand, during the isolation step, the efficiency of multisectioning is higher because: (i) a multisection creates more tasks than bisection, and (ii) there are several eigenvalues in one interval. The way we propose to use bisections or multisections is almost the same as that stated in [1].

3.2. The computation of the Sturm sequence. The recurrence (2.2) is intrinsically serial, so parallelism is not possible in this computation. The algorithm in [3] may be used, however, to vectorize the linear recurrence (2.1). Here, the computation of the regular Sturm sequence (2.1) is equivalent to solving a lower-triangular system of order $n+1$ consisting of three diagonals,

$$\begin{bmatrix} 1 & & & & \\ -a_1 & 1 & & 0 & \\ -b_2 & -a_2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & 0 & & -b_n & -a_n & 1 \end{bmatrix},$$

where $a_i=d_i-\lambda$, and $b_i=-e_i^2$. For a vector length $k<n/2$ the total number of arithmetic operations in the parallel algorithm is roughly $10n+11k$ while it is only $4n$ for the serial algorithm (we do not consider the operation that computes e_i^2, since this quantity can be provided by the user); resulting into an arithmetic redundancy which varies between 2.5 and 4.

This algorithm, therefore, is efficient only when vector operations are at least 4 times faster than sequential operations, which excludes the FX/8. The results on one processor of a CRAY X-MP are displayed in Fig. 1. In this figure, three methods are compared:

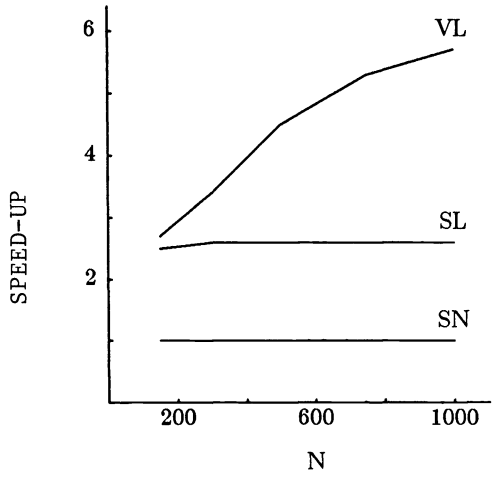


FIG. 1. *Computation of Sturm sequence on* CRAY X-MP, *speed-up over* SN. *N is the order of test matrices of* $[-1, 2, -1]$.

- Sequential computation of the linear recurrence (2.1) (SL);
- Vector computation of the linear recurrence (2.1) (VL);
- Sequential computation of the nonlinear recurrence (2.2) (SN).

Using 64 bit arithmetic on the test matrix $[-1, 2, -1]$, we have evaluated elements of the Sturm sequence (2.1) with no over- or underflow. The (SL) method is always faster than the (SN) method. The method (VL) (coded in FORTRAN) can reach a speed-up of 2.1 over the (SL) method and of 5.7 over the (SN) method.

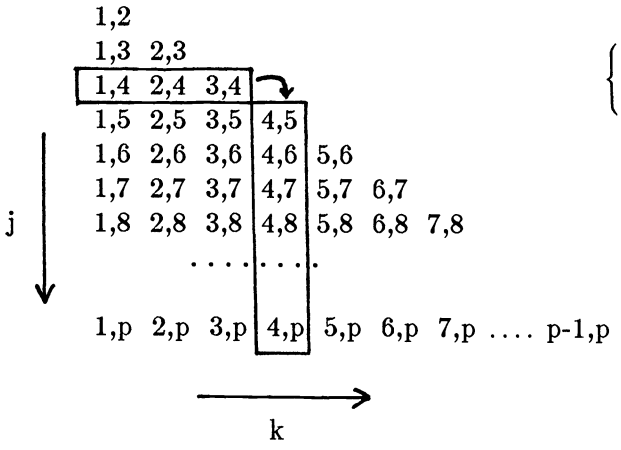
3.3. Computation of the eigenvectors and orthonormalization. The computation of an eigenvector can be started as soon as the corresponding eigenvalue is computed. So, we consider the extraction of an eigenvalue and the computation of its eigenvector as two parts of the same task, with the order of potential parallelism being dependent on the number of desired eigenvalues.

Orthonormalization is only performed on eigenvectors whose corresponding eigenvalues meet a predefined grouping criterion. The modified Gram-Schmidt method is used to orthonormalize each group. The algorithm is as follows:

do $k = 1, p$
  normalize (z_k)
  do $j = k+1, p$
    $z_j = z_j - (z_j \cdot z_k) z_k$
  od
od

where z_j and z_k are vectors, and $(z_j \cdot z_k)$ denotes their inner product. The statement in the inner loop is a vector instruction which updates the vector z_j with respect to the base vector z_k. Considering the dependency of the variables, we see that the inner loop is a **doall** loop, since all the iterations j are independent of one another for a particular k. On the other hand, the outer loop is a **doacross** loop, since any iteration k cannot start before vector z_k has been updated with respect to vectors $z_1, \cdots, z_{k-1}$ and as soon as it has been updated it is ready to be used as base vector for $z_{k+1}, \cdots, z_p$. These dependency relations are shown in Fig. 2.

Two schemes have been designed to perform the above process. One is a doall-inner-sequential-outer algorithm (called PS); the second, with some synchronization



$$\left\{ i,j \right\} \equiv \left\{ Z_j = Z_j - (Z_j \cdot Z_k)\, Z_k \right\}$$

FIG. 2. *Diagram of dependencies.*

mechanisms, is a doall-inner-pipelined-outer algorithm (called PP). Because of the synchronization overhead, the second method is to be preferred only when the order of the matrix and the size of the group are large. Table 1 shows a comparison of the two algorithms on the CRAY X-MP/48. The efficiency of the pipelined method in this case is the consequence of the low overhead of the synchronization mechanisms on the CRAY multiprocessor.

TABLE 1
Two implementations for the modified Gram-Schmidt Method.

	1 processor	2 processors	4 processors
Pipelined outer loop (PP)	26.13	13.61	8.20
Sequential outer loop (PS)	23.18	15.60	11.97
Speed-Up (PS/PP)	.89	1.15	1.46

Test performed on the CRAY X-MP for a matrix of order 1000.

If we consider processing several groups of vectors in parallel, then we have three levels of parallelism:

1) Concurrent processing of the groups;
2) Concurrent updating of vectors;
3) Vectorizing the updating step of each vector.

It is necessary to select the algorithm depending on the levels of parallelism in the architecture. For example, to orthonormalize large groups of eigenvectors, Level 2 and Level 3 can be adopted on multiprocessors like the CRAY X-MP/48 and the Alliant FX/8; whereas all three levels of parallelism can be adopted on a hierarchical structured system such as the Cedar machine [9], which contains several "clusters" of multiprocessors, each processor having vector capability.

4. Implementation and experiments.

4.1. Implementation. In this section, we describe the specific implementations of our algorithm TREPS on the multiprocessors Alliant FX/8 (consisting of 8 processors) and the CRAY X-MP/48 (consisting of 4 processors). The implementation on the Alliant is slightly different from the implementation on the CRAY.

During the orthogonalization process on the CRAY, the groups of eigenvectors to be orthonormalized are split into two classes depending on their sizes: the small groups are processed, one per processor, in parallel; and the large groups are processed one group at a time, using all the processors with the parallel-inner pipelined-outer loop algorithm. With 4 processors, a group of more than 10 eigenvalues is regarded as large.

For both implementations, the repartitioning of the intervals containing an isolated cluster is dynamic. These intervals are stored in a stack. As soon as one processor is ready, it fetches a new task (i.e., a new subinterval) from the stack. The updating of the pointer to the top of the stack is protected as a critical section by locks. It has to be pointed out that multitasking on the Alliant FX/8 is easier and less costly than that on the CRAY X-MP/48, since on the former system the creation of tasks is automatic. Moreover, on the CRAY we have used the usual trick, which consists of declaring as many tasks as there are processors, and making them active or idle depending on the number of processes involved so as to reduce the cost of task creation. This manipulation adds some complexity to the code, but it is the price to be paid if one is to obtain good performance on the CRAY X-MP/48.

4.2. Performance. To analyze the speed of this program, we start by comparing TREPS1, a version of the program which performs the extraction stage via bisection, with the EISPACK subroutines BISECT and TINVIT on the sequential machine VAX 785. For several runs with different tridiagonal matrices, we found that, even on a VAX 785, TREPS1 is no more than 20% slower than BISECT+TINVIT. In Table 2 we show such an experiment for the matrix $[-1, 2, -1]$ (unless otherwise stated, all test matrices are of this form). On the FX/8, with some synchronization directives, TREPS ran in vector-concurrent mode, with high speed-ups when the number of desired eigenvalues exceeds the number of processors. All experiments on the Alliant FX/8 are performed in double precision. In Table 3, we show the performance of TREPS1 when *all* the eigenvalues and vectors are obtained on the FX/8. In Table 4,

TABLE 2
Timings for TREPS *and* BISECT+TINVIT *Subroutines on* VAX 785.

N		TREPS1	BISC+TINV
100	Time (seconds)	4.03	3.6
	Ratio	1.11	1
300	Time (seconds)	34.13	31.58
	Ratio	1.08	1

Computations are in single precision. N is the order of the test matrix.

TABLE 3
Time and speed-up for TREPS1 *on* Alliant FX/8.

CE's	Time (second)	Speed-Up
1	115.59	1.0
2	57.12	2.0
4	28.89	4.0
8	14.78	7.8

Test matrices are of order 500.

TABLE 4
Speed-up for TREPS1 *as function of number of eigenvalues and vectors.*

	Time in seconds		
p	1 CE	8 CEs	Speed-Up
1	.15	.15	1
2	.25	.12	2.1
4	.51	.14	3.6
8	1.03	.14	7.4
10	1.29	.26	5.0
20	2.59	.42	6.2
50	6.67	.97	6.9
100	13.33	1.86	7.2
200	27.09	3.74	7.2
300	40.85	5.46	7.5

Test matrix $[-1, 2, -1]$ is of order 300. p is the number of desired eigenvalues and eigenvectors.

TABLE 5
Time for steps in TREPS1 *and* TREPS2.

N		Total	Isolation	Extr+Inv.	Orthonorm.	
500	TREPS1	14.78	.62	13.78	.38	(sec)
		100	4.2	93.2	2.6	(%)
	TREPS2	3.71	.62	2.71	.38	(sec)
		100	4.9	79.8	15.3	(%)
1000	TREPS1	67.68	3.35	54.0	10.33	(sec)
		100	16.7	73.1	10.2	(%)
	TREPS2	24.47	3.39	11.02	10.06	(sec)
		100	13.9	45.0	41.1	(%)

Computing all the eigenvalues and eigenvectors. Computations are done on Alliant FX/8 in double precision.

we measure the speed-up of TREPS1 on the FX/8 as a function of the desired p smallest eigenvalues for a matrix of order 300. On one processor, the time to compute p eigenvalues is proportional to p (in this situation the multisections become bisections, and our program is essentially equivalent to BISECT+TINVIT). On 8 processors, the speed-up over one processor increases from 1 to 5.0 when the number of desired eigenvalues and vectors varies from 1 to 10. It increases from 6.9 (for 30 eigenvalues) to 7.5 (for all of the 300 eigenvalues). Similar results are obtained with the version TREPS2, in which we use the Zeroin method (see [5]) for extraction of the eigenvalues.

Let us now look at the percentage of the computational time involved in each stage. We consider the time for grouping close eigenvalues as negligible. The step for extracting the eigenvalues and computing the eigenvectors is more time-consuming than the isolation process for TREPS1. The orthonormalization time depends heavily on the problem; in the extreme case when orthogonalization of all n vectors is necessary, the elapsed time can reach up to 50% of the whole time needed to solve the problem. Some examples are given in Table 5.

4.3. Comparison with other subroutines. In this section we compare the performance of our algorithm with BISECT+TINVIT, TQL2, and that in [4] when *all* the

eigenvalues and eigenvectors are required. We also compare our algorithm with BISECT and TQL1 when only the eigenvalues are needed. To evaluate the numerical performance, we compare the norm of the residuals, $\max_i \| Tz_i - \lambda_i z_i \|_2$, for the computed eigenvalues and eigenvectors for TREPS, BISECT+TINVIT, TQL2, and SESUPD [4]. Orthogonality of the eigenvectors is also checked by computing the $\max_{i,j} |Z^T Z - I|_{i,j}$, where $Z = [z_1, \cdots, z_n]$. Table 6 shows the results on the FX/8, machine precision of 1.11×10^{-16}, for the test matrix $[-1, 2, -1]$ of order 500. We see that for the above test matrix both the residuals and the quality of the eigenvectors and orthogonality of

TABLE 6
Residual and orthonormality of computed eigenvalues and eigenvectors.

	$\max_i \| Tz_i - \lambda_i z_i \|_2$	$\max_{i,j} \|Z^T Z - I\|_{i,j}$
TREPS1	1.00×10^{-12}	1.69×10^{-12}
TREPS2	4.12×10^{-13}	1.31×10^{-12}
TQL2	3.88×10^{-14}	2.66×10^{-14}
BISECT+TINVIT	9.75×10^{-13}	2.77×10^{-12}
SESUPD	5.87×10^{-15}	6.61×10^{-15}

Test matrices are of order 500.

TABLE 7a
Time and speed-up for computing all the eigenvalues and eigenvectors.

		Alliant		CRAY X-MP	
		1 CE	8 CE	1 CPU	4 CPU
TREPS1	time (sec)	115.6	14.8	11.06	3.04
	speed-up	1	7.8	1	3.6
TREPS2	time (sec)	25.7	3.7	1.72	.64
	speed-up	1	6.9	1	2.7
TQL2	time (sec)	486.4	103.1	6.68	—
	speed-up	1	4.7	1	—
BISECT+TINVIT	time (sec)	140.6	136.2	12.98**	—
	speed-up	1	1.0	1	—
SESUPD	time (sec)	—	17.91	—	—
	speed-up	—	—	—	—

Test matrix is of order 500.
** Probably has full orthonormalization due to the different grouping criterion used in the version of TINVIT on the Cray X-MP.

TABLE 7b
Speed-up over TQL2 *on the Alliant for computing all the eigenvalues and eigenvectors.*

i	Algorithm	$\frac{\text{Time (TQL2 on 1 CE)}}{\text{Time (algorithm } i \text{ on 8 CEs)}}$	$\frac{\text{Time (TQL2 on 8 CEs)}}{\text{Time (algorithm } i \text{ on 8 CEs)}}$
1	TREPS1	32.9	7
2	TREPS2	131.5	28
3	TQL2	4.7	1
4	BISECT+TINVIT	3.6	.8
5	SESUPD	27.1	5.8

Test matrix is of order 500.

TREPS1 and TREPS2 are close to that of BISECT+TINVIT but not as good as those of TQL2 and SESUPD [4].

In Tables 7a and 7b we give timing results comparing the performance of the above algorithms on both the FX/8 and the CRAY X-MP. Note that the time for TQL2 on one CE is 131 times slower than the time required by TREPS2 on 8 CE's, and that the time for TQL2 on 8 CE's is 28 times slower than that of TREPS2. Furthermore, TREPS2 is 4.8 times faster than SESUPD. In Tables 8a and 8b we compare both versions of our algorithm with TQL1 and BISECT for obtaining all the eigenvalues only. In Table 9 we show timing results for obtaining all the eigenvalues and eigenvectors of a random tridiagonal matrix on the Alliant FX/8.

TABLE 8a
Time and speed-up for computing all the eigenvalues.

		Alliant		CRAY X-MP	
		1 CE	8 CE	1 CPU	4 CPU
TREPS1	time (sec)	105.3	13.2	10.45	2.67
	speed-up	1	8.0	1	3.9
TREPS2	time (sec)	16.6	2.1	1.09	.4
	speed-up	1	7.9	1	2.7
TQL1	time (sec)	10.5	8.8	.87	—
	speed-up	1	1.2	1	—
BISECT	time (sec)	128.1	126.1	9.88	—
	speed-up	1	1.0	1	—

Test matrix is of order 500.

TABLE 8b
Speed-up over TQL1 *on the Alliant for computing all the eigenvalues.*

i	Algorithm	Time (TQL1 on 1 CE) / Time (algorithm i on 8 CEs)	Time (TQL1 on 8 CEs) / Time (algorithm i on 8 CEs)
1	TREPS1	.8	.7
2	TREPS2	5	4.2
3	TQL1	1.2	1
4	BISECT	.08	.07

Test matrix is of order 500.

TABLE 9
Computing all the eigenvalues and vectors of a random tridiagonal matrix on the Alliant FX/8.

	Time (second)	$\max_i \|Tz_i - \lambda_i z_i\|_2$	$\max_{i,j} \|Z^T Z - I\|_{i,j}$	Speed-up over TQL2
TREPS1	15.04	5.6×10^{-13}	5.1×10^{-12}	8
TREPS2	4.14	4.6×10^{-13}	1.3×10^{-11}	29
TQL2	120.54	2.8×10^{-14}	1.6×10^{-14}	1
BISECT+TINVIT	134.89	5.2×10^{-13}	6.4×10^{-12}	.9
SESUPD	6.65	2.5×10^{-14}	3.0×10^{-14}	18

Test matrices are of order 500.

TABLE 10
Results for full symmetric matrices on the Alliant FX/8.

Test matrix		Time (second)	$\max_{i,j} \lvert Z^T Z - I \rvert_{i,j}$
R	TREPS1	24.59	3.1×10^{-13}
	TREPS2	38.64	3.1×10^{-13}
A	TREPS1	23.96	1.1×10^{-12}
	TREPS2	12.64	5.7×10^{-13}

Test matrices are of order 500. **R** is full random symmetric matrix. $\mathbf{A} = (\mathbf{I} - 2uu^T)\mathbf{T}(\mathbf{I} - 2uu^T)$, where $\mathbf{T} = [-1, 2, -1]$.

Table 10 compares the times consumed by both versions of our algorithm on full symmetric matrices. The times indicated do not take into account the reduction to the tridiagonal form. For the matrix $\mathbf{A} = (\mathbf{I}-2uu^T)\mathbf{T}(\mathbf{I}-2uu^T)$, where $\mathbf{T} = [-1, 2, -1]$ and $u^T u = 1$, TREPS2 is very fast; but for a full symmetric random matrix **R**, TREPS2 is slower than TREPS1. This leads us to recall the implementation of the extraction algorithm. As mentioned earlier, although the Zeroin method is much faster than bisection, it may suffer from under- or overflow. Such problems are heralded by lack of convergence in the computation of some of the eigenvectors. To remedy this problem in TREPS2, we switch to bisection whenever the inverse iteration does not converge. The matrix **R** used in this experiment is an extreme case, where every single eigenvalue has first been computed by the Zeroin method, then recomputed by bisection; this explains the poor performance of TREPS2.

5. Conclusion. The algorithms TREPS1 and TREPS2 which have been presented in this paper are well suited for multiprocessors. A speed-up which is almost equal to the number of processors can be obtained as long as the number of desired eigenvalues is several times larger than the number of processors. They also achieve high efficiency because the amount of arithmetic operations for each element of data (a total of $O(n)$ elements) is high, thus avoiding the traditional bottleneck of memory bandwidth.

TREPS2, which extracts the eigenvalues using the Zeroin method, is much faster than the combination of EISPACK's BISECT and TINVIT, as well as TQL2. It can also be faster than SESUPD [4], on a multiprocessor such as the FX/8, especially when the linear recurrence (2.1) is well behaved. In fact we consider TREPS2 to be the algorithm of choice for obtaining either all the eigenvalues (it is faster than TQL1), or few of the eigenvalues and the corresponding eigenvectors of a tridiagonal matrix. Furthermore, comparisons with [4] indicate that TREPS2 is equally competitive with other multiprocessor algorithms that seek all the eigenvalues and vectors of full symmetric matrices. This is especially true since the back-transformation involved, in the full case, can be performed by matrix multiplication routines which are very efficient on both the FX/8 and the CRAY X-MP/48.

Acknowledgment. The authors would like to thank John Larson at CRAY Research for his valuable help in performing the experiments on the CRAY X-MP.

REFERENCES

[1] H. J. BERNSTEIN AND M. GOLDSTEIN, *Parallel implementation of bisection for the calculation of eigenvalues of tridiagonal symmetric matrices*, Courant Institute of Mathematical Sciences Report, New York University, New York, 1985.

[2] H.BOWDLER, R. MARTIN, C. REINSCH AND J. H. WILKINSON, *The* QR *and* QL *algorithms for symmetric matrices, Contribution* II/3, in Handbook for automatic computation, Vol. II, Linear Algebra, Springer-Verlag, Berlin, New York, 1971, pp. 227-240.

[3] 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.

[4] J. J. DONGARRA AND D. C. SORENSEN, *A fast algorithm for the symmetric eigenvalue problem*, IEEE Proc. 7th Symposium on Computer Arithmetic, Urbana, IL, 1985, pp. 338-342, this Journal, 8 (1987), to appear.

[5] G. E. FORSYTHE, M. A. MALCOM AND C. B. MOLER, *Computer Methods for Mathematical Computations*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

[6] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, 1983.

[7] H.-M. HUANG, *A Parallel Algorithm for Symmetric Tridiagonal Eigenvalue Problems*, CAC Document No. 109, Center for Advanced Computation, Univ. Illinois at Urbana-Champaign, February 1974.

[8] D. KUCK AND A. SAMEH, *Parallel computation of eigenvalues of real matrices*, IFIP Congress 1971, 2 (1972), pp. 1266-1272.

[9] D. KUCK, D. LAWRIE, A. SAMEH AND E. DAVIDSON, *Construction of a large-scale multiprocessor*, Cedar Document No. 45, Univ. Illinois at Urbana-Champaign, 1984.

[10] A. OSTROWSKI, *Solution of Equations and Systems of Equations*, Academic Press, New York, 1966.

[11] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Oxford Univ. Press, Oxford, 1965.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 2, March 1987

© 1987 Society for Industrial and Applied Mathematics

A COMPARISON OF DOMAIN DECOMPOSITION TECHNIQUES FOR ELLIPTIC PARTIAL DIFFERENTIAL EQUATIONS AND THEIR PARALLEL IMPLEMENTATION*

DAVID E. KEYES† AND WILLIAM D. GROPP‡

Abstract. Several preconditioned conjugate gradient (PCG)-based domain decomposition techniques for self-adjoint elliptic partial differential equations in two dimensions are compared against each other and against conventional PCG iterative techniques in serial and parallel contexts. We consider preconditioners that make use of fast Poisson solvers on the subdomain interiors. Several preconditioners for the interfacial equations are tested on a set of model problems involving two or four subdomains, which are prototypes of the stripwise and boxwise decompositions of a two-dimensional region. Selected methods have been implemented on the Intel Hypercube by assigning one processor to each subdomain, making use of up to 64 processors. The choice of a "best" method for a given problem depends in general upon: (a) the domain geometry, (b) the variability of the operator, and (c) machine characteristics such as the number of processors available and their interconnection scheme, the memory available per processor, and communication and computation rates. Illustrations from the first two categories are provided herein. A comparison paper [23] emphasizes the importance of the third category.

Key words. domain decomposition, substructuring, elliptic problems, Schur complement, preconditioning, parallel algorithms

AMS(MOS) subject classifications. 65N20, 65F10, 68A20

1. Introduction.

A number of methods based on domain decomposition have been proposed in recent years for the numerical solution of elliptic partial differential equations. Such methods are based upon the observation that the domain of problem definition may be regarded as the union of two or more subdomains, on each of which the restriction of the original problem may take on a particularly convenient form. Decomposition by domain also provides a natural route to parallelism. For some problems, these methods can be interesting even as serial algorithms, when the advantages that arise from isolating the subproblems can be made to outweigh the extra work involved in enforcing the proper conditions at the interfaces of the subdomains. A fortiori, they are interesting as parallel methods since reasonably large independent subtasks can readily be identified. We compare the performance of several domain decomposition methods and a class of undecomposed domain methods on a common set of two-dimensional, linear, scalar problems, and examine the parallelizability of each. Our aims are to point out the unity under certain conditions of methods which have been presented independently, and also to identify some problem characteristics which tend to favor certain methods over others in the context of parallelism.

Apart from the advantage of reformulating a large discrete problem as a collection of smaller problems which can be solved independently, there are at least two other

* Received by the editors January 6, 1986; accepted for publication (in revised form) March 31, 1986. This article was presented at the Second SIAM Conference on Parallel Processing for Scientific Computing that was held in Norfolk, Virginia on November 18–21, 1985.

† Research Center for Scientific Computation, and Department of Mechanical Engineering, Yale University, Box 2157, Yale Station, New Haven, Connecticut 06520. The research of this author was supported by the Office of Naval Research under contract N00014-82-K-0184.

‡ Research Center for Scientific Computation, Yale University, Box 2158, Yale Station, New Haven, Connecticut 06520. The research of this author was supported in part by the Office of Naval Research under contract N00014-82-K-0184, the National Science Foundation under grant MCS-8106181, and the Air Force Office of Scientific Research under contract AFOSR-84-0360.

motivations for considering domain decomposition methods, which may be present individually or in combination with the first in any given problem. All are of the "divide-and-conquer" type. Domains of irregular shape can be decomposed into subdomains of regular shape on which tensor-product-based discretization schemes can be employed, leading to discrete operators of regular structure. Also, regions of relative nonuniformity of the differential operator, whether due to coefficient variability or even substantially different physics, can be isolated into different subdomains, again resulting in exploitable locally regular structure. An example of each of these motivations is given.

The decomposition topologies considered involve both simple interfaces (with and without overlap regions) and cross-points. We compare Schur complement matrix methods (e.g., [2], [14], [20]), full partitioned matrix methods based on block Gaussian elimination not making explicit use of the reduced Schur complement system (e.g., [5]), and other methods of variational type (e.g., [19]). These classes of methods are similar at the discrete level, employing preconditioned conjugate gradient (PCG) iterations as the outer loop, and an exact equivalence between the iterates of the first two can be established under certain conditions. In all of these methods the largest implicit problems are Dirichlet or Neumann solves over the subdomains; therefore an easy handle on parallelism can be provided from an a priori dissection of the grid. However, global communication is required in forming the inner products of the PCG iterations (and also in one class of preconditioning methods), so the optimum parallel implementation is not completely straightforward. The optimum number of subdomains is generally both architecture- and problem-dependent, since the communication cost per iteration and the overall number of iterations tend to increase with the number of subdomains.

A variety of preconditioners have been proposed, for some of which there exist theoretical results showing the convergence rate of the iterations to be asymptotically independent of the spatial resolution, or only weakly dependent thereon. Some of these optimal preconditioners are exact for uniform operators, and in practice work best when the operator coefficients do not vary too much *along* the interfaces. For problems in which there is variation along the interfaces, the condition number of the system, though asymptotically independent of resolution, is larger and it is interesting to consider alternative low-bandwidth approximations to the Schur complement matrix, at least for sufficiently low resolution. We designate this type of preconditioning Modified Schur Complement (MSC), and include some examples in our comparisons.

The outline of this article is as follows. In § 2 we review some recent contributions to the domain decomposition literature, summarizing methods and key theoretical results, and introducing MSC preconditioning. Section 3 contains an experimental comparison of various methods on the same small-scale model problems, all implemented serially. Parallel implementation issues for domain decomposition methods are discussed in § 4. We present some machine timings and iteration counts obtained on the Intel Hypercube for three leading methods applied to the model problem of Laplace's equation in the unit square, and draw some conclusions in § 5. Practical limitations confine the examples to a relatively small region of the parameter space of problem size and number of subdomains. In a companion paper [23], we consider theoretically a larger region of this space for three communication topologies: ring, a two-dimensional mesh, and an n-cube. Onto these we consider the natural decomposition mappings: a decomposition into strips onto the ring, a decomposition into boxes onto the mesh, and decompositions of both types onto the n-cube. Serial complexity comparisons and parallel efficiency comparisons of the methods are presented.

2. A review of recent PCG-based domain decomposition methods. The substructuring of elliptic partial differential equations by domain has served as a practical computational technique for over twenty years (e.g., [32]), and its theoretical origins (as a method of proving the solvability of the Dirichlet problem on irregular regions) extend back to the last century [36]. A briefly annotated bibliography of various direct and iterative approaches is contained in the introduction of [2]. Here we summarize only recent domain decomposition algorithms which make use of preconditioned conjugate gradient iteration in the outer loop, which appear to have originated with [12]. There are other types of domain decomposition-related methods such as Schwarz–Jacobi [35] and Schwarz-multigrid [29] which are not considered in this article.

2.1. Problem definition. The methods will be illustrated on a model second-order, positive definite, self-adjoint elliptic Dirichlet problem on a bounded domain in R^2 with a piecewise smooth boundary:

$$Lu = f \text{ in } \Omega, \qquad a = 0 \text{ on } \partial\Omega, \tag{2.1a}$$

in the weak formulation, in which we seek $u \in H_0^1(\Omega)$ such that for all $v \in H_0^1(\Omega)$

$$A_\Omega(u, v) = (f, v), \tag{2.1b}$$

where

$$A_\Omega(u, v) \equiv \sum_{p,q=1}^{2} \int_\Omega a_{pq}(\mathbf{x}) \frac{\partial u}{\partial x_p} \frac{\partial v}{\partial x_q} \, d\mathbf{x}$$

and

$$(f, v) \equiv \int_\Omega fv \, d\mathbf{x}.$$

Given a triangulation of Ω, we define the discrete subspace of H_0^1 consisting of piecewise linear functions vanishing on $\partial\Omega$, H_{0h}^1. Note that the dimension of H_{0h}^1 is the number of vertices in the interior of Ω, defined as n. For all of the algorithms to be described but one, it is sufficient to consider the usual cardinal basis for H_{0h}^1, consisting of C^0 piecewise linear functions of the smallest possible support, denoted $\{\psi_j\}_{j \in N}$, $N = \{1, 2, \cdots, n\}$. The discrete approximation to u in H_{0h}^1 is then represented by a vector of nodal coefficients, u_h.

The Galerkin formulation of (2.1b) leads to the matrix equation

$$Au_h = f_h \tag{2.2}$$

where

$$[A]_{ij} = \sum_{p,q=1}^{2} \int_\Omega a_{pq} \frac{\partial \psi_j}{\partial x_p} \frac{\partial \psi_i}{\partial x_q} \, d\mathbf{x}, \qquad i, j \in N,$$

and

$$[f_h]_i = \int_\Omega f \psi_i \, d\mathbf{x}, \qquad i \in N.$$

The simplest decomposition on which all of the methods can be compared is that involving two simply connected subdomains. Though it is somewhat academic from the point of view of parallel processing, we consider this case first because it generalizes straightforwardly to multiple-strip decompositions and allows presentation of the basic ideas without cumbersome notation. For all of the algorithms to be described but one, the intersections of the subdomains are restricted to be interfaces of lower dimension. Referring to Fig. 1(a), we define $\gamma_{12} = \partial\Omega_1 \cap \partial\Omega_2$.

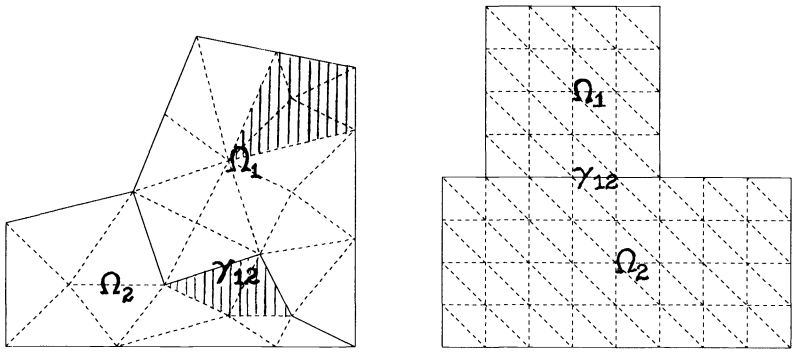


FIG. 1. *Sample domains illustrating the triangulation and substructuring described in* § 2.1. (a) *A general domain showing partitioning into two subdomains with a common interface (or separator set) lying along segments of the triangulation. Support of typical nodal basis functions* ψ_{1j}, $j \in N_1$ *and* ψ_{2j}, $j \in N_0$ *are shaded.* (b) *A model geometry for the use of fast Poisson solvers on the subdomains: a union of uniformly triangulated rectangles.*

The triangulation must be such that the segments of γ_{12} coincide with sides of the triangular elements. Let f_k denote the restriction of f to Ω_k. With a slight abuse of the usual notation, we define the discrete subspaces of functions over each subdomain which vanish on the outer boundary, but may be nonvanishing on the interface boundary, H^1_{kh}, $(k=1,2)$, and their subsets, H^1_{0kh}, of functions which vanish also on γ_{12}. The H^1_{0kh} have cardinal bases $\{\psi_{kj}\}_{j\in N_k}$, where N_k is an index set for the nodes interior to Ω_k, of dimension n_k. Denote by $\{\psi_{kj}\}_{j\in N_0}$, where N_0 is an index set of dimension n_0 for the nodes on the interface, the bases for the functions in H^1_{kh} which vanish at the interior nodes of Ω_k, but have nonvanishing trace on γ_{12}.

Equation (2.2) can be symmetrically permuted into the form

$$\begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \begin{pmatrix} u_0 \\ u_1 \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \end{pmatrix} \tag{2.3}$$

where the first block row corresponds to the unknowns defined at the nodes of the separator set γ_{12}, and where A_{11} is itself a 2×2 block diagonal matrix, one block corresponding to the nodal values in each subdomain. In (2.3) and hereafter the subscript h is dropped where there is no ambiguity between the continuous and discrete formulations. The matrix A is symmetric and positive definite, as are its diagonal blocks, by the hypotheses preceding (2.1). Consequently, the interior unknowns may be formally eliminated in forming the Schur complement system (sometimes denoted the *capacitance*[1] system)

$$Cu_0 = g \tag{2.4}$$

where $C = A_{00} - A_{01}A_{11}^{-1}A_{10}$ and $g = f_0 - A_{01}A_{11}^{-1}f_1$.

[1] A capacitance matrix formulation has been extensively used in the context of *imbedding* an irregular region in a regular one since Hockney's description of its application to electrostatics in 1970 (whence the name) [24]. In the imbedding variant, the reduced system unknowns correspond to the values on the irregularly shaped boundary. Imbedding and decomposition examples are considered together in [7]. Conjugate gradient iterations with a symmetrizer-preconditioner were applied to the imbedding variant in [31].

As the Schur complement of A_{00} in A, C is also symmetric and positive definite. Construction of the right-hand side of (2.4) requires one solve on each subdomain with homogeneous Dirichlet conditions on the interface. Having solved (2.4) for u_0, the interior unknowns can be recovered from the lower block row of (2.3), again at the cost of one solve on each subdomain with u_0 as inhomogeneous Dirichlet data.

In the two-subdomain case it is helpful for illustrative purposes to rewrite (2.3) in the expanded form

$$\begin{bmatrix} A_{00} & A_{01}^{(1)} & A_{01}^{(2)} \\ A_{10}^{(1)} & A_{11}^{(1)} & 0 \\ A_{10}^{(2)} & 0 & A_{11}^{(2)} \end{bmatrix} \begin{pmatrix} u_0 \\ u_1^{(1)} \\ u_1^{(2)} \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1^{(1)} \\ f_1^{(2)} \end{pmatrix} \tag{2.5}$$

where $u_1^{(1)}$ and $u_1^{(2)}$ are the unknowns in subdomains 1 and 2, respectively, and to decompose the interface diagonal block as

$$A_{00} = A_{00}^{(1)} + A_{00}^{(2)}$$

where

$$[A_{00}^{(k)}]_{ij} = \sum_{p,q=1}^{2} \int_{\Omega_k} a_{pq} \frac{\partial \psi_j}{\partial x_p} \frac{\partial \psi_i}{\partial x_q} \, d\mathbf{x}, \qquad i,j \in N_0.$$

The dimensions of $A_{11}^{(1)}$ and $A_{11}^{(2)}$ are n_1 and n_2, respectively, with $n = n_0 + n_1 + n_2$.

The case of Poisson's equation on a union of rectangles, uniformly triangulated as in Fig. 1(b), will be frequently considered in the sequel. For this so-called Courant triangulation, fast Poisson solvers [37] may be used to invert the $A_{11}^{(k)}$.

Since it explicitly involves the inverses of the $A_{11}^{(k)}$, the matrix operator C can be expensive to construct, generally requiring n_0 solves on each subdomain. Historically, domain decomposition was first approached in this way [32], and it remains a useful procedure when the formation of the factors of C can be amortized over a large number of right-hand sides, for instance when the same discrete linear system arises at successive steps in a time-dependent problem (e.g., [30]).

For cases in which the construction of C cannot be so amortized, for instance when a new system like (2.2) arises at each step in a nonlinear problem, more efficient methods have been developed by means of preconditioned conjugate gradient iteration, which at each step require a matrix-vector multiply involving C, without its explicit construction. Each PCG iteration still requires one solve on each subdomain, so the effectiveness of the PCG-based techniques depends on keeping the number of steps small through a preconditioning which is considerably less expensive to construct and apply than C^{-1} itself.

Of course, PCG iterations can also be used to solve the full systems in the form of (2.2) and (2.3), the type of preconditioning which is natural and efficient being different in each case. In subsequent sections we make some comparisons as to the effectiveness and generality of techniques which take each of these three formulations as their starting points. We refer to methods which depart directly from (2.2) as global matrix methods (GMM), from (2.3) as partitioned matrix methods (PMM), and from (2.4) as Schur complement matrix methods (SCM). For subsequent reference to individual steps, we present below the form of the PCG algorithm used in applications for solving the symmetric $n \times n$ linear system $\mathcal{A}x = b$ with (symmetric) preconditioner $\mathcal{B}$:

ALGORITHM PCG

Choose initial iterate:

(PCG.1) x^0, arbitrary

Compute initial residual:

(PCG.2) $r^0 \leftarrow b - \mathcal{A}x^0$

Compute preconditioned residual:

(PCG.3) $s^0 \leftarrow \mathcal{B}^{-1}r^0$

Initialize direction:

(PCG.4) $p^0 \leftarrow s^0$

Compute $\mathcal{B}$-inner-product:

(PCG.5) $\gamma^0 \leftarrow (r^0, s^0)$

For $k = 0$ **Step 1 Until** *Convergence*, **Do**

Compute matrix-vector product:

(PCG.6) $q^k \leftarrow \mathcal{A}p^k$

Compute $\mathcal{A}$-inner-product:

(PCG.7) $\tau^k \leftarrow (p^k, q^k)$

Compute step length:

(PCG.8) $\alpha^k \leftarrow \gamma^k / \tau^k$

Update solution:

(PCG.9) $x^{k+1} \leftarrow x^k + \alpha^k p^k$

Compute new residual:

(PCG.10) $r^{k+1} \leftarrow r^k - \alpha^k q^k$

Compute new preconditioned residual:

(PCG.11) $s^{k+1} \leftarrow \mathcal{B}^{-1}r^{k+1}$

Compute $\mathcal{B}$-inner-product:

(PCG.12) $\gamma^{k+1} \leftarrow (r^{k+1}, s^{k+1})$

Compute orthogonalization coefficient:

(PCG.13) $\beta^k \leftarrow \gamma^{k+1} / \gamma^k$

Update direction:

(PCG.14) $p^{k+1} \leftarrow s^{k+1} + \beta^k p^k$

End For

In addition to the storage and workspace requirements for the application of $\mathcal{A}$ and $\mathcal{B}^{-1}$, the algorithm requires storage for four vectors of length n. As is well known (e.g., [12]), the $\mathcal{A}$-norm of the error at the kth iteration of PCG is bounded according to

$$\frac{\|x^k - x\|_{\mathcal{A}}}{\|x^0 - x\|_{\mathcal{A}}} \leq 2\left(\frac{\sqrt{\kappa}-1)}{\sqrt{\kappa}+1}\right)^k$$

where κ is the condition number of $\mathcal{B}^{-1}\mathcal{A}$, and the algorithm produces exact convergence in a number of iterations which is at most the number of distinct eigenvalues of $\mathcal{B}^{-1}\mathcal{A}$. The operators $\mathcal{A}$ and $\mathcal{B}$ are said to be *spectrally equivalent* if there exist positive constants γ_0 and γ_1 independent of the discretization such that for all x, $\gamma_0(x, \mathcal{B}x) \leq (x, \mathcal{A}x) \leq \gamma_1(x, \mathcal{B}x)$.

2.2. Schur complement methods (SCM). The Schur complement methods are realized by taking $\mathcal{A}$ in Algorithm PCG to be C and selecting appropriate $\mathcal{B}$. The iterations occur on vectors of length n_0. Note that by construction the product Cp^k consists of the residuals at the nodes along γ_{12} of the original discrete operator applied to the vector which satisfies the discrete equations with homogeneous boundary conditions on $\partial\Omega$ in each of the subdomains and equals p^k on γ_{12}. Four related choices of $\mathcal{B}$ are reviewed in this subsection.

Dryja [14], [15] showed that C of (2.4) is spectrally equivalent to the matrix $K^{1/2}$, where K is the tridiagonal matrix of order n_0 with diagonal elements 2 and off-diagonal elements -1, the discrete Laplacian operator over a uniform grid in one dimension. For notational convenience, we define the average nodal spacing along the interface, $h = (n_0+1)^{-1}$. Inserting a factor of 2 for consistent scaling across the methods to follow, Dryja's preconditioner has the eigendecomposition

$$M_D \equiv 2K^{1/2} = W\Lambda_D W^T$$

where

$$[W]_{ij} = \sqrt{2h}\sin ij\pi h,$$

and

$$\Lambda_D = \operatorname{diag}(\lambda_j^D)$$

where

$$\lambda_j^D = 2\sqrt{\sigma_j},$$

with

$$\sigma_j = 4\sin^2\frac{j\pi h}{2}.$$

The condition number of $K^{1/2}$, which is equal to λ_{n_0}/λ_1, grows in proportion to the number of interfacial unknowns n_0 as this number becomes large. Through their spectral equivalence, the same is true of C, which implies that the convergence of the unpreconditioned Schur complement system iteration will deteriorate as the mesh is refined. The action of M_D^{-1} on a vector can be computed with a pair of sine transforms in $O(n_0 \log n_0)$ operations; hence the cost of the preconditioning is inexpensive in comparison with the cost of forming a matrix multiplication with a general C, which involves two-dimensional subdomain solves.

By means of FFT's in one dimension, Dryja also gave an $O(n_0 \log n_0)$ method for performing the subdomain solves for the case $L = -\Delta$ and Ω a union of two rectangular subregions, by exploiting the regular sparsity pattern of the right-hand sides arising during the PCG iterations. In this case the overall operation count for solving (2.2) is dominated by the pre- and post-processing involved in forming g of (2.4) and backsolving, instead of by solving the Schur complement system. For more general operators and domain geometries, the cost of one PCG iteration is comparable to the pre- and post-processing, namely $O(n_0^2 \log n_0)$.

Golub and Mayers [20] arrived heuristically at a preconditioner for C by starting from the observation that the elements $[C]_{ij}$ may often be well approximated as constant along a diagonal. In other words, the influence of the data at interfacial node j on the residual at interfacial node i depends (approximately) only on the discrete distance along the interface, $|i-j|$. This led Golub and Mayers to solve a discrete infinite domain Laplace problem with an infinite interface dividing two half-planes, the data on the interface and at infinity prescribed to be zero everywhere except at the origin, where it was taken as one. The ijth element of their preconditioner of Toeplitz form was then defined as the residual of the discrete Laplacian at the point on the interface at a distance $|i-j|$ from the origin. This preconditioner was tested and found superior to Dryja's on the problem considered in [20]. Though computationally complicated to construct, they noticed that a second, FFT-implementable preconditioner could be derived by replacing a term in their generating function expression for the interface residual by Dryja's preconditioner. Their result,

$$M_G = W \Lambda_G W^T$$

where

$$\Lambda_G = \operatorname{diag}(\lambda_j^G)$$

where

$$\lambda_j^G = 2\sqrt{\sigma_j + \frac{\sigma_j^2}{4}},$$

or

$$M_G = 2(K + \tfrac{1}{4}K^2)^{1/2},$$

has been also employed as an effective refinement of M_D in other investigations [2], [16].

Bjorstad and Widlund [2] showed that C, $C^{(1)} \equiv A_{00}^{(1)} - A_{01}^{(1)}(A_{11}^{(1)})^{-1}A_{10}^{(1)}$, and $C^{(2)} \equiv A_{00}^{(2)} - A_{01}^{(2)}(A_{11}^{(2)})^{-1}A_{10}^{(2)}$, are all spectrally equivalent. (Note that $C = C^{(1)} + C^{(2)}$.) Assuming that Ω is decomposed in such a way that it is computationally convenient to solve Neumann problems on one of the subdomains, say Ω_1, with zero Dirichlet data on $\partial\Omega_1 \cap \partial\Omega$ and natural boundary conditions on γ_{12}, they proposed $C^{(1)}$ as a preconditioner for C, acting on a suggestion of Dryja's. Applying $(C^{(1)})^{-1}$ to a vector p_0 of dimension n_0 requires solving the Ω_1 Neumann problem

$$\begin{bmatrix} A_{00}^{(1)} & A_{01}^{(1)} \\ A_{00}^{(1)} & A_{11}^{(1)} \end{bmatrix} \begin{pmatrix} q_0 \\ q_1^{(1)} \end{pmatrix} = \begin{pmatrix} p_0 \\ 0 \end{pmatrix},$$

whence $q_0 = (C^{(1)})^{-1} p_0$. In the case of Poisson's equation on the union of two rectangular uniformly gridded regions, as pictured in Fig. 1(b), an explicit FFT-implementable expression for $C^{(1)}$ was derived in [2]. We shall define M_B as twice $C^{(1)}$ in what follows

for purposes of comparison. Let m_1 be the number of internal grid points in the vertical direction in Ω_1. Then

$$M_B = W\Lambda_B W^T$$

where

$$\Lambda_B = \text{diag}\,(\lambda_j^B)$$

where

$$\lambda_j^B = 2\left(\frac{1+\rho_j^{m_1+1}}{1-\rho_j^{m_1+1}}\right)\sqrt{\sigma_j + \frac{\sigma_j^2}{4}},$$

where

$$\rho_j = \frac{r_{j-}}{r_{j+}}$$

and

$$r_{j\pm} = 1 + \frac{\sigma_j}{2} \pm \sqrt{\sigma_j + \frac{\sigma_j^2}{4}}.$$

This is a diagonal rescaling of M_G which takes account of the aspect ratio, m_1/n_0, of one of the subdomains. In the case where Ω is a rectangle and Ω_1 and Ω_2 are symmetrically disposed about the interface, $C^{(1)} = C^{(2)}$, $M_B = C$, and the Bjorstad-Widlund method converges in one step.

Chan has carried this circle of ideas further for the special case where Ω is a rectangle and L is the Laplacian by finding the Fourier decomposition of C itself [8]. Referring to Fig. 2(a), let m_2 be the number of internal grid points in the vertical direction in Ω_2.

Then in the previous notation,

$$M_C = W\Lambda_C W^T$$

where

$$\Lambda_C = \text{diag}\,(\lambda_j^C)$$

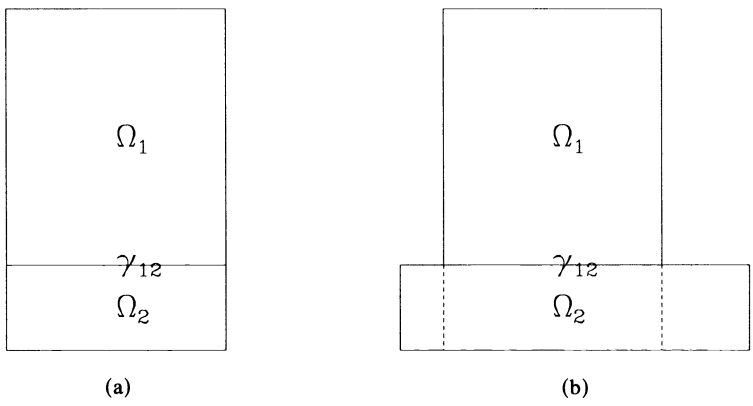


FIG. 2. *The applicability of the Chan preconditioner.* (a) *A model geometry for which* $M_C = C$. (b) *A model geometry for which* M_C *is not exact but furnishes a useful aspect ratio-sensitive preconditioner.*

where

$$\lambda_j^C = \left(\frac{1+\rho_j^{m_1+1}}{1-\rho_j^{m_1+1}} + \frac{1+\rho_j^{m_2+1}}{1-\rho_j^{m_2+1}}\right)\sqrt{\sigma_j + \frac{\sigma_j^2}{4}}.$$

By taking account of the aspect ratio of both of the subdomains, Chan's method converges in one step even in the unsymmetric case, and thus may be regarded as a direct fast Poisson solver. Of course, fast Poisson solvers not making use of domain decomposition can already be applied when Ω is a rectangle, and the serial computational complexity of Chan's approach can be made nearly equal to that of these algorithms by solving the Schur complement system in Fourier transform space [33]. However, Chan's preconditioner may also be applied in the iterative manner of the other preconditioners in a more general geometry like that of Fig. 2(b), for which a fast Poisson solver is not available, although exact only for the inscribed $n_0 \times (m_1 + m_2)$ rectangle.

In aid of visualizing C and these approximations thereto, surface plots of their elements as a function of their indices are given in Fig. 3 for the case of Poisson's equation in the unit square with $n_0 = 15$, $m_1 = m_2 = 7$. For clarity, Fig. 3(a) shows $C(= M_C = M_B)$ and the remaining plots show differences of C and various other preconditioners. Corresponding estimates of the condition numbers $\kappa(M^{-1}C)$ are listed along with convergence data in the first row of Table 1 in § 3.

The Chan and Bjorstad-Widlund methods are both exact for the Laplacian in a symmetrically decomposed rectangular domain. For a given h, the Golub-Mayers method coincides with these two methods in the infinite aspect ratio limit of the domain geometry:

$$\lim_{\substack{m_1\to\infty \\ m_2\to\infty}} \lambda_j^C = 2\sqrt{\sigma_j + \frac{\sigma_j^2}{4}} = \lambda_j^G.$$

The Dryja preconditioner is not exact for any domain geometry limit. All four of the preconditioners discussed above are *optimal* for the two-subdomain case in the sense that the condition number of the Schur complement system approaches a constant independent of h as the mesh is refined with fixed geometry. References [8], [1], [2] may be consulted for further comparisons between C, $C^{(1)}$, $C^{(2)}$, M_G, and M_D in various limits for h, m_1, and m_2, all within the context of the two-subdomain case.

References [1] and [2] are of additional interest in providing motivation from the underlying continuous problem for the $K^{1/2}$-family of preconditioners. It is shown therein that the capacitance system (2.4) is the discrete analogue of a zero jump condition along the interface for the normal derivatives of the functions which solve the continuous Dirichlet problems in each subdomain with data u_0 on the interface. For the case of two half-planes, the continuous operator acting on u_0 is shown in [1] to be equal (in Fourier space) to twice the square root of the continuous one-dimensional Laplacian. Hence, to the extent that the subdomain boundaries are "far" from the interface, one expects a multiple of the square root of the one-dimensional Laplacian to serve as an effective preconditioner for the finite domain problem. In the discrete case, this is precisely $M_D = 2K^{1/2}$. Problems with subdomains of small aspect ratio benefit from the refinements to M_D described in this section, since the boundaries are not far removed in such cases.

A disadvantage of working with the Schur complement system, independent of preconditioner, is that the subdomain solves are presumed to be carried out exactly. In the general nonseparable case of interest, fast Poisson solvers are not available, and

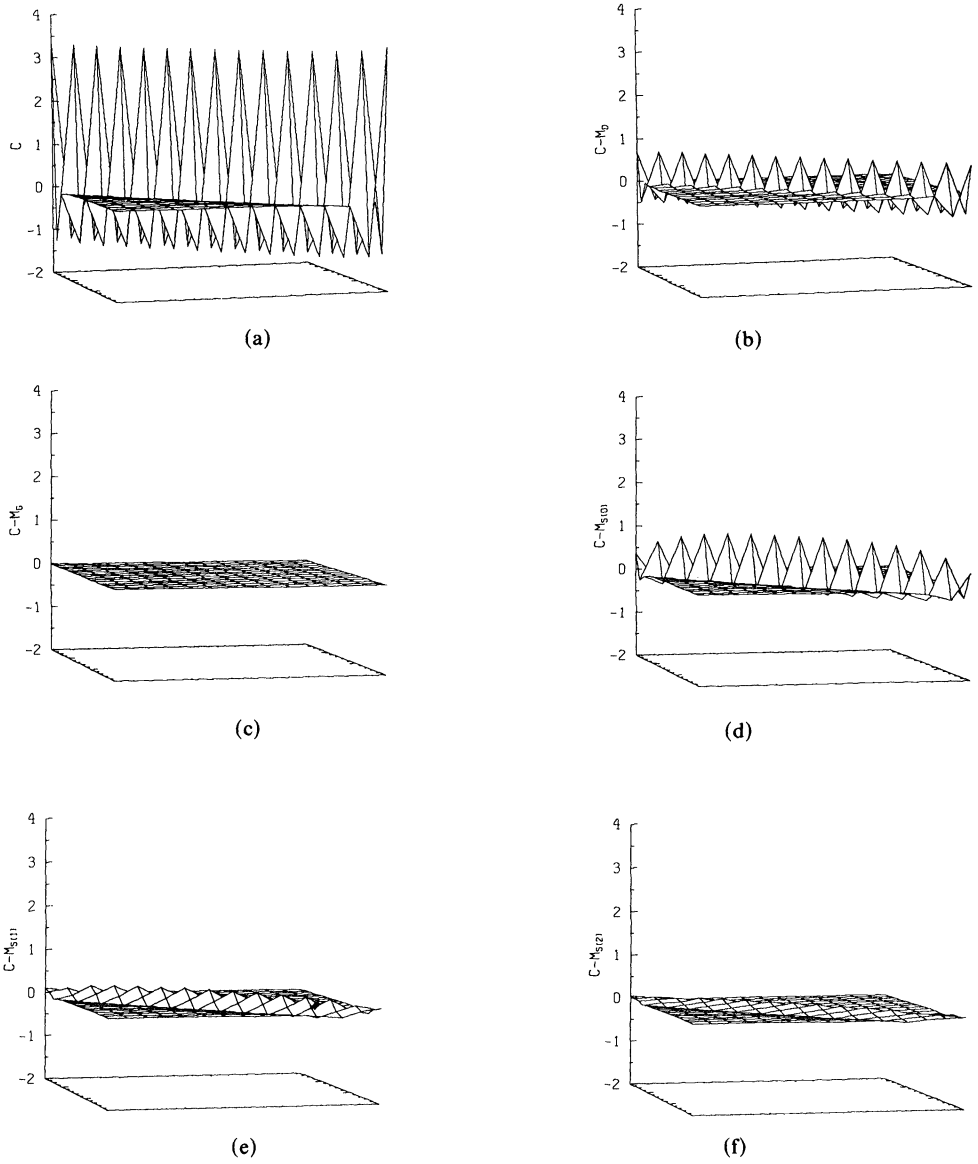


FIG. 3. *Surface plots of the elements of the Schur complement matrix C for Poisson's equation on a square uniformly discretized with* 16 *subintervals on a side, and differences of C and various preconditioners described in* §§ 2.2 *and* 2.4. (a) *C itself*; (b) $C - M_D$; (c) $C - M_G$; (d) $C - M_{S(0)}$; (e) $C - M_{S(1)}$; (f) $C - M_{S(2)}$.

one is faced with the necessity of direct sparse solvers in the computation of Cp^k, or of nested iterations. In the latter case, the convergence criterion for the inner iterations could conceivably be tuned to the rate of progress of the outer iteration to save computational work, but a more fully coupled framework of simultaneous iteration is possible, which we review in the next subsection.

2.3. Partitioned matrix methods (PMM). The partitioned matrix methods are realized by taking $\mathcal{A}$ to be A of (2.3) and selecting appropriate $\mathcal{B}$. The iterations occur on vectors of length n, rather than n_0. We begin with the following theorem, due to Eisenstat [18].

THEOREM 2.1. *Let the $n \times n$ partitioned matrix A and the $n_0 \times n_0$ Schur complement matrix C be defined as in* (2.3) *and* (2.4), *respectively, with corresponding right-hand sides f and g.*

(i) *Algorithm* PCG *applied to $Cv = g$ with initial iterate v^0 and preconditioner M is equivalent to algorithm* PCG *applied to $Au = f$ with initial iterate*

$$u^0 = \begin{pmatrix} v^0 \\ A_{11}^{-1}(f_1 - A_{10}v^0) \end{pmatrix}$$

and preconditioner

$$B = \begin{bmatrix} M + A_{01}A_{11}^{-1}A_{10} & A_{01} \\ A_{10} & A_{11} \end{bmatrix}, \tag{2.6}$$

in the sense that, for all $k \geqq 0$,

$$u^k = \begin{pmatrix} v^k \\ A_{11}^{-1}(f_1 - A_{10}v^k) \end{pmatrix}.$$

(ii) *There is no advantage to choosing an initial iterate more general than u^0 as above, in the sense that $\|u^k - u\|_A \leqq \|w^k - u\|_A$, where w^k is the kth iterate generated by* PCG *from the initial iterate*

$$w^0 = u^0 + \begin{pmatrix} 0 \\ \xi^0 \end{pmatrix}.$$

Proof. (i) Since A_{11}, M, and B are symmetric positive definite matrices, we may write $M = Q^TQ$ and $B = P^TP$, where

$$P = \begin{bmatrix} Q & 0 \\ A_{11}^{-1/2}A_{10} & A_{11}^{1/2} \end{bmatrix}.$$

The Schur complement system PCG iteration is equivalent to conjugate gradient (CG) iteration on $\hat{C}\hat{v} = \hat{g}$ with initial iterate $\hat{v}^{(0)} = 0$, where $\hat{C} \equiv Q^{-T}CQ^{-1}$, $\hat{v} \equiv Q(v - v^{(0)})$ and $\hat{g} \equiv Q^{-T}(g - Cv^{(0)})$; and similarly the partitioned system PCG iteration is equivalent to CG iteration on $\hat{A}\hat{u} = \hat{f}$ with initial iterate $\hat{u}^{(0)} = 0$, where $\hat{A} \equiv P^{-T}AP^{-1}$, $\hat{u} \equiv P(u - u^{(0)})$ and $\hat{f} \equiv P^{-T}(f - Au^{(0)})$. But note that

$$\begin{aligned}
\hat{A} &= \begin{bmatrix} Q^T & A_{10}^TA_{11}^{-1/2} \\ 0 & A_{11}^{1/2} \end{bmatrix}^{-1} \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \begin{bmatrix} Q & 0 \\ A_{11}^{-1/2}A_{10} & A_{11}^{1/2} \end{bmatrix}^{-1} \\
&= \begin{bmatrix} Q^{-T} & -Q^{-T}A_{10}^TA_{11}^{-1} \\ 0 & A_{11}^{-1/2} \end{bmatrix} \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \begin{bmatrix} Q^{-1} & 0 \\ -A_{11}^{-1}A_{10}Q^{-1} & A_{11}^{-1/2} \end{bmatrix} \\
&= \begin{bmatrix} Q^{-T}(A_{00} - A_{01}A_{11}^{-1}A_{10}) & 0 \\ A_{11}^{-1/2}A_{10} & A_{11}^{1/2} \end{bmatrix} \begin{bmatrix} Q^{-1} & 0 \\ -A_{11}^{-1}A_{10}Q^{-1} & A_{11}^{-1/2} \end{bmatrix} \\
&= \begin{bmatrix} Q^{-T}CQ^{-1} & 0 \\ 0 & I \end{bmatrix} = \begin{bmatrix} \hat{C} & 0 \\ 0 & I \end{bmatrix}.
\end{aligned}$$

Similarly,

$$\begin{aligned}
\hat{f} &= \begin{bmatrix} Q^T & A_{10}^TA_{11}^{-1/2} \\ 0 & A_{11}^{1/2} \end{bmatrix}^{-1} \begin{pmatrix} f_0 - A_{01}A_{11}^{-1}(f_1 - A_{10}v^0) - A_{00}v^0 \\ 0 \end{pmatrix} \\
&= \begin{bmatrix} Q^{-T} & -Q^{-T}A_{10}^TA_{11}^{-1} \\ 0 & A_{11}^{-1/2} \end{bmatrix} \begin{pmatrix} g - Cv^0 \\ 0 \end{pmatrix} = \begin{pmatrix} \hat{g} \\ 0 \end{pmatrix}.
\end{aligned}$$

Thus the partitioned matrix PCG iteration is equivalent to CG iteration on

$$\begin{bmatrix} \hat{C} & 0 \\ 0 & I \end{bmatrix} \begin{pmatrix} \hat{v} \\ \xi \end{pmatrix} = \begin{pmatrix} \hat{g} \\ 0 \end{pmatrix}$$

with initial iterate $\xi^{(0)} = 0$, $\hat{v}^{(0)} = 0$ and the equivalence is established.

(ii) To prove optimality, we first define $\hat{w}^k \equiv P(w^k - u^0)$, and note that

$$\hat{w}^0 = \begin{pmatrix} 0 \\ A_{11}^{1/2} \xi^0 \end{pmatrix} \quad \text{and} \quad \hat{u} = \begin{pmatrix} \hat{v} \\ 0 \end{pmatrix}.$$

Then

$$\begin{aligned}
\|w^k - u\|_A^2 &= \|\hat{w}^k - \hat{u}\|_{\hat{A}}^2 \\
&= \min_{P_k} \|(I - \hat{A}P_k(\hat{A}))(\hat{w}^0 - \hat{u})\|_{\hat{A}}^2 \\
&= \min_{P_k} \left\| \begin{pmatrix} -(I - \hat{C}P_k(\hat{C}))\hat{v} \\ (I - P_k(I))A_{11}^{1/2}\xi^0 \end{pmatrix} \right\|_{\hat{A}}^2 \\
&= \min_{P_k} \{\|(I - \hat{C}P_k(\hat{C}))\hat{v}\|_{\hat{C}}^2 + \|(1 - P_k(1))A_{11}^{1/2}\xi^0\|_2^2\} \\
&\geqq \min_{P_k} \{\|(I - \hat{C}P_k(\hat{C}))\hat{v}\|_{\hat{C}}^2\} \\
&= \min_{P_k} \|(I - \hat{A}P_k(\hat{A}))(\hat{u}^0 - \hat{u})\|_{\hat{A}}^2 \\
&= \|\hat{u}^k - \hat{u}\|_{\hat{A}}^2 \\
&= \|u^k - u\|_A^2,
\end{aligned}$$

where P_k is the set of polynomials of order k, and the second and penultimate steps rely on the optimality property of CG iteration (see [11, Chap. 3], for instance). □

By this theorem, any Schur complement system preconditioner M can be applied to the interfacial equations of the partitioned matrix by its incorporation into the matrix B as shown in (2.6). Moreover, the theorem suggests a form for the preconditioner for a more general algorithm in which is it not required that the subdomain solves corresponding to A_{11}^{-1} be carried out exactly. For instance, if A derives from a nonseparable operator L, fast Poisson solvers may be used to precondition the subdomain solves. More costly exact subdomain solves are thereby avoided. To allow for this type of generality, we take $\mathcal{B}$ to be $\tilde{B}$, where

$$\tilde{B} = \begin{bmatrix} \tilde{B}_{00} & \tilde{B}_{01} \\ \tilde{B}_{10} & \tilde{B}_{11} \end{bmatrix} \equiv \begin{bmatrix} M + \tilde{A}_{01}\tilde{A}_{11}^{-1}\tilde{A}_{10} & \tilde{A}_{01} \\ \tilde{A}_{10} & \tilde{A}_{11} \end{bmatrix}, \tag{2.7}$$

and where the $\tilde{A}_{ij}$ are determined on a problem-specific basis. A convenient form for the $\tilde{A}_{ij}$ may be derived from a permutation conformal to that of (2.3) of the discrete Galerkin equations for

$$\tilde{A}_\Omega(u, v) = (f, v) \tag{2.8}$$

where

$$\tilde{A}_\Omega(u, v) \equiv \sum_{k=1}^{2} \tilde{A}_{\Omega_k}(u, v)$$

where for each subdomain

$$\tilde{A}_{\Omega_k}(u, v) \equiv \sum_{p,q=1}^{2} \int_{\Omega_k} \tilde{a}_{pq}^k \frac{\partial u}{\partial x_p} \frac{\partial v}{\partial x_q} \, d\mathbf{x},$$

and where the constants $\tilde{a}_{pq}^{k}$ are chosen for each subdomain k in such a way that the resulting matrix $\tilde{A}$ is spectrally equivalent to A. This iteration is, of course, no longer equivalent to any reduced system iteration in the manner described above, and the scaling of M relative to the $\tilde{A}_{ij}$ becomes an issue, where it was not previously.

An efficient implementation of partitioned matrix preconditioners of the form (2.7) has been presented by Bramble, Pasciak and Schatz [5], wherein it is described how the operation $\mathcal{B}^{-1}r$ can be carried out at the cost of *two* subdomain solves per subdomain per iteration, plus the cost of applying M^{-1}. In the language of function-space decompositions, the solution u is written as the sum of a harmonic component, u^H, and a perpendicular component, u^P, such that in each subdomain k, $u^P|_k \in H^1_{0kh}$ satisfies

$$\tilde{A}_{\Omega_k}(u^P, v) = \tilde{A}_{\Omega_k}(u, v)$$

for all $v \in H^1_{0kh}$; and $u^H|_k \in H^1_{kh}$ satisfies $u^H = u$ on γ_{12} and

$$\tilde{A}_{\Omega_k}(u^H, v) = 0$$

for all $v \in H^1_{0kh}$.

In matrix terms, to solve

$$\tilde{B}\begin{pmatrix} u_0 \\ u_1 \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \end{pmatrix}$$

for u, one first solves

$$\tilde{A}_{11}u_1^P = f_1$$

for u_1^P, then

$$Mu_0 = f_0 - \tilde{A}_{01}u_1^P$$

for u_0, then

$$\tilde{A}_{11}u_1^H = -\tilde{A}_{10}u_0$$

for u_1^H and sets $u_1 = u_1^H + u_1^P$.

It is tempting to consider as a more economical alternative for $\mathcal{B}$ the matrix

$$\begin{bmatrix} M & 0 \\ \tilde{A}_{10} & \tilde{A}_{11} \end{bmatrix},$$

whose inversion allows skipping the first of the three steps above (and thus leads to an algorithm with no more subdomain solves per iteration than SCM), but this is not a symmetric preconditioner. The penalty for preserving a nest-free iterative structure in the case where the subdomain solves with A_{11} are too expensive to be done directly is thus an extra solve per subdomain with $\tilde{A}_{11}$ in the preconditioning step, and also the extra work in the multiplication and dot product steps due to the vector length of n instead of n_0.

In [6] the same authors consider a generalization of the Bjorstad–Widlund preconditioning in which the M of (2.7) is given by

$$\tilde{C}^{(1)} = \tilde{A}_{00}^{(1)} - \tilde{A}_{01}^{(1)}(\tilde{A}_{11}^{(1)})^{-1}\tilde{A}_{10}^{(1)}.$$

In matrix terms, to solve

$$\tilde{B}\begin{pmatrix} u_0 \\ u_1 \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \end{pmatrix}$$

for u, one first solves the Dirichlet problem in Ω_2:

$$\tilde{A}_{11}^{(2)} v = f_1^{(2)},$$

then the Neumann problem on Ω_1:

$$\begin{bmatrix} \tilde{A}_{00}^{(1)} & \tilde{A}_{01}^{(1)} \\ \tilde{A}_{10}^{(1)} & \tilde{A}_{11}^{(1)} \end{bmatrix} \begin{pmatrix} u_0 \\ u_1^{(1)} \end{pmatrix} = \begin{pmatrix} f_0 - \tilde{A}_{01}^{(2)} v \\ f_1^{(1)} \end{pmatrix},$$

then the Dirichlet problem in Ω_2:

$$\tilde{A}_{11}^{(2)} u_1^{(2)} = f_1^{(2)} - \tilde{A}_{10}^{(2)} u_0.$$

Since one of the subdomains solves is eliminated, the serial complexity of this method is less than that of the function-space decomposition technique just described, but the remaining subdomain solves are inherently sequential, and generalization to the many-subdomain case can only be carried out for decompositions (such as stripwise) in which the boundaries of the subdomains on which the natural boundary conditions are posed do not intersect.

2.4. Modified Schur Complement (MSC) preconditioners. Because the Schur complement matrix C is close to being tridiagonal, as illustrated for the Laplacian operator in Fig. 3, it is natural to consider the use of tridiagonal or other low-bandwidth preconditioners for it. Such an approximation of one matrix by another constrained to satisfy various sparsity requirements has often proved useful in connection with iterative methods. In this context, it was proposed independently by Chan and Resasco [9] and Eisenstat [18]. We can generate a class of interfacial preconditioners of the form

$$M_{S(k)} = \tilde{A}_{00} - E_k \tag{2.9a}$$

where E_k is a symmetric matrix with semi-bandwidth k which satisfies

$$E_k v_i = (\tilde{A}_{01} \tilde{A}_{11}^{-1} \tilde{A}_{10}) v_i \tag{2.9b}$$

for some $k+1$ vectors v_i, $i = 0, \cdots, k$. We have used $\tilde{A}_{11}$ rather than A_{11} in (2.9b) because this technique is motivated by variable coefficient problems for which exact subdomain solvers are too expensive to consider.

Note that the construction of E_k requires $k+1$ solves on each subdomain, which is in general $(k+1)/2$ times the cost of one preconditioning step with $\mathcal{B}$ of the form (2.7). For $k=0$, we consider the vector $v_0 = (1, 1, \cdots)^T$; for $k=1$, the vectors $v_0 = (1, 0, 1, 0, \cdots)^T$ and $v_1 = (0, 1, 0, 1, \cdots)^T$; and so forth. Equation (2.9b) gives $(k+1)n_0$ scalar equations for the $(k+1)(n_0 - k/2)$ distinct elements of E_k, but the overdetermination is consistent due to the symmetry of the product of matrices on the right-hand side. For the set of v_i recommended above, the diagonal elements of E_k can be read off from (2.9b) and the remaining elements can be obtained in $O(kn_0)$ operations. Note that $M_{S(k)}^{-1}$ can also be applied at the cost of only $O(kn_0)$ operations once its factorization is stored.

Surface plots of the differences between C (for the Laplacian) and $M_{S(0)}$, $M_{S(1)}$, and $M_{S(2)}$ are shown in Fig. 3 and their associated estimated condition numbers appear in Table 1.

For small k, which is the only practical limit, this class of preconditioners turns out to be markedly inferior to the optimal class described earlier for constant coefficient operators on uniform grids. However, it can be competitive when the interfaces are not placed along level curves of the coefficients. Figure 4 shows surface plots of the Schur complement matrix C and differences of C and various preconditioners for a problem with the same model geometry, but with the nonseparable operator equation

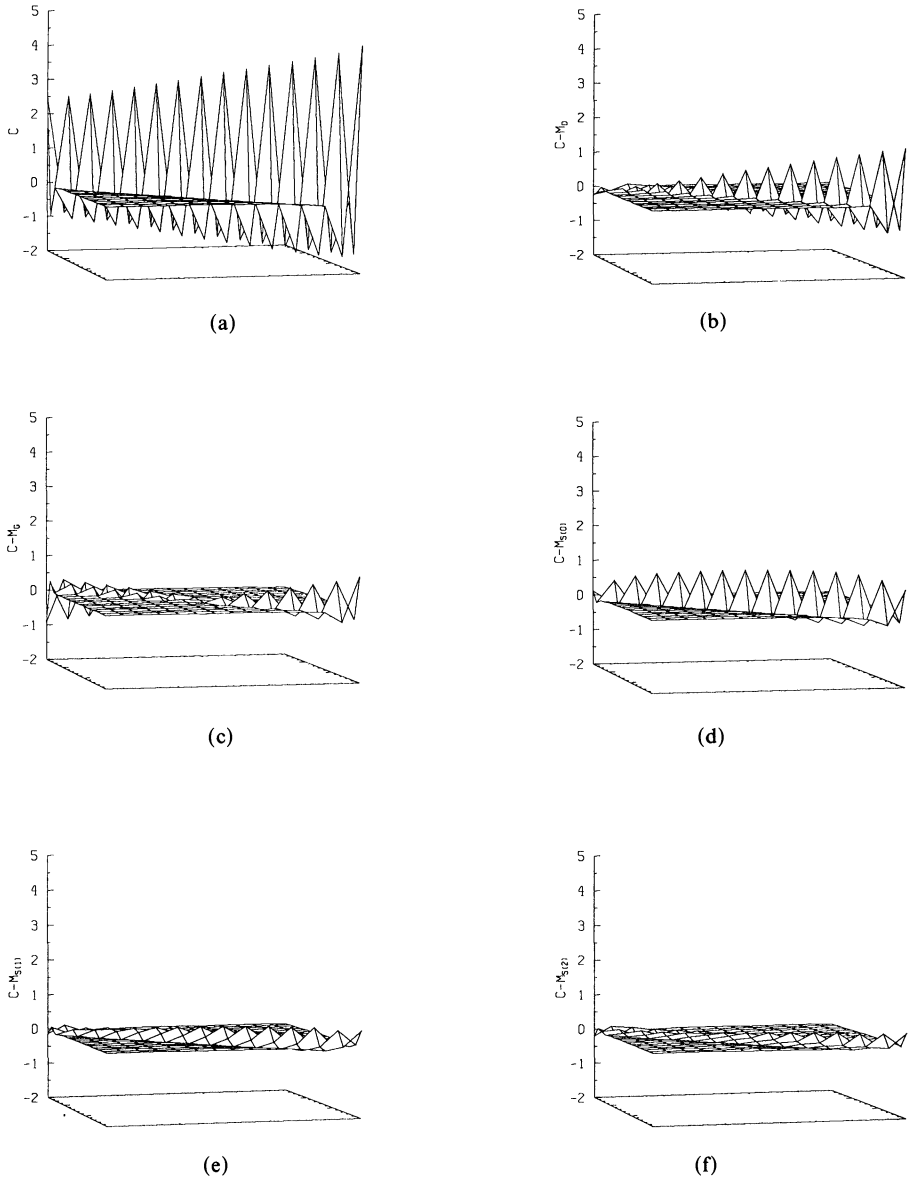


FIG. 4. *Surface plots of the elements of the Schur complement matrix C for a nonseparable operator on a square uniformly discretized with* 16 *subintervals on a side, and differences of C and various preconditioners described in* §§ 2.2 *and* 2.4. (a) *C itself*; (b) $C - M_D$; (c) $C - M_G$; (d) $C - M_{S(0)}$; (e) $C - M_{S(1)}$; (f) $C - M_{S(2)}$. (*The plot for* $C - M_C = C - M_B$ *is indistinguishable from* $C - M_G$ *on this scale.*)

$\nabla \cdot (a\nabla u) = f$, where $a = 1 + b \tan^{-1}(x - \frac{1}{2}) + c \tan^{-1}(d(y - \frac{1}{2}))$, and with all of the tilde-quantities in (2.9) obtained from the subdomain averaging of the x- and y-direction diffusion coefficients in the opposite direction to achieve separability within subdomains. Estimated condition numbers for this problem, using PMM with interfacial preconditioners from both the optimal and MSC classes appear in Table 7 of § 3.2. Observe the sensitivity of the higher-order MSC preconditioners to the variation of a along the interface, which is lacking in the optimal preconditioners. The plots are for $b = 0.65$, $c = 0.35$, $d = 10.0$.

The MSC preconditioners have the advantage of being self-scaling, relative to the $\tilde{A}_{ij}$, and can enjoy a decisive advantage over a poorly scaled optimal interface preconditioner in the PMM context. However, it is not difficult to choose a good scaling for the optimal preconditioners (see [5]).

2.5. Methods of variational type. Glowinski and several coauthors have contributed extensively to the literature of domain decomposition techniques in the course of their work on the numerical modeling of steady, compressible inviscid flow and of unsteady, incompressible viscous flow (see [19] and the references therein). Both of these subrealms of the Navier-Stokes equations may be operator-split and discretized in such a way that the subtasks of greatest computational complexity are scalar problems of Poisson type with a large number of degrees of freedom and wide variability in the coefficients in irregular geometry. We briefly describe here three algorithms of conjugate gradient type presented in [13], [19] which enter into the comparisons in § 3. Like the Schur complement methods, all of these methods iterate on degrees of freedom at the boundaries of the subdomains only, with each iteration requiring one or two exact interior solves per subdomain. Two of the methods use a partitioning of the domain into subdomains with nonintersecting interiors, as in Figs. 1 and 2. The remaining method is related to the Schwartz alternating procedure in that the subdomains overlap in regions of nonzero measure.

The algorithms are summarized below in matrix operator form, without derivation, for the model Poisson problem (2.1b). It suffices to identify the vectors in Algorithm PCG in terms of the physical variables and to specify the operators $\mathcal{A}$ and $\mathcal{B}^{-1}$.

The first two algorithms are based on the equivalence of

$$-\Delta y = f \quad \text{in } \Omega, \qquad y = 0 \quad \text{on } \partial\Omega,$$

and the problem

$$y = \arg \min_{z \in H_0^1(\Omega)} \left\{ \frac{1}{2} \int_\Omega |\nabla z|^2 \, d\mathbf{x} - \int_\Omega fz \, d\mathbf{x} \right\}.$$

Upon partitioning the domain and carrying out the minimization on each of the subdomains separately, either the normal derivative or the trace of the solution along the interface can be regarded as the principal unknown, and its value iteratively updated until continuity of the other is satisfied. Through intermediate saddle-point formulations dual conjugate gradient algorithms are derived.

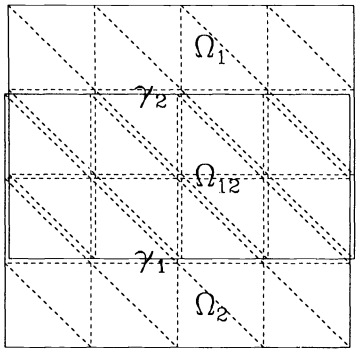


FIG. 5. *Sample domain illustrating the overlap decomposition (subdomains staggered for clarity).*

In the first algorithm ([13, § 4.1] or [19, § 2.2.2]), the unknown vector represents the discrete normal derivative of the solution along the interface adjusted for sign, $x=(-1)^{k-1}\partial u_k/\partial n_k$, $\mathscr{A}$ is the matrix $(C^{(1)})^{-1}+(C^{(2)})^{-1}$, and $\mathscr{B}$ is the matrix defined by (for $k=1$ or $k=2$)

$$[\mathscr{B}]_{ij}=h^{-1}\int_{\gamma_{12}}\psi_{kj}\psi_{ki}\,ds, \qquad i,j\in N_0.$$

In the uniform mesh case, $\mathscr{B}=I-(1/6)K$. This method we denote by Saddle-1.

In the second algorithm ([13, § 4.2]), denoted Saddle-2, the unknown vector represents the solution itself at the interface nodes, $\mathscr{A}$ is precisely the Schur complement C, and $\mathscr{B}$ is the matrix defined by

$$[\mathscr{B}]_{ij}=\sum_{k=1}^{2}\int_{\Omega_k}\nabla\psi_{kj}\cdot\nabla\psi_{ki}\,d\mathbf{x}, \qquad i,j\in N_0.$$

In the uniform mesh case $\mathscr{B}=2(I+\frac{1}{2}K)$. These preconditioners commute with their respective $\mathscr{A}$-matrices, the spectra of which can be constructed from the eigendecompositions given in § 2.2. It is readily verified that in both cases $\kappa(\mathscr{B}^{-1}\mathscr{A})$ grows asymptotically in proportion to $1/h$.

The other method ([19, § 2.3.5 and § 2.3.7]) uses a decomposition with overlapping subdomains, as illustrated for a two subdomain case in Fig. 5. The overlap region is denoted by Ω_{12} and distinct internal subdomain boundaries γ_1 and γ_2 are defined as shown. We denote this method by CG-Schwarz (or CGS).

The unknown vector represents the trace of the solution along $\gamma_1\cup\gamma_2$, and is partitioned accordingly, $u\equiv(u_1,u_2)^T$. For the case of the Laplacian, the algorithm is based on the equivalence of

$$-\Delta y=f \quad \text{in } \Omega, \qquad y=0 \quad \text{on } \partial\Omega,$$

and the problem

$$u\equiv(u_1,u_2)=\arg\min_{v\in V_1\times V_2}\left\{\frac{1}{2}\int_{\Omega_{12}}[|\nabla(y_2-y_1)|^2+|y_2-y_1|^2]\,d\mathbf{x}\right\}$$

where the $y_k\in H^1(\Omega_k)$ are the solutions of

$$-\Delta y_k=f_k \quad \text{in } \Omega_k,$$
$$y_k=0 \quad \text{on } \partial\Omega_k\cap\partial\Omega,$$
$$y_k=v_k \quad \text{on } \gamma_k,$$

and where V_k is the space consisting of the traces on γ_k of functions in $H^1(\Omega_k)$ which vanish on $\partial\Omega_{12}\cap\partial\Omega$, for $k=1,2$.

We introduce H^1_{12h}, the discrete subspace of $H^1(\Omega_{12})$ consisting of piecewise linear functions vanishing on $\partial\Omega_{12}\cap\partial\Omega$, but not necessarily on γ_1 or γ_2, and we denote a basis for it by $\{\chi_j\}$, $j\in N_{12}$. Let N_{01} and N_{02} be index sets for the nodes along γ_1 and γ_2, respectively.

The action of $\mathscr{A}$ on a vector $x\equiv(x_1,x_2)^T$ is implicitly defined as follows. First, the pair of subdomain Dirichlet problems with boundary data x_k on the γ_k,

$$A_k y_k=-C_k x_k,$$

is solved for the y_k, where

$$[A_k]_{ij}=\int_{\Omega_k}\nabla\psi_{kj}\cdot\nabla\psi_{ki}\,d\mathbf{x}, \qquad i,j\in N_k$$

and

$$[C_k]_{ij} = \int_{\Omega_k} \nabla\psi_{kj} \cdot \nabla\psi_{ki}\, d\mathbf{x}, \qquad i \in N_k, j \in N_{0k}.$$

Then the difference of the y_k over the region of common definition,

$$\delta y_k = y_k|_{\Omega_{12}} - y_l|_{\Omega_{12}},$$

is computed, where l is the complement of k in $\{1, 2\}$. Then another pair of subdomain Dirichlet problems with forcing due to the difference in the y_k over the overlap region,

$$A_k z_k = F_k \delta y_k,$$

is solved for the z_k, where

$$[F_k]_{ij} = \int_{\Omega_{12}} [\nabla\chi_j \cdot \nabla\psi_{ki} + \chi_j\psi_{ki}]\, d\mathbf{x}, \qquad i \in N_k, \quad j \in N_{12}.$$

Finally,

$$\mathcal{A}x = \begin{pmatrix} D_1\delta y_1 - E_1 z_1 \\ D_2\delta y_2 - E_2 z_2 \end{pmatrix}$$

where

$$[D_k]_{ij} = \int_{\Omega_{12}} [\nabla\chi_j \cdot \nabla\psi_{ki} + \chi_j\psi_{ki}]\, d\mathbf{x}, \qquad i \in N_{0k}, \quad j \in N_{12},$$

and

$$[E_k]_{ij} = \int_{\Omega_k} \nabla\psi_{kj} \cdot \nabla\psi_{ki}\, d\mathbf{x}, \qquad i \in N_{0k}, \quad j \in N_k.$$

We may consider different preconditioners, which are of the form $\mathcal{B} = \text{diag}\,(\mathcal{B}_1, \mathcal{B}_2)$. In § 2.3.5 of [19], an essentially unpreconditioned form of the algorithm is proposed in which $\mathcal{B}_k$ has the interfacial line integral form of $\mathcal{B}$ in the Saddle-1 method. In the experiments in § 3 we in fact report on $\mathcal{B}_k = I$. In § 2.3.7 the preconditioner

$$[B_k]_{ij} = \int_{\Omega_k} \nabla\psi_{kj} \cdot \nabla\psi_{ki}\, d\mathbf{x}, \qquad i, j \in N_{0k}$$

is used. Though not mentioned in [19], we also consider $\mathcal{B}_k = K^{1/2}$ in the numerical experiments. Note that, like the PMM, the overlap decomposition-based method requires two Dirichlet problems to be solved in each subdomain per iteration.

2.6. Generalization to multiple nonintersecting interfaces. The two-subdomain case may possess genuine interest for many problems of physical origin, but it is of limited interest when implementation of domain decomposition algorithms on ensemble architectures with a large number of processors is considered. The multiple subdomain case has been studied by Dryja and Proskurowski in [16] and [17], in which serial implementations making use of up to 32 subdomains have been tested, and by Chan and Resasco in [10]. The multiple strip case can be accommodated by our notation of (2.3) and (2.4) by letting u_0 represent all of the separator set unknowns, ordered by interface, and letting u_1 represent all of the interior unknowns, ordered by subdomain. If Ω is uniformly triangulated, and if there are p subdomains of equal size with n_0 interior gridpoints along each interface and m gridpoints across, then A_{00} is

a block matrix with $p-1$ diagonal blocks of size n_0, and A_{11} is a block matrix with p diagonal blocks of size $n_0 m$. Under our assumptions on L, the Schur complement (which is block tridiagonal with blocks of size n_0) is symmetric and positive definite, and the following is proved in [16]:

THEOREM 2.2. *Let the Schur complement C be defined as following* (2.4) *and above, and let $\tilde{M}_D$ be defined as the $(p-1)\times(p-1)$ block diagonal matrix with $n_0 \times n_0$ diagonal blocks M_D. Then, for all x,*

$$w\gamma_0(x, \tilde{M}_D x) \leqq (x, Cx) \leqq w^{-1}\gamma_1(x, \tilde{M}_D x) \tag{2.10}$$

where γ_0 and γ_1 are positive constants independent of h, and w is the minimum of the strip widths.

This makes $\tilde{M}_D$ an optimal preconditioner for C for a given decomposition, and similar block-diagonal extensions can be made for the other single-interface preconditioners. The analogous block preconditioner $\tilde{M}_G$ is also tested in [16]. The reference [17] considers a multiple subdomain generalization of the alternating Neumann–Dirichlet method, first presented for the two-strip case in [3]. Finally, [10] presents an exact eigendecomposition of the multiple-strip Schur complement matrix which generalizes Chan's method [8] to a multiple-strip domain-decomposed fast Poisson solver.

Note from (2.10) that the block preconditioning of C by $\tilde{M}_D$ suffers as the strips become thin, that is as the number of subdomains is increased for fixed domain geometry. In [10] this is traced to ignoring the off-diagonal blocks of C. By means of the exact eigendecomposition, these blocks may be shown to be insignificant in the limit of large aspect ratio subdomains ("thick" strips), but more and more significant in the limit of small aspect ratio subdomains. Since $w \propto p^{-1}$, the bound on the condition number from (2.10), $\gamma_1/(\gamma_0 w^2)$, increases like p^2. The inability of stripwise decompositions to accommodate large numbers of subdomains without a deterioration of convergence is a major weakness when it comes to large-scale parallel implementations. A more implicit means of handling the interfaces is required.

2.7. Generalization to intersecting interfaces. An extension to decompositions in which interfaces intersect in *vertices* or *crosspoints*, resulting in the formation of boxes instead of strips, is given by Bramble et al. in [5]; this development is in fact the focus of their paper. (The single interface version of the PMM presented in § 2.3 is a special case.) Constraints of space and focus prohibit the full development of their technique here, but a brief account of its implications for the discrete version of the problem is furnished below.

The key step is the further decomposition in function space of the discrete harmonic component of u, u^H. Recall from § 2.3 that $u = u^P + u^H$, where in each subdomain k, $u^P|_k$ vanishes on $\partial\Omega_k$ and satisfies

$$\tilde{A}_{\Omega_k}(u^P, v) = \tilde{A}_{\Omega_k}(u, v)$$

for all $v \in H^1_{0kh}$; and $u^H|_k$ satisfies $u^H = u$ on $\partial\Omega_k$ and

$$\tilde{A}_{\Omega_k}(u^H, v) = 0$$

for all $v \in H^1_{0kh}$. We now set $u^H = u^E + u^V$, where in each subdomain k, $u^V|_k$ and $u^E|_k$ each satisfy

$$\tilde{A}_{\Omega_k}(u^V, v) = \tilde{A}_{\Omega_k}(u^E, v) = 0$$

for all $v \in H^1_{0kh}$, and u^V is linear along the edges of $\partial\Omega_k$ and agrees with u at the vertices, and u^E vanishes at the vertices. A finite element basis for u is then constructed

which consists of the usual C^0 piecewise linear basis functions of smallest possible support defined at the nodes interior to each Ω_k and at the nodes along each edge (except at the intersections of the edges), and a special set of basis functions defined at the crosspoints with support extending out along the edges which connect the crosspoints. These special basis functions each vanish at all crosspoints but one and at all interior nodes, and are linear along the edges. The Galerkin equations involving the crosspoint degrees of freedom thus possess a nonlocal connectedness. If the nodal values of u are permuted as before so that u_0 represents the separator nodes (edges *and* crosspoints), and u_1 the interior nodes, then the computation of u_1^P and u_1^H proceed as before, as independent Dirichlet solves over each subdomain. However, the calculation of u_0 from

$$Mu_0 = f_0 - \tilde{A}_{01} u_1^P$$

proceeds in another series of independent solves as follows. M is a block diagonal matrix with a diagonal block corresponding to the nodes along each edge. These blocks are each essentially matrices of the form M_D, leaving scaling considerations aside. The final block corresponds to a diagonally dominant difference equation for the internal crosspoint nodes and has a sparsity structure identical to a graph of the decomposition: each edge connecting one vertex i to another vertex j contributes a nonzero to the ith row in column j. For a decomposition of the unit square into uniform square subdomains, this matrix is simply the discrete Laplacian, to within a scaling factor. The right-hand side involves inner products between the interfacial nodal coefficient vectors. The reader is referred to [5] for the complete details. We conclude by paraphrasing Theorem 1 and Remark 2.6 therein as follows, in which it is assumed that the underlying triangulation and decomposition into subdomains are quasi-uniform of sizes h and d, respectively.

THEOREM 2.3. *Let the matrix A be defined as in* (2.3) *and above, and let $\tilde{B}$ be defined as in* (2.7), *where M is the separator set preconditioner of* [5]. *Then, for all x,*

$$\gamma_0(x, \tilde{B}x) \leqq (x, Ax) \leqq \gamma_1(x, \tilde{B}x) \tag{2.11}$$

where γ_0 and γ_1 are positive constants such that for some positive constant c_1,

$$\frac{\gamma_1}{\gamma_0} \leqq c_1\left(1 + \ln\left(\frac{d}{h}\right)^2\right).$$

If, instead, the vertex difference equation block of M is replaced by a weighted identity operator then for some positive constant c_2,

$$\frac{\gamma_1}{\gamma_0} \leqq c_2 d^{-2}\left(1 + \ln\left(\frac{d}{h}\right)^2\right).$$

The implications of this theorem for the many subdomain decomposition are illustrated by the following special case. Consider a square domain uniformly discretized with n subintervals on a side and uniformly decomposed into p square subdomains of $n/\sqrt{p}$ subintervals on a side. Then $d \propto \sqrt{p}$ and $d/h = n/\sqrt{p}$. Therefore, by Theorem 2.3, when the crosspoints are treated implicitly the condition number is bounded by $c_1(1+\ln(n/\sqrt{p})^2)$; and when they are decoupled the condition number is bounded by a $c_2 p(1+\ln(n/\sqrt{p})^2)$. In the former case the condition number is bounded by a constant if subdomains are introduced as required with increasing resolution to keep the number of subintervals on a subdomain side fixed. In the latter case, the condition number grows in proportion to the number of subdomains.

A simpler alternative for handling the crosspoints in the decoupled case is to employ the usual cardinal finite element basis for all of the interior degrees of freedom, including the crosspoints, along with a weighted identity operator for the corresponding block of M. This method requires no special computations to form the right-hand side of the crosspoint system. The examples of § 3 labeled "PMM without vertex coupling" employ this simpler alternative.

3. Experimental comparisons of domain decomposition techniques. The methods described in § 2 were tested on a variety of model problems chosen to reveal their relative strengths and weaknesses. Four of the problems we consider are posed on the unit square, symmetrically divided into two strips or four boxes by straight segments bisecting opposite sides. The operators include the cases of uniform, discontinuous, and smoothly varying nonseparable coefficients. The other problems are posed in a small aspect ratio rectangle, and in a T-shaped region.

These tests were carried out on a VAX/785 in single precision (24-bit mantissa) using the experimental interpretive language CLAM (Conversational Linear Algebra Machine) [21]. By providing a convenient symbolic interface to the LINPACK and EISPACK libraries, CLAM allowed relatively quick "breadboarding" of the various algorithms. The tests are not as comprehensive as some of those available in the literature for individual problems and methods (e.g., [2], [5], [16], [17]) because of size, speed, and precision limitations, nor do they embrace as wide a scope of domain geometries or operators. The chief value of these results is tutorial, in that they permit algorithmic comparison on a common set of problems from common initial iterates with common convergence criteria and measures, all of which may vary or be left unstated from paper to paper.

3.1. Conventions in conducting tests and reporting results. To complete the specification of Algorithm PCG of § 2.1, which is the basis for all of the experiments, the initial iterate is always taken to be zero everywhere, and convergence is based on the relative size of the unnormalized Euclidean norm of the true residual. Since these practices are not uniform in the testing of such algorithms, we comment briefly upon them. In practical applications of PCG, the most conveniently monitored convergence criterion is the norm of the preconditioned residual, $(r, \mathcal{B}^{-1}r)^{1/2}$, since the square of this quantity is already required to advance the algorithm. However, when comparisons across a variety of preconditioners $\mathcal{B}$ are carried out, the convergence criterion should not be affected by the type of preconditioner. Therefore the norm of the unpreconditioned residual was calculated at every iteration of the serial tests, as well. We declare convergence when $(r^k, r^k)^{1/2} < \varepsilon(r^0, r^0)^{1/2}$, as opposed to the absolute criterion used in some of the early domain decomposition literature, in an attempt to remove dependence on the resolution of the problem. In our test problems the shape of the right-hand side f is generally such that its domain average decreases by an appreciable factor as the uniform mesh is refined over several powers of two and points further from the symmetrically located maximum of f are brought in. This is reflected in a reduced initial residual, since the initial iterate is zero, and fewer iterations are required to reach an absolute residual tolerance. If anything, we should insist that an iterative method work *harder* as the resolution increases in order to realize the potential of the lower truncation error of the discretization, otherwise the additional resolution is wasted from the continuous viewpoint. As a minimum requirement, we insist that the residual be decreased by a constant factor. In all of the tables to follow, $\varepsilon = 10^{-4}$.

The principal results that we report for each experiment are the number of iterations to convergence, I, the estimated condition number of the preconditioned system, κ,

and the average reduction per iteration of the residual, ρ. These are complementary indications of the rate of convergence of the method. As the only one of these three quantities which depends solely upon the operator, the true condition number is an attractive measure. However, a large condition number may lead to an overly pessimistic appraisal of a method if the eigenvalues are unevenly distributed, and an exact condition number is expensive to compute. The iteration count is the most useful "bottom line" convergence measure for use in conjunction with cost-per-iteration complexity estimates to determine overall algorithmic complexity. However, this is convergence criterion-dependent. The average rate of reduction per iteration, defined by $((r^I, r^I)/(r^0, r^0))^{1/(2I)}$, where I is the number of iterations to convergence, is a more reliable estimator of the convergence rate than the condition number, without being as strongly dependent upon the particular convergence criterion as I, and it does not exhibit the threshold effect that I does. However, ρ still retains a dependence on the initial iterate. Since none of these three measures is ideal, we report all of them.

The condition number estimate in all of our tables is obtained as a by-product of the PCG iterations by the method of Lanczos [27], as implemented in [28], at a small computational overhead. For each k in the loop (PCG.6)-(PCG.14), the diagonal and subdiagonal elements of a symmetric tridiagonal matrix are formed according to

$$d_k = \frac{1}{\alpha^k} + \frac{\beta^{k-1}}{\alpha^{k-1}}, \qquad s_{k+1} = -\frac{\sqrt{\beta^k}}{\alpha^k},$$

respectively. The extreme eigenvalues of this matrix often accurately approximate those of $\mathcal{B}^{-1}\mathcal{A}$ even if terminated with k considerably less than n. We have checked these estimates against the true condition numbers for the $h^{-1} = 16$ row of the examples in Tables 1 and 7. Very close agreement obtains in Table 1, where at least the first two significant figures of κ_{true} and κ_{Lanczos} agree for all of the methods. In Table 7 the Lanczos-derived estimates shown overestimate the true condition number by 11-46%.

Finally, we note that although the finite element formulation has been chosen throughout this article in order to provide a uniform theoretical background for the different methods, the results in Tables 5, 6, and 7 were actually generated using the standard second-order finite difference method to construct the discrete A and $\tilde{A}$. (These alternative formulations do not generally coincide at the discrete level when the operator is not piecewise constant.)

3.2. Stripwise decomposition tests. The first test problem is Poisson's equation

$$\nabla^2 u = f,$$

in the unit square divided symmetrically into two strips, where f is chosen so that $u = 16xy(1-x)(1-y)$. This problem is essentially the same as the first example in [16], [17]. It may be solved conveniently by an SCM. Surface plot comparisons of the preconditions in Table 1 with C were given in Fig. 3.

For this domain geometry, and for all but one to follow, the Schur complement matrix preconditioners M_C and M_B are identical, so there is no need for separate tabulation. The optimality properties of M_C, M_G, and M_D are in evidence, with the number of iterations independent of problem size. As the mesh is refined, ρ actually shows a slight improvement for M_G and M_D, but this is not regarded as significant. The MSC preconditioners are inferior, and exhibit steadily deteriorating ρ. The last

TABLE 1

$\nabla^2 u = f$ on the unit square, divided symmetrically into two strips. Results for SCM *as a function of problem size and interface preconditioner.*

h^{-1}		M_C	M_G	M_D	$M_{S(0)}$	$M_{S(1)}$	$M_{S(2)}$	I
16	I	1	2	3	5	5	4	8
	κ	1.00	1.09	1.30	1.76	1.49	1.30	13.1
	ρ	3.7 (−6)	9.0 (−4)	3.2 (−2)	1.5 (−1)	8.8 (−2)	4.2 (−2)	1.1 (−1)
32	I	1	2	3	7	6	6	12
	κ	1.00	1.09	1.32	2.47	2.07	1.76	26.1
	ρ	1.4 (−5)	8.0 (−4)	2.7 (−2)	2.4 (−1)	2.0 (−1)	1.4 (−1)	4.4 (−1)
64	I	1	2	3	9	8	7	17
	κ	1.00	1.09	1.34	3.52	2.94	2.48	52.4
	ρ	8.4 (−5)	8.0 (−4)	2.2 (−2)	3.3 (−1)	3.0 (−1)	2.5 (−1)	5.7 (−1)

TABLE 2

$\nabla \cdot (a \nabla u) = 0$ on the unit square, divided symmetrically into two strips, where $a = 1.0$ in one subdomain and 0.1 *in the other. Results for* SCM *as a function of problem size and interface preconditioner.*

h^{-1}		M_C	M_G	M_D	$M_{S(0)}$	$M_{S(1)}$	$M_{S(2)}$
16	I	1	3	4	5	4	4
	κ	1.00	1.09	1.35	1.76	1.48	1.29
	ρ	4.3 (−6)	2.8 (−3)	8.7 (−2)	1.3 (−1)	9.4 (−2)	4.8 (−2)
32	I	1	2	4	6	6	5
	κ	1.00	1.09	1.36	2.47	2.07	1.76
	ρ	9.0 (−6)	9.0 (−4)	9.1 (−2)	2.0 (−1)	1.6 (−1)	1.2 (−1)
64	I	1	2	4	7	7	6
	κ	1.00	1.09	1.37	3.28	2.93	2.48
	ρ	5.0 (−5)	7.9 (−3)	9.3 (−2)	2.5 (−1)	2.3 (−1)	1.9 (−1)

TABLE 3

$\nabla^2 u = f$ on a low aspect ratio rectangle, divided asymmetrically into two strips. Results for SCM *as a function of problem size and interface preconditioner.*

h^{-1}		M_C	M_B	M_G	M_D	$M_{S(0)}$	$M_{S(1)}$	$M_{S(2)}$
32	I	1	3	3	3	5	4	4
	κ	1.00	1.27	2.10	2.00	1.63	1.45	1.28
	ρ	7.6 (−6)	1.6 (−2)	2.2 (−2)	3.3 (−2)	1.1 (−1)	9.4 (−2)	6.6 (−2)
64	I	1	3	3	3	6	6	5
	κ	1.00	1.27	2.09	2.05	2.24	1.96	1.64
	ρ	4.3 (−5)	1.6 (−2)	2.2 (−2)	2.6 (−2)	1.9 (−1)	1.7 (−1)	1.3 (−1)

TABLE 4

$\nabla^2 u = f$ on an asymmetric T-shaped region, divided into two strips. Results for SCM *as a function of problem size and interface preconditioner.*

h^{-1}		M_C	M_G	M_D	$M_{S(0)}$	$M_{S(1)}$	$M_{S(2)}$
16	I	3	2	4	4	4	3
	κ	1.08	1.07	1.43	1.41	1.21	1.12
	ρ	9.5 (−3)	9.9 (−3)	9.3 (−2)	7.4 (−1)	5.2 (−2)	2.4 (−2)
32	I	3	3	5	5	5	4
	κ	1.11	1.11	1.47	1.72	1.60	1.38
	ρ	1.7 (−2)	1.9 (−2)	1.1 (−1)	1.2 (−1)	1.0 (−1)	7.3 (−2)
64	I	3	3	5	6	6	5
	κ	1.14	1.14	1.49	2.49	2.26	1.91
	ρ	3.3 (−2)	3.3 (−2)	1.1 (−1)	2.2 (−1)	1.9 (−1)	1.5 (−1)

TABLE 5

$\nabla \cdot (\underline{a} \nabla u) + ku = f$ on the unit square, divided symmetrically into two strips, where $a_{11} = e^{xy}$, $a_{22} = e^{-xy}$, $a_{12} = a_{21} = 0$, and $K = (1+x+y)^{-1}$, with subdomain preconditioning by the Laplacian. Results for PMM *as a function of problem size and interface preconditioner and for* GMM *as a function of problem size.*

h^{-1}		M_C	M_G	M_D	$M_{S(0)}$	$M_{S(1)}$	$M_{S(2)}$	GMM
8	I	7	7	7	7	7	7	7
	κ	2.42	2.42	2.53	2.41	2.41	2.41	2.42
	ρ	2.2 (−1)	2.2 (−1)	2.4 (−1)	2.3 (−1)	2.2 (−1)	2.2 (−1)	2.2 (−1)
16	I	8	8	8	8	8	8	8
	κ	3.25	3.24	3.23	3.15	3.24	3.24	3.25
	ρ	3.0 (−1)	3.1 (−1)	3.1 (−1)	3.2 (−1)	3.1 (−1)	3.1 (−1)	3.0 (−1)
32	I	9	9	10	10	10	10	9
	κ	3.92	3.89	4.04	4.57	3.99	4.03	3.92
	ρ	3.6 (−1)	3.6 (−1)	3.6 (−1)	4.0 (−1)	3.8 (−1)	3.7 (−1)	3.6 (−1)

TABLE 6

$\nabla \cdot (\underline{a} \nabla u) + ku = f$ on the unit square, divided symmetrically into two strips, where $a_{11} = e^{xy}$, $a_{22} = e^{-xy}$, $a_{12} = a_{21} = 0$, and $K = (1+x+y)^{-1}$, with subdomain preconditioning by a locally averaged separable variable-coefficient operator. Results for PMM *as a function of problem size and interface preconditioner and for* GMM *as a function of problem size.*

h^{-1}		M_C	M_G	M_D	$M_{S(0)}$	$M_{S(1)}$	$M_{S(2)}$	GMM
8	I	5	5	6	5	5	4	5
	κ	1.47	1.47	1.98	1.46	1.37	1.31	1.52
	ρ	1.1 (−1)	1.2 (−1)	1.8 (−1)	1.2 (−1)	9.9 (−2)	8.6 (−2)	1.2 (−1)
16	I	5	5	6	6	6	5	5
	κ	1.52	1.52	2.09	1.90	1.71	1.54	1.66
	ρ	1.3 (−1)	1.3 (−1)	1.9 (−1)	2.0 (−1)	1.5 (−1)	1.3 (−1)	1.5 (−1)
32	I	5	5	6	8	7	7	6
	κ	1.52	1.51	2.14	2.64	2.33	2.06	1.81
	ρ	1.4 (−1)	1.4 (−1)	1.9 (−1)	2.9 (−1)	2.6 (−1)	2.0 (−1)	1.8 (−1)

TABLE 7

$\nabla \cdot (a\nabla u) = f$ on the unit square, divided symmetrically into two strips, where $a = 1 + b \tan^{-1}(x - \frac{1}{2}) + c \tan^{-1}(d(y - \frac{1}{2}))$, with subdomain preconditioning by a locally averaged separable variable-coefficient operator. Results for PMM *as a function of problem size and interface preconditioner and for* GMM *as a function of problem size.*

h^{-1}		M_C	M_G	M_D	$M_{S(0)}$	$M_{S(1)}$	$M_{S(2)}$	GMM
8	I	6	6	7	6	5	5	7
	κ	1.87	1.89	2.41	2.00	1.79	1.81	2.79
	ρ	1.7 (−1)	1.7 (−1)	2.2 (−1)	1.7 (−1)	1.5 (−2)	1.5 (−2)	2.5 (−1)
16	I	6	6	7	7	6	6	8
	κ	1.94	1.97	2.59	2.69	2.31	2.09	3.47
	ρ	2.0 (−1)	2.0 (−1)	2.5 (−1)	2.6 (−1)	2.1 (−1)	1.8 (−1)	2.9 (−1)
32	I	7	7	7	9	8	7	9
	κ	2.10	2.12	2.62	3.80	3.25	2.78	3.85
	ρ	2.2 (−1)	2.2 (−1)	2.6 (−1)	3.6 (−1)	3.1 (−1)	2.7 (−1)	3.3 (−1)

column, for which $\mathcal{B} = I$, shows the proportionality of the condition number to the number of degrees of freedom in the unpreconditioned system.

The second problem involves a physical interface which coincides with the subdomain boundary. The equation is:

$$\nabla \cdot (a\nabla u) = 0$$

where the diffusion coefficient a is 1.0 in one of the subdomains and 0.1 in the other, and where u obeys the Dirichlet condition $u = xy$ on the boundary of the unit square. By partitioning the domain so that each subdomain has a uniform diffusion coefficient, fast solvers may be exploited. Again, an SCM is appropriate for this problem, the results of which are shown in Table 2.

The optimal methods perform well in spite of the jump discontinuity. (The decrease in the number of iterations required with M_G between $h^{-1} = 16$ and 32 is not regarded as significant in this sub-asymptotic range.) The MSC methods fare slightly better than in the featureless case, but still cannot compete with the optimal methods on so ideal a problem.

The third problem is again Poisson's equation, a compressed version of the first problem posed in the rectangle $(0, 1) \times (0, 3/8)$, with an interface at $y = \frac{1}{4}$ and f chosen so that $u = (1024/9)x(1-x)y(3/8-y)$. At $h^{-1} = 32$, the upper subdomain is only three interior grid points high. Table 3 contains the results.

The subdomain aspect ratios in this problem are more typical of those in multistrip decompositions of domains of roughly unit aspect ratio. This is the only problem to exhibit a distinction between M_C, which is again exact, and M_B.

As our only example of a geometry which is unsuitable for an undecomposed fast Poisson solver, we consider the asymmetric T-shaped domain problem from [2]. The domain is the unit square with two rectangles removed, as shown to scale in Fig. 1(b). The equation is Poisson's equation with inhomogeneous Dirichlet boundary data and right-hand side chosen so that $u = x^2 + y^2 - x\,e^x \cos y$. Though the overall geometry is nonseparable, the subdomains can each be handled by a fast Poisson solver, so an SCM is appropriate, as shown in Table 4.

The optimal methods have no trouble handling geometrical irregularity such as this. The MSC methods are more competitive than before due to the spoiling of

symmetry which prevents the use of any exact preconditioning, but they continue to be inferior for sufficiently fine resolution.

The fifth problem involves generalization in another direction: a nonseparable operator. We consider ELLPACK problem #1 [34]:

$$\nabla \cdot (\underline{a} \nabla u) + Ku = f$$

where $a_{11} = e^{xy}$, $a_{22} = e^{-xy}$, $a_{12} = a_{21} = 0$, $K = -(1+x+y)^{-1}$, and where f is chosen so that $u = \frac{3}{4} e^{xy} \sin(\pi x) \sin(\pi y)$.

For this problem, we use a PMM and present results for two different forms of the partitioned matrix preconditioner $\mathcal{B}$ in two separate tables. The condition number estimates pertain to $\kappa(\tilde{B}^{-1}A)$, where $\tilde{B}$ is constructed according to (2.7), by using the different M-blocks shown at the head of the table columns. Results for the preconditioning arising from taking $\tilde{A}$ of (2.7) to be the Laplacian operator are shown in Table 5. (CLAM's capacity is exceeded by our modular implementation of the PMM at $h^{-1} = 64$, so results for the very coarse problems with $h^{-1} = 8$ are added to Tables 5, 6, and 7 in order to provide a third stage for comparison.) The Laplacian is a rather coarse-grained preconditioner, which does not take advantage of the sensitivity of the domain decomposition technique to local operator properties. We also consider a more conventional use of the PCG technique, the GMM of preconditioning a nonseparable problem by one which can be implemented by a fast Poisson solver over the entire domain. The results, displayed in the last column, are identical to those of M_C, which is, of course, the decomposed equivalent of the Laplacian.

The problem is a difficult one for all of the methods, as evidenced by the fact that for the coarsest mesh case, the number of iterations for all methods is fully equal to the number of unknowns on the interface. (Of course, we are now iterating on vectors with length $O(h^{-2})$, but recall that for the first four problems considered, the PMM and SCM implementations have identical iteration counts much lower than the number of interfacial unknowns.) For problems of this difficulty, the MSC preconditioners do not fare appreciably worse than the optimal ones, and achieve comparable condition numbers even at the moderate resolutions investigated. However, the performance of the MSC preconditioners deteriorates with increasing resolution more rapidly than the optimal ones by any of the three measures in the tables. Furthermore, in comparing iteration counts, it should be recalled that the construction of $M_{S(k)}$ requires $k+1$ solves per subdomain, the arithmetic complexity of which is roughly equivalent to that of $k+1$ extra iterations which make no appearance in the tables.

Table 6 shows the improvement possible by generating the preconditioner blocks $\tilde{A}_k$ by subdomain averaging the original operator A, giving a PMM finer granularity than a GMM. Let $\langle a_{11} \rangle_y(x)$ and $\langle K \rangle_y(x)$ denote subdomain averages of the respective coefficients with respect to y and $\langle a_{22} \rangle_x(y)$ and $\langle K \rangle_x(y)$ the same with respect to x. In each subdomain k, $\tilde{A}_k$ has the separable form

$$[\tilde{A}_k]_{ij} = \int_{\Omega_k} \left(\langle a_{11} \rangle_y \frac{\partial \psi_j}{\partial x} \frac{\partial \psi_i}{\partial x} + \frac{1}{2} \langle K \rangle_y \psi_j \psi_i + \langle a_{22} \rangle_x \frac{\partial \psi_j}{\partial y} \frac{\partial \psi_i}{\partial y} + \frac{1}{2} \langle K \rangle_x \psi_j \psi_i \right) d\mathbf{x}.$$

With the local averaging, the best of the optimal PMMs begin to break away from the GMM even though only two subdomains are employed. A more graphic illustration of the improvements offered by the PMM in the many subdomain case is given in Example 3 of [5].

Our final stripwise decomposition example is

$$\nabla \cdot (a \nabla u) = f$$

on the unit square, where $a = 1 + b \tan^{-1}(x - \frac{1}{2}) + c \tan^{-1}(d(y - \frac{1}{2}))$, using the values for b, c, and d listed in § 2.4, where surface plots of the Schur complement matrix and various approximations thereto were given. In this problem there is a significant difference in the subdomain averages of a, which would tend to favor the PMMs over a GMM, and a significant variation of a along the interface, $y = \frac{1}{2}$, which would tend to favor MSC preconditioning.

A separable subdomain-averaged PMM preconditioner was constructed in the manner described above. In this example, the results of which appear in Table 7, all of the PMMs are better than or comparable to the GMM; however, the MSC advantage begins to disappear even at fairly coarse resolution.

3.3. Boxwise decomposition tests. We carried out serial tests of the two decomposition strategies involving crosspoints on the first problem of the previous subsection, namely Poisson's equation on the unit square. Table 8 contains the results for the uncoupled crosspoint version of the algorithm, and Table 9 contains the results for the coupled crosspoint version. Since there is only one crosspoint in this example, the two algorithms differ only in the assembling of the right-hand side of the crosspoint equation, as described in § 2.7.

TABLE 8

$\nabla^2 u = f$ on the unit square, divided into four equal boxes. Results for PMM *without vertex coupling, as a function of problem size and interface preconditioner.*

h^{-1}		M_C	M_G	M_D	$M_{S(0)}$	$M_{S(1)}$	$M_{S(2)}$
16	I	5	5	6	9	7	8
	κ	8.60	8.59	8.89	12.8	12.5	12.1
	ρ	6.9 (−2)	6.9 (−2)	2.0 (−1)	2.7 (−1)	2.4 (−1)	2.2 (−1)
32	I	6	6	7	10	10	10
	κ	12.4	12.4	12.1	24.4	22.7	21.9
	ρ	1.1 (−1)	1.1 (−1)	2.2 (−1)	4.4 (−1)	3.3 (−1)	4.4 (−1)
32	I	6	6	7	15	13	13
	κ	15.1	15.1	15.9	45.4	40.4	37.0
	ρ	2.0 (−1)	2.0 (−1)	2.4 (−1)	5.3 (−1)	4.8 (−1)	4.8 (−1)

TABLE 9

$\nabla^2 u = f$ on the unit square, divided into four equal boxes. Results for PMM *with vertex coupling after Bramble et al., as a function of problem size and interface preconditioner.*

h^{-1}		M_C	M_G	M_D	$M_{S(0)}$	$M_{S(1)}$	$M_{S(2)}$
16	I	5	5	6	8	7	7
	κ	6.38	6.38	5.77	10.4	9.96	9.52
	ρ	6.7 (−2)	6.8 (−2)	1.9 (−1)	2.8 (−1)	1.9 (−1)	2.4 (−1)
32	I	5	5	6	11	9	11
	κ	7.13	7.23	8.20	19.8	18.3	16.7
	ρ	1.2 (−1)	1.2 (−1)	2.1 (−1)	3.9 (−1)	3.2 (−1)	3.8 (−1)
64	I	6	7	7	15	13	12
	κ	13.4	13.4	12.7	36.5	32.4	29.5
	ρ	2.1 (−1)	1.5 (−1)	2.2 (−1)	5.0 (−1)	4.6 (−1)	4.6 (−1)

Even though there is only a single crosspoint, the difference between employing local and nonlocal basis functions for the crosspoint degree of freedom is evident in comparing the tables. The condition numbers are uniformly smaller in the nonlocal case, though the overall number of iterations is not greatly affected. The growth of condition number for Dryja's interface preconditioner with crosspoint coupling (the third column of Table 9) follows the theoretical bound $c_1(1+\ln(n/\sqrt{p})^2)$ with a c_1 of approximately 1.

A word is in order concerning the construction of the MSC preconditioners in the case of multiple interfaces (four in these examples). We require that M be block-diagonal as with the other preconditioners, to preserve the independence of the interfacial solves. In order to avoid solving the Dirichlet problems on each subdomain for each degree k of approximation in MSC(k), we economize by varying the components of the vectors v_i of (2.9b) on all interfaces simultaneously, rather than by treating the interfaces one-by-one. The algorithm described in § 2.4 is then applied block-by-block to generate the coefficients of the interfacial blocks of $M_{S(k)}$. The vectors v_i always have zeros corresponding to the crosspoint location and the crosspoint block is unaffected by the choice of interfacial blocks. As in the case of Poisson's equation on two strips, the MSC methods cannot compete with the optimal methods.

3.4. Variational methods tests. The methods of § 2.5 were applied to the model problem of Table 1, with the results shown in Table 10. A two-strip decomposition was used for the saddle-point methods. A decomposition of the type pictured in Fig. 5 was employed for the CG-Schwarz methods. The overlap region was centered on the horizontal bisector of the square and its width was one-eighth of the length of a side. This geometrical width was held constant as the mesh was refined, thus the discrete size of the overlap region increased. Numerical studies of the convergence of Schwarz-type methods which we do not report in detail here show this to be an important consideration. Fixing the discrete size of the overlap region instead, thus allowing the geometrical width to shrink with increasing resolution, causes substantial deterioration of convergence. For instance, the growth of the condition number for the unpreconditioned system CGS/I is superlinear in h^{-1} in this case, as opposed to the linear behavior of κ with h^{-1} in the third column of Table 10.

We note that the $K^{1/2}$ preconditioning is very effective on the CG-Schwarz method, the spectrum of the discrete operator $\mathcal{A}$ of which has not been analyzed theoretically to date. However, it is apparent by comparison with Table 1 that on such simple

TABLE 10

$\nabla^2 u = f$ on the unit square, divided symmetrically into two strips, with and without overlap. Results as a function of problem size and method/preconditioning combination.

h^{-1}		Saddle-1	Saddle-2	CGS/I	CGS/$I+\frac{1}{2}K$	CGS/$K^{1/2}$
16	I	4	5	6	4	3
	κ	5.47	4.41	21.5	7.21	1.89
	ρ	8.4 (−2)	1.1 (−1)	1.9 (−1)	4.6 (−2)	1.1 (−2)
32	I	7	7	9	5	2
	κ	10.6	8.65	44.4	14.1	1.49
	ρ	2.1 (−1)	2.3 (−1)	3.2 (−1)	1.5 (−1)	7.0 (−3)
64	I	9	10	12	7	2
	κ	20.5	17.3	89.9	29.0	1.47
	ρ	3.3 (−1)	3.7 (−1)	4.6 (−1)	2.6 (−1)	4.4 (−3)

problems these methods are inferior to the SCM with optimal preconditioning. It should be recalled in comparing iteration counts across these methods that the CGS method requires twice as many subdomain solves per iteration as the SCM.

4. Parallel experiments with domain decomposition algorithms. In order to test the convergence of the methods on a larger number of subdomains and to test the degree of parallelization that is possible in practice, two PMM methods, with and without vertex coupling and with M_D as the interfacial preconditioner, were programmed for the Intel Hypercube. For simplicity and economy of coefficient storage, we considered only Poisson's equation on the unit square. For programming convenience, efficient use of the processors and economy of data transfer between nodes, we wrote a single program to run on each of the nodes, rather than writing special purpose programs to do the major identifiable operations in Algorithm PCG, such as interior solves or edge solves. The same program was used to run both strips and boxes, with different compile-time constants to reshape the arrays. The number of processors was left dynamic.

4.1. Domain-to-processor mapping and dataflow analysis. The square domain is uniformly discretized with n subintervals on a side and divided into p subdomains in one of two ways. In the case of domains consisting of strips, each domain is an $n \times n/p$ rectangle. In the case of boxes, each domain is an $n/\sqrt{p} \times n/\sqrt{p}$ square. For simplicity and efficiency we require n and p to be powers of 2. By restricting ourselves to this limiting case of decomposition regularity, we are able to pass over the complex issue of load balancing. The work per processor is balanced a priori and does not change during the course of the solution process in response to any adaptive mechanism. It should be noted that regular decompositions of this type are the exception in applications. The tradeoffs between load balancing efficiency and communication efficiency in multiprocessor implementations of adaptive algorithms for partial differential equations have begun to receive attention elsewhere [4], [22]. We note however, that some geometrical adaptivity can be accommodated within the context of *logically* uniform decompositions by means of generalized tensor-product gridding (see, e.g., the examples in [5]).

Four types of nodes are to be distinguished in the decomposition, for purposes of keeping track of the data flow: interior nodes, separator set nodes, nodes which are adjacent to a separator, and nodes which are adjacent to an exterior boundary. For the strip decomposition, we associate the gridpoints along each interface with one of the two subdomains it separates so that each processor (except for one) is assigned one of the interfaces of length n. For definiteness in what follows, let the strips run north-south, and let the east face of each subdomain contain the separator nodes, the west face containing nodes which are adjacent to a separator, with the obvious exceptions of the west-most and east-most subdomains. For the box decomposition, we associate the gridpoints along each interface with one of the two subdomains it separates and each vertex with one of the four subdomains it separates so that each processor (except for those along two of the domain boundaries) is assigned two interfaces of length $n\sqrt{p}$ and one crosspoint. For definiteness, let the east face and south faces of each subdomain contain the separator nodes, let the southeast corner contain the crosspoint, and let the west and north faces contain nodes which are adjacent to a separator, again with the obvious boundary exceptions. We map p strips onto the hypercube by labeling the strips from one end of the domain to the other in binary reflected Gray code order [25]. We map p boxes onto the hypercube by forming the tensor-product of two such rings, along the row and column directions of the

subdomain "grid." In this way all of the local exchanges in the physical domain are also local exchanges between processors, any two of which are connected if and only if their binary local representations differ in exactly one bit.

We now analyze the data flow in each of the major steps in Algorithm PCG. We concentrate on the boxwise decomposition, since strips may be regarded as a special case thereof.

The operation (PCG.7) of applying the distributed $\mathcal{A}$ matrix consists of: (1) sending the vector of data along each processor boundary not adjacent to a physical boundary to the neighboring processor; (2) computing the interior components of the product from the 5-point star formula; (3) receiving the vector of data coming across each boundary and computing the boundary components of the product. Note that it is not necessary to distinguish between separator nodes and those adjacent to a separator in this step.

The operation (PCG.11) of solving with $\mathcal{B}$ consists of: (1) solving for u^P in the subdomain interior (the subdomain interior includes the processor boundary nodes that are adjacent to a separator); (2) sending the vector of data along each north and west processor boundary not adjacent to a physical boundary to the neighboring processor; (3) receiving the vector of data coming across each south and east processor boundary and forming the right-hand side of the edge separator system and the local summands of the right-hand side of the crosspoint system; (4) sending the appropriate scalar summands to the processors to the north and west, if any; (5) solving the south and east edge systems for u^E on the separator set; (6) receiving the summands coming from the processors to the south and east, if any, and forming the right-hand side of the crosspoint system; (7) scattering the local component of the right-hand side of the crosspoint system to all other processors, and receiving their components in turn; (8) solving an identical global crosspoint system in each processor; (9) extending the crosspoint data linearly along the separator set edges to form u^V; (10) summing u^V and u^E along the separator set to form u^H on the separator set; (11) sending the vector of u^H data along each south and east boundary not adjacent to a physical boundary to the neighboring processor; (12) receiving the vector of data coming across each north and east processor boundary and forming the right-hand side of the interior u^H system; (13) solving for u^H in the subdomain interior and summing u^H and u^P. Steps (4), (6)-(7), and (9)-(10) are unnecessary in the decoupled form of the algorithm, and step (8) is replaced by a scalar divide local to each processor containing a crosspoint in this case.

The operation of performing a dot product in (PCG.7) and (PCG.12) consists of two traversals of a binary spanning tree, arbitrarily rooted in processor "0". First, the local portion of the dot product is computed. These partial sums are then passed from leaf to root, and summed along the way. The root processor performs the final sum and sends the data back from root to leaf.

4.2. Numerical results. We present data obtained on an Intel iPSC-6 for the strip form and the two box forms of the algorithm described above. For simplicity of coefficient generation and economy of storage (as memory proved to be constraining), we considered only Laplace's equation on a square. The problem is the same as that in Tables 1, 8, and 9.

Our conventions for the tests on the iPSC differ from those in § 3 in that the preconditioned residual is used to monitor convergence, rather than the actual residual. This is the practical choice since we are not comparing different preconditioners against each other in this section, but rather carrying out larger scale tests on selected algorithms.

Since monitoring the true residual would require an extra global dot product per iteration, the reported execution times would exhibit an unnatural penalty associated with increased subdivision of the domain. The relative reduction required in this different norm was set to be the same as in § 3, namely 10^{-4}.

The notation employed in the tables follows that of § 3 except that ρ is now defined as $((r^I, \mathcal{B}^{-1}r^I)/(r^0, \mathcal{B}^{-1}r^0))^{1/(2I)}$. T is the actual execution time (computation plus communication) of the domain decomposition algorithm, which does not include the initial distribution of the problem data or the post-convergence analysis of the results. Our timings for various cube dimensions (as reflected in p, the number of processors employed in the calculation) were all obtained on a cube of uniform declared dimension (namely six) with some processors idle, in order to avoid nonuniform host-processor polling rates. Where applicable, s measures the ratio of two execution times: T for the next smaller number of processors on the same size problem divided by the current T. It is thus a local measure of speedup, and a natural one for domain decomposition algorithms, but it is not the *speedup* as it would be defined in a strict sense, since the algorithm changes with the number of processors (see [23]).

We comment briefly on the sparsity of the tables. Our program requires a minimum of 8 subintervals on a side for each subdomain. Therefore, the number of processors that can be employed on a given problem is bounded above by the resolution of the mesh. Unfortunately, the number of processors is also bounded below by the limited local memory of each node. The largest number of degrees of freedom that could be accommodated per subdomain was between 2^{11} and 2^{12}; therefore, our largest problems, involving 2^{14} or 2^{16} nodes, could not be run on small numbers of processors. These two restrictions imposed a banded structure on the region of $h-p$ parameter space in which it was feasible to run experiments.

The results of a series of runs using strips are given in Table 11. The $p=2$ column of this table is comparable to the M_D column of Table 1. Due to the different, more lenient convergence criterion, one less iteration is required in the more highly refined cases, and the Lanczos estimate of the condition number is obviously sensitive to I for I as small as 2 or 3 in this problem. Note the effect of Theorem 2.2: as the number of processors is increased while the refinement is held constant, κ goes up asymptotically in proportion to the square of the number of processors (subdomains). The number of iterations required does not behave as badly as the condition number. From the limited data, it appears to go up roughly in proportion to the number of processors, with fixed refinements. The increasing number of iterations asymptotically defeats the power of the additional processors as far as s is concerned, however, in that the local speedups do not approach 2.

The results of a series of runs using boxes without crosspoint coupling are given in Table 12. The $p=4$ column of this table may be compared to the M_D column of Table 8; again the preconditioned residual convergence criterion is more lenient, but the condition number estimates agree almost exactly. In this table d/h is constant along a diagonal; therefore by Theorem 2.3 we would expect to see asymptotically a proportionality between κ and p along a diagonal, which is approximately the case. The number of iterations required goes up sublinearly with the number of processors (subdomains), with fixed refinement. For this table and the next, an "ideal" algorithm would display local speedups s of 4.

The results of a series of runs using boxes with crosspoint coupling are given in Table 13. The $p=4$ column of this table may be compared to the M_D column of Table 9, subject to the caveats above, and the $p=16$ column shows close agreement with Table 6.2 of [5]. According to Theorem 2.3, the condition number should be bounded

TABLE 11

$\nabla^2 u = f$ in the unit square, divided into equal strips. Results for PMM *with M_D on the interfaces, as a function of problem size and number of processors.*

h^{-1}		$p=1$	2	4	8	16
16	I	1	3			
	κ	1.00	1.26			
	ρ	5.0 (−6)	2.9 (−2)			
	T	4.85	3.39			
	s	—	*1.43*			
32	I	1	2	5		
	κ	1.00	1.09	3.29		
	ρ	1.7 (−5)	8.8 (−3)	9.9 (−2)		
	T	25.0	13.6	9.60		
	s	—	*1.84*	*1.42*		
64	I		2	4	8	
	κ		1.09	3.29	13.0	
	ρ		4.8 (−3)	8.5 (−2)	2.8 (−1)	
	T		70.8	43.0	27.9	
	s		—	*1.65*	*1.54*	
128	I				8	18
	κ				13.0	51.9
	ρ				2.8 (−1)	5.9 (−1)
	T				150	114
	s				—	*1.32*

by a constant along a diagonal, which appears to be the case. The number of iterations virtually levels off and the condition number actually *decreases* as more processors are added with fixed refinement.

A tendency portrayed by several rows of Tables 11-13 is that adding processors at fixed mesh resolution eventually becomes inefficient. This is due to two factors: increasing the number of subdivisions of the domain tends to increase the iteration count for some methods; and even on a per iteration basis, communication becomes dominant over computation in this limit on message-passing machines such as the Hypercube. However, the tables illustrate that significant overall speedups are obtainable in spite of the inefficient use of processors. Furthermore, high efficiency regions can be found. For a theoretical analysis of the efficiency of domain decomposition algorithms in a much larger region of $h-p$ parameter space, see [23], [26].

5. Conclusions. Several basic algorithmic forms, different decomposition topologies, and a variety of preconditioners for the separator set equations, each with advantages and disadvantages, make the study of domain decomposition surprisingly rich.

Because of its lower computational complexity, the SCM is to be preferred when it is applicable, namely when the subdomain solves can be handled exactly. In this case the full PMM iterates reduce to the SCM iterates and the latter can be obtained more cheaply. The PMM is a more flexible algorithm for general symmetric scalar problems, but many generalizations are needed, especially to nonsymmetric systems of equations, where some generalized form of preconditioned conjugate gradients, such as the generalized minimum residual method, will come into play, and where the optimal preconditioners for the symmetric positive definite version of the algorithm will have less value.

TABLE 12

$\nabla^2 u = f$ in the unit square, divided into equal boxes. Results for PMM *with M_D on the interfaces and without vertex coupling, as a function of problem size and number of processors.*

h^{-1}		$p=1$	4	16	64
16	I	1	6		
	κ	1.00	8.89		
	ρ	5.0 (−6)	1.6 (−1)		
	T	4.85	4.02		
	s	—	*1.21*		
32	I	1	6	11	
	κ	1.00	12.1	25.3	
	ρ	1.7 (−5)	1.8 (−1)	3.6 (−1)	
	T	25.0	19.2	8.86	
	s	—	*1.30*	*2.17*	
64	I		6	12	17
	κ		15.9	32.2	94.4
	ρ		1.8 (−1)	4.0 (−1)	5.7 (−1)
	T		94.8	37.6	15.3
	s		—	*2.52*	*2.46*
128	I			13	19
	κ			40.5	120
	ρ			4.7 (−1)	6.1 (−1)
	T			192	60.3
	s			—	*3.18*
256	I				22
	κ				146
	ρ				6.5 (−1)
	T				319

The optimal preconditioners are to be preferred to the MSC preconditioners over the full range of examples contained herein, but the latter offer more possibilities for generalization in conjunction with problems not dominated by a Laplacian-type term, and show at least some promise in dealing with coefficient irregularity.

In terms of convergence properties, boxwise decompositions have been shown preferable to stripwise decompositions for large numbers of processors, and the crosspoint-coupled version of the algorithm is preferable from the point of view of iteration count.

The limited range of parameter space over which parallelized tests can actually be carried out with present hardware does not allow for conclusive experimental comparisons of the algorithms. However, our parallelization analysis [23] reveals the important result that for machine parameters which are reasonably achievable and for reasonably large problems, boxwise decompositions can be made more efficient than stripwise decompositions per iteration. The combination of their better convergence properties, and the fact that when domain decomposition is efficiently parallelizable at all the more implicit crosspoint-coupled version of the algorithm is not much less efficient per iteration than the decoupled version, argues for further research in the direction of more implicit preconditioners.

Acknowledgments. We would like to thank all of the members of the numerical analysis community at Yale for their stimulation during our informal "Reading Group

TABLE 13

$\nabla^2 u = f$ in the unit square, divided into equal boxes. Results for PMM *with M_D on the interfaces and with vertex coupling after Bramble et al., as a function of problem size and number of processors.*

h^{-1}		$p=1$	4	16	64
16	I	1	6		
	κ	1.00	5.74		
	ρ	5.0 (−6)	1.3 (−1)		
	T	4.85	4.28		
	s	—	*1.13*		
32	I	1	5	7	
	κ	1.00	7.88	7.01	
	ρ	1.7 (−5)	1.5 (−1)	2.6 (−1)	
	T	25.0	16.6	7.39	
	s	—	*1.51*	*2.25*	
64	I		6	7	6
	κ		10.7	10.2	7.16
	ρ		1.5 (−1)	2.5 (−1)	2.1 (−1)
	T		95.1	24.7	16.7
	s		—	*3.85*	*1.48*
128	I			7	7
	κ			14.1	10.5
	ρ			2.5 (−1)	2.4 (−1)
	T			111	36.3
	s			—	*3.06*
256	I				8
	κ				14.5
	ρ				2.7 (−1)
	T				139
	s				—

for Elliptic System Substructuring," which met during the spring of 1985. We are especially grateful for the contributions of Stan Eisenstat, Tony Chan, and Diana Resasco, which are only incompletely reflected in the bibliography, and for Doug Baxter's tireless assistance with the Hypercube programming. We also thank the referees for their thorough comments and improvements.

REFERENCES

[1] C. R. ANDERSON, *On domain decomposition*, Technical Report, Math. Dept. and Comput. Sci. Dept., Stanford Univ., 1985.

[2] P. E. BJORSTAD AND O. B. WIDLUND, *Iterative methods for the solution of elliptic problems on regions partitioned into substructures*, Technical Report 136, Courant Institute of Mathematical Sciences, New York Univ., September 1984. (See also the 1985 version with an expanded section of numerical examples.)

[3] ———, *Solving elliptic problems on regions partitioned into substructures*, in Elliptic Problem Solvers II, G. Birkhoff and A. Schoenstadt, eds., Academic Press, New York, 1984, pp. 245-256.

[4] M. J. BERGER AND S. BOKHARI, *A partitioning strategy for non-uniform problems on multiprocessors*, Technical Report 85-55, ICASE, November 1985.

[5] J. H. BRAMBLE, J. E. PASCIAK AND A. H. SCHATZ, *The construction of preconditioners for elliptic problems by substructuring*, I, Math. Comp., to appear.

[6] ———, *An iterative method for elliptic problems on regions partitioned into substructures*, Math. Comp., to appear.

[7] 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.

[8] T. F. CHAN, *Analysis of preconditioners for domain decomposition*, Technical Report 408, Comput. Sci. Dept., Yale Univ., August 1985.

[9] T. F. CHAN AND D. RESASCO, *A survey of preconditioners for domain decomposition*, Technical Report 414, Comput. Sci. Dept., Yale Univ., September 1985. In Proc. IV Coloquio de Matemáticas del CINVESTAV, Workshop in Numerical Analysis and its Applications, Taxco, Mexico, August 18–24, 1985.

[10] ———, *A domain-decomposed fast Poisson solver on a rectangle*, Technical Report 409, Comput. Sci. Dept., Yale Univ., October 1985.

[11] R. CHANDRA, *Conjugate gradient methods for partial differential equations*, Technical Report YALEU/DCS-RR-129, Comput. Sci. Dept., Yale Univ., January 1978.

[12] 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 Proceedings of the Symposium on Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1975, pp. 309–332.

[13] Q. V. DINH, R. GLOWINSKI AND J. PERIAUX, *Solving elliptic problems by domain decomposition methods with applications*, in Elliptic Problem Solvers II, G. Birkhoff and A. Schoenstadt, eds., Academic Press, New York, 1984, pp. 395–426.

[14] M. DRYJA, *A capacitance matrix method for Dirichlet problem on polygonal region*, Numer. Math., 39 (1982), pp. 51–64.

[15] ———, *A finite element-capacitance method for elliptic problems on regions partitioned into subregions*, Numer. Math., 44 (1984), pp. 153–168.

[16] M. DRYJA AND W. PROSKUROWSKI, *Fast elliptic solvers on rectangular regions subdivided into strips*, in Advances in Computer Methods for Partial Differential Equations—V, R. Vichnevetsky and R. S. Stepleman, eds., IMACS, 1984, pp. 360–368.

[17] ———, *Capacitance matrix method using strips with alternating Neumann and Dirichlet boundary conditions*, Appl. Numer. Math., 1 (1985).

[18] S. EISENSTAT, personal communication, 1985.

[19] R. GLOWINSKI, Q. V. DINH AND J. PERIAUX, *Domain decomposition methods for nonlinear problems in fluid dynamics*, Comp. Meths. Appl. Mech. Engrg., to appear.

[20] G. H. GOLUB AND D. MAYERS, *The use of pre-conditioning over irregular regions*, 1983. Lecture at Sixth Int. Conf. on Computing Methods in Applied Sciences and Engineering, Versailles, France, December 1983.

[21] W. D. GROPP, *Numerical linear algebra on workstations*, in Proc. Army Research Office Workshop on Microcomputers in Scientific Computing, 1985.

[22] ———, *Dynamic grid manipulation for* PDEs *on hypercube parallel processors*, Technical Report YALEU/DCS/RR-458, Comput. Sci. Dept., Yale Univ., March 1986.

[23] W. D. GROPP AND D. E. KEYES, *Complexity of parallel implementation of domain decomposition techniques for elliptic partial differential equations*, this Journal, 1986, submitted.

[24] R. W. HOCKNEY, *The potential calculation and some applications*, Meth. Comput. Phys., 9 (1970), pp. 135–211.

[25] S. L. JOHNSSON, *Communication efficient basic linear algebra computations on hypercube architectures*, Technical Report 361, Comput. Sci. Dept., Yale Univ., September 1985.

[26] D. E. KEYES AND W. D. GROPP, *A comparison of domain decomposition techniques for elliptic partial differential equations and their parallel implementation*, Technical Report YALEU/DCS/RR-448, Comput. Sci. Dept., Yale Univ., December 1985.

[27] C. LANCZOS, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Res. NBS, 45 (1950), pp. 255–282.

[28] D. P. O'LEARY AND O. WIDLUND, *Capacitance matrix methods for the Helmholtz equation on general three-dimensional regions*, Math. Comp., 33 (1979), pp. 849–879.

[29] J. OLIGER, W. SKAMAROCK AND W.-P. TANG, *Schwarz alternating procedure and* S.O.R. *accelerations*, Technical Report, Comput. Sci. Dept., Stanford Univ., 1985.

[30] A. T. PATERA, *A spectral element method for fluid dynamics: Laminar flow in a channel expansion*, J. Comput. Phys., 54 (1984), pp. 468–488.

[31] W. PROSKUROWSKI AND O. WIDLUND, *A finite element-capacitance matrix method for the Neumann problem for Laplace's equation*, this Journal, 1 (1980), pp. 410–425.

[32] J. S. PRZEMIENIECKI, *Matrix structural analysis of substructures*, AIAA J., 1 (1963), pp. 138–147.

[33] D. RESASCO, personal communication, 1985.

[34] J. R. RICE, E. N. HOUSTIS AND W. R. DYKSEN, *A population of linear second order, elliptic partial differential equations on rectangular domains—Part* I, Technical Report 2078, Mathematics Research Center, Univ. of Wisconsin, Madison, May 1980.

[35] G. RODRIGUE AND J. SIMON, *Jacobi splittings and the method of overlapping domains for solving elliptic* P.D.E.*'s*, in Advances in Computer Methods for Partial Differential Equations—V, R. Vichnevetsky and R. S. Stepleman, eds., IMACS, 1984, pp. 383-386.

[36] H. A. SCHWARZ, *Gesammelte mathematische Abhandlungen*, Springer, Berlin, 2, 1890, pp. 133-134.

[37] P. N. SWARZTRAUBER AND R. A. SWEET, *Efficient* FORTRAN *subprograms for the solution of separable elliptic partial differential equations*, ACM Trans. Math. Software, 5 (1977), pp. 352-364.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 2, March 1987
© 1987 Society for Industrial and Applied Mathematics
004

A PARALLEL ADAPTIVE NUMERICAL SCHEME FOR HYPERBOLIC SYSTEMS OF CONSERVATION LAWS*

BRADLEY J. LUCIER† AND ROSS OVERBEEK‡

Abstract. We generalize the first author's adaptive numerical scheme for scalar first order conservation laws to systems of equations in one space dimension. The resulting numerical method generates highly nonuniform, time-dependent grids, and hence is difficult to execute efficiently on vector computers such as the Cray or Cyber 205. In contrast, we show that this type of algorithm may be executed in parallel on alternate computer architectures. We describe a parallel implementation of the algorithm on the Denelcor HEP, a multiple-instruction, multiple-data (MIMD) shared memory parallel computer. This program dynamically partitions the problem domain so as to achieve effective load balancing among the available processes.

Key words. adaptive grids, MIMD computation, domain decomposition

AMS (MOS) subject classifications. 65M05, 68B99

Introduction. In [22] the first author presented an adaptive numerical scheme for the numerical approximation of the scalar hyperbolic conservation law

$$\text{(C)} \qquad \begin{aligned} &u_t + f(u)_x = 0, \qquad x \in \mathbb{R}, \quad t > 0, \\ &u(x, 0) = u_0(x), \qquad x \in \mathbb{R}. \end{aligned}$$

The numerical method used spatial adaptation to refine the grid near singularities in the solution. The novelty of the method lay in the criteria used to adapt the grid, the provision of both a priori and a posteriori stability estimates for the complete adaptive scheme, and the experimental demonstration that asymptotic speed-up results for some problems when compared with a method using the same finite difference operator on a fixed, uniform grid. Although the adaptive method is unsuitable for execution on vector computers such as the Cray or Cyber 205 because it uses a highly nonuniform, time-dependent computation grid, the algorithm may be executed in parallel on alternate computer architectures. In this paper we extend the algorithm for scalar conservation laws to hyperbolic systems of conservation laws, and we describe a parallel implementation of the algorithm on the Denelcor HEP, a multiple-instruction, multiple-data (MIMD) shared memory parallel computer [16]. We employ a dynamic tree partitioning algorithm that achieves approximate load balancing among the tasks presented to the available processes. Because our implementation uses a macro-preprocessor to avoid mention of the specific hardware synchronization primitives provided on the HEP, our programs may be transported to any shared memory multiprocessor on which the synchronization macro package has been installed.

There has been much recent interest in adaptive or moving grid methods applied to evolution equations, and specifically to conservation laws. Papers describing numerical schemes or analyses may be found in [2]–[5], [9], [13]–[14], [15]–[23], [25]–[27], [29], [33], [35]–[36]. Papers making specific mention of data structures for adaptive computation include [1], [4], [5], [22], and [32].

* Received by the editors November 27, 1985; accepted for publication (in revised form) May 13, 1986. This article was presented at the Second SIAM Conference on Parallel Processing for Scientific Computing that was held in Norfolk, Virginia on November 18–21, 1985.

† Department of Mathematics, Purdue University, West Lafayette, Indiana 47907. The work of this author was partially supported by the National Science Foundation under grant DMS-8403219.

‡ Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois 60439. The work of this author was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under contract W-31-109-Eng-38.

1. Background: The sequential algorithm for scalar equations. We briefly describe the method for scalar conservation laws presented in [22], to which the interested reader is referred for more details. The numerical method is based on the following algorithm, similar to well-known algorithms for adaptive linear approximation [6]-[7], for choosing a grid on which to approximate a function u defined on an interval $[a, b]$.

ALGORITHM M. This algorithm chooses grid points at which to approximate a bounded function u defined on $[a, b]$ that is constant outside $[a, b]$. Let ε be a small parameter.

(1) The grid points consist only of the points a and b and the centers of admissible intervals. Admissible intervals are defined by (2) and (3) below.

(2) The interval $[a, b]$ is an admissible interval.

(3) For any admissible interval I, let $3I = \{x \mid \operatorname{dist}(x, I) \equiv \inf_{y \in I} |x-y| < |I|\}$. If $|I| \geqq \varepsilon$ and

$$|I| \int_{3I} [|u_{xx}| + |f''(u)| u_x^2]\, dx \geqq \varepsilon, \tag{1}$$

then the left and right halves of I are admissible intervals. The above integral is finite if u_{xx} is a finite measure; otherwise, it is to be interpreted as infinite. Note that $3I$ is an *open* interval. The approximation to u is chosen to be the piecewise linear interpolant of u at the given grid points.

This algorithm chooses a grid that satisfies three criteria that we feel are important for adaptive numerical methods that use piecewise linear functions as approximations to solutions of evolution equations:

(1) Under plausible assumptions about the structure of the solution u of (C) (that $u \in BV(\mathbb{R})$, and that between discontinuity curves $u_x \in BV(\mathbb{R})$), it may be shown that u can be approximated well in $L^1(\mathbb{R})$ by continuous piecewise linear functions on this grid.

(2) It may be shown formally that the spatial operator $f(u)_x$ may be approximated well in $L^1(\mathbb{R})$ by piecewise constant functions on this grid.

(3) One observes empirically that the error incurred in the regridding process from one time step to the next is small on this grid; there are heuristic reasons, based on the assumed local smoothness of u away from discontinuity curves, for believing this.

The numerical scheme solves (C) on an interval $[a, b]$ instead of $\mathbb{R}$. To ensure that interactions with the boundaries of the interval $[a, b]$ may be neglected, we assume that the widths of the leftmost and rightmost minimal interval are greater than ε. We also assume that $f \in C^2$ and that $\|f'\|_{L^\infty(\mathbb{R})} \leqq 1$; the latter condition may always be achieved by a change in the time scale. The numerical scheme is as follows:

(1) A grid $\{x_i^0\} = \mathbf{M}^0$ and initial approximation U^0 is chosen, based on the initial data $u_0(x)$ ($U^n(x_i^n) = U_i^n$).

(2) For each $n \geqq 0$:

(a) The approximate solution is advanced from time t^n to t^{n+1} at all the grid points $x_i^n \in \mathbf{M}^n$ using, in our case, the Engquist-Osher finite difference operator [10]:

$$\frac{\overline{U_i^{n+1}} - U_i^n}{\Delta t} + \frac{f^+(U_i^n) - f^+(U_{i-1}^n)}{h_i^n} + \frac{f^-(U_{i+1}^n) - f^-(U_i^n)}{h_{i+1}^n} = 0. \tag{F}$$

The flux f has been separated into its increasing (f^+) and decreasing (f^-) parts, $f = f^+ + f^-$. To maintain the stability of the scheme, the time step for the finite difference scheme (F), Δt, is chosen to be $\varepsilon/4$; $h_i^n = x_i^n - x_{i-1}^n$;

(b) The grid selection algorithm is applied to the piecewise linear function $\overline{U^{n+1}}$ with values $\overline{U_i^{n+1}}$ at the points $x_i^n \in \mathbf{M}^n$ to yield a new grid $\mathbf{M}^{n+1}$ and approximation U^{n+1}.

The greatest difficulty in a sequential implementation of the method is the organization of the data structures and the implementation of Algorithm M (which, strictly speaking, is not algorithmic at all, but definitional). Because the grid selection is based on recursive subdivision of the basic interval $[a, b]$, the natural data structure through which to organize the grid information is a binary tree. Each point in the grid, or equivalently, each admissible interval, corresponds to a node in the tree; therefore, we will speak of nodes, grid points and intervals interchangeably. The information stored in each node is presented in Fig. 2 of [22]; we give a brief discussion here of the main points. The left and right *children* of a node are the nodes corresponding to the left and right halves of the interval associated with that node; the *parent* relation is the inverse of the child relation. Two other pairs of links are needed to calculate the integral in (1). If a point x_i is centered in the interval $I = (x_l, x_r)$, then links are stored in x_i to the points x_l and x_r, which are called the left and right *boundary* points of x_i. As well, if an interval I is not a leaf in the tree (if it has been divided by Step 3 in Algorithm M), links are stored to the adjacent intervals of the same width as I immediately to the left and right of I. We choose to call these links *sibling* links. (Because these links correspond to adjacent nodes at the same depth in the tree representing the grid, *cousins* may be a more appropriate term!) It may be shown that nonleaf nodes *always* have adjacent left and right siblings; this will prove important in the parallel version of the algorithm.

After a calculation that determines the initial grid $\mathbf{M}^0$, the computations of each time step are organized as follows. In the first part of each time step, the finite difference scheme calculates U^{n+1} defined on $\mathbf{M}^n$, as in Step 2(a) above. In the final part of each time step, a new grid is chosen based on the function $U^{n+1} \equiv \bar{U}$. This grid selection algorithm is divided into two phases. In the first phase (which is illustrated in Fig. 4 of [22]), a standard recursive algorithm, similar to algorithms for adaptive quadrature, calculates for each grid point $x_i^n \in \mathbf{M}$ centered in the interval $I = (x_l, x_r)$, the values of *left int*, *right int*, and *int*, which are the values of (1) integrated over the intervals (x_l, x_i^n), (x_i^n, x_r) and (x_l, x_r), respectively. This calculation is considerably simplified by the fact that $\bar{U}$ is piecewise linear. For example, $\bar{U}_{xx}$ consists solely of multiples of delta measures located at each grid point x_i^n, so that if x_i^n is a leaf node,

$$\int_{(x_l, x_r)} |\bar{U}_{xx}|\, dx = |\bar{U}_x^+(x_i) - \bar{U}_x^-(x_i)|,$$

where $\bar{U}_x^+$ is the right-hand limit of $\bar{U}_x$ and $\bar{U}_x^-$ is the left-hand limit of $\bar{U}_x$. The above expression will be denoted by $x_i^n \to uxx$. (In general, a right arrow designates a subsidiary property of a node; it could be a numerical value associated with the node, or a link to another node, such as a node's parent or child.) Similarly, $\bar{U}_x$ is constant between grid points, so that if we define $\bar{f}'(s) = |f''(s)|$ for all s,

$$\int_{(x_l, x_{l+1})} |f''(\bar{U})|\, \bar{U}_x^2\, dx = (\bar{f}(\bar{U}(x_{i+1})) - \bar{f}(\bar{U}(x_i))) \frac{\bar{U}(x_{i+1}) - \bar{U}(x_i)}{x_{i+1} - x_i}.$$

After these integrals are calculated, the second phase of the grid modification algorithm begins. The tree associated with $\mathbf{M}^n$ is traversed in a top-down, breadth-first ordering, examining each node in turn. If a node x_i^n is a left child, it is not difficult to see that with the definitions given in the previous paragraph of the various links, the integral

in (1) may be calculated as

(2) $x_i^n \to \text{parent} \to \text{int} + x_i^n \to \text{parent} \to \text{left sibling} \to \text{right int} + x_i^n \to \text{left boundary} \to \text{uxx}.$

(See Fig. 5 of [22].) The boundary and sibling links are included in the data structure specifically for this computation. A breadth-first traversal is used because this calculation depends on the sibling links of the parent, which must be present. Step 3 of Algorithm M is then applied to decide whether the node x_i^n should have children. If a leaf node x_i^n should have children, they are added to the tree and the appropriate links in the tree are updated; if an interior (nonleaf) node should not be subdivided, then the subtrees headed by its children are removed. It may be shown that at every stage in the calculation all interior nodes have adjacent left and right siblings, so that (2) may be calculated.

2. An adaptive method for hyperbolic systems. We now consider when u in (C) is a vector of unknowns in $\mathbb{R}^m$, and the equation takes the form

$$\text{(S)} \qquad \begin{aligned} &u_t + F(u)_x = 0, \qquad x \in \mathbb{R}, \quad t > 0, \\ &u(x, 0) = u_0(x) \in \mathbb{R}^m, \qquad x \in \mathbb{R}. \end{aligned}$$

We assume that for all u the Jacobian matrix $A(u) = \partial F(u)$ has m real and distinct eigenvalues $\lambda_1(u) < \lambda_2(u) < \cdots < \lambda_m(u)$, with associated right eigenvectors $r_1(u), r_2(u), \cdots, r_m(u)$. Furthermore we assume that each eigenvalue is either *genuinely nonlinear* in the sense of Lax or is linearly degenerate, that is, the eigenvectors $r_k(u)$ may be normalized so that $\nabla_u \lambda_k(u) \cdot r_k(u)$ is either identically 1 (which is a convexity condition) or is identically 0 (see [17]). Again we assume a time scaling such that $|\lambda_i(u)| \leqq 1$ for all i and all u that will arise in the computation.

Several finite difference schemes (see [12], [28], [31]) have been devised for such problems that satisfy certain auxiliary conditions, such as having Riemann invariants. In our algorithm we chose a generalization of the Engquist-Osher scheme, introduced by Osher and Solomon in [30], which is based on a method described in [17] for solving the Riemann problem for (S). The Riemann problem has the special initial data

$$u_0(x) = \begin{cases} u_l & \text{for } x \leqq 0, \\ u_r & \text{for } x > 0. \end{cases}$$

Because the structure of Osher and Solomon's finite difference operator determines the grid selection algorithm (as we believe it should), their scheme is described below.

Near each point $u \in \mathbb{R}^m$ one may find a local covering $\sigma(\cdot, u) : [-S, S]^m \to \mathbb{R}^m$, with $\sigma(0, u) = u$, by solving successively the differential equations:

$$\frac{d\Gamma_k}{d\tau} = r_k(\Gamma_k) \quad \text{for } \tau \text{ between 0 and } s_k, \; k = m, \cdots, 1,$$

with $\Gamma_m(0) = u$ and $\Gamma_k(0) = \Gamma_{k+1}(s_{k+1})$. Set $\sigma(s, u) = \Gamma_1(s_1)$. (The paths $\Gamma_k(\tau)$ are related to the solution of the Riemann problem with $u_l = u$ and $u_r = \sigma(s, u)$.) Because the matrix whose columns consist of the eigenvectors $r_k(u)$ is nonsingular, for small enough S the mapping σ exists, is one-to-one, and covers a local neighborhood of u. In other words, any state v near u may be connected to u by a unique path $\{\Gamma_k\}$. For many physical problems, and for the Euler equations for gas dynamics in particular, it may be shown that any two physically possible states may be connected by such a path. This property is necessary for the finite difference scheme of Osher and Solomon to make sense, and we restrict our attention to systems (S) that have this property.

Note that for the conservation law (C), the spatial difference operator $(f(U_i^n) - f(U_{i-1}^n))/h_i^n$ may be written as

$$\begin{aligned}\frac{f(U_i^n)-f(U_{i-1}^n)}{h_i^n} &= \frac{f^+(U_i^n)-f^+(U_{i-1}^n)}{h_i^n} + \frac{f^-(U_i^n)-f^-(U_{i-1}^n)}{h_i^n} \\ &= \frac{1}{h_i^n}\int_{U_{i-1}^n}^{U_i^n} (f'(u) \vee 0)\, du + \frac{1}{h_i^n}\int_{U_{i-1}^n}^{U_i^n} (f'(u) \wedge 0)\, du,\end{aligned} \tag{3}$$

where $a \vee b = \max(a, b)$ and $a \wedge b = \min(a, b)$. The scheme (F) may be thought of as distributing the first term on the right of (3) to the right endpoint of the interval (x_{i-1}^n, x_i^n) and the second term to the left. The basic idea of the Osher–Solomon finite difference method for systems is to apply the difference scheme (F) along each curve Γ_k^i connecting U_{i-1}^n to U_i^n; i.e., $U_i^n = \sigma(s, U_{i-1}^n)$. For the system (S), the wave speeds $\lambda_k(u)$ correspond to $f'(u)$ and the paths $\{\Gamma_k^i\}$ replace the simple path of integration, so that

$$\frac{F(U_i^n)-F(U_{i-1}^n)}{h_i^n} = \frac{1}{h_i^n}\sum_{k=1}^{m}\int_{\Gamma_k^i} (\lambda_k(u) \vee 0) r_k(u)\, du + \frac{1}{h_i^n}\sum_{k=1}^{m}\int_{\Gamma_k^i} (\lambda_k(u) \wedge 0) r_k(u)\, du. \tag{4}$$

The finite difference scheme for systems, then, consists of distributing Δt times the first sum to U_i^n at the right endpoint of the interval (x_{i-1}^n, x_i^n) and Δt times the second sum to U_{i-1}^n, at the left. Osher and Solomon showed that these integrals may be calculated simply and effectively for many problems of interest.

One may instantly devise an adaptive method for systems, based on Algorithm M, once one finds a suitable substitute for the condition (1). The condition we choose, although heuristic, reduces to (1) in the special case of a scalar conservation law, and may be shown to be effective when solving constant coefficient, linear problems.

We first note that when the flux f of (C) is convex or linear and u is a piecewise linear function defined on a grid $\{x_i\}$, we may write

$$I_1(i) \equiv \int_{x_{i-1}}^{x_i} |f''(u)| u_x^2\, dx = |f'(u(x_i)) - f'(u(x_{i-1}))| \frac{|u(x_i) - u(x_{i-1})|}{h_i}; \tag{5}$$

similarly,

$$I_2(i) \equiv \int_{x_i-\delta}^{x_i+\delta} |u_{xx}|\, dx = \left| \frac{u(x_{i+1}) - u(x_i)}{h_{i+1}} - \frac{u(x_i) - u(x_{i-1})}{h_i} \right| \tag{6}$$

in the limit of small δ. We generalize (5) and (6) for systems (with the assumption of genuinely nonlinear or linearly degenerate eigenvalues) by noting that the eigenvalues $\lambda_k(u)$ play the part of the derivative $f'(u)$, and so for systems we set

$$I_1(i) \equiv \sum_{k=0}^{m-1} |\lambda_k(\Gamma_k^i(0)) - \lambda_k(\Gamma_{k+1}^i(0))| \frac{\|\Gamma_k^i(0) - \Gamma_{k+1}^i(0)\|_1}{h_i}, \tag{7}$$

and

$$I_2(i) \equiv \left\| \frac{u_{i+1} - u_i}{h_{i+1}} - \frac{u_i - u_{i-1}}{h_i} \right\|_1, \tag{8}$$

where $\|v\|_1 = \sum_{k=1}^m |v_k|$ for $v \in \mathbb{R}^m$. When $m = 1$ definitions (5) and (6) coincide with (7) and (8). The adaptive scheme for systems uses Algorithm M to choose the grid, substituting (7) and (8) for (5) and (6) whenever the latter two quantities arise in the scalar computation of (1).

A complete analysis of the adaptive scheme using these substitutes for the integral (1) may be carried out for the constant coefficient, linear case, that is, for

$$\begin{aligned} &u_t + Au_x = 0, \qquad x \in \mathbb{R}, \quad t > 0, \\ &u(x,0) = u_0(x) \in \mathbb{R}^m, \qquad x \in \mathbb{R}, \end{aligned} \tag{9}$$

where A is a constant matrix. We may write $A = R\Lambda R^{-1}$, where Λ is the diagonal matrix with nontrivial entries λ_k, and R is the matrix whose columns consist of the right eigenvectors of A. The finite difference scheme that we use may now be written as

$$\frac{\overline{U_i^{n+1}} - U_i^n}{\Delta t} + \frac{A^+(U_i^n - U_{i-1}^n)}{h_i^n} + \frac{A^-(U_{i+1}^n - U_i^n)}{h_{i+1}^n} = 0, \tag{10}$$

where $A^+ = R\Lambda^+ R^{-1}$ and Λ^+ is the diagonal matrix with diagonal entries $\max(\lambda_k, 0)$; $A^- = R\Lambda^- R^{-1}$, with $\Lambda^- = \Lambda - \Lambda^+$. To analyze the system, we introduce the change of dependent variables $u = Rv$; by this device, the system (9) is transformed to the decoupled system

$$\begin{aligned} &v_t + \Lambda v_x = 0, \qquad x \in \mathbb{R}, \quad t > 0, \\ &v(x,0) = R^{-1}u_0(x); \end{aligned}$$

the scheme (4) is equivalent to

$$\frac{\overline{V_i^{n+1}} - V_i^n}{\Delta t} + \frac{\Lambda^+(V_i^n - V_{i-1}^n)}{h_i^n} + \frac{\Lambda^-(V_{i+1}^n - V_i^n)}{h_{i+1}^n} = 0.$$

Because the components of this system are uncoupled, and the system is linear (so that $I_1(i) = 0$), the analysis presented in Theorem 5.2 of [22] may be applied separately to each component of v to prove the following theorem, which we state here without proof.

THEOREM. *Let each component of u_0 have a first derivative that is of bounded variation, and let $u(x, t)$ be the solution of* (9). *Assume, moreover, that Algorithm* M *is modified so that if an interval is smaller than $\varepsilon^{1/2}$, then it may no longer be divided by Step* 3; *let $\Delta t = \varepsilon^{1/2}/4$. If $n\Delta t = T$, and U^n is the solution of the adaptive grid algorithm above, then*

$$\|u(T) - U^n\|_{(L^1(\mathbb{R}))^m} \leq 3(T+1)m\Delta t \|R\|_1 \|R^{-1}\|_1 \|u_0'\|_{(BV(\mathbb{R}))^m}.$$

$\|R\|_1$ is the operator norm of R when applied to $\mathbb{R}^m$ with the $\|\cdot\|_1$ norm.

The computer implementation for systems of equations is organized somewhat differently than the implementation for a scalar equation. Each time step begins with $\overline{U^n}$ defined on $\mathbf{M}^{n-1}$. Two passes are then made through the tree. The first pass through the tree calculates the "integral" in (1) over each of the intervals (x_l, x_i^{n-1}), (x_i^{n-1}, x_r) and (x_l, x_r) for every interval $I = (x_l, x_r)$ with $x_i^{n-1} = (x_l + x_r)/2$ in $\mathbf{M}^{n-1}$. In so doing, it calculates for each leaf x_i^{n-1} in $\mathbf{M}^{n-1}$ the values of $\Gamma_k^i(s_k^i)$ and $\Gamma_k^{i+1}(s_k^{i+1})$ and substitutes (7) and (8) for (5) and (6), which are used in the scalar computation. For systems such as the Euler equations, calculating the intersection of the curves Γ_k takes much longer than calculating (7) and (8), so that the supplementary calculations to choose the grid are very cheap. The recursive algorithm used in the scalar computation is again used here to accumulate the value of (1) for interior nodes. The second pass through the tree combines the tree modification part of Algorithm M, which uses the calculation (2) requiring the sibling links, and the calculation of $\overline{U^{n+1}}$ from U^n using the Osher–Solomon finite difference scheme. In this second pass, the tree is traversed in a breadth first ordering, as before, to add and remove nodes where necessary to obtain $\mathbf{M}^n$. The breadth first ordering implies that each node's parent's sibling links

are always available when needed in (2). In addition, whenever one discovers a leaf x_i^n in $\mathbf{M}^n$, then one knows that the left and right neighbors (boundaries) of x_i^n cannot be changed by further modifications to the tree. This allows one to calculate the contributions of the sums in (4) on each interval (x_{i-1}^n, x_i^n) and (x_i^n, x_{i+1}^n) to the values of U_{i-1}^{n+1}, U_i^{n+1}, and U_{i+1}^{n+1}. Thus, it is at this time that the finite difference scheme is applied locally to U^n to obtain U^{n+1}.

3. Parallel implementation. Several changes had to be made to the original data structure and computational sequence found in [22] to implement an efficient parallel version of the adaptive algorithm. We will begin our discussion by describing in more detail the recursive calculation of the intergral (1) over each interval (x_l, x_i), (x_i, x_r), and $(x_l, x_r) = I$ associated with a node $x_i = (x_l + x_r)/2$ in the grid.

The basic algorithm is very simple (cf. Fig. 4 in [22]):

```
Calculate Integrals (I) {
    do calculations common to all nodes x_i
    if (I is a leaf) {
        calculate left int, right int, and uxx from basic formulae
        set int := left int + right int + uxx
    } else {
        Calculate Integrals (I → left child)
        Calculate Integrals (I → right child)
        set left int := I → left child → int
        set right int := I → right child → int
        calculate uxx from basic formula
        set int := left int + right int + uxx
    }
}
```

By maintaining a stack of nodes waiting to be processed and introducing in each node I a counter $I \rightarrow count$ that keeps track of the number of children of that node that have been processed, the recursion may be removed in a standard way; see, for example, [37, pp. 316ff.]. We include the following algorithm only because we use it as a model for later algorithms. It assumes that the stack has been initialized and that the root of the tree has been pushed onto the stack.

```
while (the stack is not empty) {
    pop I from the stack
    while (I is not a leaf) {
        do calculations common to all intervals
        set I → count := 0
        push I → right child onto the stack
        set I := I → left child
    }
    /* at this point I is a leaf */
    do calculations common to all intervals
    calculate left int, right int, and uxx from basic formulae
    set int := left int + right int + uxx
    set done := false
    while (I is not the root node and not done) {
        set I := I → parent
        if (I → count = 0) {
            /* only one child of I has been processed */
```

```
                set I → count := 1
                set done := true
            } else {
                /* both children of I have been processed */
                set left int := I → left child → int
                set right int := I → right child → int
                calculate uxx from basic formula
                set int := left int + right int + uxx
            }
        }
    }
```

To implement this algorithm in parallel, we follow the strategy of Lusk and Overbeek [24] (and, obviously, Brinch Hansen [8] and others), and set up a *monitor* to control access to the stack by a fixed number of processes executing identical copies of the above code in parallel. A monitor is a shared data structure that can be accessed by only one process at a time, together with routines that access and modify that structure. In our case, the stack of nodes waiting to be processed has associated with it two routines, *push a node onto the stack*, and *request a node from the stack*. When a process requests a node from the stack, either (a) it is successful and a node is popped from the stack, or (b) all processes are waiting to get a node from the stack and the stack is empty. In the latter case, the *request* monitor routine returns a special code to all processes that indicates that the integral calculation stage of execution is finished because there are no other processes executing that can add a node to the stack. When the special termination code is sent out, one process, the *organizer*, sets up the next part of the computation, while the *worker* processes wait until the stack is nonempty and there is more work to be done. The monitor routines are implemented as macros that are expanded into in-line code by a macro preprocessor, so that little overhead is expended in the synchronization code. It is true, however, that the code in the monitor is executed sequentially, so whenever one routine is pushing a node onto the stack, for example, no other process may enter the monitor.

Because processes may not access the global (shared) stack in parallel, it is important to minimize contention among processes using the stack. This can be accomplished by having each process complete the computation for a subtree if this subtree is smaller than a maximum size. This requires one to keep track of the size of the subtree headed by each node in the tree; this information is stored in a field that we call *subtree size*. The integral calculation algorithm may be further abstracted as follows:

```
while (the global stack is not empty) {
    pop I from the global stack
    while (I → subtree size > max size) {
        do preliminary calculations
        push I → right child onto the global stack
        set I := I → left child
    }
    /* at this point I heads a subtree with max size or fewer nodes */
    process the subtree headed by I
}
```

This strategy works well when the tree is balanced, that is, when the two subtrees headed by the children of a node are roughly of the same size. In this case, each

process computes results for about *max size* to (*max size*)/2 nodes. However, the trees that are generated by our grid algorithm are very unbalanced, because the tree is deep and narrow near discontinuities in the solution (see [22, Fig. 3]). This causes some processes to have little work to do between trips to the monitor, thereby decreasing the amount of inherent parallelism. Thus, a dynamic tree partitioning algorithm that breaks up the tree into parts of roughly the same size is necessary to ensure that all processes have about the same work load between trips to the monitor.

We developed the top-down tree partitioning algorithm presented in Fig. 1, which is similar to a bottom-up algorithm noted in [11]. In the tree modification pass of the computation, complete subtrees may be removed by the algorithm, thereby making a bottom-up scheme impractical—it is useless to examine nodes that should have already been removed by a previous part of the calculation. This algorithm ensures that each process works on between (*max size*)+1 and 3(*max size*), inclusive, nodes before returning to the monitor. It also ensures that every node on the global stack heads a subtree of size at least (*max size*)+1. The partitioning algorithm employs a local stack on which to hold unprocessed subtrees. In practice, each worker process executes the code in Fig. 1 imbedded in a loop that repeats endlessly, with the worker processes waiting between time steps when there are no nodes on the shared stack. At the end of the program a special signal is sent to all worker processes that allows them to terminate gracefully.

There are two conflicting requirements when considering the value of the parameter *max size*. *Max size* should be small enough so that the tree is partitioned into many pieces and all processes end work at about the same time; it should be large enough so that little time is spent by processes waiting to enter the monitor. One might find that only for larger trees can these conflicting requirements be met—there may just not be enough inherent parallelism available in the computation of a small tree.

Because the tree partitioning algorithm does not guarantee that the nodes will be processed in any particular order (other than top-down), it was difficult to apply this

```
TREE PARTITION ALGORITHM {
    Let stack size denote the number of nodes in the
        subtrees stored temporarily on the local stack
    pop I from global stack
    set stack size := 0
    while (stack size ≦ max size and stack size + I → tree size > 3 (max size)) {
        process I as an interior node
        let min tree be the smaller of the subtrees of the two children of I
        let max tree be the larger of the subtrees of the two children of I
        if (min tree → tree size + stack size > 3 (max size)) {
            push min tree onto the global stack
        } else {
            push min tree onto the local stack
            set stack size := stack size + min tree → tree size
        }
        set I := max tree
    }
    if (I → tree size + stack size > 3 (max size)) {
        push I onto the global stack
    } else {
        push I onto the local stack
    }
    Process all subtrees on the local stack
}
```

FIG. 1. *Tree partition algorithm.*

algorithm directly to the second pass through the tree in each time step. It was during this pass that we traversed the tree in breadth first order while evolving $\mathbf{M}^{n-1}$ into $\mathbf{M}^n$ according to Algorithm M and advancing the solution using the finite difference operator. A breadth first traversal of the tree is strongly suggested because the decision whether to modify a node by adding or removing subtrees depends on the calculation in (2), which depends in turn on the availability of the left and right siblings of that node's parent. If one does use a breadth first traversal, however, then the maximum parallelism available during the second pass is limited to the number of nodes at a given depth in the tree. Near shocks the tree is very narrow (cf. Fig. 3 of [22]), and very little parallelism is available. As an indication of this, we note that our first implementation used a breadth first traversal of the tree and could achieve a speed-up of at most a factor of 1.5 independently of the number of processes invoked in the computation. Clearly another strategy is in order.

It is at this point that we use the fact that the grid modification process may be interpreted as a tree transformation, where the initial tree, $\mathbf{M}^{n-1}$, satisfies certain invariants because it has been developed by the same process at the previous time step. In particular, it is known that every node in $\mathbf{M}^{n-1}$ has a parent either on the outer edge of the tree structure or with adjacent left and right siblings, so that every node originally in $\mathbf{M}^{n-1}$ may be examined in parallel (subject to the condition that the tree be traversed top-down, so that nodes that will be removed during this time step are removed before their children are examined). Thus, if we allow ourselves to remove subtrees at will, but to add only a layer of nodes one node thick to the edge of the tree, all the nodes originally in $\mathbf{M}^{n-1}$ may be examined and modified in parallel. Because sibling links are horizontal, and not vertical, we cannot add these links to new nodes until all the nodes in $\mathbf{M}^{n-1}$ have been processed; one can show that even under this condition children may be added to leaf nodes in parallel with the processing of the rest of the tree. The tree partitioning algorithm in Fig. 1 may now be used to balance the execution load among the processes.

After the nodes in $\mathbf{M}^{n-1}$ have been processed, and the tree has been transformed into a state intermediate to $\mathbf{M}^{n-1}$ and $\mathbf{M}^n$, it remains to add the sibling links to the new nodes and to apply Step 3 of Algorithm M and the finite difference operator to the nodes that have been added in this step. This "clean-up" part of the second pass is done sequentially, using a breadth first, top-down traversal of only the nodes that have been added during this time step. Because it is rare that a subtree of height two or more is added to a node during one time step (it was never observed in any computer simulation), this part of the computation, though sequential, takes very little time.

4. Computational results and discussion. Our method was applied to approximate the one-dimensional Euler equations of gas dynamics. The physical quantities in this system are the density ρ, the momentum m and the specific energy e, so that

$$u = \begin{pmatrix} \rho \\ m \\ e \end{pmatrix} \quad \text{and} \quad F(u) = \begin{pmatrix} m \\ \dfrac{m}{\rho}\left(m + (\gamma-1)\left(e - \dfrac{1}{2}\dfrac{m^2}{\rho}\right)\right) \\ \dfrac{m}{\rho}\left(e + (\gamma-1)\left(e - \dfrac{1}{2}\dfrac{m^2}{\rho}\right)\right) \end{pmatrix}.$$

To compare the results with experiments by Sod [34], we set $\gamma = 1.4$; the initial values of $\rho(x)$ were chosen to be 1 for $x \leqq 0$ and 1/8 for $x > 0$; the momentum was initially

0 everywhere; and $e(x)=2.5$ for $x\leqq 0$ and $e(x)=0.2$ for $x>0$. The grid refinement algorithm was modified slightly for this problem. To make the grid slightly less uniform, Step 3 of Algorithm M was changed for m-dimensional systems so that if $|I|\geqq\varepsilon$ and

$$|I|\int_{3I}[|u_{xx}|+|f''(u)|u_x^2]\,dx\geqq m\varepsilon$$

(with the appropriate substitute for the integral when $m>1$) then the left and right halves of I are admissible intervals. To avoid an artificial time scaling, the time step Δt was set to $\varepsilon/16$. Figures 2(a) through 2(c) show the numerical approximations to the density ρ, the pressure $p=(\gamma-1)(e-(m^2/2\rho))$, and the internal energy $((e/\rho)-\frac{1}{2}m^2/\rho^2)$ at time 0.15 when solved on the interval $[-1, 1]$ with $\varepsilon=1/256$. These graphs are drawn with the same horizontal and vertical scales as the graphs in [34]. A total of 193 grid points were used to approximate the solution at the final time; the minimum grid spacing was 1/1024, while the largest spacing was 1/8. The results in [34] were obtained with a uniform grid of 100 points, so we use twice the number of grid points in this experiment. Figure 3 shows the node placement superimposed upon a graph of the density ρ; note that the horizontal scale is expanded by a factor of three, so that the contact discontinuity appears much more spread out.

Some comparison should be made between this and other methods. It is the purpose of this paper to illustrate that grid adaptation, developed with careful consideration of the underlying finite difference operator, can be effective when solving problems with moving singularities. The finite difference scheme that we chose, the Osher–Solomon scheme, is a first order scheme that gives relatively good resolution to shocks. If we denote the grid spacing on a uniform grid by h, the scheme has $O(h)$ error where the solution is smooth and smears moving contact discontinuities to a width of $O(h^{1/2})$. An examination of the condition (1) in Algorithm M shows that our adaptive scheme sets the local grid intervals near contacts and shocks to $O(\varepsilon)$ and the local grid intervals in the smooth part of the flow to $O(\varepsilon^{1/2})$, so that the errors in each part of the flow are of the same order of magnitude. (Shocks are resolved to a width of $O(\varepsilon)$, but because the error in the shock speed depends, in general, on the error in the generally smooth flow on each side of the shock, the L^1 error near the shock is again $O(\varepsilon^{1/2})$.) It is our contention that solving the problem more accurately in one area of the flow than in another is generally unnecessary, and so this is the correct balance between local grid sizes in each region of the domain for this finite difference operator.

If a formally second order TVD scheme [12] were used as the underlying finite difference operator, a new set of trade-offs would be implied. The error in the smooth area of the flow would be $O(h^2)$; if the scheme resolved moving contacts in a fixed number of grid points, then the error near contacts would be $O(h)$; the error near shocks would again be $O(h)$. This implies that the local grid spacing near singularities in the flow should be $O(\varepsilon)$, while the grid spacing in the smooth areas of the flow should be $O(\varepsilon^{1/2})$. This would result, however, in an overall error of $O(\varepsilon)$.

The methods of Oliger [3], Bolstad [5] and Berger [4] use locally adapted temporal as well as spatial increments. We maintain that although this technique is useful, it would achieve only a constant factor speed-up over the results presented here because of the first order accuracy of our finite difference operator. This may be seen by counting how many grid intervals of each width are in the grid. Near the contact, there are $O(\varepsilon^{-1/2})$ intervals of width ε that require a time step of order ε. In the smooth part of the flow there are $O(\varepsilon^{-1/2})$ intervals of width $O(\varepsilon^{1/2})$ that require a time step of

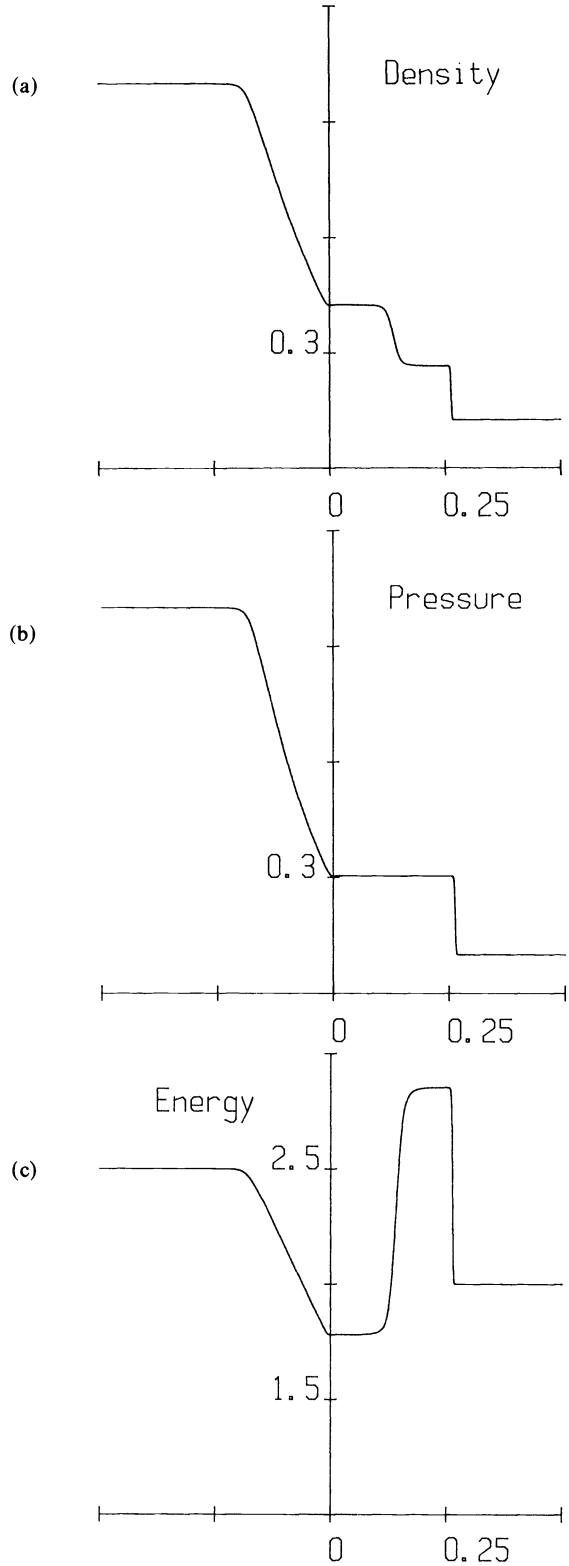


FIG. 2. *Graphs of* (a) *the density* ρ, (b) *the pressure* p, *and* (c) *the internal energy* E *at time* 0.15 *for the example given in text.*

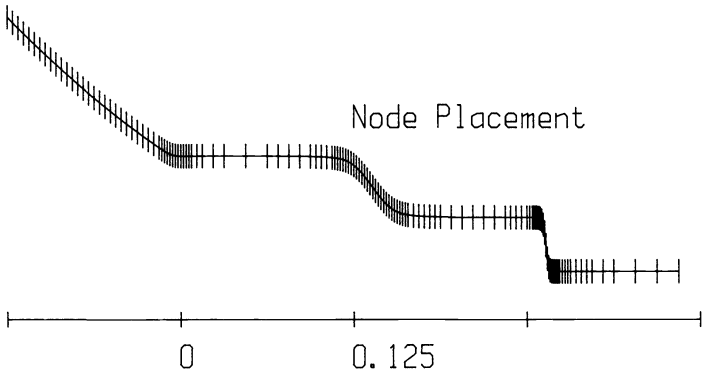


FIG. 3. *The density ρ for x between $-1/8$ and $3/8$. Tics mark the placement of the grid points. Note that the horizontal scale here is three times the horizontal scale in Fig. 2.*

$O(\varepsilon^{1/2})$. Thus, $O(\varepsilon^{-3/2})$ operations are required whether or not we use larger time steps in the smooth part of the flow. This difference in the constant factor is of course an important practical consideration but it does not result in asymptotically greater order CPU times. The situation changes drastically when formally second order difference operators are used. In this case the same analysis shows that locally adaptive time steps *are* effective in reducing CPU times.

The methods of Oliger and his students employ local, uniform spatial subgrids over which the computation can be effectively vectorized. It is easily seen from Fig. 3 that even though we allow an essentially arbitrary grid, in our example the grid naturally organizes itself into subregions of uniform grid spacing. Because of this, it is reasonable to believe that their methods will outperform our method on this problem; i.e., they may achieve a smaller error at a lower computational cost (if their adaptive grid criteria chooses the grid correctly—at least in Bolstad's early work, the ratio of maximum to minimum grid spacing is bounded by a constant, and so the argument used above to estimate the error in various parts of the region is not valid). Based on the "fingering" phenomena documented by Glimm and his co-workers, we suspect that inhomogeneities in media and higher spatial dimension may lead to grids that are not modeled well by small numbers of overlapping, uniform, subgrids. This must be verified experimentally, of course.

The algorithm was implemented in *C* on the Denelcor HEP using the synchronization macro package described in [24]. The architecture of the HEP is described in [16]; for our purposes each Process Execution Module (PEM) may be seen as a computer with an execution pipeline of length eight that executes instructions from processes in a common execution queue. One instruction from the queue is started every cycle that an instruction is waiting to be executed. When an instruction is completed, the next instruction from the same process is entered into the instruction queue. Processes waiting on semaphores busy-wait in a separate queue. In a one PEM system, all main memory accesses except for accesses to asynchronous variables and indexed stores go through a local memory interface that has a queue length of eight; all other memory accesses go through the general memory switch, which has an effective queue length of 24–30 (see [15]). One HEP may have several PEMs.

The speed-ups resulting from the parallel algorithm are presented in Table 1 and Fig. 4. Table 1 presents the clock times for the execution of the program for the Euler equations with $\varepsilon = 1/256$. Execution times were not directly measurable, and the clock

TABLE 1

Execution times		
Number of processes	Time (in seconds) for first pass	Time (in seconds) for second pass
1	165.589419	51.099661
2	86.486981	27.329106
3	60.209457	19.657619
4	48.109058	16.012172
5	40.244957	13.783819
6	35.079278	12.341126
7	32.348853	11.445739
8	29.046026	10.726242
9	27.370288	10.190968
10	25.556640	9.809485
11	24.225887	9.481623
12	23.330546	9.264341
13	22.942822	9.215509
14	22.728492	9.123390
15	22.503358	9.183468

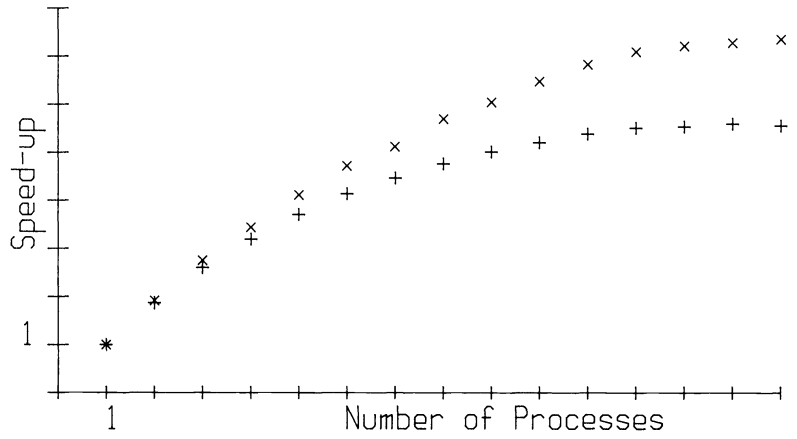


FIG. 4. *Speed-up versus number of processes for the Euler equations example. The points × represent the first pass of each time step, the points + represent the second.*

times were partially vitiated by the periodic execution of a system disk management routine for a fraction of a second. Figure 4 presents the speed-up versus the number of processes for the same problem.

Because of the various queues in the HEP architecture, with their different lengths, it is difficult to estimate the maximum possible speed-up for this algorithm independent of the task synchronization time. However, some indication may be given by an examination of the assembly code for the program. Ignoring synchronization code, fewer than 13% of all nonsynchronization instructions in the main execution path of the program are indexed store instructions. Furthermore, much of the time is spent in the library routines *pow*( ) and *sqrt*( ) (used to calculate the position of the points $\{\Gamma_k^i(s_k^i)\}$), which have mainly register to register instructions. If one assumes that the *sqrt* routine takes about 15 instructions and the *pow* routine takes about 40 instructions

(conservative estimates), this further lowers the fraction of instructions that use the general memory switch to an estimated 9%. A crude estimate of the maximum speed-up possible in a program may be obtained by taking an average of the queue lengths of the instructions executed, weighted by the number of instructions that go through each queue. In our case this would be less than $0.91 \times 8 + 0.09 \times 24$, or about 9.44. The final speed-up of 7.36 achieved for the first computational passes of each time step, on a problem of relatively small size, is very good.

The second computational pass of each time step did not speed up as much as the first because of the spatial dimension of the underlying physical problem. When a shock moves along the x-axis, most often only two or four nodes are added in front of the shock in one time step. These nodes are added at the deepest part of the tree, after most of the tree has been processed. There is therefore about a factor of two parallelism available during the latter part of the tree modification and finite difference pass. This effect would disappear for two-dimensional problems, for which approximately $O(N^2)$ nodes would be added along a moving front at each time step, where the average grid size where the solution is smooth is $O(N^{-1})$. Thus, a much greater amount of parallelism would be available in the two-dimensional problem.

5. Conclusions. We present a programming methodology and tree partitioning algorithm that allows us to efficiently compute the solution of a nonlinear partial differential equation with moving discontinuities in parallel on a currently available machine. It has been shown empirically in [22] that in some cases the present adaptive method achieves asymptotic speed-up over uniform fixed grid methods with the same difference scheme. This work shows that even though the fixed grid scheme may be simple to execute on vector machines, parallel processing offers hope of executing adaptive grid methods at supercomputer speeds. The use of these techniques with better difference schemes may improve the approximation schemes for the problems by an order of magnitude. Because the amount of parallelism available depends on tree size, two-dimensional problems, with larger and more complex grids, offer the promise of even more parallel efficiency.

Acknowledgments. Danny Sorensen offered much help and many suggestions during the course of this work. The computer coding was done on the Denelcor HEP at Argonne National Laboratory. The last two references to more recent papers were suggested by a referee.

REFERENCES

[1] R. E. BANK, *The efficient implementation of local mesh refinement algorithms*, in Adaptive Computational Methods for Partial Differential Equations, I. Babuska, J. Chandra and J. Flaherty, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1983, pp. 74–84.

[2] J. B. BELL AND G. R. SHUBIN, *An adaptive grid finite difference method for conservation laws*, J. Comput. Phys., 52 (1983), pp. 569–591.

[3] M. BERGER AND J. OLIGER, *Adaptive mesh refinement for hyperbolic partial differential equations*, J. Comput. Phys., 54 (1984), pp. 484–512.

[4] M. BERGER, *Data structures for adaptive mesh refinement*, in Adaptive Computational Methods for Partial Differential Equations, I. Babuska, J. Chandra and J. Flaherty, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1983, pp. 237–251.

[5] J. H. BOLSTAD, *An adaptive finite difference method for hyperbolic systems in one space dimension*, Ph.D. dissertation, Lawrence Berkeley Lab. LBL-13287 (STAN-CS-82-899), 1982.

[6] C. DE BOOR, *Good approximation by splines with variable knots*, in Spline Functions and Approximation Theory, A. Meir and A. Sharma, eds., ISNM, v. 21, Birkhauser-Verlag, New York, 1973, pp. 57–72.

[7] ———, *Good approximation by splines with variable knots* II, in Lecture Notes in Mathematics 363, Springer-Verlag, New York, 1974, pp. 12-20.
[8] P. BRINCH HANSEN, *Operating System Principles*, Prentice-Hall, Englewood Cliffs, NJ, 1973.
[9] J. DOUGLAS, JR. AND M. F. WHEELER, *Implicit, time-dependent variable grid finite difference methods for the approximation of a linear waterflood*, Math. Comp., 40 (1983), pp. 107-122.
[10] B. ENGQUIST AND S. OSHER, *Stable and entropy satisfying approximations for transonic flow calculations*, Math. Comp., 34 (1980), pp. 45-75.
[11] G. N. FREDERICKSON, *Data structures for on-line updating of minimum spanning trees, with applications*, SIAM J. on Comput., 14 (1985), pp. 781-798.
[12] A. HARTEN, *High resolution schemes for hyperbolic conservation laws*, J. Comput. Phys., 49 (1983), pp. 357-393.
[13] A. HARTEN AND J. M. HYMAN, *Self-adjusting grid methods for one-dimensional hyperbolic conservation laws*, J. Comput. Phys., 50 (1983), pp. 235-269.
[14] G. W. HEDSTROM AND G. H. RODRIGUE, *Adaptive-grid methods for time-dependent partial differential equations*, in Multigrid Methods, W. Hackbusch and U. Trottenberg, eds., Springer-Verlag, New York, 1982, pp. 474-484.
[15] H. F. JORDAN, HEP *architecture, programming, and performance*, in Parallel MIMD Computation: the HEP Supercomputer and its Applications, J. S. Kowalik, ed., MIT Press, Cambridge, MA, 1985, pp. 1-41.
[16] J. S. KOWALIK, Ed., *Parallel* MIMD *Computation: the* HEP *Supercomputer and its Applications*, MIT Press, Cambridge, MA, 1985.
[17] P. D. LAX, *Hyperbolic systems of conservation laws and the mathematical theory of shock waves*, SIAM Regional Conference Lectures in Applied Mathematics, Number 11, 1972.
[18] R. J. LEVEQUE, *A large time step generalization of Godunov's method for systems of conservation laws*, SIAM J. Numer. Anal., 22 (1985), pp. 1051-1073.
[19] ———, *Convergence of a large time step generalization of Godunov's method for conservation laws*, Comm. Pure Appl. Math., 37 (1984), pp. 463-478.
[20] ———, *Large time step shock-capturing techniques for scalar conservation laws*, SIAM J. Numer. Anal., 19 (1982), pp. 1091-1109.
[21] B. J. LUCIER, *A moving mesh numerical method for hyperbolic conservation laws*, Math. Comp., 46 (1986), pp. 59-69.
[22] ———, *A stable adaptive numerical scheme for hyperbolic conservation laws*, SIAM J. Numer. Anal., 22 (1985), pp. 180-203.
[23] ———, *Error bounds for the methods of Glimm, Godunov and LeVeque*, SIAM J. Numer. Anal., 22 (1985), pp. 1074-1081.
[24] E. L. LUSK AND R. A. OVERBEEK, *Use of monitors in* FORTRAN: *a tutorial on the barrier, self-scheduling DO-loop, and askfor monitors*, in Parallel MIMD Computation: the HEP Supercomputer and its Applications, J. S. Kowalik, ed., MIT Press, Cambridge, MA, 1985, pp. 367-411.
[25] K. MILLER, *Alternate modes to control the nodes in the moving finite element method*, in Adaptive Computational Methods for Partial Differential Equations, I. Babuska, J. Chandra and J. Flaherty, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1983, pp. 165-184.
[26] ———, *Moving finite elements*, II, SIAM J. Numer. Anal., 18 (1981), pp. 1019-1057.
[27] J. OLIGER, *Approximate Methods for Atmospheric and Oceanographic Circulation Problems*, in Lecture Notes in Physics 91, R. Glowinski and J. Lions, eds., Springer-Verlag, New York, 1979, pp. 171-184.
[28] S. OSHER AND S. CHAKRAVARTHY, *High resolution schemes and the entropy condition*, SIAM J. Numer. Anal., 21 (1984), pp. 955-984.
[29] S. OSHER AND R. SANDERS, *Numerical approximations to nonlinear conservation laws with locally varying time and space grids*, Math. Comp., 41 (1983), pp. 321-336.
[30] S. OSHER AND F. SOLOMON, *Upwind difference schemes for hyperbolic systems of conservation laws*, Math. Comp., 38 (1982), pp. 339-374.
[31] J. PIKE, *Grid adaptive algorithms for the solution of the Euler equations on irregular grids*, to appear.
[32] W. C. RHEINBOLDT AND C. K. MESZTENYI, *On a data structure for adaptive finite element mesh refinements*, ACM Trans. Math. Software, 6 (1980), pp. 166-187.
[33] R. SANDERS, *The moving grid method for nonlinear hyperbolic conservation laws*, SIAM J. Numer. Anal., 22 (1985), pp. 713-728.
[34] G. A. SOD, *A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws*, J. Comput. Phys., 27 (1978), pp. 1-31.
[35] A. J. WATHEN, *Mesh-independent spectrum in the moving finite element equations*, to appear.
[36] A. J. WATHEN AND M. J. BAINES, *On the structure of the moving finite element equations*, IMA J. Numer. Anal., to appear.

[37] D. K. KNUTH, *The Art of Computer Programming. Volume* 1: *Fundamental Algorithms*, second edition, Addison-Wesley, Reading, MA, 1973.

[38] W. D. GROPP, *Local uniform mesh refinement on loosely-coupled parallel processors*, Yale Univ. Tech. Report YALEU/DCS/RR-352, 1985.

[39] W. D. HENSHAW, *The numerical solution of hyperbolic systems of conservation laws*, California Inst. of Technology, Ph.D. thesis, 1985.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 2, March 1987
© 1987 Society for Industrial and Applied Mathematics

INTERPRETING PARALLEL PROCESSOR PERFORMANCE MEASUREMENTS*

HARRY F. JORDAN†

Abstract. This paper discusses execution time versus number of simultaneous operations in parallel computing systems. The main focus is on shared memory multiprocessors. A model for execution time as a function of the number of processes used in a computation is developed. The model addresses the effect of sequential code, code which can be executed by only a limited number of processes, hardware limits to speedup, critical section synchronization overhead and the influence of task granularity. The model is shown to correspond very closely to experimental measurements of execution time on the HEP pipelined, shared memory multiprocessor. Use of the model as an analysis tool in complex parallel programs is indicated.

Key words. parallel processor, performance, speedup, synchronization, multiprocessor

AMS (MOS) subject classifications. 68A05, 68J10

1. Introduction. Programs written to be executed by a large number of processes are often formulated so that they can be run with different numbers of processes. In addition to the desire to measure speedup by running the program with differing numbers of processes, pressure for process number independence arises from several other sources. Portability considerations and the fact that some processors may be unavailable at a given time in a multiprocessor system argue for making the number of processes a parameter in a parallel program. Furthermore, with large numbers of processes, it is difficult to formulate programs having a detailed dependence on the number of processes executing them. In this paper, evidence is presented that, given a program which can be executed by any number of processes, its execution time as a function of this number gives significant insight into the program structure. It can be used to estimate program parameters such as degree of parallelization, task granularity and synchronization overhead. Some of these parameters may be hard to predict directly from the program code while the known values of others can be used to support the accuracy of the performance models presented below.

2. Effects of sequential code. The simplest model shows the general shape of the execution time versus number of processes curve for an imperfectly parallelized program. The model, discussed by Ware [1] in 1972, can be viewed as separating a program into a perfectly parallelized and a strictly sequential section. For small numbers of processes, many parallel programs exhibit a behavior of execution time versus number P of processes of the form

$$T(P) = c_1 + \frac{c_2}{P}.$$

A simple least squares fit to determine c_1 and c_2 can be applied to a suitable portion of the execution time versus number of processes curve to estimate the degree of parallelization. Sharp corners in this rapidly decreasing portion of the curve can be caused by sections of the program in which parallelism is limited to, say, K processes. The execution time then behaves as

$$T(P) = c_1 + \frac{c_2}{\min(K, P)} + \frac{c_3}{P}.$$

* Received by the editors November 18, 1985; accepted for publication (in revised form) April 18, 1986. This article was presented at the Second SIAM Conference on Parallel Processing for Scientific Computing that was held in Norfolk, Virginia on November 18-21, 1985.

† Computer Science Department, Indiana University, Bloomington, Indiana 47405-4101.

The existence of a sharp change in slope of $T(P)$ is a good indication that a significant portion of the execution time is devoted to a program section which is limited in the number of computations that can be done in parallel.

If multiprogramming or pipelining is used to allow more parallel computations to be specified than there are physical hardware units, the speedup is limited by the number U of hardware units. The denominator in the parallel time term changes from P to min (U, P) much as in the limited program parallelism case. The $T(P)$ curve in this case has a flat section representing the minimum execution time possible with the hardware. Any increase in the number of processes can only increase execution time as a result of extra computation, synchronization or process management associated with more processes. For the purposes of performance measurement it may be quite interesting to observe the $T(P)$ curve for P larger than that giving the minimum execution time, since this gives an indication of overhead associated with each process on parallelization.

The effect of limited parallelism is easier to see in the speedup curve, which shows a predominantly linear behavior with slope changing at $P = K$. An example of this behavior was seen in measurements made on a parallelized version [2] of the MA28 sparse LU-decomposition program originally written by Duff [3]. This program was written for the HEP [4] pipelined multiprocessor. It was known from analysis of the code that the search for an optimal pivot using the Markowitz criterion contained a parallelism limit. A critical section needed to parallelize the pivot search loop had a size which was one sixth of that of the loop body for the matrix size and percent sparsity in the measurement run. Thus, no more than six processes could be productively used in the pivot search. The other large parallel section, the reduction of nonpivot rows, did not have as low a parallelism limit. The speedup curve for this program is shown in Fig. 1. The parallelism limit at six processes is clearly visible. The hardware

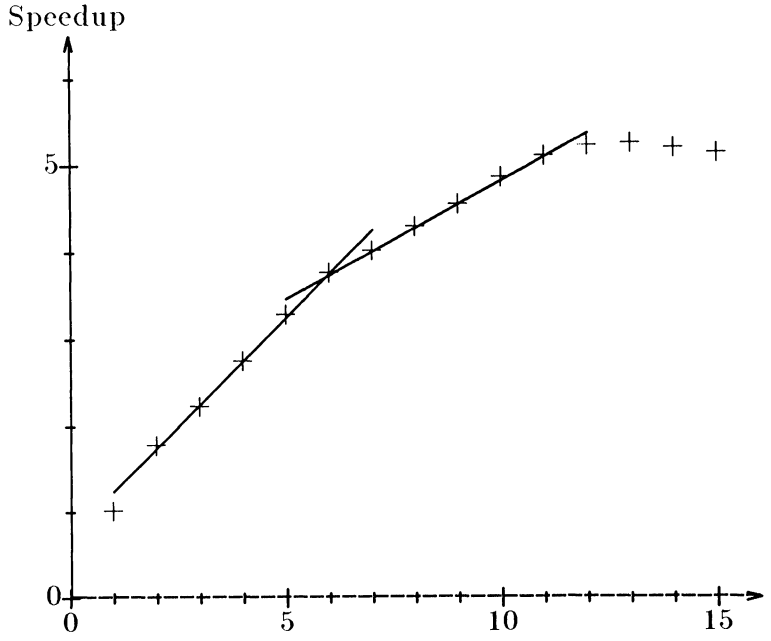


FIG. 1. *Speedup in a parallelized sparse LU-decomposition.*

parallelism limitation, U, on the HEP computer is 10-12, depending on instruction mix, and is also apparent in Fig. 1.

3. Synchronization overhead. Several common synchronization operations can be viewed functionally as critical sections of code, which can be executed by only one process at a time. We can alter the Ware model to allow for this. Assume that a fraction f of the sequential code cannot be parallelized and that the cost of parallelizing the fraction $(1-f)$ of the code is a critical section of length t_c in each process. The amount of useful work done by each process in the parallel section is

$$\frac{(1-f)\times T(1)}{P}.$$

It may be possible for all other processes to complete their critical sections while a given process is doing its fraction of useful work. This can only occur when the amount of useful work exceeds the number $P-1$ of other processes times t_c. Conflict for critical section access can occur in this case, but a justification for using the best case analysis of perfect overlap is that, if the critical section is made up of short segments executed in a loop, the delays occurring on the first pass cause the processes to fall into synchronism. Thus, no delays occur on subsequent passes through the loop, and the best case timing is well approximated. In this case, the total execution time can be written

$$T(P)=f\times T(1)+(1-f)\times\frac{T(1)}{P}+t_c,$$

provided

$$(1-f)\times T(1)\geqq P\times(P-1)\times t_c.$$

When the amount of time a process spends doing useful work is shorter than the total time spent by other processes sequentially passing through their critical sections, the useful work is masked by critical section delay, and the best case time becomes

$$T(P)=f\times T(1)+P\times t_c,\qquad (1-f)\times T(1)<P\times(P-1)\times t_c.$$

Combining the cases, the best case analysis gives

$$T(P)=f\times T(1)+t_c+\max\left((1-f)\times\frac{T(1)}{P},(P-1)\times t_c\right).$$

This expression has a minimum at

$$P_m\approx\frac{1}{2}+\left(\frac{(1-f)\times T(1)}{t_c}\right)^{1/2},\qquad \frac{t_c}{(1-f)\times T(1)}\ll 1$$

and rises linearly with P for $P>P_m$.

The above analysis assumes no limit on the number P of processes that may execute in parallel. In a multiprogrammed or pipelined parallel processor the limit U on the number of parallel hardware units must be taken into account. The addition of further processes does not speed the completion of the $(1-f)\times T(1)$ workload which can be parallelized but does increase the number of processes executing critical sections. Taking this limit into account, the execution time model becomes

$$T(P)=f\times T(1)+t_c+\max\left(\frac{(1-f)\times T(1)}{\min(P,U)},(P-1)\times t_c\right).$$

Thus a linear rise in execution time as P becomes large is an indication of critical section type synchronization, and the length of the critical section can be estimated from the slope of the linear rise.

A parallel program to solve the Laplace equation with Dirichlet boundary conditions on a square was written to test a parallel Successive Over Relaxation (SOR) algorithm [5] on the HEP computer. Multiple processes were used to relax the rows of a 100 by 100 grid, and the maximum change in solution value over the grid was computed cooperatively by processes relaxing different rows by using a critical section. The execution time showed all the salient features of the above formula. An estimate of the model parameters from the data yielded the formula

$$T(P) = 9.15 + 0.73 + \max\left(\frac{215.10}{\min(P, 10.51)}, 0.73(P-1)\right).$$

This model is plotted as a continuous line in Fig. 2 along with the measured execution times, marked by "+". The parallelism limit U was determined from the height of the flat section and t_c from the slope of the linearly rising tail while $f \times T(1)$ and $(1-f) \times T(1)$ came from a least squares fit of the simple Ware model formula to data in the range $1 \leqq P < U$. The last digits of the estimated parameters were affected slightly by the analysis to be presented in the next section. While the code for the Laplace SOR was not massive, detailed instruction counts were not done, but the estimated parameters are quite consistent with program section sizes estimated from the Fortran source code and from the parallelism limits measured in other HEP programs.

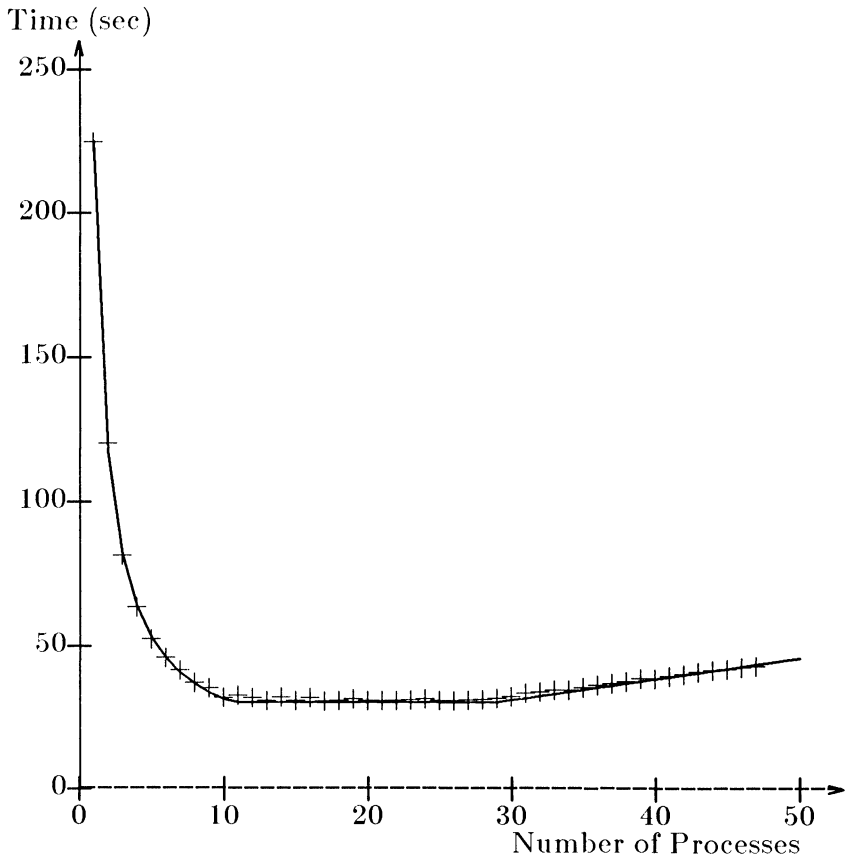


FIG. 2. *Laplace* SOR *measurements and model.*

4. Task granularity correction. Another measurable effect occurs when the parallel work has a fixed granularity. If a parallel program consists of serial work $T_s = f \times T(1)$ and parallel work of N indivisible units of size t_u, then N is said by Mohan [6] to be the logical parallelism of the program. A simple program with this behavior might be one with some sequential code followed by a DOALL [7] of order N. The machine model for the execution of this program has two parallelism parameters. P is the instruction parallelism and represents, in an MIMD machine, the number of instruction streams used by the program and, in an SIMD machine, the vector length for a single vector instruction (length of vector registers where this applies). U is a limit on the number of hardware units that can operate in parallel on the P "processes" associated with the instruction parallelism. We assume a vanishing "process switch" overhead as occurs in pipelined processing where "process switch" merely corresponds to issuing the next unit of computation into the pipeline. We also ignore the filling and emptying of the pipe which is really a part of process synchronization. It is absent, for example, in vector chaining and in pipelined multiprocessors when there is no interprocess interaction. Since the units t_u are not subdividable, the N units are divided by the process structure into $\lfloor N/P \rfloor$ units, on which P processes work simultaneously, and $N \bmod P$ units, on which at most $N \bmod P$ processes can be active. Including this effect, the execution time is given by a modified version of the hardware limited Ware model

$$T(P) = T_s + \frac{P\lfloor N/P \rfloor t_u}{\min(P, U)} + \frac{(N \bmod P)t_u}{\min(N \bmod P, U)}.$$

For $P < U$, the granularity effect results in flat portions of the execution time curve between successive values of P which divide N evenly, since the addition of further processes does not reduce execution time until the next even divisor is reached. In a hardware limited parallelism situation, it is helpful to view the effect as a difference between the execution times including and ignoring the granularity. The difference between the time with arbitrarily divisible work and that with N indivisible units is

$$\delta(P) = \frac{(N \bmod P)t_u}{\min(N \bmod P, U)} - \frac{(N \bmod P)t_u}{\min(P, U)}.$$

This difference can be viewed as a correction to the hardware limited Ware model. $\delta(P)$ is zero either if P divides N evenly or if $U \leqq N \bmod P$. When nonzero, its value can be written

$$\delta(P) = t_u\left(1 - \frac{N \bmod P}{\min(P, U)}\right).$$

In the range of P where hardware is not a limitation, $P < U$, the correction term is always nonzero except when P evenly divides N. In the hardware limited region, the correction is zero unless $N \bmod P < U$. In this region, deviations from the hardware limited Ware model should be observed whenever the total work divides across processes with a small remainder.

If there is a flat region of minimum execution time, these divisibility corrections can be easily seen. If the logical parallelism of the program is not obvious from the code it may be possible to deduce it from a remainder analysis. Since the size of the correction $\delta(P)$ is a simple fraction of t_u, an even easier parameter to extract is the size, in execution time, of the granule.

Mohan's thesis [6] thoroughly explores the granularity correction as the effect of logical parallelism. His analysis does not cover a hardware parallelism limit and is thus characteristic of the situation for $P < U$. Flat plateaus in execution time are clearly shown in both his models and simulations. The Laplace SOR program for HEP, described above, clearly exhibits the effects of the granularity correction, both in the hardware limited portion and in the rising tail resulting from critical section conflict. Figure 3 shows a detailed view of the measured execution times (again denoted "+") along with a solid line plot of the model equation including the correction term

$$T(P) = 9.15 + 0.73 + \max\left(\frac{215.10}{\min(P, 10.51)}, 0.73(P-1)\right) + \delta(P).$$

The parameter t_u in $\delta(P)$ was not estimated separately but taken to be

$$t_u = \frac{(1-f) \times T(1)}{100},$$

since processes execute 100 tasks corresponding to rows of the grid. It can be seen that the overall fit of the complete equation is quite good. As expected, the positions of the visible granularity peaks are very well predicted since they depend only on remainder analysis. The height of the peaks is somewhat less well predicted.

5. Conclusions. Some fairly simple models have been exhibited above which not only predict the performance of programs with simple structure on parallel processors, but which can also be used to analyze performance measurements on more complex programs for the purpose of estimating parallelism related parameters of the program

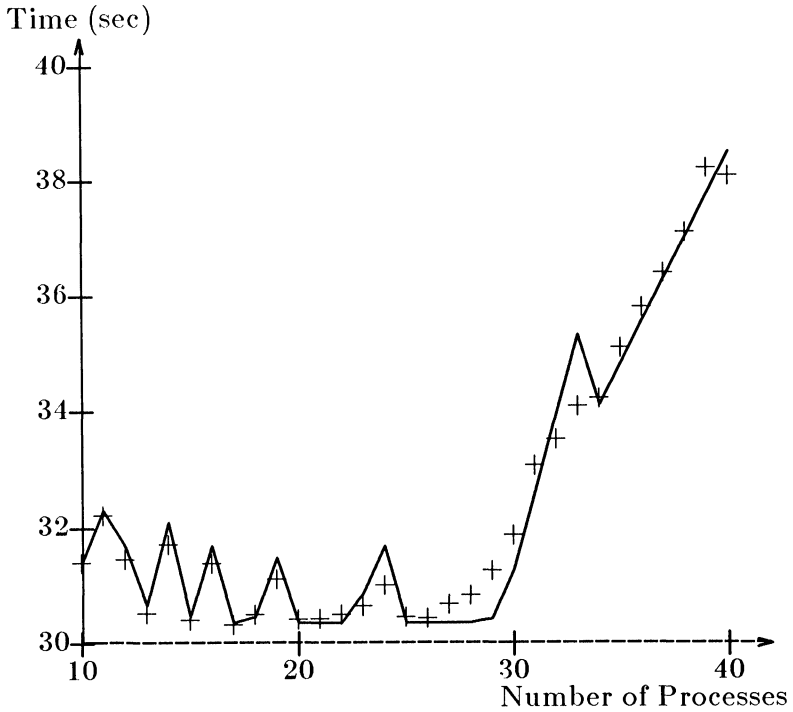


FIG. 3. *Laplace* SOR *measurements and model with granularity term.*

structure. These models have been observed to be very accurate in practice and, in simple cases, the parameters estimated from the $T(P)$ curve have matched well with values obtained by direct analysis of the code.

The task granularity model is particularly accurate in giving the positions of the resultant peaks. This effect also appears in register based vector machines in programs where vectors are longer than the vector registers. The hardware parallelism limit also shows up as the effect of pipeline length in pipelined vector machines. It is harder to study in the vector case because pipeline startup strongly influences the measurements in the region where this limit takes effect.

The synchronization overhead is particularly interesting because many synchronization mechanisms impose the same type of process waiting involved in the critical section model. The hardware parallelism limit in the pipelined MIMD architecture characteristic of the HEP has the effect of suppressing other influences and making the synchronization overhead apparent at large numbers of processes.

REFERENCES

[1] W. H. WARE, *The ultimate computer*, IEEE Spectrum, March 1972, pp. 84–91.

[2] G. ALAGHBAND AND H. F. JORDAN, *Parallelization of the* MA28 *Sparse Matrix Package for the* HEP, Report CSDG 83-3, Computer Systems Design Group, Electrical and Computer Enging. Dept., Univ. Colorado, Boulder, 1983.

[3] I. S. DUFF, MA28—*A set of* FORTRAN *subroutines for sparse unsymmetric linear equations*, AERE Report R.8730, H.M. Stationery Office, London, 1977.

[4] H.F. JORDAN, *Experience with pipelined multiple instruction streams*, Proc. IEEE, Vol. 72, No. 1, January 1984, pp. 113–123.

[5] N. R. PATEL AND H. F. JORDAN, *A parallelized point rowwise successive over-relaxation method on a multiprocessor*, Parallel Computing, 1, 3, 4, December 1984.

[6] JOSEPH MOHAN, *Performance of parallel programs: Model and analysis*, Ph.D. thesis, Carnegie-Mellon Univ., Pittsburgh, PA, 1984.

[7] D. A. PADUA, D. J. KUCK AND D. H. LAWRIE, *High speed multiprocessors and compilation techniques*, IEEE Trans. Comput., C-29 (1980), pp. 763–776.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 2, March 1987
© 1987 Society for Industrial and Applied Mathematics

HYPERCUBE ALGORITHMS AND IMPLEMENTATIONS*

OLIVER A. MCBRYAN† AND ERIC F. VAN DE VELDE‡

Abstract. Parallel algorithms are presented for important components of computational fluid dynamics algorithms along with implementations on hypercube computers. These programs, used to solve hyperbolic and elliptic equations, achieve high efficiency on 5- and 7-dimensional hypercubes.

For elliptic equations, a parallel preconditioned conjugate gradient method is described which has been used to solve pressure equations discretized with high order finite elements on irregular grids. A parallel full multigrid method and a parallel fast Poisson solver are also presented. Hyperbolic conservation laws have been discretized with parallel versions of finite difference methods and with the random choice method.

The performance of these algorithms is analyzed in terms of machine efficiency, communication time, bottlenecks and software development costs. A key aspect of this work is the development of a library of parallel operators for distributed vectors and matrices, efficient for both full and sparse data. The implementation of these operators on hypercubes is described along with measurements of communication effects. Using the library, the PDE algorithms mentioned above have been implemented on both serial computers and on hypercubes without *any* code modification. All interprocess communication is hidden in library routines. A general parallel computer simulator is described along with its use in the development of the algorithms.

The relation of the model problems solved here to the more complex physical problems encountered in real fluids is discussed. Techniques are developed for comparing the behavior of an algorithm on different architectures as a function of problem size and local computational effort.

Key words. hypercube, parallel, linear algebra, multigrid, conjugate gradient, PDE, matrix

AMS(MOS) subject classifications. 65W05, 65F05, 65F10, 65M99, 65N99, 65N30

1. Introduction.

In this paper we describe the implementation of partial differential equation solvers on hypercube processors, part of an ongoing effort to exploit parallelism in the solution of equations arising in Computational Fluid Dynamics. We present results of computations run on the Caltech Mark II Hypercube parallel processor and on the Intel iPSC d7 processor. These machines have respectively 32 and 128 processors connected on 5- and 7-dimensional hypercube networks. We have previously described the implementation of some of these algorithms on the shared memory Denelcor HEP parallel computer [1]–[8].

Our goal is to develop algorithms that make close to optimal use of available parallelism on a range of realistic problems. Our studies have concerned the development of parallel algorithms for elliptic, parabolic and hyperbolic equations. These algorithms include important components of real fluid dynamics codes. All of our algorithms are for equations with two space dimensions, though extension to three dimensions is straightforward in each case. While most of the computational results described in the paper relate to PDE solution, the techniques used are much more general. The PDE solvers are written entirely in terms of a general-purpose library of parallel operations which automates the construction of distributed data structures and provides parallel operations on them. Much of the paper (§§ 3–6) will concentrate on this library, describing its implementation on hypercubes, and the techniques used to make it efficient for such architectures. In § 1.1 we give an overview of this library,

* Received by the editors January 16, 1986; accepted for publication (in revised form) May 22, 1986. This article was presented at the Second SIAM Conference on Parallel Processing for Scientific Computing that was held at Norfolk, Virginia, November 18–21, 1985.

† Courant Institute of Mathematical Sciences, New York University, New York, New York 10012, and Los Alamos National Laboratory, Los Alamos, New Mexico 87545. The work of this author was supported in part by U.S. Department of Energy contract DE-AC02-76ER03077, by National Science Foundation grant DMS83-12229, and by the T-Division, Los Alamos National Laboratory.

‡ Courant Institute of Mathematical Sciences, New York University, New York, New York 10012.

followed in § 1.2 by an introduction to the PDE problems we have solved. Section 1.3 discusses the relation of the model problems solved in this paper to parallelization of complex physical problems and introduces a method for performance analysis that provides useful comparison of parallelization efficiency between different algorithms and machines. In § 1.4 we make some comments relevant to comparison of our studies on the Caltech Hypercube and on the Intel iPSC. Section 1.5 surveys the remainder of the paper in more detail.

1.1. Portable parallel software and libraries. A primary long-range aim of our work is the development of portable programming methodologies for numerical applications. The goal is to be able to use the same code on a wide range of different processors, including serial, SIMD and MIMD machines, with both shared memory and message-passing architectures. We report progress in this area. In particular, we have implemented a library of vector and matrix operations for hypercubes. This library deals with globally distributed vectors and matrices. It includes allocation routines for such objects, standard linear algebra operators such as inner products, vector maximum, vector sums and products, matrix transpose, matrix-vector and matrix-matrix products and rank one updates. The allocation routines allow vectors and matrices to be distributed in various ways across the processors, and routines are provided to convert between different representations. The library is efficient for both sparse matrices such as those encountered in discretizing partial differential equations and for full matrices. A second library involves routines for object-level communication, both between hypercube nodes as well as with the host processor. All of our applications are implemented in terms of these two libraries.

There are several advantages that result from the use of these libraries, beyond the normal improvement in software modularity. Some of the library routines involve substantial and complex inter-process communication on the hypercube—for example the *inner_product*, *transpose* and *matrix_vector* routines for distributed vectors and matrices. These library routines may be written for different parallel processors as well as for serial and vector processors and represent a powerful mechanism for hiding communication and network dependence in programs. As a result, most of our applications algorithms can be run on both serial machines and hypercubes with *no* code modification. In particular we identify a class of algorithms, which we term *transpose-split algorithms* which have this property. We present a general discussion of such algorithms as well as several applications to hyperbolic and elliptic problems.

We have also observed that a large number of algorithms, when written in terms of our library, take the form of SIMD rather than MIMD programs. In particular the code running in each node is identical and does not refer in any way to location in the network (of course such dependencies are buried in the library routines). Furthermore we do not use cooperating host programs which only tend to complicate the clarity of algorithms, and introduce critical sections into programs. We believe that by casting programs in an SIMD form, portability is greatly enhanced, and programs can be easier to debug and understand.

A critical issue here is the semantics of the vector library routines. We illustrate this with the example of the inner product of two distributed vectors. In our approach, the inner product operation is called simultaneously by all processors, and all return with the same global value. We do not allow the possibility that a single processor requests a global inner product. Consequently all processors compute quantities such as inner products that are used to determine when to terminate an iteration. Each processor independently decides to terminate its iteration, although of course all

terminate at the same point since the same inner product value is obtained at all nodes. An alternate approach would be to assign one processor (or the host) as a control processor to monitor convergence. On detecting sufficient accuracy that processor would then notify all others to terminate. This scheme does not have an SIMD form since one processor is distinguished. There is also no real time savings in this scheme because although most processors avoid computing with the inner product, they have to wait while the control processor computes the inner product and notifies them to continue iterating or to terminate.

1.2. Elliptic and hyperbolic equations. Once the parallel libraries are available, implementation of various solution methods for PDE becomes straightforward. All of the basic parallel data structures required for storing data across the processors are automatically generated by calls to library routines. Further library routines are available to perform all operations on these data structures that require any communication. The remaining code is then related to only one processor, and is just standard serial code—we have not developed any new numerical methods for solving PDE. The algorithms we have used include random choice and finite difference methods for hyperbolic equations, multigrid methods for Poisson-type equations and preconditioned conjugate gradient methods for arbitrary-order finite element discretizations of elliptic equations on irregular grids.

The hyperbolic equations are the easiest to parallelize, and in fact we have parallelized various hyperbolic solvers by adding as little as one line of code to the corresponding serial code (see § 8.3). We have used several parallelization strategies for hyperbolic equations and we compare these from the point of view of computational efficiency. The simplest method is not the most efficient. However it is quite sufficient as computation already substantially dominates communication costs even for this method. From a software point of view we prefer the simpler approach—trading a small amount of efficiency for a large decrease in software complexity. Furthermore we have used the same parallelization strategy, which we call *transpose splitting parallelization*, for various other applications, including an FFT-based fast Poisson solver (see §§ 7 and 13). The basic observation underlying this approach is that an efficient parallel matrix transpose can be used to parallelize a numerical algorithm of ADI or operator splitting type.

Another general parallelization technique, which we call *areal decomposition*, is applicable to both hyperbolic equations and to multigrid methods and other relaxation schemes. Here a grid is decomposed into rectangular subgrids which are distributed in an areal (or volume) manner to a set of processors. Extra boundary layers are provided around each subgrid and are used to store duplicate data recording values at grid points in neighboring processors out to a specified distance, typically 1 grid point. A basic library routine is used to globally allocate such data structures and another routine (*shuffle*) can be called to update the boundary duplicates at any time. The combination of the data structure and the *shuffle* operation simulates a shared memory extremely closely—in fact it does simulate one exactly for grid operators whose stencil range does not exceed the buffer range. Efficiency depends on the aspect ratio of the subgrids used and we measure this area-perimeter effect in several applications.

1.3. Real problems and performance comparisons. For the most part we have studied model problems which have simpler governing equations than those for realistic physics. We have studied the issue of how effectively real problems may be parallelized. Real physical equations differ from our model equations in complexity. For example

expensive equations of state may need to be solved at every grid point. Because of the resulting increase in computation time per grid point, with little or no increase in communication time, computational efficiency will in general be higher for many realistic problems than for our model problems. Since the model problems already behave extremely well, the prognosis for the real problems looks excellent.

We have studied this question using a model hyperbolic code that does nothing but compute *NFLOPS* floating point operations per grid point and performs the standard inter-process communication typical of hyperbolic solvers. We then study the computational efficiency of this code as a function of grid size, number of processors and *NFLOPS*. The conclusion is that attaining high efficiency is extremely easy even on very coarse grids, in fact efficiencies over 90% are attainable on the Intel iPSC with *NFLOPS* of under 100, on grids so coarse that there are only a few dozen grid points per processor (see § 9 for details). Caltech Hypercube efficiencies are generally even higher. We obtain efficiency performance curves for various grid sizes that essentially characterize the behavior of the hypercube in question for a very wide range of numerical processes. Given an estimate of *NFLOPS* we can predict performance of an algorithm on various grids or with increasing numbers of processors. Furthermore comparing such curves for different algorithms gives an immediate comparison of the algorithms over the range of parameters. Plotting equivalent curves for different processors would provide a useful way to compare hardware designs over a range of different problems.

Variations in hardware design can play a major role in the behavior of algorithms, even among machines with the same network configuration. Critical parameters include the amount of memory per processor, the network communication rate, the overhead for sending short messages and the overall balance between these quantities and the processor speed. This has shown up very clearly in our work with the Caltech Hypercube and the Intel iPSC. Despite the larger memory and somewhat faster processing speed of the iPSC, the Caltech machine is more successful for most algorithms because there is almost no overhead for short messages, whereas with the iPSC all messages up to 1024 bytes in length take essentially the same time. The question of communication startup cost is closely tied to the available memory. In the current iPSC design, the memory available (about 250K bytes per node) is such that for many algorithms, communication is just beginning to be efficient even in the largest problems that can fit on the processors. For example, if a square subgrid of maximal size is stored on an iPSC processor, it will contain about 2^{16} real-valued grid points and consequently have edges containing about 256 grid points (1024 bytes of data). A communication in which a whole edge is communicated at once to another processor is then barely in the efficient range. In practice, memory is required for program storage and other data so that the situation is actually less favorable. The Caltech Hypercube does not suffer from this problem. In fact on that machine even individual boundary data points of a subgrid may be efficiently transferred to a neighboring processor. It follows that such *areal decompositions* of grids are substantially more favorable on the Caltech machine.

In analyzing algorithms in the sequel we will comment where appropriate on these issues. Such considerations play an especially important role in the asymptotic analysis of algorithms. An otherwise accurate analysis may be seriously flawed in practice if communication startup overhead is omitted. In § 2.3 we introduce three critical parameters α, β and γ which characterize the communication and computation performance of parallel computers. These quantities are used repeatedly throughout the paper.

1.4. Caltech Hypercube vs. Intel iPSC. We have referred above to advantages of the Caltech design over the Intel iPSC. The iPSC has advantages too, such as more

memory and processors. The results of our experiments performed on the Caltech and Intel Hypercubes are not for the most part directly comparable. We describe now some of the reasons for this.

We performed all of the Caltech Hypercube experiments prior to any of the iPSC experiments. The former experiments were performed with generally far inferior algorithms to the latter. In part we had learned from the Caltech experiments, and in part we were forced to develop better algorithms by the difficulties of dealing with the iPSC communications overhead. As an example, when we first moved a multigrid code from the Caltech Hypercube to the Intel iPSC it ran 100 times *slower* on the latter machine! Analysis showed that control messages being sent between the host computer and the node processors were dominating all computation. The fundamental problem was that control messages were typically a few words long and were therefore hundreds of times less efficient than on the Caltech cube. Another problem appeared to be that the host is time-shared and may not be ready to interrupt itself immediately on receipt of a message from the cube, effectively increasing communication costs with nodes. A third problem was that boundary data on the edges of subgrids were transferred to neighboring processors one point at a time. The combination of these three effects generated the hundred-fold slow-down. The final form of the iPSC algorithms involve no exchange of messages with the host, and buffer all boundary data to avoid sending single points. This should be taken as a very serious demonstration of the importance of coupling algorithm and hardware design. We have not had an opportunity to return to Caltech to repeat the old experiments with our optimized software. Undoubtedly efficiencies would be much closer to optimal in all cases with the new software. It is also difficult to compare the machines because of the different memory sizes. The largest problems that can fit in the Caltech Hypercube are relatively small iPSC problems, and as a result suffer more from communication startup effects than do larger iPSC problems.

1.5. Overview of contents. In § 2 we review the hardware and software environment of the Caltech and Intel Hypercubes. In § 3 we introduce two hypercube simulators we have developed that were crucial to the work described in this paper. In § 4 we describe the organization of the hypercube network into various topologies, including grids, trees and a hierarchy of canonical rings. Section 5 concentrates on the vector library introducing distributed vector formats and the operations defined on them, in particular, *inner_product* and *shift_vector.* Section 6 describes the various distributed matrix representations and the matrix library, including the important *transpose, matrix_vector* and *matrix_matrix* operations.

The second half of the paper deals with applications of the software described in §§ 3–6. In § 7 we introduce the concept of transpose-split algorithms, indicate their implementation on hypercubes and an analysis of their communication vs. computation costs. Section 8 discusses the solution of hyperbolic equations using several parallelization schemes. Section 9 presents the analysis of computational efficiency as a function of work per grid point and of grid size referred to earlier.

Section 10 serves as an introduction to the elliptic equation solvers in the remaining sections. Section 11 presents an implementation of a full multigrid elliptic equation solver for hypercubes. Section 12 describes a parallel preconditioned conjugate gradient solver for arbitrary-order finite element (or finite-difference) equations. Finally § 13 provides an application of transpose splitting to the development of a fast solver for the Poisson equation, which also serves as a parallel preconditioner for the conjugate gradient solver.

In each of the implementation sections we present results of actual hypercube implementations, along with analyses of efficiency and communications overhead.

It is difficult in a paper of this type to provide complete references to related work. The numerical analysis involved is not new, and we have given a few references to the literature in each case. The decomposition of PDE problems on parallel machines has been actively explored in recent years by many groups. For example, areal distribution schemes have been used frequently and we do not attempt to catalogue these. The development of portable linear algebra libraries may be traced back to LINPACK [9] and to related developments such as the BLAS routines [10]. Work on extension of LINPACK to vector and parallel machines has also been underway for some time [11]-[17]. Application of parallel computers to the solution of PDE has been discussed frequently [18]-[23]. Issues of software portability in parallel environments have been addressed by several groups, in particular we reference the work of Lusk and Overbeek [24], and Jordan [25]. A large source of algorithms for hypercubes are found in the many publications of the Caltech/JPL group, in particular see [26] and the annual reports of the CCCP project [27]. Application of hypercubes to linear algebra and to solution of elliptic equations is discussed in several reports from Yale [28]-[32]. For a very complete list of references to related work on parallel numerical algorithms see the excellent review of Ortega and Voigt [33].

2. Architecture and software.

2.1. The Caltech Cosmic Cube. The Cosmic Cube is a parallel processor developed by Geoffrey Fox and Charles Seitz [26], [34] at Caltech. The Caltech Mark II Hypercube consists of 2^D ($D=5$ *or* 6) independent processors, each with its own local memory. There is no shared memory available; the processors cooperate by message passing. Messages are passed over an interconnection network which is a *hypercube* in a space of dimension D, see Fig. 1. Processors are located at the vertices of the D-dimensional hypercube and adjacent vertices of the cube are connected by a communication channel along the corresponding edge. All data exchange between processors occurs in 8-byte packets along these cube edges which are asynchronous full duplex channels. In addition to the 2^D node processors, there is a host processor which acts as a control processor for the entire cube and also provides the interface between the cube and a user. All I/O to and from the cube must pass through the host, which is connected to

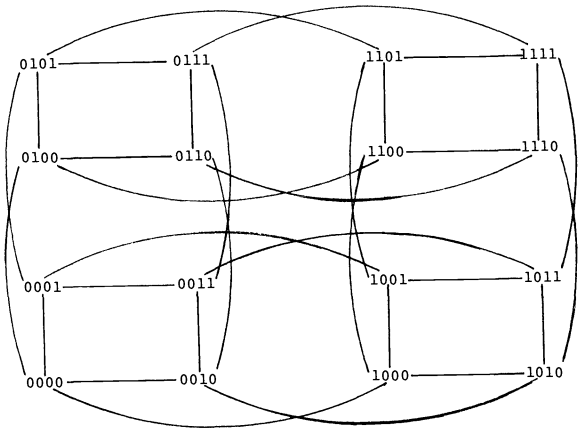


FIG. 1. *Representation of a* 4*-dimensional hypercube.*

one corner of the cube by an extra communication channel. The original Caltech design consists of a 64-node 6-dimensional hypercube utilizing Intel 8086/8087 processors with 128K bytes of memory at each node. This architecture has the advantage of being easily fabricated from standard components, and may be scaled up to much larger sizes (in powers of 2) with almost no change in design. Because of these features, machines of this type are likely to become widely available in the immediate future, whereas development of highly parallel global memory machines will take substantially longer.

A more advanced Caltech cube called the Mark III is now under development. This will have much faster processors at the nodes (Motorola 68020) and local memory per node will reach several megabytes. Other enhancements will be incorporated based on the experience with the prototype.

2.1.1. Caltech Hypercube programming. There are two fundamentally different communication modes available on the hypercube. In the *interrupt driven mode*, processors are interrupted by messages arriving from the communication channels. These messages are preceded by sufficient identification and destination information so that the processor can either forward them to another channel (if the current processor is not the destination) or process the incoming message (if the message has arrived at its destination). In the *crystalline operating system* messages are not preceded by address information. As a result, each processor has to know in advance exactly what communication pattern to expect. The latter system is unquestionably more efficient, although it is clearly also more restrictive. For the computations described in this paper the crystalline operating system was quite adequate. The parallelization of other algorithms (e.g. the local grid refinement algorithms discussed in our related papers [4], [6]) will likely require some interrupt driven communication protocols. For the remainder of the discussion we will refer only to the crystalline operating system when discussing Caltech Hypercube software.

The software for the cube consists of an operating system kernel, a copy of which resides in each processor, as well as a run-time library providing user access to the communication facilities. Typically, identical copies of a user program are down-loaded to all processors where they execute concurrently. All scheduling is accomplished through communication calls, so that some care is required to prevent locking situations from occurring.

As discussed previously, the D-cube has 2^D vertices with D edges radiating from each. Thus each processor sees D channels connecting it to its neighbors. The cube nodes are numbered in the range $[0, 2^D - 1]$, such that the D-digit binary representations of physically adjacent nodes differ only in 1 bit. The channels emanating from a node may then be numbered $0, 1, \cdots, D-1$ according to which bit differs in the binary node representations at either end of the channel. There is also an extra channel from node 0 to the intermediate host (referred to as the *IH* below) through which all communications to and from the cube pass. Data to be communicated between processors is sent in 8-byte packets, which are sufficient to encode all scalar data types. A set of system calls are available to node-resident programs which implement the required communication primitives for these packets. Similar system calls are available on the host to provide communication with the cube. In order to give the flavor of these system calls, we list some of the most important ones, along with their functions, in Table 1. In the table the notation *IH* and *ELT* are used, respectively, for the intermediate host and for the nodes, *data* represents an 8-byte item, *CUBE* denotes the channel to or from the *IH* to the hypercube and P is the number of processors.

TABLE 1
Crystalline cube communications.

wtIH(data,CUBE)	Write a data packet from the IH to node 0
rdsig(data)	Data sent from the IH is read by each node
wtres(data)	Data is sent from a node to the IH
rdbufIH(data, CUBE,P)	Read the union of data sent by all nodes to the IH into array datas
wtELT(data,chan)	Send data to the cube neighbor on channel chan
rdELT(data,chan)	Read data from the cube neighbor on channel chan
shift(ind,inc,outd,outc)	Write buffer outd onto channel outc, then read from channel inc into buffer ind

The *shift* operator allows either the in or out channel to be specified to be a nonexistent channel, denoted *NULLCHAN*. In this case *shift* simply omits the corresponding read or write operation. This feature is especially useful in the treatment of nonperiodic grid boundaries where a read or write beyond the boundary is not desired. We have found that most hypercube communication may be written in terms of the *shift* operator alone. In part this is because the *shift* operator actually includes the *rdELT* and *wtELT* operations which are special cases corresponding to setting *outc* = *NULLCHAN* and *inc* = *NULLCHAN*, respectively.

One additional routine is very useful in the simulation of many physically interesting problems, such as those derived from discretizations of partial differential equations on regular grids. An important feature in such discretizations is that there is typically only nearest-neighbor connectivity among the variables of interest. For efficient use of the hypercube, it is then very desirable to map the grid onto the cube in such a way that neighboring grid points (in 2- or 3-dimensional space) are mapped onto adjacent nodes of the cube. Communications overhead will be minimized by such a mapping. Accomplishing such a mapping is nontrivial and in general impossible. For example, there is no such mapping of a 3-dimensional grid onto a 5-cube since the grid requires a local connectivity of 6 at each node. A general purpose routine called *whoami* has been developed by John Salmon at Caltech [35] based on *binary gray codes*, which generates a suitable mapping of the above type in most cases where one is possible. The *whoami* call is usually executed at the start of any grid-oriented program, and in addition to creating a suitable mapping of the grid to the cube nodes it returns communication channel information for each of the *grid-neighbors* of each processor. This allows the programmer to think entirely in *grid space* rather than in the less intuitive *edge space* of the cube.

A hypercube program consists of two separate programs: an *intermediate host program* and an *element program*. The intermediate host program initiates the computations and returns the results to the user, but does not interfere with the core of the computations. These are described by the element program, identical copies of which are executed in all processors of the hypercube simultaneously.

2.2. The Intel iPSC computer. The Intel Corporation has recently marketed the first commercial realization of the hypercube design, based largely on the Caltech cosmic cube. The machine, known as the iPSC, comes in three models—the d5, d6 and d7. These have, respectively, 32, 64 and 128 processors. The individual processors are the Intel 80286/80287 with up to 512Kb of memory, and the interconnections are provided by high-speed Ethernets, using an Intel Ethernet chip. The intermediate host machine, which is both the control processor and the user interface, is an Intel 310 microcomputer running a UNIX system (Xenix). In addition to the Ethernets along

cube edges, a global communication channel is provided from the intermediate host machine to the individual processors. This feature is useful for debugging and to a limited extent for control purposes. Besides the UNIX system on the host, software for the system consists of a node-resident kernel providing for process creation and debugging along with appropriate communications software for interprocessor exchanges, and for host to processor direct communication. Combined computing power of a 128-node system can be over 5 MFLOPS, which along with the 64 Mbytes of memory available, provides a relatively powerful computer.

2.2.1. iPSC programming. The software environment for the Intel iPSC is distinctly different from the crystalline operating system described above. To begin with, the operating system supports multiple processes at each cube node, identified by their process identity number *pid.* All communication primitives can address an arbitrary process on an arbitrary node. The underlying message passing system includes automatic routing of messages between any two processes. This frees the user from developing complex routing schemes in his software, but at the expense of some extra communication overhead. We present a list of the iPSC communication calls in Table 2.

A further flexibility is the availability of both synchronous and asynchronous communication modes. The system supports a concept of *virtual channel,* unrelated to the physical channels connecting nearest neighbor nodes. A process can communicate with several other processes simultaneously by opening several virtual channels (using the routine *copen*()) and then exchanging messages using asynchronous communication calls. All messages have a user-defined integer attribute, called *type,* which is assigned by the sender. A receiver may request messages by type, but not by source process or source node. Fortunately the range of the type attribute is large enough ([0,32767]) to allow the source of a message to be encoded in its type. Messages of any size up to 16384 bytes may be sent, although the overhead for message transmission severely discourages sending small messages, a point which we return to in a later section. To send a message, the message pointer and length are supplied along with the destination node and process and the *type* attribute (*send*() and *sendw*()). To receive a message, a type and a message buffer and desired length are supplied, and on receipt of the message the actual length, source node and source process identity (*pid*) are returned

TABLE 2
iPSC *communication routines.*

Node routines:	
chan = copen(pid)	open a virtual channel for process *pid*
send(chan,type,mesg,len,node,pid)	send a *type* message to *pid* on *node*
recv(ci,type,msg,len,cnt,node,pid)	read message of type *type*
length = probe(chan,type)	are there messages of type *type*?
status(chan)	is channel free yet?
flick()	nonbusy wait
sendw(chan,type,mesg,len,node,pid)	blocking send to *pid* on *node*
recvw(ci,type,msg,len,cnt,node,pid)	blocking read of *type type*
syslog(pid,string)	print a message on the host
Host routines:	
sendmsg(chan,type,mesg,len,node,pid)	blocking send to *pid* on *node*
recvmsg(ci,type,msg,len,cnt,node,pid)	blocking read

(*recv*() and *recvw*()). To support asynchronous transmissions, it is possible to determine if a previous message has completed on a specific virtual channel (with *status*()), and to determine if there is a message of a specific type pending at a node (with *probe*()). One very unfortunate feature of the system is that the host communication primitives are less powerful. For example it is not possible to request receipt of a message at the host by *type* and in fact one is forced to read messages in totally arbitrary order. As a consequence, it is usually necessary to develop a software buffering scheme for messages at the host so that incoming messages may be stored until one of an appropriate type arrives. This is a serious inconvenience and will hopefully be remedied in future versions of the iPSC software.

2.2.2. Computation and communication costs. Two characteristics of the current iPSC design are the slow communication rate and the high overhead for short messages. In fact messages of length 1 and 1024 bytes take essentially the same time. As a measure of the slowness we note that a message of length 16384 bytes takes 12 seconds to traverse a nearest-neighbor ring of 128 processors, or over 17 seconds using a ring in random (sequential) order. The cost of sending a message of length 1 byte to a neighboring processor is approximately 5.3 ms while longer messages require about 5.5 ms per 1024 byte segment. Figure 2 displays the cost of sending packets as a function of packet size. These numbers are approximate and were obtained by sending 30 consecutive messages from node 0 to its 6 neighbors on a 6-dimensional cube. An uncertainty is involved because the system clock is accurate only to 16 ms. Larger numbers of messages could not be sent to obtain better statistics without a time delay due to an operating system bug. Messages were sent using both the asynchronous *send* routine and the synchronous *sendw* routine, though there was little difference as indicated by the pair of curves in Fig. 2. This slow communication speed is way below the hardware limits of the Ethernet connections and suggests that much time is wasted in operating system overhead. Despite this fact we have found that the iPSC can be used with high efficiency on a wide range of problems because of the substantial memory available per node. To indicate the processor speed, we note that a *C for* loop with no body requires about 11 μs per point, while a loop with a typical floating point computation such as $a = a + b * c$ requires about 67 μs per point. Thus we rate the

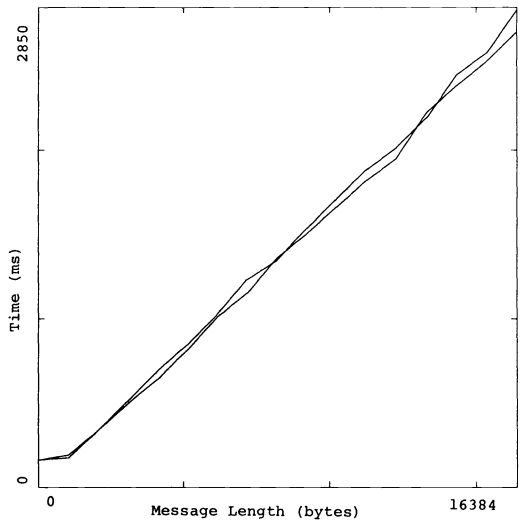


FIG. 2. *Synchronous and asynchronous message sending time as a function of the message length on the* iPSC.

processor at about .03 Mflops though this estimate might vary by a factor of about 2 in different situations. We summarize processor speed characteristics in Table 3.

We present a study of timing costs for messages of different sizes on a 128 processor d7 in Fig. 3. The times measured are for passage of messages of various sizes around a complete ring of 128 processors. Each processor reads a message from its clockwise neighbor on the ring, then sends the message to its anti-clockwise neighbor. The sequence starts and ends at processor 0 which records the time between sending the message and final receipt. The upper curve is for a ring of nonnearest neighbor processors, ordered as $0, 1, \cdots, 127$, while the lower curve is for a nearest-neighbor ring. Superimposed on the lower curve is a second curve, generated on the iPSC using Caltech Hypercube communication calls. The latter calls were implemented using a simulator we have written for the Caltech Hypercube on the iPSC hardware (see § 3.2). As can be seen, overhead for using the simulator is completely negligible.

2.3. Basis for communication cost analyses. In subsequent sections, we analyze communication costs for many algorithms. Generally we assume that the cost to transfer

TABLE 3
iPSC *performance.*

C **for** loop: empty body	10.9 μs per point
C loop to copy real numbers	15.7 μs per point
C **for** loop $a = a + b * c$	67.4 μs per point
send 0 bytes	5.3 ms
send 1024 bytes	5.9 ms
send 2048 bytes	11.2 ms
send 16384 bytes	90 ms
clock() system call	155 μs
clock() time unit	16 ms

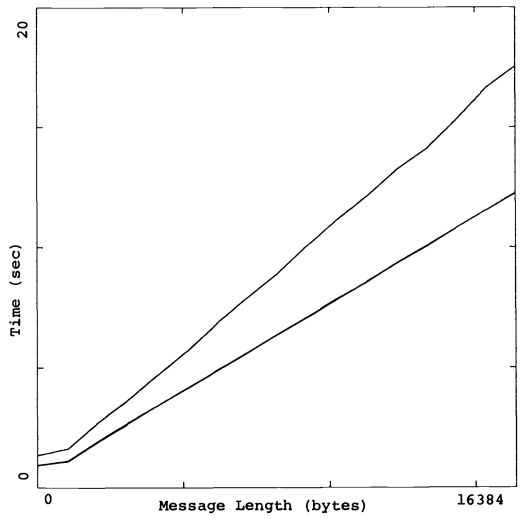


FIG. 3. *Time for different size messages to travel around a ring in a* 128 *node* iPSC. *The top curve is for the ring in which the processor numbers are increasing. The bottom curve is a nearest neighbor ring (based on binary reflected gray codes). Superimposed on the lower curve are the times obtained by using the Caltech simulator on the* iPSC *for the nearest neighbor ring.*

a segment of k real numbers between two neighboring processors is of the form:

$$ST(k) = \alpha + \beta k.$$

This is accurate for the Caltech Hypercube, but is a simplification for the Intel iPSC since the formula does not model the communication cost correctly over the whole range of permissible message lengths. From Fig. 3 we notice that messages shorter than 1024 bytes (256 reals) all take essentially the same time. This is an important case which we have included in our analyses by using different values α_{long} and β_{long} for long messages, and α_{short} and β_{short} for short messages.

We have derived estimates for the coefficients α and β from the data displayed in Fig. 3. The data for the lower curve in this figure was obtained by measuring the time necessary to send a message around a 128 node nearest neighbor ring. The timing represented there is then the curve $T = 128\ ST(k)$. From this we have deduced that, with times measured in microseconds,

$$\alpha_{short} = 6625, \qquad \beta_{short} = 8.28,$$
$$\alpha_{long} = 3477, \qquad \beta_{long} = 22.5.$$

These numbers are in sharp contrast with the cost γ to perform a typical arithmetic operation, which from Table 3 is seen to be of order 30 μs. In particular the ratio $\alpha_{short}/\gamma = 220$ indicates that communication of single data items is hundreds of times slower than a corresponding computation. Another parameter that appears in the analysis of some algorithms is the length λ words of a buffer used to accumulate short messages for communication in a single packet. Ideally λ should be chosen such that $\alpha/\lambda < \beta$. On the iPSC we have used $\lambda = 4096$.

For comparison we present here the corresponding data for the Caltech Hypercube. In that case a single value of α and β suffice to cover the whole range, and we find:

$$\alpha = 92, \qquad \beta = 40.$$

Thus message startup overhead becomes small as soon as even a few words are communicated between nodes, although communication rates for very long messages are about twice as slow as on the iPSC. Computation rates γ for the Caltech Hypercube are comparable to those for the iPSC. The buffer of length λ is not required at all.

The four parameters α, β, γ and λ appear in efficiency estimates throughout the paper without further comment. We also use γ as a measure of integer computation speed, for example, it is used to measure the cost of copying arrays. In addition we use N to denote the size of vectors or the dimension of arrays and we use $P = 2^D$ to denote the number of processors.

3. Hypercube simulators. An important aspect of our hypercube work was the development of a hypercube simulator for the Intel iPSC. It is difficult to develop programs for a parallel computer such as a hypercube in a reasonable period of time without access to a simulator for the machine. A simulator allows programs to be developed on faster and more robust computers, tested under conditions of moderate parallelism (we have used 4 to 16 processor simulations) and then moved to the target hardware with reasonable assurance that issues of synchronization and control are correct. One can then concentrate on the many remaining issues that can prevent successful execution, such as faulty node compilers, defective processor hardware or bad communication lines. In addition, by instrumenting a simulator one can obtain useful statistics when real codes are run. We have instrumented all of our codes, and statistics on a per-processor basis are collected on many items such as numbers and sizes of messages sent and received, amount of memory used and so on. This information

is extremely helpful in pinpointing communication bottlenecks in the code. These statistics are collected by each node separately and are automatically sent on program termination to the host machine for storing in a history file. We have also instrumented the standard communication calls used on the real hypercube so that history files are obtained during actual runs as well. In § 3.3 we present a set of higher level communication routines that we have found especially useful on the Caltech Hypercube, where messages are normally of length 8 bytes.

We describe now two hypercube simulators we have developed, one for the Intel iPSC, and one for the Caltech Hypercube. These simulators may be run across a series of machines, with sets of nodes assigned to different machines. This allows for much faster simulations. We have also simulated the Caltech Hypercube on the Intel iPSC, which allows Caltech Hypercube programs to be ported directly to the iPSC without any code modification.

3.1. An Intel iPSC simulator for UNIX. We have developed an Intel iPSC simulator that uses the UNIX operating system to simulate both hypercube processes and inter-process communication. The simulator actually represents a general model of message-based parallel computation and can be used to simulate various different architectures. We use the operating system to implement processes, the directory tree to implement nodes and the file system to implement message passing. Each hypercube node is assigned a subdirectory of the running directory, with the directory name being the hypercube node number. The hypercube host is assigned to a host subdirectory. Each process running on that node is assigned to a subsubdirectory, again numbered in relation to the number of processes on the node. Thus process *pid* on node *node* resides in directory *./node/pid.*

The simulator starts by creating all of the relevant directories and starting up the requested programs in the appropriate directories. The processes then communicate with each other, and with the host, by exchanging files containing their messages. The procedure here is simple: a process creates a message in a message file with a name *source.seq_num* that identifies its source and also a *sequence number* from the source to the destination. The sequence number is incremented each time the source sends a message to that destination. Each process records the sequence number of the last message sent to each of the other processes, initialized to 0. Furthermore each process remembers the next expected message sequence number from each of the other processes, again initialized to 0. Once a message is complete it is moved to the appropriate destination directory. The receiving node looks for appropriately named files in its directory and on reading a file, adds the message to a linked list of messages. This list is ordered by source node, and within that by sequence number. When a message is requested of a particular type (by a *recv* or *recvw*), the list is searched for a message of that type whose sequence number is the next expected one from its source. This ensures that messages from the same source are received by programs in the order sent. If no such message is found, the process continues reading and storing incoming messages until a suitable one appears.

The simulator is independent of the mechanism for message routing. Thus various message routing systems may be added to simulate actual hardware designs. The simplest message routing uses file renaming to transmit a file to its destination directory. This simulates a fully connected network hardware in that all processes are connected to each other in a completely equivalent fashion. We have also used the *socket*-based interprocess communication facilities of 4.2bsd UNIX to implement message transmission as an alternative to file passing. However the file mechanism has the advantage

of working over networks of heterogeneous systems and with various flavors of UNIX. In addition to the fully connected network, we have also simulated a hypercube network. In this case messages are passed through a sequence of *nearest-neighbor directories* in order to reach a target. We stress that for most purposes this is completely unnecessary and significantly slows the simulation since most messages must be handled several times. In fact the fully interconnected network is perfectly adequate for iPSC simulations since the iPSC software supports symmetrical communication among all processes, completely hiding the hardware.

3.2. A Caltech Hypercube simulator on the Intel iPSC. Developing a Caltech Hypercube simulator for the iPSC also proved to be useful. This allows identical codes (i.e. Caltech Hypercube codes) to be run on both machines, with a software overhead that we have measured to be under 1%.

Caltech processes are implemented as iPSC processes, with only one per node allowed. Since Caltech processes work with channel numbers, we first create a mapping of channel numbers to node numbers. The basic communication primitives such as *rdELT* and *wrtELT* (see Table 1) are then implemented using the iPSC *recvw* and *send* primitives (see Table 2). Note that Caltech protocols of reading and writing channels imply that when receiving a message the message is requested from a specific source, the node at the other end of the requested channel. On the other hand iPSC protocols do not allow for requesting receipt of messages by source location, but only by message type. We overcome this problem by using the iPSC message type to encode the source node of a message when sent. A receiver then requests a message with type equal to the desired source node number. The only complexity encountered is at the host processor. At the host, the iPSC protocol does not even allow asking for messages of a specific type; one simply reads messages as received. This issue is dealt with using a complicated buffering scheme where incoming messages are stored and ordered with respect to both sequence number and type. A Caltech Hypercube simulator for UNIX was previously written at Caltech [27] but the one developed here is different in that it is targeted at another parallel machine—the iPSC. By combining the Caltech Hypercube simulator for the iPSC with the iPSC simulator for UNIX, we have created a new Caltech Hypercube simulator for UNIX.

3.3. An object-level communication library. The Caltech Hypercube communication software deals with messages in terms of 8-byte units. This tends to lead to unpleasant code where 4 integers, or 2 floating point numbers or the like are stored into an 8-byte array for transmission. Resulting code is hard to read and prone to bugs, especially if the receiver decodes the bytes differently from the sender's encoding. On the Intel iPSC we have seen that sending lots of short messages is extremely inefficient.

We have developed a simple higher-level interface to these routines that allows objects (data-structures) of arbitrary size to be communicated between nodes and with the host with essentially optimal efficiency. In Table 4 we indicate the calling sequence of some of these routines. In addition to sending linear arrays and structured types (as in *C*) between processors, this library also supports communication of vectors and matrices between processes and with the host. In particular there are routines to send a globally distributed vector or matrix to the host, frequently used in printing the final results of a program. The *send_local_vectors* routine allows r vectors $v_1, \cdots, v_r$, with addresses stored in the array v and with lengths $n_1, \cdots, n_r$ stored in the array n to be sent to a neighbor. On the Caltech Hypercube this routine simply sends each vector in short segments, whereas on the Intel iPSC the vectors are first copied and concate-

TABLE 4
Communication routines.

send_structure_to_node(chan,struct,size)
recv_structure_from_node(chan,struct,size)
send_local_vectors_to_node(chan,*r*,*n*,*v*)
recv_local_vectors_from_node(chan,*r*,*n*,*v*)
send_matrix_to_node(chan,mat)
recv_matrix_from_node(chan,mat)

nated into an array of length 4096 bytes which is sent each time it fills with a single *send* call. The last segment sent may of course be shorter. Unless all of the vectors are already over 1024 bytes, this results in much more efficient iPSC communication, at the cost of copying the vectors. If some of the vectors supplied to the routine are already long, then they are sent directly to avoid the extra copying overhead. The *recv_local_vectors* routine just splits and copies incoming 4096-byte segments into r local vectors. Similarly the *send_matrix_to_node* routine buffers the rows of the matrix into large messages on the iPSC. These two routines are actually part of the vector library described in § 5.

The only nontrivial issue in the other routines is the handling of quantities whose size is not a multiple of 8 bytes. In these cases it is necessary to send a last packet containing some zero data in order that the full communication be a multiple of 8 bytes. On receiving messages this can require buffering the message first, and then copying it to its destination. In particular this occurs on reading the union of messages sent by all nodes to the host, if the messages are not a multiple of 8 bytes.

4. Hypercube interconnection topologies. The interconnection network of the hypercube can be organized in a variety of ways depending on the application. In the following subsections we consider some of these topologies since they are essential for many of the later algorithms. The cases of most importance to us are mappings of grids, trees and rings to the cube network.

4.1. Grids on hypercubes. Grids are mapped onto the hypercube using the *whoami* facility developed by J. Salmon [35]. The routine uses Binary Gray Code (BGC) sequences [36], [37] to number processors. These are sequences of numbers such that consecutive terms in the sequence differ by only one bit in their binary expansions. For example, one 4-bit BGC is the sequence (by rows):

0000 0001 0011 0111 1111 1110 1100 1000
1001 1011 1010 0010 0110 0100 0101 1101.

As mentioned earlier, nodes on the hypercube with processor numbers that differ in only one bit in their binary expansion are physical neighbors. If we interpret the terms in a BGC sequence as processor numbers, then nearest neighbors in the sequence will also be physical neighbors. This allows 1-dimensional grids to be mapped onto the hypercube using only nearest-neighbor connections. The generalization to higher dimensional grids is done by mapping each dimension to a suitable subcube and using the 1-dimensional mappings for those subcubes.

Certain applications require periodic grids to be mapped to the cube. The above BGC sequence is not suitable for this purpose because its last element is not a neighbor of the first one. A subclass of Gray codes, the Binary Reflected Gray Codes (BRGC), have the desired property [35]. In the sequel we will always use a canonical BRGC defined recursively as follows. With one bit the BRGC sequence is 0 1. With d-bits we

take the $d-1$ bit sequence, followed by the same sequence reversed in order and with each dth bit set. Thus with two bits we have the sequence 00 01 11 10 and with four bits the sequence is:

0000 0001 0011 0010 0110 0111 0101 0100
1100 1101 1111 1110 1010 1011 1001 1000.

This is the standard processor ordering used by the Caltech Hypercube research group [27]. With a BRGC ordering the subcubes used for a grid whose dimensions are each a power of 2 will all be periodic, allowing periodic grids of such dimensions to be mapped to the cube. The BRGC sequence also has the property that two elements that are a power of 2 apart differ in at most two bits [38].

4.2. Trees on hypercubes. There are several ways to embed a tree onto a hypercube. The most natural trees are either binary trees or D-ary trees where D is the cube dimension. In the following discussion we choose the root of the tree to be at node 0, but any other point can be used as easily.

The observation that allows a balanced binary tree to be mapped to the hypercube, is that a D-cube may be regarded as two $(D-1)$-cubes with corresponding processors from each connected by an extra channel. Numbering the processors in the two $(D-1)$-cubes as $p_0, \cdots, p_0+2^{D-1}-1$, and $p_0+2^{D-1}, \cdots, p_0+2^D-1$, respectively, we connect processor p_0+p in the first subcube with processor p_0+p+2^{D-1} in the second subcube. The binary tree on the D-cube is then defined recursively as having the lowest numbered processor as its root and the two $(D-1)$-cubes as its left and right subtrees. Note that this is a logical binary tree only—some processors occur several times at tree nodes. For example, processor 0 occurs D times, being the root of D subtrees. Each processor is located exactly once at a leaf of the tree.

An alternative tree mapping is to represent the network as an unbalanced D-ary tree. This mapping is based on the obvious fact that D connections emanate from every node. This tree is a physical tree—each node in the tree corresponds to a unique processor. The tree may be defined either by giving the children of each node, or by specifying the parent of each node. The children of a node are the processors whose node numbers are obtained by setting in turn each of the low-order unset bits in the D-digit binary representation of the node number. Alternatively, the parent of a node is located by unsetting the lowest-order set bit in the binary representation of the node number.

These two logical trees are in reality two different ways of representing the same set of connections. Depending on the application it can be helpful to consider trees from either viewpoint. Figure 4 represents the 4-cube as a tree from these two points of view.

4.3. The hypercube network as a hierarchy of rings. As indicated above, it is possible to map a ring structure onto the hypercube by giving the processors the position of their processor number in a Binary Reflected Gray Code sequence. As a property of the binary gray codes, processors that are neighbors in this ordering (logical neighbors) are also neighbors on the Hypercube (physical neighbors). We will use the terms *logical distance* and *physical distance* in this sense throughout the rest of the paper. As noted in § 4.1, the BRGC ordering has the further property that two processors at logical distance 2^d, $d=1, \cdots, D-1$, in the ring are at a physical distance 2. For each d in the range $1, \cdots, D-1$ there is then a collection of subrings of processors such that processors at logical distance 2^d in the subring are at physical distance 2. We call these the *level d subrings* of the hypercube. Every processor is on exactly one subring of level d. A subring connecting processors at logical distance 2^d in the main ring contains

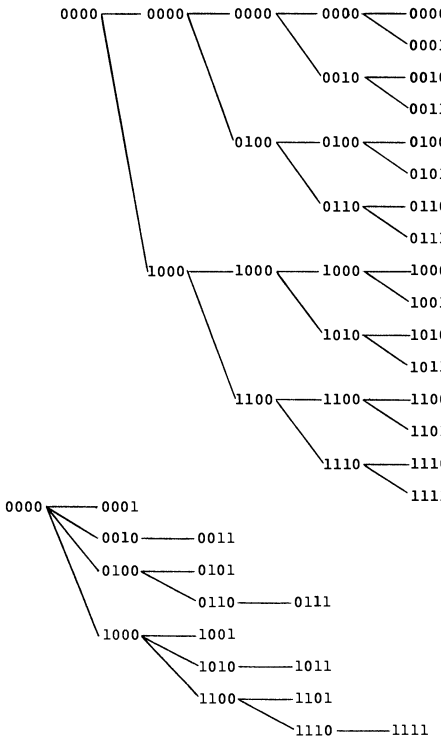


FIG. 4. *Nodes of a 4-dimensional hypercube as a balanced binary tree and as an unbalanced tree of order 4.*

2^{D-d+1} nodes, and hence there are $2^D/2^{D-d+1}=2^{d-1}$ such subrings covering the hypercube. In Fig. 5, we display on the left the 4 logical hierarchies of rings in a 16 node (4-dimensional) hypercube which has been ordered with the BRGC. On the right, we show the actual physical communication channels used to implement the logical structures. Note that except for the level 0 ring, logical neighbors on the subrings always are separated by distance 2. We see that the main ring structure is used for the rings at levels 0 and 1. Level 2 consists of 4 logical subrings. The physical links at level 2 however form only 2 subrings, corresponding to the fact that 1 logical link requires 2 physical links. It should be noted that parallel communication is possible on all subrings at the same level.

We have implemented a library subroutine that, analogously to the *whoami* routine mentioned earlier, returns the channel information needed to use the hypercube as a hierarchy of rings. This routine starts by creating a level 0 ring, by calling the *whoami* routine to construct a 1-dimensional periodic grid. From this ring the other levels can then easily be constructed. On the Caltech Hypercube, the information returned consists of $D-1$ pairs of channels: a left and a right channel for each physical ring level. This gives a complete description of the subring hierarchy since each processor is a member of just one physical subring at each level.

5. Parallel vector operations. We describe the development of a library of vector operations on a hypercube. This description is valid for a wide range of parallel processors, including shared memory and even serial machines. All that we assume is that the processors may be organized into a 1-dimensional *ring* of processors in which each processor can communicate efficiently with its neighbors on either side. On a hypercube there are many ways to imbed such a ring onto the processor network in

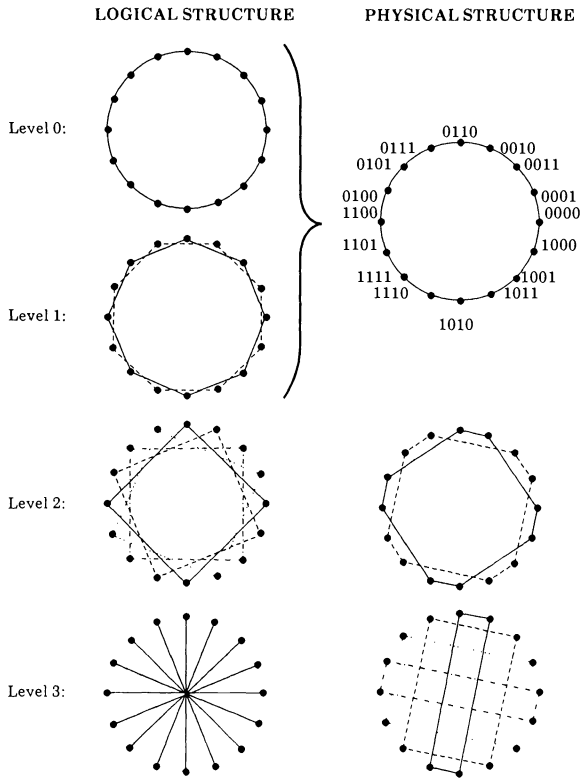


FIG. 5. *The logical hierarchies of rings in a* 16*-node hypercube and the communication channels used to implement them.*

such a way that only nearest-neighbor connections are involved (see § 4). In order to simplify notation, we assume in the following that the processor ordering $0, 1, \cdots, P-1, 0$ is already such an optimal ring.

An important feature of this library is that individual processors work entirely with a local vector and are never aware of the location of other parts of the distributed vector. This is true even with operations such as *inner_product* which involve communication between different processors. Furthermore the operations on local vectors are themselves vector operations. Thus the library could take full advantage of an available vector processor on each node.

5.1. Distributed vectors. We use a consistent scheme for distributing all vectors over the P processors. We consider here only vectors of length $N \geqq P$. We divide the vector into P contiguous and roughly equal-length segments. It is then possible to distribute a segment to each of the P processors. We will call a vector stored in this way, a *distributed vector.* We will assume that every distributed vector of the same length is distributed in the same way over the processors, i.e. segments of different vectors residing in the same processor have the same length. The actual distribution formula we have used is as follows. We divide the vector length N by the number of processors:

$$N = hP + r, \qquad 0 \leqq r < P.$$

The first r processors will be assigned segments of length $h+1$ elements and the remaining $P-r$ processors will be assigned segments containing h elements. In the case that P divides N exactly, all segments are then of size $h = N/P$.

To create a distributed vector of length N, all processors call an allocation routine:

$$v = \textit{allocate_vector}(N, \textit{type}).$$

Several types of vector representation are supported, of which the most important types are called *SIMPLE* and *SHIFT*. The *SIMPLE* form represents each vector segment as an array of length h or $h+1$. In addition to performing the above computation for the segment length in each processor, this routine allocates storage for the array and returns a pointer to a data structure v which represents the vector. The data structure records the type, local segment length and a pointer to the start of the vector. In the *SHIFT* format, each vector segment is located in a buffer of length $2h$ within which it is allowed to slide around (see the following section on shifting vectors). Initially the segment is centered in the buffer. In this case, the data structure v records the type and local segment length, pointers to the start of the segment both before and after shifting, as well as the buffer extremities. Whenever it is intended to apply the *shift_vector* operation to a vector, the vector should be allocated of type *SHIFT*. A library routine *convert_vector* is available to convert a vector from one type to another.

5.2. Operations on distributed vectors. We have implemented a large set of operations on such distributed vectors. Application of any such operation is accomplished by having all of the processors call the appropriate routine simultaneously, with the appropriate vector data structures as arguments. Vectors of different types may be freely mixed, as appropriate conversions are applied when needed. We list sample routines from the library, along with their purpose, in Table 5.

TABLE 5
Basic vector routines.

allocate_vector	allocate local segment of a vector
delete_vector	deallocate storage for a vector
convert_vector	change type of vector
zero_vector	$v_i = 0$
shift_vector	$v_i = v_{(i-s) \bmod N}$
max_vector	$\max_{0 \le i < N} v_i$
sum_vector	$\sum_{i=0}^{N-1} v_i$
inner_product	$\sum_{i=0}^{N-1} u_i v_i$
copy_vector_to_vector	$v_i = u_i$
add_scalar_times_vector_to_vector	$u_i = u_i + s v_i$
add_vector_to_scalar_times_vector	$u_i = s u_i + v_i$
add_vector_times_vector_to_vector	$u_i = u_i + v_i w_i$

Most of these routines are trivial to parallelize. However *shift_vector*, *inner_product*, *max_vector* and *sum_vector* involve communication between processors. We discuss these four routines in more detail in the following subsections. The remaining routines involve no communication and are optimally parallel: the corresponding operation is performed in each processor on the local segments of the vector arguments, using the standard serial code. Thus we will not discuss routines such as

add_vector_times_vector_to_vector

any further.

5.3. The vector shift algorithm. The *shift_vector* operation consists of sliding each element of the distributed vector forward by s elements, placing the last s elements of the vector at the start. It is essential that this operation be very efficient since it is used by many of our other algorithms. For this operation we require that each vector

segment is of at least length one, or, in other words, that each processor contains at least one element of the vector.

We discuss first the simple case of a short shift of length 1. Most of the shifting will occur internal to a processor. If there are h elements in a segment, the cost of sliding the segment forward is $O(h)$. In addition, each segment has to execute communication calls to send its last segment component to the following processor and to receive the last component from the previous processor. All of the internal shifting within nodes may be done in parallel, and similarly for the shifting of the segment ends, so that overall the cost of the operation appears to be $O(h)$, which is unacceptable.

We have implemented the shift operation as an $O(1)$ operation at the cost of using some extra memory in each node. The idea is to imbed the local segment of each vector in a larger buffer which extends a distance $O(h)$ (for example $h/2$) on either side of the segment. For this purpose, vectors are allocated by the *allocate_vector* routine using type *SHIFT*, see the previous section. Vectors are then shifted by simply leaving them in place, adding the new end element from a neighboring processor to the appropriate end of the local vector. The effect of this is to cause the vector to slide around in its buffer with successive shifts. The pointer to the start of the vector in the corresponding vector data structure is incremented for each shift. After $O(h)$ elementary shifts, the vector may have slid to the end of its buffer in each node. At this point an $O(h)$ operation is required to copy it back to its initial point in each processor. However one can regard this $O(h)$ operation as amortized over the $O(h)$ shifts that preceded it, so that the overall cost of a shift is still effectively $O(1)$. This copy back to the original position is automatically detected and performed by the software. When shifted vectors are involved in other vector operations, the appropriate offsets are automatically used for each vector.

The shift operation for large shifts, consisting of many one element shifts in succession, can be made even more efficient by using the full hypercube hierarchy of rings at many levels in addition to the top level ring used above. As this optimization is somewhat complex we assign the discussion to a separate section.

5.4. The optimized vector shift algorithm. Based on previous observations, we can optimize the *shift* operation for the case in which the number of shifts is large. In fact we will show that large shifts may be performed $P/\log P$ times faster than if the simple shift algorithm above were used. In the notation of the previous section, we require that $r=0$ for this algorithm, so that the length N of the vector is an exact multiple of the number of processors: $N=hP$. Thus every processor contains a segment of length h of the vector.

We introduce some useful terminology. An *element shift* will denote the logical operation that shifts a vector by one element. An *element transfer* will denote the physical communication operation between two processors required to transfer a vector element from one processor to another. Analogously, we call a *segment shift* the logical operation of shifting a vector h times. A *segment transfer* is then the communication operation between processors to transfer a segment. Note that a segment shift results in each processor transmitting its complete segment to the following processor in the ring, receiving in turn the complete segment of the preceding processor. We assume that an element transfer takes time $O(1)$ while a segment transfer takes time $O(h)=O(N/P)$. We will show that any shift can be performed in time at most $O(N \log P/P)$.

Suppose we need to shift a vector by m element shifts, where without loss of generality we assume $m>0$. If we write

$$m = m_s h + m_e, \qquad 0 \leqq m_e < h,$$

we can reduce the m element shifts to m_s segment shifts and m_e element shifts. Note that if $m_s \geqq P$, we can reduce it further by letting $m_s = m_s \bmod P$, since a shift of P segments is a shift of $N = hP$ elements and thus is equivalent to not shifting at all. A further immediate savings is possible by observing that when $m_s > P/2$, it is better to shift the segments in the opposite direction $m_s - P$ times. Suppose that all these reductions are done on m_s. Then the resulting m_s will be in the range $-P/2+1, \cdots, P/2$.

The m_e element shifts are performed separately as in the previous section. In fact they may be lumped together into one for the internal shift, which then becomes an $O(1)$ operation using the buffering mechanism described earlier. Because of the need, mentioned earlier, to occasionally copy a shifted segment back to its initial position, a better estimate of the cost of m_e internal shifts would be that it is $O(m_e)$. The m_e last elements of each segment will be sent to the following segment, resulting in a communication cost that is $O(m_e)$. Therefore the total cost of m_e element shifts is $O(m_e)$.

We now use the hierarchy of rings to speed the segment shifts. Consider the binary representation $b_{D-2} \cdots b_1 b_0$ of m_s. For each $i > 0$ such that b_i is set, we need to perform 2^i segment shifts, which reduce to just one segment shift on the logical subrings of level i. As discussed in § 4.3 this may be accomplished with just 2 shifts on the physical subrings of level i. This has the effect of performing 2^i segment shifts while taking the time of only 2 segment transfers. Bit 0 is an exception because if set it requires only one segment shift of the main ring, taking the time of one segment transfer. Thus even in the worst case, where all $D-1$ bits in m_s are set and where the number of element shifts is $h/2$, the total time is at most:

$$T_{shift} = \{2(D-2)+1\}T_{segment} + h/2\,T_{element}$$
$$= O(\log P)O(N/P) + O(N/P).$$

Note that these estimates neglect any overhead related to sending short messages. In § 6.6.1 we present a more complete analysis of these costs. We summarize these results as two theorems:

THEOREM 1. *The time $T_{shift}(m)$ to shift a vector of length N by a distance m on a hypercube with P processors satisfies*

$$T_{shift}(m) \leqq \alpha(1+d(m)) + 2\beta d(m)N/P + (\beta+\gamma)m \bmod (N/P),$$

where $d(m)$ is the number of set bits in the binary expansion of m div N/P.

THEOREM 2. *Upper bounds for the maximum time $T_{shift}^{\max}$ and average time T_{shift}^{ave} to shift a vector of length N by a distance m on a hypercube with P processors are*:

$$T_{shift}^{\max} \leqq (2\beta \log P + \gamma)N/P + 2\alpha \log P,$$

$$T_{shift}^{\text{ave}} \leqq (\beta \log P + \gamma)N/P + \alpha \log P.$$

The algorithm above fails if the length of the vector is not a multiple of the number of processors. In this case there are r segments of length $h+1$ and $P-r$ segments of length h. With segment shifts, the segments of length $h+1$ are transported to arbitrary processors. To bring the vector back to its original form (with the longer segments in the logically early processors) requires $O(N)$ element transfer operations, destroying any efficiency obtained by using segment shifts. One solution in such cases is to solve a slightly larger problem such that the length of the vectors is a multiple of P.

5.4.1. Timings for the optimized vector shift. We have measured the effectiveness of the optimized vector shift by timing shifts of various sizes for vectors of varying

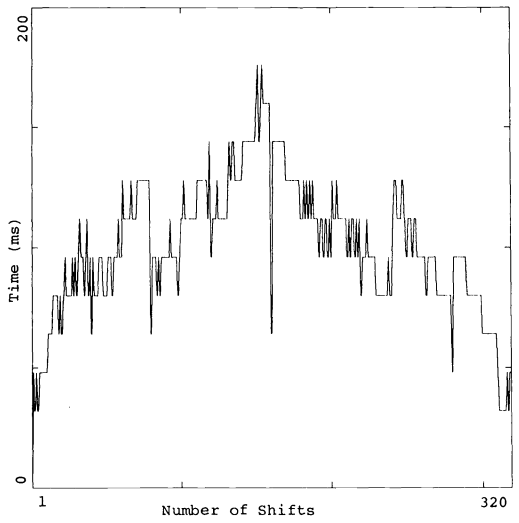


FIG. 6. *Time to shift a vector of length* 320, *distributed over the* 32 *node* iPSC (10 *elements per processor*), *as a function of the number of element shifts.*

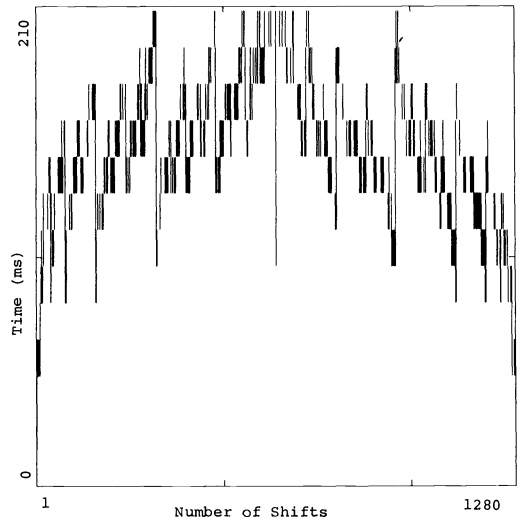


FIG. 7. *Time to shift a vector of length* 1280, *distributed over the* 128 *node* iPSC (10 *elements per processor*), *as a function of the number of element shifts.*

length and with different numbers of processors. Figure 6 shows the time required to shift a vector of length 320 on a 32-node iPSC. The graph shows the shift time as a function of the shift length, which varies from 1 to 319. Figure 7 presents the same statistics for a vector of length 1280 on a 128 node iPSC, again shifted by distances 1 through 1279. These two cases are useful to compare since the number of elements h per node segment is the same in each case, namely 10. We note that the cost of performing shifts on the larger cube is only slightly more than on the 32-node cube. Specifically the slowest shift in the 128 node case is only about 10% slower than in the 32 node case. Furthermore the variation from longest to shortest shift is comparable. Finally in Fig. 8 we present shift statistics for a vector 100 times longer, i.e., of length

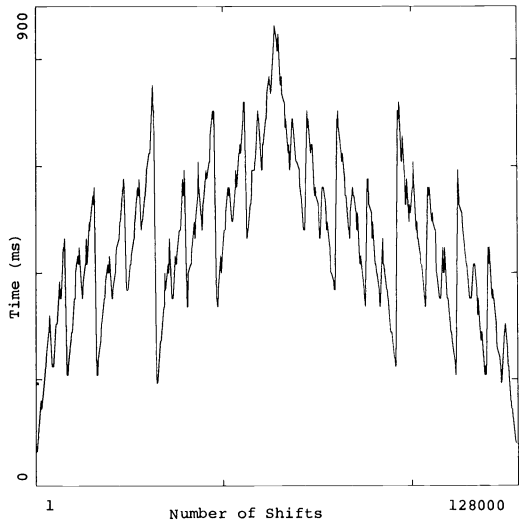


FIG. 8. *Time to shift a vector of length* 128000, *distributed over the* 128 *node* iPSC (1000 *elements per processor*), *as a function of the number of element shifts.* (*Only representative sample is displayed.*)

128000 so that there are 1000 elements per node. The shift times for the shorter vectors are dominated by the high message start-up cost of the iPSC.

5.5. Inner product. The semantics of the inner product operation in our library needs careful discussion. Each processor calls the *inner_product* routine simultaneously, providing as arguments the two vector segments local to the processor. The *inner_product* routine returns in *every* processor with the value of the inner product of the two complete distributed vectors. No processor is distinguished in these semantics.

The implementation of the routine does of course distinguish processors. The inner product operation is complicated because it involves communication between the processors to sum inner products of the segments local to each processor, and further communication to broadcast the final sum back to all of the nodes. On cube architectures, this should be done by summing over a *tree* of processors, since if one processor is used to accumulate all partial inner products, it will result in a critical section. A D-ary tree is mapped onto the processor network with node 0 as the root. The parent of a node is defined by zeroing the lowest nonzero bit of the node number while the children of a node are obtained by setting in turn exactly one of the low-order zero bits of the node number. The inner product routine then takes the form:

```
real inner_product (v1, v2):
mysum = local_inner_product(v1, v2)
for each child
    read child_sum from child
    mysum = mysum + child_sum
end for
if (node ≠ 0) send mysum to parent
if (node ≠ 0) read sum from parent
for each child
    send sum to child
end for
return sum
```

With this implementation the cost of an inner product will be $O(N/P)+O(\log P)$, with the $O(N/P)$ reduced further if vector hardware is available on the nodes. Clearly all of the local sums may be computed in parallel in time $O(N/P)$, proportional to the local segment length $h=N/P$. The sum of the partial sums over the D-ary tree may be computed in time D, if the unit of time consists of 1 real addition plus communication of a real number to a neighbor. To see this, note that the unbalanced D-ary tree rooted at node 000..0 contains $(D-1)$-ary, $(D-2)$-ary, $\cdots$, 0-ary subtrees rooted at its children 100..0, 010..0, 000..1, respectively. Assume as an induction principle that the result is true for each subtree of fanout $0, 1, \cdots, (D-1)$. The hypothesis is clearly true for $D=0$ and $D=1$. By the induction hypothesis, the sum from node 100..0 will arrive at 000..0 at time $D-1$, from node 010..0 will arrive at time $D-2$ and so on. Each partial sum can then be added to the total in 000..0 while the next partial sum arrives. It follows that all summation will be completed by time D, completing the induction. The broadcast of the result to the individual processors reverses the above steps and also requires time $O(D)$.

Following the steps in the proof in detail and using the formulation in § 2.3 for the cost of sending messages, we arrive at the result:

THEOREM 3. *The cost of performing an inner product of two vectors of length N on a hypercube with P processors satisfies*

$$T_{inner_product} \leqq 2\gamma N/P+(2\alpha_{short}+2\beta_{short}+\gamma)\log P.$$

We assign the same semantics to other related operations such as *max_vector* and *sum_vector*. Again *max_vector* will be called by each node with the local segment of the vector as argument and returns with the maximum component of the corresponding *global* vector in time $O(\log P)$.

In Fig. 9 we display the execution time of the *inner_product* routine for vectors of increasing size, from 1 to 950 elements per processor, on the 128 processor iPSC. The initial point on the curve is related to the high startup cost for short messages on the iPSC. Since the communication cost is independent of vector size, the curve is essentially linear, reflecting the time taken to compute the local inner products within

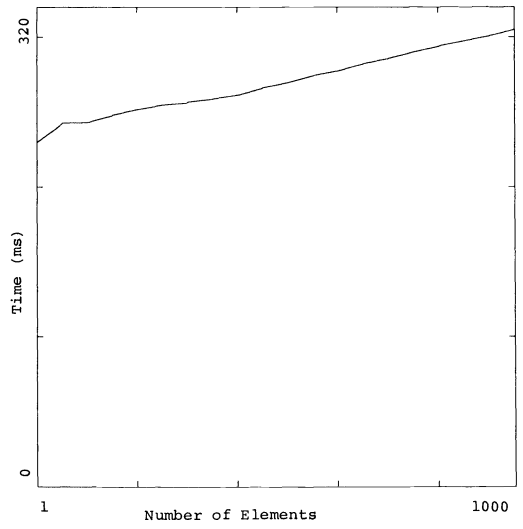


FIG. 9. *Time to compute the inner product of a distributed vector on the* 128 *node* iPSC *as a function of the number of elements per processor.*

each node. As can be seen, even for vectors of length over 120000 elements the inner product is dominated on the iPSC by the communication cost.

6. Parallel matrix operations. Distributed matrices are of special importance in numerical algorithms and we use several different representations for them depending on the application. By far the most important routines in this library are the *transpose* and *matrix_vector* operations. The library also contains routines to allocate and print matrices, to exchange submatrices among nodes, and to convert matrices from one representation to another. We summarize in Table 6 the basic matrix operations of importance. Higher level operations such as addition and multiplication of matrices are easily constructed from the basic set.

TABLE 6
Basic matrix routines.

allocate_matrix	allocates global matrix in each node
delete_matrix	deletes an allocated matrix
convert_matrix	convert between matrix formats
send_matrix_to_cube	host sends a matrix to the nodes
recv_matrix_from_host	host reads a global matrix
send_matrix_to_node	send piece of matrix to another node
recv_matrix_from_node	receive piece of matrix from another node
transpose	transpose a matrix in place
matrix_vector	global matrix times distributed vector
matrix_matrix	product of global matrices
rank1_update	rank 1 update of matrix
areal_parameters	set areal decomposition parameters
shuffle	exchange areal boundaries with neighbors

As with distributed vectors, distributed matrices are represented by small data structures in each processor that encode the location of the data, the type of the matrix representation, the array dimensions and other relevant information. This allows matrices of different types to be handled consistently. Since each processor is aware of the data representation it is working with, it is not necessary to depend on some control processor to handle flow control for matrix operations. The most important representation types are *CONTIGUOUS_ROW*, *DISTRIBUTED_ROW*, *CONTIGUOUS_COLUMN*, *DISTRIBUTED_COLUMN*, *DIAGONAL* and *AREAL*. The allocation routine has the form:

$$mat = allocate_matrix(rows, cols, type, nsparse, sparse).$$

If *nsparse* is nonzero it indicates that the matrix is sparse, and the value of nsparse then indicates the number of nonzero rows, columns or extended diagonals according to the value of *type*. The argument *sparse* is a pointer to an array of *nsparse* integers indicating which rows, columns or extended diagonals are nonzero. Sufficient storage is set aside to store the matrix, and the data structure *mat* when returned includes the sparsity information as well as the size of and pointer to the local matrix segment stored in each processor. The sparse format supported here does not include general sparse matrices, but is adequate for a large class of matrices whose sparsity structure has a diagonal form. For type *AREAL* matrices the sparsity information is ignored, but certain other information is obtained in this case from the *areal_parameters* routine which must be called before any matrix of this type is allocated, see § 6.3.

6.1. Row and column matrix formats. The simplest representations of an $N \times N$ matrix are row or column oriented. We make use of two different row representations. As usual we decompose N as: $N = hP + r$, $0 \leqq r < h$. In the *CONTIGUOUS_ROW* representation, each row is stored entirely in one processor. The first r processors contain $h+1$ consecutive rows of the matrix, which we store contiguously as an $(h+1) \times N$ matrix, while the remaining $P - r$ processors contain h rows of the matrix, stored as an $h \times N$ matrix. In the *DISTRIBUTED_ROW* representation, each row is stored as a distributed vector, with the vector segments in each processor from consecutive rows stored contiguously. This representation is fully defined by the distributed vector format discussed previously and we say no more about it here as a result. We note that the N vector segments stored in one processor form an $N \times (h+1)$ or $N \times h$ matrix. The corresponding column representations are obvious.

The two representations are in fact closely related. Given a *CONTIGUOUS_ROW* representation we can convert it to the *DISTRIBUTED_COLUMN* form by simply transposing each of the submatrices stored in individual processors. Similar transpositions convert a *DISTRIBUTED_ROW* format to *CONTIGUOUS_COLUMN* format. Since no communications are involved, and all local transpositions may occur in parallel, the time to effect these transformations is $O(N^2/P)$.

6.2. The diagonal form of a matrix. The third representation of a matrix we call the *DIAGONAL* form. This form is especially convenient for matrices which have a small bandwidth or which contain only a few nonzero diagonals. We consider an $N \times N$ matrix A with rows indexed by i, columns by j, both i and j in the range $0, \cdots, N-1$. The kth diagonal of A, where k is in the range $-(N-1), \cdots, (N-1)$, consists of those elements $A_{i,j}$ such that $j - i = k$. This diagonal contains $N - |k|$ elements. We construct *extended diagonals* D_m indexed by m in the range $0, \cdots, (N-1)$ as follows. For each $k > 0$, consider the diagonals k and $k - N$ which have, respectively, $N - k$ and k elements. It is therefore possible to construct a vector of length N, which we will index by $m = k$ and call an extended diagonal, consisting of diagonal $k - N$ in the first k components of the vector and diagonal k in the last $N - k$ components. The complete matrix A is uniquely represented by the N extended diagonals of length N. For the rest of the paper we will refer to these extended diagonals simply as diagonals.

It is convenient to introduce a matrix D whose rows are the extended diagonals of A (ordered by the index m introduced above). More compactly, we can describe D as an *index transformation* of the matrix A:

$$A_{i,j} = D_{(j-i) \bmod N, j}.$$

This notation will be used frequently in the sequel without further comment. Figure 10 displays a 6×6 matrix A and its diagonal form D.

The diagonal form of a matrix is stored on a parallel machine by representing each extended diagonal as a distributed vector. In other words the matrix D is stored in *DISTRIBUTED_ROW* form.

$$A = \begin{bmatrix} 00 & 01 & 02 & 03 & 04 & 05 \\ 10 & 11 & 12 & 13 & 14 & 15 \\ 20 & 21 & 22 & 23 & 24 & 25 \\ 30 & 31 & 32 & 33 & 34 & 35 \\ 40 & 41 & 42 & 43 & 44 & 45 \\ 50 & 51 & 52 & 53 & 54 & 55 \end{bmatrix} \qquad D = \begin{bmatrix} 00 & 11 & 22 & 33 & 44 & 55 \\ 50 & 01 & 12 & 23 & 34 & 45 \\ 40 & 51 & 02 & 13 & 24 & 35 \\ 30 & 41 & 52 & 03 & 14 & 25 \\ 20 & 31 & 42 & 53 & 04 & 15 \\ 10 & 21 & 32 & 43 & 54 & 05 \end{bmatrix}$$

FIG. 10. *Conversion of a matrix A to diagonal form D.*

An important aspect of the diagonal form is that it can be constructed from the simple *CONTIGUOUS_COLUMN* or *DISTRIBUTED_ROW* form without any communication. To see this, note that from the expression above for the elements of D, the jth column of the diagonal form is a permutation of the jth column of the matrix A. Since D is stored in *DISTRIBUTED_ROW* form, each column of D is entirely within one processor (see § 6.1). As seen previously, a *CONTIGUOUS_COLUMN* or *DISTRIBUTED_ROW* matrix also has its columns local to processors. It follows that if A is in *DISTRIBUTED_ROW* form, conversion of A to the *DIAGONAL* form D involves only in-column permutations internal to each processor.

6.3. The areal form of a matrix. The *AREAL* form of a matrix is used to distribute a rectangular $N_x \times N_y$ matrix in a block-rectangular form to processors assigned to a logical rectangular $p_x \times p_y$ grid. Each block is surrounded by a dummy boundary of width b rows or columns. The boundary allows the matrix to be treated as if stored in a global memory, provided operations on it have a stencil of radius at most b. Figure 11 displays a typical areal matrix distribution in which the submatrices are chosen to be square and are surrounded by boundaries of width 1. Prior to allocating matrices of type *AREAL* the logical grid dimensions and the boundary width must be specified by a call to the library routine *areal_parameters*:

$$areal_parameters(p_x, p_y, per_x, per_y, b).$$

This routine constructs the logical grid mapping into the hypercube and stores the resulting connectivity information and boundary width b. The routine returns an error value if the logical grid cannot be mapped to the hypercube using only nearest neighbor connections. The grid is periodic in the x or y direction according as per_x or per_y is nonzero. The mapping of a logical grid to the hypercube network has been discussed in § 4.1. Subsequent calls to *allocate_matrix* with type *AREAL* use the information stored during the last *areal_parameters* call, and this information is copied and stored in the matrix data structure returned by the allocation call.

For simplicity we assume in this section that we can factor the number of processors as $P = p_x p_y$ and that N_x and N_y are multiples of p_x and p_y, respectively: $N_x = n_x p_x$,

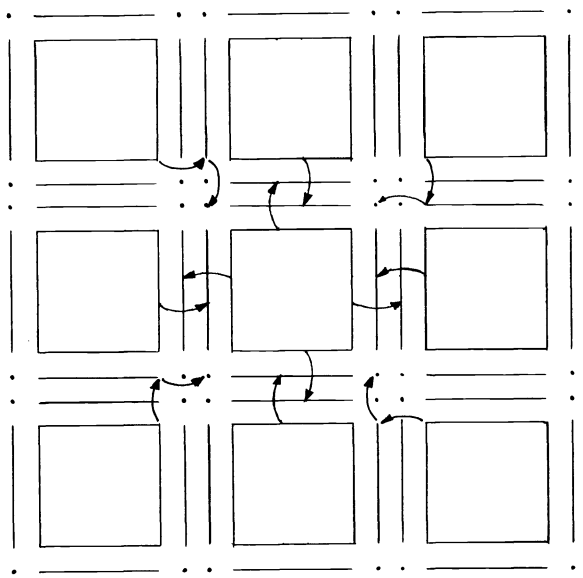


FIG. 11. *The shuffle operation on an areally distributed grid.*

$N_y = n_y p_y$. The processors may be labeled by a grid index as P_g, $g = (i, j)$ with $0 \leqq i < p_y$, $0 \leqq j < p_x$. The required matrix A is then conceptually split into P equal sized rectangular $n_x \times n_y$ submatrices A_g, with each assigned to a separate processor. Matrix element $a_{I,J}$ is assigned to processor $P_{I/n_y, J/n_x}$. The *allocate_matrix* routine allocates storage for these arrays in the processors along with storage for the boundary. Each submatrix is centered in an allocated matrix of size $(n_x+2b)\times(n_y+2b)$. Thus the block A_g is indexed as $A_{g;i,j}$ where $-b \leqq i < n_y + b$, and $-b \leqq j < n_x + b$. The rows and columns in the ranges $[-b, -1]$ and $[n_x, n_x+b-1]$ or $[n_y, n_y+b-1]$ correspond to the dummy boundary rows or columns.

The primary purpose of the boundary points is to allow copies of some matrix elements stored on neighboring processors to be available locally. The *shuffle* library operation:

$$shuffle(matrix, width, sides),$$

updates the boundary data for a submatrix by exchanging boundary rows or columns with all neighboring processors. Only the outer *width* boundary rows or columns are exchanged, although generally *width* will be chosen to coincide with the boundary width b of the matrix. The *sides* parameter specifies which of the 4 sides of the rectangles to exchange. Four symbolic constants NORTH, SOUTH, EAST, WEST are used, with values 1, 2, 4 and 8. Any set of sides may be specified by orring together the corresponding symbolic constants.

An important point to note is that the shuffle operation on the EAST or WEST sides involves sending noncontiguous data, assuming the submatrices are stored by rows. Sending the individual boundary elements is too expensive on the Intel iPSC and so these elements are first copied to a buffer of appropriate size before sending. Similarly *shuffle* receives buffered sides from neighbors, then copies the data to the appropriate noncontiguous boundary locations. We have observed that the cost of these copy operations can be a substantial fraction of the communication time; see for example Table 3.

Applications of the areal distribution of a matrix will be encountered in §§ 8 and 11. In these applications almost all communication is handled by the *shuffle* operation. An important issue is the aspect ratio of the submatrices A_g allocated to each processor. Figure 12 represents mappings of logical 8×4, 16×2 and 32×1 grids onto a 32-node hypercube, and their effect on the aspect ratio of submatrices of a distributed matrix. In principle, the shuffle operation is proportional in cost to the perimeter of a subgrid, and can be minimized for fixed subgrid area by ensuring that the subgrids are near square. We remark that on the Intel iPSC, where short messages are as expensive as those of length 1024 bytes, the shuffle operation is actually a fixed (large) cost operation. The largest real square submatrix that can be stored in an iPSC processor has about 64K elements (256K bytes) and consequently sides of length 256 elements or 1024 bytes. Thus a shuffle operation on one side consumes exactly one send and receive communication call, no matter how small the matrices are. This seriously affects the behavior of many algorithms, and also makes it essentially impossible to measure area/perimeter effects. On the Caltech Hypercube on the other hand such area/perimeter effects are readily observed, see for example Fig. 24. We summarize the overheads for *shuffle* communication in:

THEOREM 4. *The shuffle operation on a matrix of dimension $N_x \times N_y$, distributed areally on a $p_x \times p_y$ 2-dimensional mesh of processors takes time*:

$$T_{shuffle} \leqq 2\beta(N_x/p_x + N_y/p_y) + 2\gamma N_y/p_y + 4\alpha,$$

where the γ term represents the cost of copying noncontiguous boundary columns.

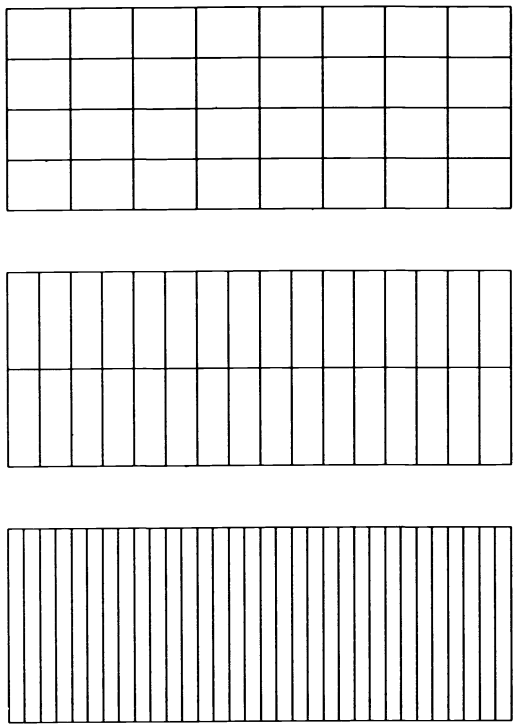


FIG. 12. *Assignment of equal area grid blocks to* 32 *processors but with different area-perimeter ratios.*

6.4. Matrix-vector multiplication. In the case of the distributed-row matrix format, matrix-vector multiplication is essentially trivial. In this case the multiplication reduces to a set of standard inner products of distributed vectors. Some communication may be saved by defining processor r to be the root of the inner product tree (rather than processor 0) when multiplying row r of the matrix with the vector—this saves the broadcast of the result to many processors that do not really need it. The time required for the matrix-vector operation is $O(N^2/P + N \log P)$. This algorithm may be extended to provide a matrix multiplication algorithm for distributed matrices. The row representation is only sensible for matrices that are more or less full, so we move on now to a discussion that is appropriate also for many sparse matrices.

We will show that the diagonal representation of a matrix, combined with the *shift_vector* operation, allows the matrix multiplication to be performed efficiently and in a highly portable fashion. To introduce the algorithm, we consider first the case of a dense matrix A where we wish to compute the product $Y = AX$:

$$\begin{aligned} Y_i &= \sum_{j=0}^{N-1} A_{i,j} X_j \\ &= \sum_{j=0}^{N-1} D_{(j-i) \bmod N, j} X_j \\ &= \sum_{m=0}^{N-1} D_{m,(i+m) \bmod N} X_{(i+m) \bmod N}. \end{aligned}$$

Denote by D_m the mth diagonal of A in diagonal form, by V^k the vector obtained by shifting a vector V, k times, and by $\pm$ the vector product of two vectors: $(u \pm v)_i = u_i v_i$.

Then the above equations can be written in vector form as:

$$Y = \sum_{m=0}^{N-1} D_m^m \overset{*}{+} X^m.$$

In the mth term of this sum both D_m and X are shifted m times. Instead of shifting both the diagonals and the vector X by the same amount, we can more efficiently shift the result vector Y by m in the opposite direction. This leads to the final form of the matrix-vector algorithm:

$Y = 0$
for $m = 0, \cdots, N-1$
    $Y = Y + D_m \overset{*}{+} X$
    $Y = Y^{-1}$
end

The first line in the loop of this procedure is nothing but the *add_vector_times_vector_to_vector* procedure, the second line is the *shift_vector* procedure. Parallelization of both of these operations has already been discussed above. Thus in order to parallelize this procedure is suffices to store the extended diagonals, D_m, X and Y as distributed vectors. The computation time of the matrix-vector operation consists of $O(N^2/P)$ arithmetic operations, as is usual for a full matrix, and the time for N shift operations on the vector Y, which is $O(N)$. Communication overhead—the time spent in shifts—will be negligible compared to that in computation for large matrices, defined as those with $N \gg P$.

This algorithm is perfectly acceptable for full matrices. In the case of a sparse matrix with fill-in along diagonals, such as occurs frequently in PDE discretizations, considerable savings can be accomplished. One obvious saving is that it is unnecessary to multiply with zero diagonals. However the right-hand side still must be shifted once for every diagonal. This results in savings on the number of arithmetic operations. The communication cost however remains N shift operations, although many of these can now be grouped together into longer shifts.

If the bandwidth b of the matrix A is limited, we can save on the communication cost by reducing the number of necessary shifts from N to $3b$. Matrices with bandwidth b are characterized in their diagonal forms by having a cluster of extended diagonals with indices in the range $0, \cdots, b$ and a cluster of diagonals with indices in the range $N-b, \cdots, N-1$. In this case we split the loop of vector operations describing the matrix times vector operation into two parts in the order $m = N-b, \cdots, N-1$ followed by $m = 0, \cdots, b$. We start by initializing Y to zero, which we regard as implicitly shifted $N-b-1$ times. The first loop requires $b-1$ shifts, the second b shifts. Finally, in order to get the right-hand side to its correct (unshifted) place we need to shift b times in the opposite direction. Thus we obtain the algorithm:

$Y = 0$
for $m = N-b, \cdots, N-1$
    $Y = Y + D_m \overset{*}{+} X$
    $Y = Y^{-1}$
end
for $m = 0, \cdots, b$
    $Y = Y + D_m \overset{*}{+} X$
    $Y = Y^{-1}$
end
$Y = Y^b$

This brings the total communication cost to $3b-1$ shifts which is a very substantial saving over the full matrix case. Note that many diagonals within these nonzero bands may vanish. Shifts associated with zero diagonals should again be combined into larger shifts.

As an example, in typical planar PDE discretizations over an $M \times M$ grid the resulting matrices are of size $M^2 \times M^2$ so that $N = M^2$. Usually there are $O(1)$ nonzero diagonals and the bandwidth b is $O(M)$. Thus time spent on communication will be $O(M)$, against time $O(M^2/P)$ spent on computation. For large problems, defined as those with $M \gg P$, communication time will be dominated by computation time.

Extending the analysis above to include communication startup costs we obtain:

THEOREM 5. *On a P-processor hypercube, matrix-vector multiplication for a matrix of dimension N, with bandwidth b and with d nonzero diagonals, requires time*:

$$T_{matrix_vector} \leqq 2d\gamma N/P + \alpha d + 3\beta b.$$

In case the matrix is symmetric some savings can obviously be accomplished in memory required. However, this will result in increased communication since the diagonals have to be shifted as well as the right-hand side in one part of the calculation.

6.5. Matrix-matrix multiplication. The diagonal form is also very well suited to matrix multiplication:

$$C \leftarrow C + A * B,$$

with A, B and C square matrices of dimension N. In describing the algorithm, we will use a simplified notation in which lower case characters refer to elements of the matrix, while upper case characters refer to elements of the diagonal form. Thus, we have that:

$$a_{i,j} = A_{(j-i) \bmod N, j}.$$

We will further simplify expressions by performing all index arithmetic modulo N for the rest of this section, so that the above formula reduces to:

$$a_{i,j} = A_{j-i,j}.$$

As in the previous section, we use singly subscripted capital letters to represent the extended diagonals. We also use the superscript notation introduced above for vector shifts and $\pm$ for element-wise vector products.

With the above notation, the classical matrix multiplication formula becomes:

$$C_{j-i,j} = \sum_{k=0}^{N-1} A_{k-i,k} B_{j-k,j}.$$

With $p = j - i$, and $q = j - k$ we have:

$$C_{p,j} = \sum_{q=0}^{N-1} A_{p-q,j-q} B_{q,j}.$$

In vector form, using diagonals of A and B and the vector product, this formula reduces to:

$$C_p = \sum_{q=0}^{N-1} A_{p-q}^{-q} \pm B_q.$$

It is now easily shown that the following algorithm implements the operation, using only fully parallel operations:

$$\begin{array}{l}\text{for } q=0,\cdots,N-1\\ \quad\text{for } p=0,\cdots,N-1\\ \quad\quad C_p \leftarrow C_p + A_{p-q} \mathbin{*} B_q\\ \quad\quad A_{p-q} \leftarrow A_{p-q}^{-1}\\ \quad\text{end for}\\ \text{end for}\end{array}$$

The first line in the loop is the *add_vector_times_vector_to_vector* procedure, which is trivially parallel. The second line is the *shift_vector* procedure. Note that after the operation, all matrices, including A, are in their original form. Only the diagonals of A need to be stored as distributed vectors of type *SHIFT*.

For full matrices, this operation requires one communication operation (time to transfer one floating point variable) for every $2N$ floating point operations. More precisely, the following theorem is valid:

THEOREM 6. *On a P-processor hypercube, matrix-matrix multiplication for full matrices of dimension N, takes time*

$$T_{matrix_matrix} \leqq \gamma N^3/P + (\alpha+\beta)N^2,$$

where the γ term represents the combined cost of a floating point multiply and addition.

For $N \gg P$ this algorithm is therefore asymptotically optimally parallel. For sparse matrices with fill-in along diagonals the loops can be easily adapted so that shifts can be grouped together when zero diagonals of B are encountered and to skip the loop body entirely whenever A_{p-q} is zero. Another refinement is possible, important even for full matrices on machines like the iPSC with high message startup cost. Shifts on all diagonals should be done within the same vector shift operation so that all values can be transmitted in one communication operation after appropriate buffering. This requires a modification to the *shift_vector* routine allowing a list of vectors to be specified.

6.6. Rank one update. The diagonal form also is suitable for implementation of rank one updates:

$$A \leftarrow A + xy^T.$$

The elements of the matrix $B = xy^T$ are $b_{i,j} = x_i y_j$. It follows that the diagonal form of B has elements $B_{k,j} = x_{j-k} y_j$, where we have used the notation of the previous section as well as indexing modulo N. This may be expressed in terms of vector shifts and element vector products as:

$$B_k = x^{-k} \mathbin{*} y.$$

This immediately implies the following algorithm for a rank one update:

$$\begin{array}{l}\text{for } k=0,\cdots,N-1\\ \quad A_k \leftarrow A_k + x \mathbin{*} y\\ \quad x \leftarrow x^{-1}\\ \text{end for}\end{array}$$

This algorithm requires one shift for every $2N$ floating point operation. All of the arithmetic operations are optimally parallel, hence we have the following:

THEOREM 7. *On a P-processor hypercube, the rank one update of an $N \times N$ matrix takes time*

$$T_{rankone} \leqq \gamma N^2/P + (\alpha+\beta)N$$

where the γ term represents the combined cost of a floating point multiply and addition.

6.7. Parallel matrix transpose. The matrix transpose operation allows parallelization in a portable and efficient way for many 2-dimensional numerical methods, see §§ 7, 8 and 13. The goal of this operation is to convert a row distributed matrix into a column distributed matrix. We present two transpose algorithms and their implementations. The first algorithm uses the diagonal form of the matrices introduced above and the vector shift operation. For this algorithm to be efficient, the optimized vector shift algorithm which we discussed previously is crucial. The second algorithm we call the block transpose method. Both algorithms presented require a communication time proportional to $N^2 \log P/P$ floating point number transfers, where N is the dimension of the matrix and $P=2^D$ is the size of the hypercube. These estimates are based on the assumption that the cost of communications is linear in the amount of data transferred. As discussed previously this assumption is not valid for the iPSC. In particular the transpose based on the shift algorithm is strongly affected by the startup cost for communication, and consequently appears much slower on the iPSC than the block transpose.

6.8. The shift transpose algorithm. There are two key points to understanding this transpose algorithm. First, it is easy, and requires no communication between processors, to convert a matrix from the row distributed form to the diagonal form. This issue has already been addressed in § 6.2 where the diagonal form was introduced. Secondly, it is straightforward to transpose the diagonal form. We now examine the second point in greater detail.

The element $A_{i,j}$ of the matrix A is equivalent to element $D_{(j-i) \bmod N,j}$ of the diagonal form D. The transpose A^T of the matrix A, has a diagonal form DT such that

$$DT_{(j-i) \bmod N,j} = A^T_{i,j} = A_{j,i} = D_{(i-j) \bmod N,i}.$$

From this it follows that

$$DT_{k,j} = D_{N-k,i} = D_{N-k,(j-k) \bmod N}.$$

This means that the kth diagonal of DT is the $(N-k)$th diagonal of D, shifted by k. Note that once $k > N/2$ it is cheaper and equivalent to shift by $N-k$. As a result, the cost of this algorithm, using the simple shift operation (see § 5.3), is $N(N-1)/4$ element transfers, which makes it comparable in cost to transposing the whole matrix on a serial machine. Thus this algorithm makes no effective use of parallelism! It is therefore essential to use the optimized shift operation. We show in the next subsection that the number of communication operations is then reduced asymptotically to $(\log P - 1)N^2/P$, which, apart from the log P factor, indicates almost optimal use of parallelism. For a 5D cube this is about a factor of 2 better than using the straight shift while for a 7D cube the gain is about a factor of 5.3. We will assume from now on that the shift transpose only uses the optimized version of the shift vector operation. Note however that the length of vectors must then be a multiple of the number of processors.

In the implementation of this transpose algorithm for a matrix A of *DISTRIBUTED_ROW* type, we never actually store the matrix D. Instead of storing the whole diagonal form, we allocate two distributed vectors of type *SHIFT*. At the kth step,

extended diagonals k and $N-k$ of A are constructed in these vectors and respectively shifted by k and $-k$. Once these shift operations are performed, the diagonals are copied back to the matrix, taking care that the diagonals are interchanged correctly.

6.8.1. Communication cost for shift transpose.

THEOREM 8. *The shift transpose of an N by N matrix on a P-processor hypercube takes time*:

$$T_{shift_transpose} \leqq (\beta \log P + \gamma) N^2/P + \alpha N \log P.$$

Proof. The communication cost for transposing an $N \times N$ matrix using this algorithm is incurred by shifting the mth (extended) diagonal m times, with m taking on the values $0, \cdots, N-1$. For each m, we can find m_s and m_e such that $m = m_s h + m_e$ $(0 \leqq m_e < h)$, where $h = P/N$. The m element shifts required for diagonal m can be obtained by performing m_s segment shifts and m_e element shifts. As m ranges from 0 through $N-1$, m_e assumes each of the values $0, \cdots, h-1$, P times and m_s assumes each of the values $0, \cdots, P-1$, h times.

From the range of values assumed by m_e, we can determine the total cost T_e associated with the element shifts:

$$T_e = P \sum_{k=1}^{h-1} ST(k),$$

where we have used notation introduced in § 2.3. $ST(k)$ is the cost of communicating a segment of length k to a neighbor processor.

We focus our attention now on the segment shifts. Since segment shifts over more than $P/2$ processors are better done as segment shifts in the opposite direction, we can divide the set of values assumed by m_s into three subsets: $[1, P/2-1]$, $[-P/2+1, -1]$ and the single value $P/2$. As indicated earlier in discussing the optimized shift algorithm, the actual number of segment transfers to be performed to complete a segment shift over m_s processors is most easily determined by considering set bits in the binary representation of m_s: bit 0 represents 1 segment transfer, the other bits represent 2 physical segment transfers. Since $P = 2^D$, the values in the first range are represented by $D-1$ bits and as the range is traversed each bit is set half of the time, i.e., in $P/4$ cases. It follows that the total number of physical segment transfers for the first range of values of m_s is: $(2D-3)P/4$. The second range of m_s values requires by symmetry an identical number of segment transfers, while the final value $m_s = P/2$ requires 2 segment transfers. Since each value m_s is assumed h times and the cost of a segment transfer is $ST(h)$ it follows that the total cost of all segment shifts required in the transpose algorithm is:

$$T_s = ((D-1.5)P+2)hST(h).$$

Adding together the quantities T_e and T_s derived above, and using the relation $ST(k) = \alpha + \beta k$ introduced in § 2.3, we obtain as the total time for the shift transpose of a full matrix:

$$\begin{aligned} T_{shift} &= ((D-1.5)P+2)h(\alpha+\beta h) + P(h-1)(\alpha+0.5\beta h) \\ &\leqq \beta N^2 \log P/P + \alpha N \log P. \end{aligned}$$

The term proportional to α gives the effect of message startup costs and can dominate for small N on machines such as the iPSC where $\beta \ll \alpha$. To complete the proof of the theorem we add in the computation cost related to the copying of data internal to processors.

The shift transpose algorithm is well suited to diagonally sparse matrices, whereas in the matrix-vector multiplication algorithm only the nonzero diagonals actually require shifting.

6.9. The recursive block transpose algorithm. An algorithm for transposing a matrix in a time of order $N^2 \log P/P$ is possible without requiring that the size of the matrix be a multiple of the number of processors (as was necessary for the optimized shift operations used in the previous section). Note that a 2×2 matrix is transposed by exchanging its upper right and lower left elements. This leads to the following serial algorithm to transpose a matrix [39], [28]:

$$\textit{Transpose}(M):$$

$$M = \begin{bmatrix} M_{00} & M_{01} \\ M_{10} & M_{11} \end{bmatrix}$$

$$M_{01} \leftrightarrow M_{10}$$

$$\begin{array}{ll} \textit{Transpose}(M_{00}) & \textit{Transpose}(M_{01}) \\ \textit{Transpose}(M_{10}) & \textit{Transpose}(M_{11}) \end{array}$$

A matrix with dimension $N = 2^L$ can be transposed by recursive application of this procedure through L levels. At the final level all matrices are 1×1 and so are trivially transposed. We indicate the first three levels in this procedure in Fig. 13, where the arrows represent block interchanges.

To implement this algorithm on a hypercube we store the matrix M in contiguous-row form and assume that its dimension N is divisible by the processor number $P = 2^D$: thus $N = h2^D$. The case where the size of the matrix is not a multiple of the number of processors does not pose significant problems. We may order the processors either according to their natural order or according to the BRGC ordering. With the natural ordering, we give up nearest neighbor connectivity of adjacent rows; however processors that are a distance 2^d apart in processor numbers differ in only one bit in their binary representation and are therefore nearest neighbors on the network. With the BRGC ordering, nearest neighbor connectivity is preserved at the cost of doubling the communication cost between such processors. As discussed in §§ 4.1 and 4.3, with BRGC order the physical distance between processors at logical distances of a power of 2 is only 2. The experiments we present in later sections were performed with the natural ordering.

At level L in the algorithm the matrices are of dimension $h2^{D-L}$ and are stored as contiguous-row distributed matrices over 2^{D-L} processors. Furthermore these processors are a contiguous sequence of processors and form a $(D-L)$-dimensional subcube of the hypercube. At level D all submatrices are local to one processor and may then be transposed by the normal serial algorithm. Thus we terminate the recursion at level D.

The crucial point in analyzing this algorithm is that the interchange of the upper right and lower left subblock at level L involves communication between processors with processor numbers differing by 2^{D-L-1}; hence the two communicating processors are at a physical distance of 1 (or 2 if BRGC ordering is used). In principle the transpose of the 4 subblocks can be done in parallel. However since the 4 blocks are distributed over only 2 subcubes, the effective parallelism is only 2. Equivalently, the two upper blocks span the same set of processors; the two lower blocks also span a common set of processors. The total time for this algorithm is asymptotically $O(N^2 \log P/P)$ as shown in the following subsection.

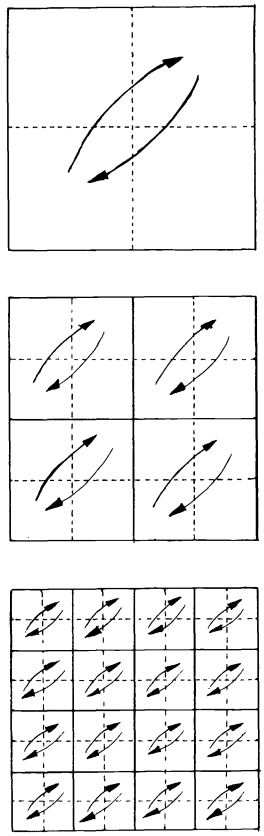


FIG. 13. *Schematic representation of the block transpose algorithm as it would be executed on a 2-dimensional hypercube (4 processors).*

6.9.1. Communication cost for the block transpose algorithm.

THEOREM 9. *The block transpose of an N by N matrix on a P-processor hypercube takes time*:

$$T_{block_transpose} \leqq ((\alpha/\lambda + \beta + \gamma) \log P + 1.5\gamma)N^2/P + 2(\alpha + \beta\lambda)P.$$

Proof. The fundamental communication taking place in the block transpose algorithm is the transfer of a block of k rows of length l (the submatrices) from one processor to a nearest neighbor processor (assuming natural ordering). The cost associated with this transfer, which we denote by $BT(k, l)$, depends on the implementation details of the transfer, and we consider three possibilities. The most efficient solution is to send a contiguous array of kl elements, entailing a block transfer cost of:

$$BT^1(k, l) = ST(kl),$$

where $ST(k)$ is the segment transfer cost introduced in § 2.3. The rows of the individual submatrices are not stored contiguously however. This makes it more convenient to send over individual rows of the matrix one by one, in which case we obtain

$$BT^2(k, l) = kST(l).$$

Because of the high cost of sending short messages, this procedure is very expensive on the iPSC. We avoid this by using a buffering scheme to collect the noncontiguous

rows before transmission. In practice this is accomplished by calling the *send_local_vectors* routine from the vector library, see § 3.3. With a limited buffer length λ, and including time necessary to copy the data into the buffer, the estimated communication time is:

$$BT^3(k, l) = (kl)/\lambda ST(\lambda) + \sigma ST((kl) \bmod \lambda) + \gamma kl,$$

where in the first term integer division by λ is assumed and σ is 0 if kl is a multiple of λ and is 1 otherwise.

Consider now a $2^d h \times 2^d h$ matrix on a d-dimensional hypercube. To exchange the upper right submatrix with the lower left submatrix we use the above transfer operation. Both submatrices have dimensions $2^{d-1}h \times 2^{d-1}h$, each distributed over 2^{d-1} processors. Thus each processor contains a block of h rows of length $2^{d-1}h$. Thus the total interchange of the two submatrices requires time $2BT(h, 2^{d-1}h)$.

Let B_d be the *communication* time required to block transpose a matrix of dimension $2^d h \times 2^d h$ on a d-dimensional hypercube, which we call a level d transpose. As indicated above, the level d transpose is reduced to a block exchange plus four level $(d-1)$ transpose operations. Using the block transfer time introduced above, and the fact that the effective parallelism in the submatrix transposes is 2, we therefore obtain the following recursion relation:

$$B_d = 2B_{d-1} + 2BT(h, 2^{d-1}h),$$

with an initial value $B_0 = 0$. It follows that:

$$B_D = P \sum_{i=0}^{D-1} 2^{-i} BT(h, 2^i h).$$

We can now substitute the appropriate expression for BT^i in terms of ST and use the relation $ST(k) = \alpha + \beta k$ from § 2.3. We obtain as bounds for communication cost in transposing an $N \times N$ matrix, where $N = Ph = 2^D h$:

$$B_D^1 \leqq 2\alpha P + \beta N^2 \log P/P,$$

$$B_D^2 \leqq 2\alpha N + \beta N^2 \log P/P,$$

$$B_D^3 \leqq 2(\alpha + \beta\lambda)P + (\alpha/\lambda + \beta + \gamma)N^2 \log P/P.$$

We notice that for asymptotically large N, the buffered mechanism is the most expensive as expected since for large enough N the individual rows exceed the buffer length. Since the first method is not practical due to the matrix storage scheme, the second method is therefore indicated for large matrices. For smaller matrices on a machine like the iPSC the third method has a great advantage because of the smaller coefficient of the term proportional to α, the message startup cost. In fact using actual iPSC measurements we find that for all but the largest matrices that will fit on the hardware, the term proportional to α plays a dominant role, making the buffered scheme a necessity. To complete the proof of the theorem, we add the computation cost involved in performing P serial transposes of $N/P \times N/P$ matrices in each processor, which consumes time $3/2\gamma N(N/P - 1)$.

6.10. A comparison of transpose algorithms on the iPSC. We now compare the best block transpose method with the shift transpose. From the higher order terms in

N it would seem that both methods are equivalent as far as communication time is concerned. The linear term in the shift transpose cost however dominates for practical problems on the iPSC. Even for the largest matrix that can be stored on the iPSC (2048×2048), the segment length of the distributed vectors used in the algorithm is only 16. This means that all communications taking place are well under the effective minimum message length of 1024 bytes or 256 floating point variables. For the block transpose, the number of floating point variables per message is proportional to N^2/P^2. For the largest problem this is 256 elements per smallest submatrix encountered. Therefore only the values of α and β for long messages play a role if the buffering scheme described above is used. For smaller problems, the short message values influence timings in the exchange of the smaller submatrices. These remarks are borne out by the coefficient of the term proportional to α in Theorems 8 and 9. In the shift transpose case this coefficient is proportional to N while in the block transpose case it is proportional to P and is in fact therefore independent of matrix size.

For clarity, we have neglected the "computation" cost in some formulae above, although this is included in the theorem statements. The basic computation unit for the transpose algorithms is the assignment of an indexed floating point variable, i.e., a statement of the form: $x = y_i$. Assuming this statement takes time γ, we have expressed the computation time as a multiple of γ in Theorems 8 and 9. For the algorithms studied, executed on the iPSC, these costs turn out to be negligible. For example we find that the "computation" cost on a 128 node iPSC for a 2048×2048 matrix is about 10% of the measured transpose time. Thus the transpose is dominated by communication.

6.11. Results and timings. In the figures for this section we always display times as a function of matrix *size*, i.e., of the number of elements N^2. On a serial machine the cost of matrix transpose is a linear function of its size. In Fig. 14 we display the cost of transposing a matrix on the 5D Caltech Hypercube as a function of the size of the matrix. The timings displayed use the transpose algorithm based on the diagonal form and the nonoptimized vector shift operation.

Figure 15 presents corresponding timings for the optimized shift transpose algorithm and the block transpose algorithm as a function of matrix size N^2 on the 128 processor iPSC. Note that, for the largest matrix that can be stored on the iPSC (dimension 2048), the block transpose is much faster than the shift transpose, despite the fact that the latter is asymptotically slightly faster. The linear form of the block transpose curve indicates that that algorithm has already attained its asymptotic behavior. In Fig. 16 we display the timing for the block transpose on a better scale where it becomes clear that the curve has in fact approached its asymptotic rate after an initial high startup overhead for small matrices.

7. Transpose splitting algorithms. We introduce here a general class of algorithms which may be solved on parallel machines with almost no modification to an appropriate serial program. We term these algorithms *Transpose Splitting Algorithms* because the application of a grid point operation is split into two parts: one applied to the x-direction and one to the y-direction. Examples are operator-splitting hyperbolic methods and ADI methods, though many other algorithms may be recognized to be of this type.

Consider a 2-dimensional function $M(x, y)$ on a grid, where x, y label the rows and columns of grid points. We store the values of M as a contiguous-row distributed matrix (see § 6.1), with one or more complete rows of points per processor. Many numerical algorithms amount to applying one or more operations $O(x, y)$ successively

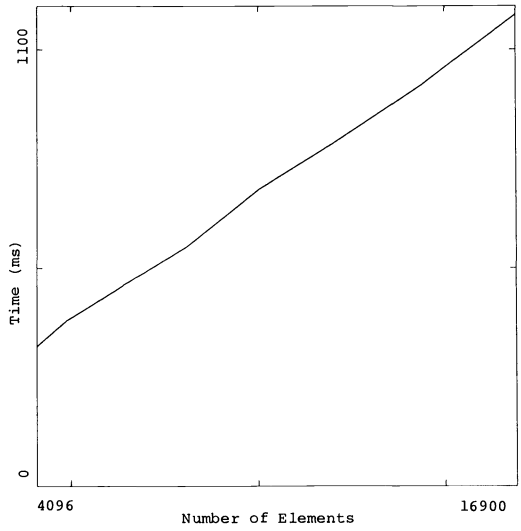


FIG. 14. *Cost of transposing a matrix on the* 32 *node Caltech Hypercube as a function of the number of elements. Transposing using the nonoptimized vector shift algorithm.*

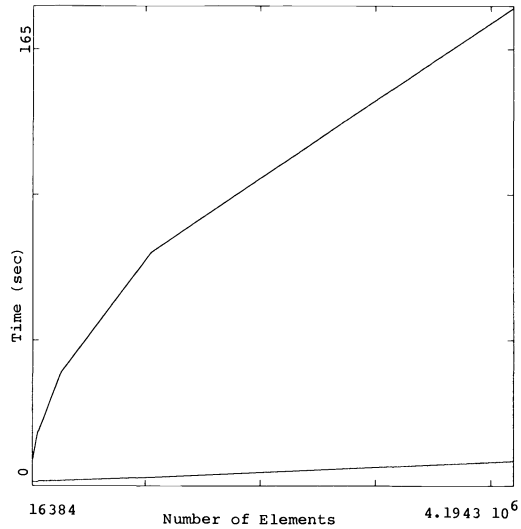


FIG. 15. *Time to transpose a matrix with the shift transpose (upper curve) and the block transpose (lower curve) algorithm, respectively, on the* 128 *node* iPSC *as a function of the number of elements.*

to M at each point x, y, which we denote by the notation $O() \cdot M()$. This would be represented in code as a loop of the form:

$$\text{for } \textit{each } x, y\text{: } O(x, y) \cdot M().$$

Suppose that the operation O may be split into x and y parts which we represent schematically as:

$$O(x, y) = B(y) \cdot A(x),\ x, y \textit{ in grid.}$$

Now assume that $A(x) \cdot M(x, y)$ applied at point x, y uses only values $M(x, y)$ from the same row y containing the point. Similarly we assume that $B(y)$ applied at x, y

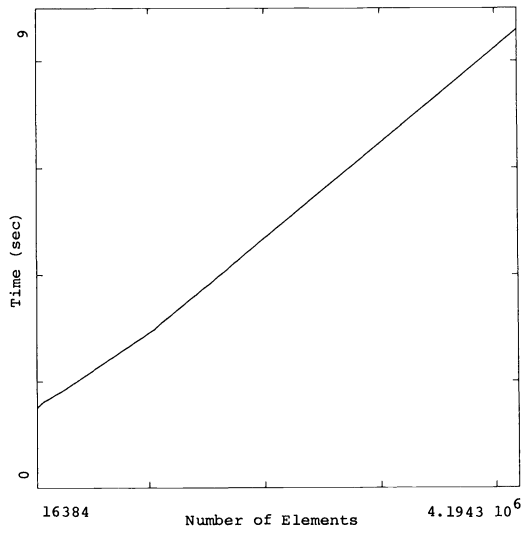


FIG. 16. *Time to transpose a matrix with the block transpose algorithm on the* 128 *node* iPSC *as a function of the number of elements.*

uses only values $M(x, y)$ from the same column x. An equivalent code segment for application of O would then be:

```
for each y
    for each x: A(x) · M()
for each x
    for each y: B(y) · AM()
```

Note that in the application of A the inner loop is along a row, and since $A()$ involves only elements from the same row, and since each row is entirely in one processor, it follows that the inner loop involves no communication whatever. The outer loop, the loop over rows, also involves no communication and is furthermore completely parallelized. Thus A may be applied to M without communication and in an optimally parallel fashion.

Unfortunately the loop with B is not so nice. Here there is communication every time that B involves adjacent column elements that happen to be in different processors. However there is a trivial trick that removes all communication from the B loop and also parallelizes it. The idea here is first to apply a transpose to M, or rather to the matrix $A \cdot M$. Then the roles of x, y in the B loop are reversed and the relevant code becomes:

```
N = transpose(A · M)
for each y
    for each x: B(x) · N()
```

All communication between processors in the application of O to M is now hidden in the transpose operation. Outside of this the code is optimally parallel.

Even for situations in which $A(x)$ applied at point x, y uses values of $M(x, y)$ from another row, this algorithm can still provide a useful approach to parallelism. To accomplish this one stores the matrix M in AREAL form but in the extreme case where the processors are assigned as a $1 \times P$ matrix. Then a vertical *shuffle* is used to

ensure that the data required by $A(x)$ when operating on boundary rows of subgrids are available as needed. After a transpose and *shuffle* the data required for B will also be available as needed.

8. Hyperbolic equations. The hyperbolic equations we have studied are of the form of conservation laws:

$$\frac{\partial u}{\partial t} = -\nabla \cdot F(u).$$

The nonlinearity of the function $F(u)$ may result in the formation of discontinuities (shocks) in a finite time. Hence, the numerical methods used to solve such equations must also be able to handle discontinuities of the solution. The random choice method [40] is one solution method that is sensitive to shocks. We have also implemented simple finite-difference schemes such as the Lax-Wendroff method [41].

We have parallelized the hyperbolic solution in several different ways. In one approach, the parallelization of these methods is a typical application of the transpose splitting strategy, which we have discussed as a general method in § 7. The application to hyperbolic equations will be given in § 8.3. Another approach is based on decomposition of the rectangular grids into rectangular subgrids of near-square aspect ratio, with each subgrid assigned to a separate processor, see §§ 6.3 and 8.2. We have implemented the first approach on the Caltech Hypercube. We compare both methods on the Intel iPSC and conclude that the former involves substantially less development effort, while yielding comparable efficiency to the latter approach.

Our test problems have been model hyperbolic equations—scalar equations with simple nonlinearities. Real physical problems involve complex equations of state and may require substantial amounts of numerical computation per grid point. Real problems typically involve only the same communication costs as the model problems however, unless they are discretized using very high-order schemes. To assess the behavior of hypercubes on such codes, we use the same hyperbolic solvers mentioned above, but with a dummy nonlinear function that simply executes n floating point operations. We preserve all communication operations exactly as they would be in the hyperbolic solver. This allows us to measure efficiency as a function of work done per grid point, and of the number of grid points. The results of these studies are very favorable. If significant amounts of work are done per grid point, hypercubes can provide near optimal efficiency even on very coarse grids, down to a few dozen points per processor. The crucial point here is that the model problems already perform well. This ensures that real problems will perform even more successfully. In these studies we use the *computational efficiency* of a program, defined as the ratio of computation time to execution time of the program, as a measure of communication performance. These results are discussed in detail in § 9, although the methods used to parallelize the programs are developed in this section.

8.1. Discretization methods. We have developed parallel hyperbolic equation solvers based on both finite difference and random choice methods. While standard difference schemes vectorize easily, this may not be so for the random choice method since different random sequences may be used in different spatial locations and because the underlying Riemann solutions generally involve complex logic. However random choice parallelizes as easily as do difference methods. In addition, realistic physical problems involve complex equations of state which may be difficult to vectorize, but

again these parallelize readily. These are examples where MIMD parallelism has a real advantage over SIMD parallelism.

The random choice method (RCM in the sequel) is a general purpose solution method for hyperbolic conservation laws, allowing formation of shocks without affecting convergence of the method. A fundamental building block of the method is the 1-dimensional Riemann solver for the conservation law under consideration. Given a step function at time t describing two constant states connected with each other through a shock, the Riemann solver returns the solution for the given conservation law at time $t+\Delta t$.

The RCM uses a Riemann solver to construct approximate solutions of 1-dimensional conservation laws for all time. The function describing the state at time t is approximated by a sequence of step functions. At each jump of the step function a Riemann problem is solved, resulting in a solution that is no longer piecewise constant. The process is then repeated. To extract a step function from an approximate solution, a random point is chosen in each of the elementary Riemann solutions. It can be shown [40], [42] that this method converges to the correct solution even in the case that shocks are present in the system. In two dimensions, the RCM can still be applied using operator splitting, first performing 1-dimensional random choice solutions along each row, and then along each column.

In parallelizing this method we have distributed the state variables $u(x, y)$ in several ways. We have represented the solution as both an areal-distributed grid function, using a distributed matrix of type *AREAL*, and as a row-distributed grid function, using a distributed matrix of type *CONTIGUOUS_ROW*—see §§ 6.1 and 6.3 for discussion of these matrix types. The communication pattern varies with the distribution scheme and we analyze the communication costs of each scheme in the following subsections. In the areal case communication is provided by the *shuffle* operation whereas for the row case the *transpose* operation is used. There is one further issue involved in the parallelization. As is normal for explicit hyperbolic solution methods, the length of the time-step Δt is constrained by a Courant condition based on the maximum wave speed. The computation of the maximum wave speed uses the *max_vector* routine to compute the global maximum of the local processor maxima and broadcast the results back to the individual processors. Because of the semantics of the *max_vector* routine, all processors can compute the new time step independently of each other, although of course they obtain the same result. As a result the algorithm is completely independent of node location in the network. With both distribution methods all communication is handled entirely by the vector and matrix libraries.

The same parallelization schemes can be applied to finite difference schemes. In addition to random choice we have used the Lax-Wendroff method [41]. The key point is that both methods involve only local stencils of a few grid points. For the row-distribution method it is also important that the discretization scheme be amenable to operator splitting.

8.2. Areal decomposition. In the areal decomposition, a logical $p_x \times p_y$ rectangular grid is mapped onto the hypercube network as discussed in § 4.1. Thus the processors may be labeled by a grid index as p_g, $g = (i, j)$ with $0 \leqq i < p_y$, $0 \leqq j < p_x$. For simplicity we assume in this section that we can factor the number of processors as $P = p_x p_y$ and that the computational domain is a rectangular $N_x \times N_y$ grid where $N_x = p_x n_x$ and $N_y = p_y n_y$. A grid function $U = \{U_g\}$ representing the solution values of the hyperbolic equation on the $N_x \times N_y$ grid is now allocated as a distributed matrix of *AREAL* type with boundary width 1 as described in §§ 6.1 and 6.3. Thus the component U_g is

indexed as $U_{g;i,j}$ where $-1 \leqq i \leqq n_y$, $-1 \leqq j \leqq n_x$, with the rows and columns numbered -1 and n_x or n_y being the extra boundary rows. To perform the hyperbolic time step computation on the grid, a *shuffle* operation, see § 6.3, is first performed in order to fill the boundary perimeter of each U_g with the appropriate values from the neighboring subgrids. Following this, the standard sequential hyperbolic method is performed in the interior of each subgrid, i.e., for rows and columns in the range $[0, n_y - 1] \times [0, n_x - 1]$. In the course of this computation the maximum wave speed within that subgrid is computed, and is stored into the local component of a distributed vector of length P. The final phase in the solution is the computation of a new time step, which is based on a Courant condition involving the maximum wave speed. To compute the maximum wave speed, the *max_vector* operation is applied to the vector of local maxima, which provides the global maximum value to all of the processors (see § 5.5 for the semantics of *inner_product* and *max_vector*). This allows all of the processors to compute the next time step simultaneously, using only vector/matrix library routines for all communication activities, and ensuring an SIMD form for the computation in cases where the local discretization operation is translation independent.

8.2.1. Communication costs. For simplicity of analysis, we begin by assuming that the number of processors is a perfect square, $P = p^2$, and that the computational domain is a square grid of N points, with $N = n^2 p^2$. Thus there are n^2 grid points per processor. For any hyperbolic scheme, the computation to be performed per grid point is a local operation in grid space and thus we regard the basic computation per grid point as an $O(1)$ operation. We stress that it may in fact be a large number if the physics or the discretization method is complex. Consequently the cost of the hyperbolic sweep over a subgrid is $O(n^2)$, proportional to the area of the subgrid. Since all subgrids are computed in parallel, this is also the cost of the sweep over the whole grid. The cost of the shuffle exchange operation at the start of the time step is $O(n)$, since the amount of data to be transferred is proportional to the perimeter of the subgrids, and the communications for different subgrids are performed in parallel. Finally the time step computation involves a cost of $O(\log P)$, which is essentially negligible since the maximum is evaluated over a tree. It follows that the computation time per time step for the complete hyperbolic solution is $O(n^2)$, i.e., $O(N/P)$, and that communication costs are negligible compared to computation costs provided only that sufficiently many grid points are stored in each processor, *independent* of the number of processors. Equivalently, the computational efficiency of the algorithm is given by:

$$E = T_{computation} / T_{execution} = \frac{1}{1 + O(1/n)}.$$

If the time step had been evaluated by accumulating the maxima in a single processor, then the time step computation would be $O(P)$ and might actually dominate for problems such that $N < P^2$. We remark that machines which involve a high overhead for sending short messages provide especially poor performance for the areal algorithm on coarse grids, since the subgrid perimeters may be smaller than the effective minimum packet size. In particular, on the Intel iPSC the effective minimum packet size is 1024 bytes, so that a subgrid edge must contain at least 256 points for fully efficient communication. Thus a subgrid would contain 65536 grid points, close to the maximum available user memory (about 275K bytes including the program). Below this size the above analysis is not strictly accurate, since the cost of a boundary shuffle should be taken as $O(\alpha + \beta n)$ in the notation of § 2.3, resulting in a much lower efficiency for small n when α is large as in the iPSC case. In this case the efficiency has the asymptotic

form:

$$E = \frac{1}{1 + O(\alpha/n^2 + \beta/n)}.$$

Returning to the more general case of rectangular subgrids, we now study the behavior of the computational efficiency as a function of the aspect ratio $a = n_x/n_y$, the ratio of height to width of the subgrid, where we assume without loss that $n_x \leqq n_y$. The computation cost is $O(n_x n_y)$, whereas the communication cost is $O(n_x + n_y)$, ignoring the time step cost. It follows that the computational efficiency of the algorithm is:

$$E = T_{computation}/T_{execution} = \frac{1}{1 + O(1/n_x + 1/n_y)}.$$

This expression is maximized for fixed subgrid size $n_x n_y$ when $n_x = n_y$, which corresponds to the case of square subgrids. The efficiency is minimized by allowing n_x to be as small as possible, which occurs if $p_x = P$ and $p_y = 1$ corresponding to the bottom configuration in Fig. 12. This corresponds to a linear decomposition in which whole columns are distributed to each processor, so that the state variables are effectively represented as a contiguous-column matrix. Assuming that the overall grid is square so that $N_x = N_y$, it follows that $n_x = \sqrt{N}/P$, and $n_y = \sqrt{N}$. Correspondingly the minimal efficiency may then be expressed as:

$$E = \frac{1}{1 + O(P/n_x n_y)^{1/2}},$$

where we note that $n_x n_y$ is the number of grid points per processor. The efficiency may no longer be close to 1 if the number of processors exceeds the number of grid points per processor. In particular, for fixed processor memory size (which limits the subgrid size $n_x n_y$), the efficiency goes to 0 as larger problems are solved with correspondingly more processors.

8.3. Linear decomposition + transpose. For this section we assume that the hyperbolic discretization method is of operator splitting type and consists of a grid sweep with a horizontal stencil operator in the x direction, followed by a grid sweep with a vertical stencil operator in the y direction. We are then in a position to apply the transpose splitting parallelization discussed in § 7. Thus the state variables are distributed as a matrix in *CONTIGUOUS_ROW* form, in the notation of § 6.1. In particular, N/P rows of grid points are assigned to each processor. The hyperbolic sweep (1-dimensional) is now applied to each row, involving no communication cost since all required stencil values are in a single row and are thus local to a processor. Following this, the matrix of state values is transposed, so that effectively it is now in *CONTIGUOUS_COLUMN* form. Then the hyperbolic sweep is applied in the y-direction, which now means along rows, and thus again no communication is required. Finally the time step is computed using the *max_vector* routine. In the following time step we perform the y-sweep before the x-sweep which saves on performing one transpose per time-step. This algorithm differs from the corresponding algorithm on a shared memory machine or on a serial machine only in one line—the presence of the matrix transpose. Indeed, even on a serial machine there may be an advantage in performing the transpose if there is a hardware cache present.

8.3.1. Communication costs. As usual, the computation cost is proportional to the number of grid points per processor and is thus $O(N/P)$. The only communication

costs in this version of the algorithm are the transpose and the negligible $O(\log P)$ cost of the *max_vector* routine. As discussed in § 6.7, the transpose may be performed using the block transpose algorithm in $O(N \log P/P)$ time. It appears that communication is comparable to computation on all size machines. In fact communication is always small compared to computation, because the stencil operation performed per grid point in a nonlinear conservation law is typically nontrivial, i.e., has a large value for its $O(1)$ cost compared to the corresponding $O(1)$ cost for the communication. We note that since the block transpose communicates matrix elements in long sequences (see § 6.7), this algorithm behaves well even on machines with a high overhead for communicating short messages. This is borne out in the iPSC experiments reported below.

8.4. Timing and results for hyperbolic equations. In Fig. 17, the time used on the 32 node Caltech Hypercube to perform a random choice method iteration step is given as a function of the size N of the problem. The method used was the operator splitting algorithm with transpose. The lower curve in this figure is the time needed to transpose the solution matrix. The transpose algorithm in these timings used the nonoptimized vector shift operation. Even in this case, the communication cost (transpose) is already dominated by the computation cost.

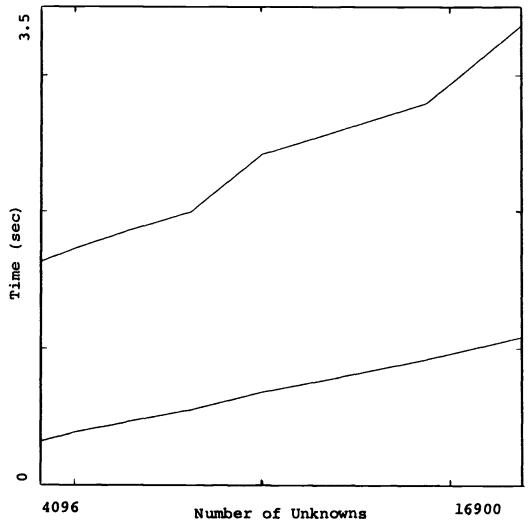


FIG. 17. *Iteration time for the random choice method on the 32 node Caltech Hypercube versus size of the problem. The lower curve is the time necessary to transpose the matrix (shift transpose with nonoptimized vector shift).*

Analogously, Figs. 18 and 19 give the iteration time and the computational efficiency of the random choice method as a function of the size of the problem on the 128 node iPSC. In these iPSC calculations the recursive block transpose algorithm was used instead of the shift transpose used in the Caltech calculations. In Fig. 18 we note that as the grid becomes large both the computation and transpose times are converging to straight lines as expected from the previous discussion—times proportional to the number of grid points N. However the transpose time has a smaller slope, ensuring that it will always be dominated by computation time. For very small grid sizes however one can clearly see from Fig. 18 that the transpose cost dominates the

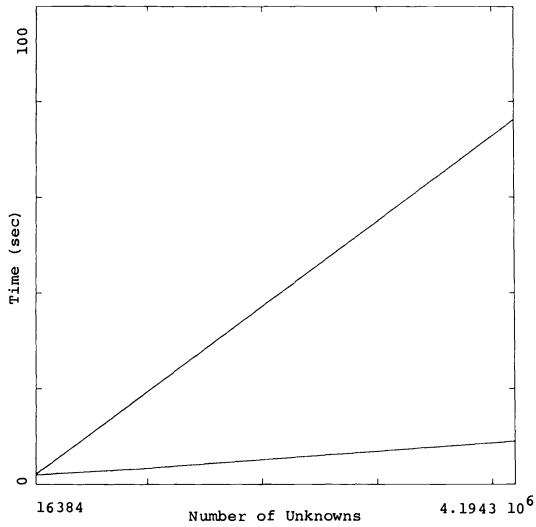


FIG. 18. *Iteration time for the random choice method on the* 128 *node* iPSC *versus size of the problem. The lower curve is the time spent in communication (transpose time with block transpose algorithm and computation of global maximum speeds for the Courant condition).*

computation due to the high communication startup cost on the iPSC. This shows up more clearly in Fig. 19 where computational efficiencies are poor for very coarse grids. For the Lax-Wendroff method using either of the transpose algorithms, we obtained equivalent results.

We have also used the areal decomposition technique for each of the methods, with rectangular subgrids ranging in shape from near square to thin strips. As an example, we display in Fig. 20 the scaled time step cost for the areal-distributed Lax-Wendroff method on a 32 node iPSC as a function of the number of unknowns in the problem. Here the time per time step has been divided by the number of grid

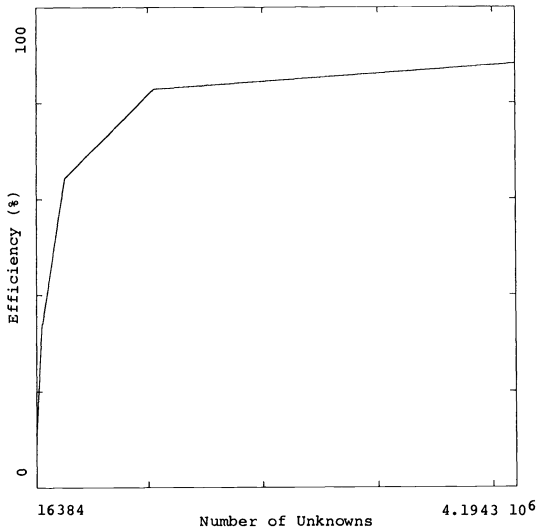


FIG. 19. *Computational efficiency of the random choice method using matrix transpose versus size of problem on the* 128 *node* iPSC.

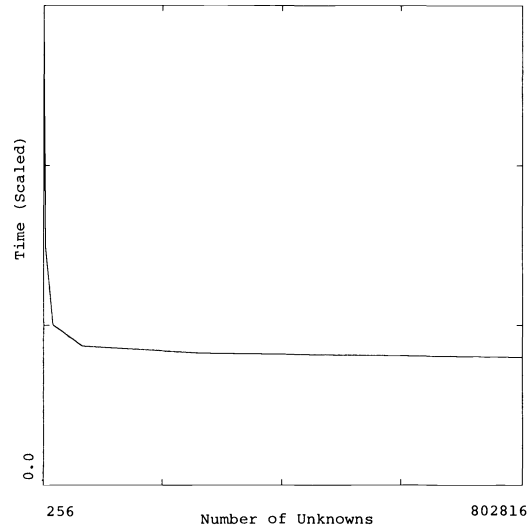


FIG. 20. *Effective processing time per grid point as a function of the problem size (number of unknowns) for the areal decomposition Lax-Wendroff method on the* 32 *node* iPSC.

points to arrive at a measure of effective processing time per grid point and to allow comparison of different grid sizes. The areal decomposition in this case corresponded to mapping the 32 processors onto an 8×4 rectangular grid. The scaled time decreases rapidly to a limiting value as the grid size increases, corresponding to increasing computational efficiency. The scaled time is proportional in fact to the inverse of computational efficiency.

Table 7 indicates the effect of the area-perimeter law as a function of subgrid rectangle shape on the iPSC for a fixed-size solution. The 128 processors are mapped to logical rectangular $p_x \times p_y$ grids of dimensions 16×8, 32×4, 64×2 and 128×1, respectively, and the 4,096,000 unknowns are then distributed to the processors as indicated previously. As discussed above it is extremely difficult to measure the area-perimeter effect on the iPSC because startup time dominates communication. This is borne out clearly in the table. The near-square 16×8 grid is slightly faster than the following two more rectangular mappings, but the linear distribution is fastest. The explanation is that the linear distribution is a limiting case and involves special effects. For example, no communication is required across two of the sides in a linear distribution. Thus only the first three entries in the table are really relevant and the observed area-perimeter effect is small. For comparison with the Caltech Hypercube, we refer to the corresponding Fig. 24 where aspect ratio effects are quite pronounced.

TABLE 7
iPSC *area-perimeter effect.*

p_x	p_y	Time (secs/iteration)
16	8	94.576
32	4	95.248
64	2	95.376
128	1	91.968

9. Comparison of algorithms and architectures. In this section we discuss the amount of computational work per grid point that is necessary to maintain efficient use of the hypercube in typical numerical procedures. For this purpose we define the *computational efficiency* E of a program running on a parallel computer as the ratio of computation time to total execution time:

$$E = \frac{T_{computation}}{T_{execution}}.$$

This ratio has the desirable property that efficiency increases if communication time can be overlapped with computation. It does not however measure efficiency relative to the best serial algorithm for the same problem. Note that the classical definition for efficiency is the speedup obtained divided by the number of processors in a system. Assuming that the computation time using one processor is approximately equal to P times the parallel computation time, then the above computational efficiency and the classical efficiency are approximately equal. The above assumption is valid for most algorithms in this paper.

The computational efficiency of a program with localized communication will generally depend strongly on the grid size. As the number of grid points per processor increases, so does efficiency. The efficiency will also depend on the number of floating point operations performed per grid point; if more computation is performed, efficiency will be increased. In § 9.1 we describe measurements we have performed of efficiency as a function of work per grid point for the hyperbolic solvers described in § 8 with both areal and transpose algorithms. Section 9.2 discusses the general applicability and utility of these measurements for the comparative evaluation of architectures and algorithms.

9.1. Efficiency and work per grid point. To measure these effects, we have used the basic transpose-based hyperbolic solver described previously. We replaced the low-level two or three point hyperbolic difference step by a call to a subroutine that executes some number (*NFLOPS*) of floating point multiplications. By running the resulting program with different grid sizes and values of *NFLOPS* we obtain performance curves for the iPSC that measure efficiency with respect to both grid size and work per point. We summarize these results with the curves in Fig. 21, obtained on a 128 processor iPSC.

We conclude that computational efficiency is excellent in almost all cases ($E \geqq 50\%$). Even on very coarse grids, such as 128×128 grids where there are only 128 grid points per processor, reasonable efficiencies may be achieved by executing less than a hundred operations per grid point. (In fact we have had excellent efficiencies with even 32 points per node.)

The hyperbolic equations we have solved in this paper have been model problems, where the cost of computation at a grid point is just a few multiplications. From the graphs in Fig. 21 it is clear that the solution of realistic physical equations using real equations of state or other material properties will be extremely efficient, even on very coarse grids. We stress again that there are three other factors that are likely to further increase computational efficiency:

(a) the current iPSC design has a very high communication overhead;
(b) future machines will have much larger memories, and so many more grid points per processor;
(c) the transpose algorithm is far from optimal with respect to communication.

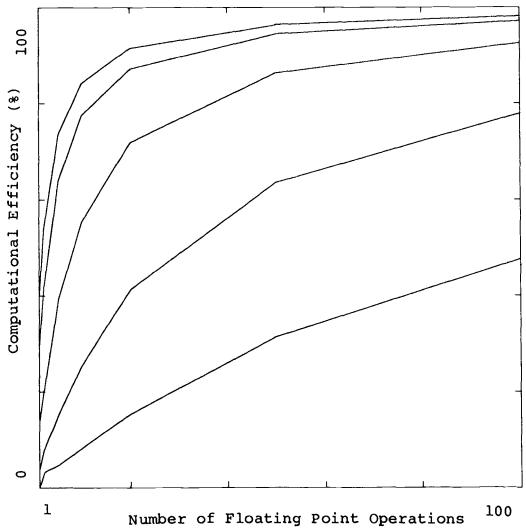


FIG. 21. *Computational efficiency on the* 128 *node* iPSC *for an areal sweep with block matrix transpose versus number of floating point operations executed per grid point. The curves from top to bottom are for grid sizes*: 2048*2048, 1024*1024, 512*512, 256*256, 128*128.

As an illustration of point (c) we repeated the above experiment using the areal decomposition approach discussed in § 8.1. The resulting efficiency curves are presented in Fig. 22 and are seen to be substantially better than in the transpose case.

We conclude therefore that the vast majority of regular-grid hyperbolic methods will parallelize with an efficiency close to 1, provided only that the memory available on each processor is large enough to store the code, associated data and results needed by that processor for a reasonably large number of grid points. It is important to note that for methods such as random choice where the computational work varies from

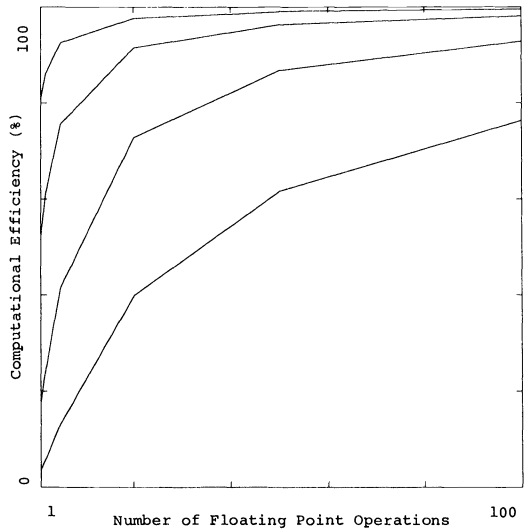


FIG. 22. *Computational efficiency on the* 128 *node* iPSC *for an areal sweep with areal decomposition technique versus number of floating point operations executed per grid point. The curves from top to bottom are for grid sizes*: 1024*1024, 512*512, 256*256, 128*128.

point to point in a grid, the efficiency as described above will be misleading in that some processors may in fact be idle while others are completing their locally expensive computations. Thus unless load-balancing or time-sharing of processors is employed, efficiencies for such methods will be lower than predicted unless the value of *NFLOPS* used in the predictions corresponds to the worst case possible at any grid point.

9.2. Architecture comparisons. There are many criteria that may be used to compare various parallel algorithms and architectures. The efficiency curves introduced in the previous section are one such mechanism which allow computational efficiency of each of a number of relevant algorithms to be compared across a range of machines. We intend to repeat the measurements described in this section for other parallel computers to provide a basis for comparison between machines. We have already measured the curves for two different algorithms, as seen in Figs. 21 and 22, and those graphs allow immediate comparison of the algorithms across a wide range of problem characteristics (*NFLOPS* and grid sizes). In fact the graphs are applicable to just about any situation where an areal sweep is performed over a rectangular grid and where only nearest-neighbor connectivity is involved between points. As an example, Fig. 22 is directly applicable to the areal multigrid relaxation discussed in § 11—one simply computes the number of floating point operations per grid point in the relaxation, and can then read off the expected efficiency on different grid sizes from the graph.

10. Elliptic equations and solution methods. The basic elliptic equation we have studied is of the form:

$$\nabla \cdot (-\vec{K} \cdot \nabla P)(x, y) = F(x, y).$$

Here K has discontinuities, possibly of order a thousand or more, across a given set of curves and in typical applications may represent a fluid density, permeability or dielectric constant. The quantity of most interest is not the pressure or potential P but the velocity or flux $-\vec{K} \cdot \nabla P$. The right-hand side F may contain arbitrary point and line sources. Boundary conditions may be Dirichlet, Neumann or mixed. We also allow a parabolic term on the left, provided an implicit time discretization is used. In the latter case, the following discussion will apply to the individual time steps.

Discontinuities of coefficients imply discontinuities in the solution gradient. Discretization of the equation on a rectangular grid leads to bad pressure and velocity solutions due to such discontinuities. For this reason it is essential to adapt the grids locally. In the resulting grids, discontinuities lie only along edges of triangles. The cost of grid generation is negligible compared to equation solution. We allow the use of curved edged triangles (isoparametrics) to provide high order boundary fitting. In In general, our grids consist of unions of rectangles and triangles, with triangles primarily used to fit boundaries and interior interfaces. For details of the grid construction methods used, we refer to our papers [43], [44], [45].

We have used finite element methods to discretize these equations, though similar studies could be applied to finite difference methods. To provide sufficient accuracy, we allow high order elements up to cubics (on triangles) or bicubics (on rectangles). We have discussed the solution of singular elliptic equations by these techniques in [43], [45] and the use of parallelism in the context of a tree of refinement grids elsewhere [6]. We have also previously described the solution of such systems on a parallel machine with global shared memory [4], [6]. We discuss here the implementation of fast solution methods on hypercube parallel processors. We describe the implementation of the *conjugate gradient method* and an *FFT-based fast Poisson solver* which we

have used as a preconditioner for the conjugate gradient method. We have also implemented a *full multigrid method* based on a 5-point operator discretization of the equations. Using either of these methods, the solution cost in total operations performed is essentially proportional to the number of unknowns, while at the same time allowing near optimal use of parallelism.

Execution time is strongly dominated by the solution phase and it is the parallelization effort for this which we describe in the following sections. We point out however that the finite element construction is also easily parallelized: the construction of the element matrices can proceed essentially independently on each element, fully utilizing available parallelism. Vectorizing such code is difficult, because the operations to be performed at individual finite element nodes are complicated, and because the elements are neither regular nor homogeneous (rectangles and triangles intermixed). Thus we will focus on parallel solution of the resulting equations and refer to our papers for details of the numerical analysis and of the discretization approach [43]–[47].

11. Parallel multigrid. We have developed a parallel multigrid algorithm and tested it on both the Caltech and Intel Hypercubes. Multigrid methods are near-optimal for a wide range of computations on serial or vector processors. Our test problem was a Poisson equation with Dirichlet boundary conditions on a square, discretized with a 5-point finite difference operator, but the implementation is considerably more general. A multigrid algorithm for the Poisson equation in 3 dimensions on the Caltech Hypercube has been developed independently by Clemens Thole [48].

The basic multigrid idea [49]–[51], [47], [20], [22] involves two aspects: the use of relaxation methods to dampen high-frequency errors and the use of multiple grids to allow low-frequencies to be relaxed inexpensively. A simple *Two-grid Iteration* involves a number of relaxations on a fine grid to reduce high-frequency errors in an initial guess of the solution, a projection of remaining errors to a coarser grid where they are solved for exactly, and then an interpolation back to the fine grid and addition to the solution there. This solution would now be exact but for errors introduced by the projection and interpolation processes. The solution is then improved by repeating the procedure.

The Two-grid Iteration is converted to the *Multigrid Iteration* (MGI) by recursively applying the two-grid iteration in place of the exact solution on the coarse grid. The number of times that the coarse grid iteration is repeated before returning to the fine grid is important for convergence rates—typical values used are once, known as *V-cycles*, or twice, known as *W-cycles*.

Improved convergence can be obtained by choice of a good initial guess for the solution. A simple strategy would be to solve the equations on a coarse grid using the Multigrid Iteration, and interpolate the solution to the fine grid as the initial guess for the Multigrid Iteration there. Recursively applying this idea leads to the *Full Multigrid Iteration* (FMG) which performs a sequence of Full Multigrid solutions on increasingly finer grids, using the solution of each as an initial guess for the next.

Our studies are concerned with the case where there are many fine grid points per processor and we will assume this to be the case throughout the exposition. The algorithms presented in this section are valid for any architecture in which the processors can be arranged as a 2-dimensional periodic or nonperiodic array, depending on the nature of the boundary conditions of the elliptic problem. We have developed both a parallel Red-Black Gauss–Seidel relaxation operator and a parallel alternating line relaxation algorithm. The latter relaxation is important for problems which have variable coefficients, and is implemented using the parallel transpose operation.

Computational efficiency of the multigrid relaxation decreases with coarser grids because the perimeter to area ratio becomes less favorable as the number of grid points is reduced. Consequently we find that V-cycles are somewhat more favored than W-cycles on a hypercube compared to a serial computer. Figure 23 shows a comparison of V- and W-cycle speedup curves for the same sized grid which clearly demonstrates the superior computational efficiency of the V-cycle with respect to communication overhead. Similarly it is advantageous to terminate grid subdivision at a somewhat finer grid than one might use on a serial machine. Note that in Fig. 23 the curves begin only at speedups of 2 or 4. We were unable to solve the problem with that grid size on only one processor due to memory limitations. If the smallest number of processors that could be utilized was p then we arbitrarily assigned speedup p to the p-processor time.

11.1. Distributed grids for multigrid. Having discretized the PDE using finite differences, the resulting equations on any grid level involve the solution or error on a rectangular grid of points. Parallelizing multigrid amounts to distributing these grids over the processors and implementing the communications needed as a result between processors. We accomplish the distribution by representing the solution (or error) on each grid level as a distributed matrix of type *AREAL*, see §§ 6.1 and 6.3. In particular the hypercube is organized as a fixed rectangular mesh of $P = p_x p_y$ processors for all grid levels, using the matrix library call:

$$areal_parameters(p_x, p_y, 0, 0, 1),$$

which provides a boundary of width 1 surrounding all sides of each subgrid allocated in subsequent *allocate_matrix* calls for type *AREAL*, as well as nonperiodic boundary conditions. We will use the term *extended subgrid* to denote such a subgrid with boundary, and will refer to the nonboundary points of the subgrid as its *interior*. We assume that the grids to be used are of dimensions $2^{l_x}p_x \times 2^{l_y}p_y$ for some l_x, l_y. With this choice, the interior of a level l subgrid assigned to a processor may be identified with a subset of the extended level $l+1$ subgrid assigned to that processor. Similarly

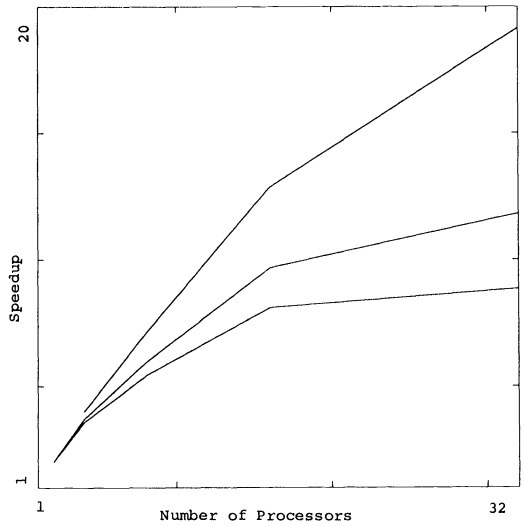


FIG. 23. *The top curve is speedup versus number of processors for V cycle multigrid on a* 128 *by* 128 *grid with* 4 *levels on the Caltech Hypercube. The two lower curves are for V and W cycle multigrid on a* 64 *by* 64 *grid with* 3 *levels.*

the interior of the level $l+1$ subgrid in a processor is contained in the extended subgrid of level l.

11.2. Inter-process communication. Along the boundary of the subgrids at any level some communication will be required by the multigrid relaxation operation. For example, a relaxation at boundary points of a subregion will require the boundary data of the adjoining subregion. The *shuffle* operation from the matrix library, also described in § 6.3, provides exactly the required interchange of boundary data. By calling *shuffle* one can assure that each boundary edge at the start of any operation (relaxation or intergrid transfer) holds a copy of the closest interior edge of the adjacent subgrid. For a 5-point operator, a boundary width of 1 suffices. The only possible problem here is the updating of corner nodes which a priori have to be copied from a nonneighboring processor. These are not actually needed in a five-point problem, but in more general situations are accommodated by adding an extra corner node shuffle. The *shuffle* operation is thus seen to be the basic communication routine necessary for multigrid to work on distributed grids.

Intergrid transfers between multigrid levels (injection and interpolation) are entirely local to each processor because of the containment relationships between the grids presented in the previous section. Thus no communication is involved in these steps. However before any updated grid values are used it is essential to perform a *shuffle* operation to preserve the integrity of the duplicate boundary data. Thus we use the *shuffle* operation not only before each relaxation but also before every intergrid transfer.

It is in fact possible to choose a slightly different areal distribution of a matrix which avoids most of the shuffle operations at the cost of a small amount of memory. In this representation the interior edges of subgrids are actually duplicated in neighboring processors, as well as having a further boundary edge as above. With this choice, the interior of the level $l+1$ subgrid contains all points of the interior of the level l subgrid, obviating any need to perform shuffles after intergrid transfers. In this case a *shuffle* is required only before every relaxation, resulting in a substantial decrease in communication.

To control termination of iteration in adaptive multigrid algorithms, norms of error values need to be calculated. These are evaluated by computing a local inner product in each processor, storing it in the local component of a distributed vector of length P, and then calling *sum_vector* on this distributed vector, which computes the squared norm and distributes it to the processors (see the discussion of the semantics of *inner_product* in § 5.5). If the alternate areal representation of the previous paragraph is used, it is important to assign a weight of 0.5 to interior edge points and a weight of 0.25 to interior corner points in computing local norms since these points are duplicated as interior points in neighboring subgrids.

11.3. The relaxation operation. The relaxation operation is straightforward to implement in each of the subregions. Standard Gauss-Seidel relaxation is discarded on numerical grounds of slower convergence. It introduces artificial discontinuities along the boundaries of the subregions as the parallel version uses the original values of the boundary duplicates. Note that in the limit of subregions consisting of only one grid point, this would reduce to a Jacobi relaxation. Instead we utilize a Red-Black ordering of the grid points. This can be implemented such that the parallel algorithm will, after each iteration step, give the same result as the corresponding serial algorithm. The procedure consists of shuffling, relaxing the red points, shuffling again and then relaxing the black points.

11.4. Intergrid transfers. The error (the residual equation to be precise) is projected to the coarse grid using half-injection—the errors at the black grid points are identically zero after the previous relaxation step. For this step as discussed previously, it is necessary that the extended fine grid subregions contain the nonextended coarse grid regions.

Having solved the error equation on the coarse grid the solution on the fine grid has to be updated by addition of a suitable interpolation of the computed coarse grid error. We have used linear interpolation at this point. To do this in parallel without communication, the extended coarse grid subregion must contain the nonextended fine grid subregion. We update the boundary duplicates in the fine grid subregions following the interpolation using a *shuffle* operation.

11.5. Results and timings for multigrid. In Fig. 23 we compare timings for 3-level V-cycle and W-cycle multigrid iterations with finest grid of dimension 64×64, and also a 4-level V-cycle iteration on a 128×128 grid. These computations were performed on the Caltech Hypercube with from 2 up to 32 nodes. The curves represent speedup attained with increasing number of processors (the 128 grid problem would not fit on 2 processors). We notice that the speedup for the W-cycle is worse than that for the V-cycle. This is due to the fact that in the W-cycle algorithm, one spends a larger segment of the computation time on the coarser grids. In spite of this, it can be expected that for some equations it may be profitable to use the W-cycle instead of V-cycle because of its faster convergence rate. This issue however is too dependent on the particular type of equations solved to be discussed in this context. The V-cycle curve for the finer grid shows substantially improved performance compared to the coarser grid. All of these curves correspond to relatively very coarse grids since the number of grid points per processor is at most 512 on the finest grid.

In Fig. 24 we illustrate the effect of Area/Perimeter considerations on the Caltech Hypercube with 32 processors. The hypercube was organized as logical rectangular grids of dimensions 32×1, 16×2 and 8×4 (see Fig. 12) and we display the resulting

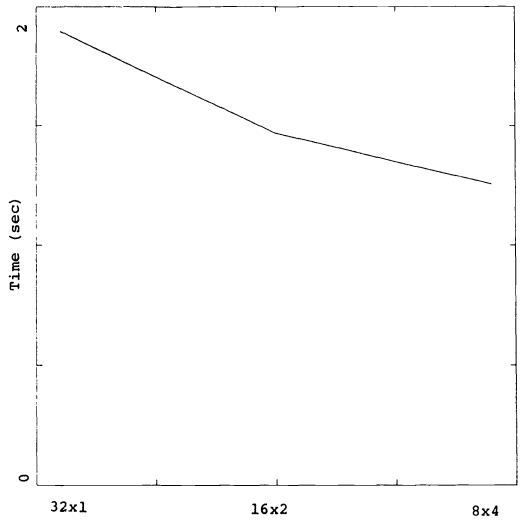


FIG. 24. *Area-perimeter considerations for multigrid on the* 32 *node Caltech Hypercube. We compare times for a V cycle multigrid iteration on a* 128 *by* 128 *grid with* 2 *levels, when the domain is split in* 32 *by* 1, 16 *by* 2 *and* 8 *by* 4 *rectangles, respectively.*

computation time for a multigrid iteration on a 128×128 grid with 2 grid levels. The improvement in timing as the subregions approach squares is obvious. The success of this demonstration is related to the fact that even for short messages communication costs on the Caltech Hypercube are strictly proportional to message length.

To illustrate that communication overhead decreases with an areal distribution as problem size increases, we compare in Fig. 25 the scaled time of a multigrid iteration as a function of grid size. The scaled time is obtained by dividing the iteration time by a quantity proportional to the number of grid points and is therefore proportional to effective work per grid point for multigrid *including* communication overhead. Specifically we compare the times for 16 iterations on a 64 by 64 grid, 4 iterations on a 128 by 128 grid and 1 iteration on a 256 by 256 grid on the 32 node Caltech Hypercube. Scaled time is over twice as small for the finest grid as for the coarsest.

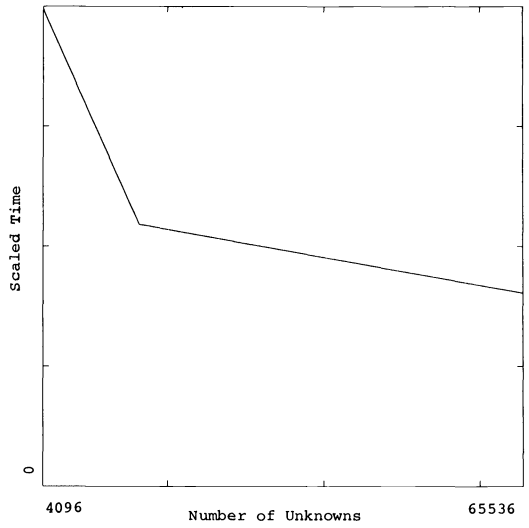


FIG. 25. *Multigrid, 3 levels, V cycle on the Caltech Hypercube. We compare execution times of different sized problems by scaling the times by the number of floating point operations executed.*

Our multigrid results for the Intel iPSC, are cast in a somewhat different form. We solved problems with varying grid sizes on a 128 node iPSC. To compare the iteration times, we calculated the number of *useful* floating point operations (additions or multiplications, but not memory accesses) that were executed in each of the runs. We divided these by the time it took for an iteration to complete. This defines an attained *floprate* for each run. In Fig. 26 we plot the attained floprate for V-cycles as a function of grid size, going from 4096 unknowns to over 1.6 million unknowns. We see that for the largest problems we attained about 3.75 megaflops.

12. Parallel conjugate gradient. Discretization of elliptic partial differential equations in two dimensions by finite element or finite difference methods leads to systems of equations with sparse coefficient matrices. The fill-in of the matrix tends to follow diagonals and the bandwidth is about $dN^{1/2}$, where N is the dimension of the matrix and d is the degree of the finite elements used for the discretization. Furthermore typically only $O(1)$ diagonals have nonzero elements. The parallel conjugate gradient method we have developed on the hypercube solves systems of equations with such

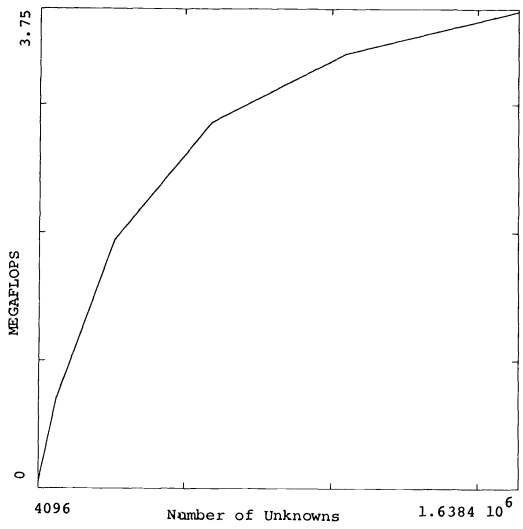


FIG. 26. *Performance of* 2 *level V cycle multigrid method on the* 128 *node* iPSC. *Number of floating point operations executed per second as function of problem size.*

coefficient matrix structures. This allows us to parallelize the solution of finite element discretizations of arbitrary and even variable degree with high efficiency.

12.1. The serial algorithm. The preconditioned conjugate gradient method [52]-[55] finds the solution of the system of equations $Ax = f$, to a specified accuracy ε by performing the following iteration on the vector x, which has been appropriately initialized:

$r = f - Ax$
$p = Br$
loop
  $s = \langle r, Br\rangle / \langle p, Ap\rangle$
  $r = r - s \cdot Ap$
  $x = x + s \cdot p$
  $rbr = \langle r, Br\rangle$
  $s = \langle r, Br\rangle / old\langle r, Br\rangle$
  $p = Br + s \cdot p$
until *converged*

Here B is an approximate inverse of A, which is assumed to be positive definite symmetric, and $\langle x, y\rangle$ denotes the inner product of vectors x and y. The *preconditioning* operator B can be effective in improving substantially the convergence rate of the algorithm. In § 13, we will describe a parallel preconditioner for the conjugate gradient algorithm, concentrating in this section on the unaccelerated conjugate gradient method where B is just the identity operator.

12.2. Parallelism in conjugate gradient. We parallelize the algorithm by exploiting parallelism in every operation of the iteration. We have discussed previously how to parallelize each operation in the algorithm above (with the identity preconditioning). All of the vectors in the algorithm are allocated using the *allocate_vector* library routine

(see § 5) with type *SHIFT*. The matrix A is allocated using the *allocate_matrix* library routine (see § 6) with type *DIAGONAL* and with the sparsity structure of the nonzero extended diagonals explicitly specified. Each line in the algorithm above is expressed with the corresponding vector or matrix library routine. In particular the communication-intensive operation $p \to Ap$ is implemented directly using the *matrix_vector* library routine, so that all communication is performed by the optimized *shift_vector* routine. It then suffices to compile the above program with the vector and matrix libraries introduced earlier. Note that the resulting program shows no dependence on a control processor. Each processor obtains the required inner products for convergence control using only a local subroutine call. All processors make the decision to stop iterating, and they will all do so simultaneously since they all receive the same value for the inner products.

12.3. Timing and results for conjugate gradient. In Fig. 27 we display the execution time per iteration step as a function of the number of processors used for the conjugate gradient method on a 64×64 grid on the Caltech Hypercube with from 2 up to 32 nodes. In Fig. 28 we compare the floprates attained in the execution of the conjugate gradient iteration on a 5D Caltech cube for different size grids. The floprates are determined by dividing the actual number of floating point operations executed by the time to perform the iteration. We counted the number of multiplications in the algorithm to be about $(\textit{number of diagonals}+5)N$ per iteration and about an equal number of floating point additions take place. These computations were performed with the nonoptimized vector shift algorithm, and are for relatively coarse grids.

In Fig. 29 we display the analogous graph for the 128 node iPSC for problems with 4096 up to 409600 unknowns. These computations were performed with the optimized shift algorithm and so are not directly comparable to the Caltech Hypercube results. Counting all useful floating point operations (multiplications and additions), the 128 node iPSC attained about 2.5 megaflops in conjugate gradient iteration. The

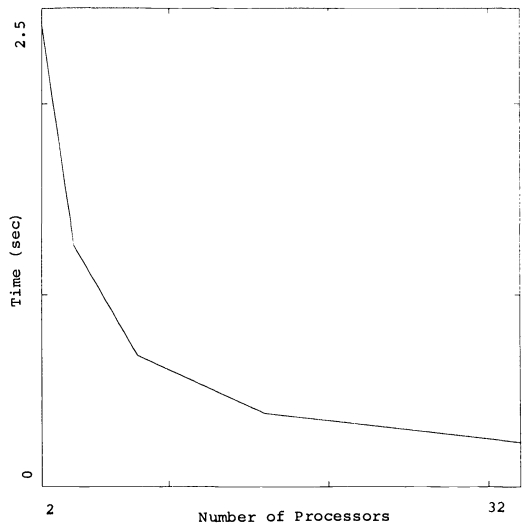


FIG. 27. *Conjugate gradient on a 64 by 64 grid. Time per iteration step versus number of processors used on the Caltech Hypercube.*

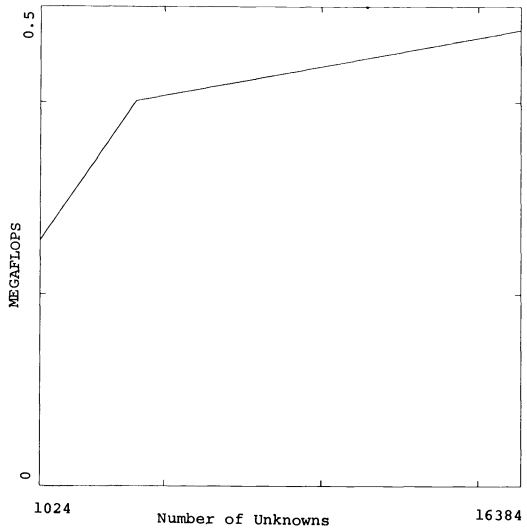


FIG. 28. *Performance of the conjugate gradient method on the* 32 *node Caltech Hypercube. We display the number of floating point operations executed per second for problems with* 1024 *up to* 16384 *unknowns.*

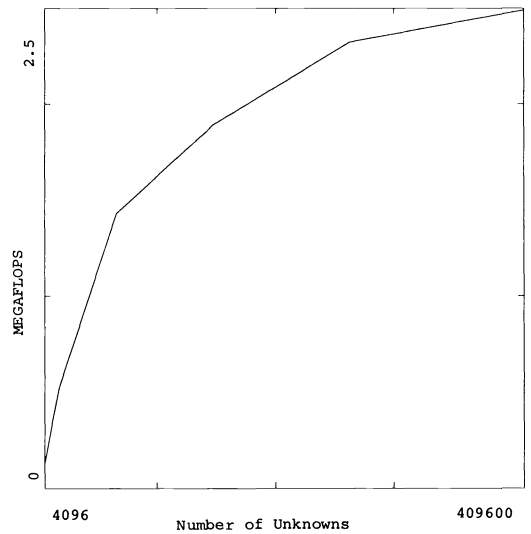


FIG. 29. *Performance of the conjugate gradient method on the* 128 *node* iPSC. *We display the number of floating point operations executed per second for problems with* 4096 *up to* 409,600 *unknowns.*

matrix used for the purpose of timings had 5 diagonals corresponding to linear finite elements.

13. A parallel fast Poisson solver. We have developed a parallel preconditioner for use in conjunction with the parallel conjugate gradient algorithm described in § 12. The algorithm is based on the Fast Fourier Transform and reduces the solution of the Poisson equation:

$$-\Delta\Phi(x, y) = F(x, y),$$

on a rectangle to Fourier transforms and tridiagonal solvers [56]. This is a standard method frequently used on serial computers and the interest here is entirely in how to parallelize it. An attraction of our parallelization method is that neither the FFT

nor the tridiagonal solvers need to be parallelized. Assuming a 5-point discretization of the Poisson equation in a rectangle on an $m \times n$ grid, the solution may be obtained on a serial machine in three steps using the algorithm:

$$\begin{array}{ll} \textit{for col in } 1, \cdots, m & \textit{fft}(n, F(\cdot, \textit{col})) \\ \textit{for row in } 1, \cdots, n & \textit{tridiag}(m, t(\textit{row}, \cdot), F(\textit{row}, \cdot)) \\ \textit{for col in } 1, \cdots, m & \textit{fft}(n, F(\cdot, \textit{col})) \end{array}$$

This algorithm returns the exact solution of the discrete Poisson equation in $O(nm \log n)$ operations on a serial computer. Here the matrix $F = F(\cdot, \cdot)$ is a discretization of the source function $F(x, y)$, while $F(row, \cdot)$ and $F(\cdot, col)$ denote the vectors corresponding to the indicated row or column of the matrix F. The matrix $t(\cdot, \cdot)$ contains the coefficients of the tridiagonal systems, obtained analytically by Fourier transforming the 5-point matrix. The routine *fft*(n, v) computes a real (sine or cosine) Fast Fourier Transform of a vector v of length n. Similarly the routine *tridiag*(m, t, v) solves a tridiagonal system $Tu = v$, overwriting the m-component vector v with the solution u.

We have found that such a solver is a useful preconditioner even for higher-order finite elements on logically rectangular grids—one can use the Poisson solver for a rectangular grid corresponding in dimensionality to the nodal points of the finite elements, see McBryan [43], [47].

Vectorization of this algorithm involves considerable effort because it requires vectorizing the FFT routine which is nontrivial especially if the grid dimension n is more general than a power of 2. However parallelization is straightforward. The matrix F is stored as a distributed matrix of type *CONTIGUOUS_COLUMN*, while the matrix t is precomputed and stored as a distributed matrix of type *CONTIGUOUS_ROW*, see § 6.1. Each *fft* and *tridiagonal* operation is assigned to an individual processor with each processor executing the normal serial code for that operation on one or more columns or rows, respectively. This allows the full available parallelism to be applied effectively.

As we have noted in our previous Denelcor HEP studies, to complete the parallelization of this algorithm on a shared memory parallel computer it suffices to place a *barrier* between the first and second step, and again between the second and third step since it is necessary that all processors complete work on each step before any processor proceeds to the following step. There is a complication however on a hypercube stemming from the fact that the memory on the hypercube is not shared among the processors. In the *fft* steps it is necessary to distribute the matrix F in contiguous-column form, so that the *fft* operations are local to individual processors. Similarly in the tridiagonal step it is necessary to distribute the matrix F in contiguous-row form so that each tridiagonal solution involves only values from one processor. To get from a row-distributed matrix to a column-distributed matrix requires a matrix transpose.

It follows that the above algorithm may be parallelized on a local-memory processor by distributing F in contiguous-column form and by inserting an extra *transpose*(F) step between the first and second step and between the second and third step. This along with the *barrier* described previously are the only modifications required to the serial algorithm and in fact all communication for the Poisson solution is hidden in the two transpose operations. On shared memory computers the *transpose* steps are not needed. However on machines with fast data caches there may be an advantage to using the transpose anyway, since it ensures that the *fft* and *tridiagonal* solutions work entirely on contiguous data. Another approach to parallelizing the fast Poisson solver, more cumbersome from a programming point of view, would be to adapt the Fourier transform algorithm to handle distributed vectors. This is nontrivial to perform unless the grid dimension is a power of 2.

Acknowledgments. We wish to thank Geoffrey Fox at Caltech for providing access to the Caltech Hypercube and Shirley Enguehard at Caltech for much helpful advice. We also express our appreciation to Los Alamos National Laboratory and to Intel Scientific Computers for access to their iPSC Hypercube systems. In particular we would like to thank Ralph Brickner (Los Alamos National Laboratory) for his patience through many late night and weekend system reboots and Cleve Moler (Intel) for providing documentation and other useful information.

REFERENCES

[1] O. McBRYAN AND E. VAN DE VELDE, *Parallel algorithms for elliptic equations*, Proc. 1984 ARO Novel Computing Environments Conference, Stanford University, Society for Industrial and Applied Mathematics, 1986.

[2] O. McBRYAN, E. VAN DE VELDE AND P. VIANNA, *Parallel algorithms for elliptic and parabolic equations*, Proc. Conference on Parallel Computations in Heat Transfer and Fluid Flows, Univ. Maryland, Baltimore, MD, November 1984.

[3] O. McBRYAN, *State of the art of multiprocessors in scientific computation*, Proc. European Weather Center Conference on Multiprocessors in Meteorological Models, EWCMF, Reading, England, December 1984.

[4] O. McBRYAN AND E. VAN DE VELDE, *Parallel algorithms for elliptic equation solution on the* HEP *computer*, Proc. Conference on Parallel Processing using the Heterogeneous Element Processor, Univ. Oklahoma, Tulsa, OK, March 1985.

[5] ———, *Elliptic equation algorithms on parallel computers*, Comm. Appl. Numer. Math., 2 (1986), pp. 311–318.

[6] ———, *Parallel algorithms for elliptic equations*, Comm. Pure Appl. Math., 38 (1985), pp. 769–795.

[7] ———, *The multigrid method on parallel computers*, Proc. 2nd European Multigrid Conference, Cologne, October 1985, GMD Studie Nr. 110, J. Linden, ed., GMD, July 1986.

[8] ———, *Hypercube algorithms for computational fluid dynamics*, Proc. Oak Ridge meeting on Hypercube Architectures, August 1985, Society for Industrial and Applied Mathematics, to appear.

[9] J. DONGARRA, J. BUNCH, C. MOLER AND G. STEWART, LINPACK *User's Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979.

[10] C. LAWSON, R. HANSON, D. KINCAID AND F. KROUGH, *Basic linear algebra subprograms for* FORTRAN *usage*, ACM TOMS, 5 (1979), pp. 308–323.

[11] J. J. DONGARRA, J. DU CROZ, SVEN HAMMARLING AND R. J. HANSON, *A proposal for an extended set of Fortran basic linear algebra subprograms*, ACM SIGNUM, January 1985.

[12] D. B. GANNON AND J. PANETTA, *The systolic level* 2 BLAS, in New Computing Environments: Parallel, Vector and Systolic, A. Wouk, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1986.

[13] J. J. DONGARRA AND R. E. HIROMOTO, *A collection of parallel linear equations routines for the Denelcor* HEP, Parallel Computing, 1 (1984).

[14] D. P. O'LEARY AND G. W. STEWART, *Data-flow algorithms for parallel matrix computations*, Univ. Maryland Tech. Report 1366, January 1984.

[15] J. J. DONGARRA, A. H. SAMEH AND D. C. SORENSEN, *Some implementations of the* QR*-factorization on an* MIMD*-machine*, in Parallel Computing, October 1985.

[16] M. T. HEATH AND D. C. SORENSEN, *A pipelined Givens method for computing the* QR*-factorization of a sparse matrix*, in Proc. Workshop on Parallel Processing using the HEP, S. Lakshmivarahan, ed., Univ. Oklahoma, Tulsa, OK, March 1985.

[17] D. C. SORENSEN, *Buffering for vector performance on a pipelined* MIMD *machine*, MCS-TM-29, Argonne National Laboratory, Argonne, IL, April 1984.

[18] C. GROSCH, *Performance analysis of Poisson solvers on array computers*, in Infotech State of the Art Report: Supercomputers, R. Hockney, ed., vol. 1, Infotech Int. Ltd., 1979, pp. 147–181.

[19] 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.

[20] A. BRANDT, *Multi-grid solvers on parallel computers*, ICASE Technical Report 80-23, NASA Langley Research Center, Hampton, VA, 1980.

[21] L. ADAMS, *Iterative algorithms for large sparse linear systems on parallel computers*, NASA Report CR-166027, NASA Langley Research Center, Hampton, VA, 1982.

[22] D. GANNON AND J. VAN ROSENDALE, *Highly parallel multi-grid solvers for elliptic* PDEs: *An experimental analysis*, ICASE Technical Report 82-36, NASA Langley Research Center, Hampton, VA, 1982.

[23] L. ADAMS AND H. F. JORDAN, *Is* SOR *Color-Blind?*, ICASE Report No. 84-14, NASA Langley Research Center, Hampton, VA, 1984.
[24] E. L. LUSK AND R. A. OVERBEEK, *Implementation of monitors with macros*, Technical Memorandum ANL-83-97, Argonne National Laboratory, Mathematics and Computer Science Div., Argonne, IL, December 1983.
[25] H. E. JORDAN, *Structuring parallel algorithms in an* MIMD, *shared memory environment*, Univ. Colorado CS Report, Denver, CO, September 1984.
[26] C. L. SEITZ, *The cosmic cube*, Comm. ACM, 28 (1985), pp. 22–33.
[27] Caltech/JPL *concurrent computation project annual report* 1983–1984, Caltech Report, December 1984.
[28] L. JOHNSSON, *Data permutations and basic linear algebra computations on ensemble architectures*, Yale Univ. Research Report DCS/RR-367, New Haven, CT, February 1985.
[29] T. F. CHAN AND R. SCHREIBER, *Parallel networks for multi-grid algorithms: architecture and complexity*, Yale Univ. Research Report, New Haven, CT, DCS 262, 1983.
[30] Y. SAAD AND M. H. SCHULTZ, *Data communication in hypercubes*, Yale Univ. Research Report, New Haven, CT, DCS/RR-428, October 1985.
[31] ———, *Direct parallel methods for solving banded linear systems*, Yale Univ. Technical Report, CS Dept., New Haven, DCS/RR-387, CT, 1985.
[32] S. L. JOHNSSON, *Communication efficient basic linear algebra computations on hypercube architectures*, Yale Univ. Research Report, New Haven, CT, DCS/RR-361, September 1985.
[33] J. M. ORTEGA AND R. G. VOIGT, *Solution of partial differential equations on vector and parallel computers*, ICASE Report 85-1, January 1985.
[34] G. C. FOX AND S. W. OTTO, *Algorithms for concurrent processors*, Phys. Today, 37 (1984), pp. 50–59.
[35] J. SALMON, *Binary gray codes and the mapping of a physical lattice into a hypercube*, Caltech Concurrent Processor Report (CCP)Hm-51, 1983.
[36] E. N. GILBERT, *Gray codes and paths on the n-cube*, Bell System Tech. J., 37 (1958), p. 915.
[37] M. GARDNER, *Mathematical games*, Scientific American, August 1972, p. 106.
[38] T. CHAN, Copper Mountain, Colorado, April 1985. Remark at 2nd Multigrid Conference.
[39] J. O. ECKLUNDH, *A fast computer method for matrix transposing*, IEEE Trans. Comput., C-21 (1972), pp. 801–803.
[40] A. CHORIN, *Random choice solution of hyperbolic systems*, J. Comput. Phys., 22 (1976), pp. 517–533.
[41] R. RICHTMYER AND K. MORTON, *Difference Methods for Initial Value Problems*, Wiley-Interscience, New York, 1952.
[42] J. GLIMM, *Solutions in the large for nonlinear hyperbolic systems of equations*, Comm. Pure Appl. Math., 18 (1965), pp. 697–715.
[43] O. MCBRYAN, *Computational methods for discontinuities in fluids*, Lectures in Applied Mathematics, vol. 22, American Mathematical Society, Providence, 1985.
[44] ———, *Elliptic and hyperbolic interface refinement in two phase flow*, in Boundary and Interior Layers, J. J. H. Miller, ed., Boole Press, Dublin, 1980.
[45] ———, *Shock tracking for* 2*d flows*, in Computational and Asymptotic Methods for Boundary Layer Problems, J. J. Miller, ed., Boole Press, Dublin, 1982.
[46] J. GLIMM AND O. MCBRYAN, *A computational model for interfaces*, Adv. Appl. Math., 6 (1985), pp. 422–435.
[47] O. MCBRYAN, *Fluids, discontinuities and renormalization group methods*, in Mathematical Physics VII, Brittin, Gustafson and Wyss, eds., North-Holland, Amsterdam, 1984, pp. 481–494.
[48] C. THOLE, *Experiments with Multigrid on the Caltech Hypercube*, GMD Internal Report, October 1985.
[49] A. BRANDT, *Multi-level adaptive solutions to boundary-value problems*, Math. Comp., 31 (1977), pp. 333–390.
[50] W. HACKBUSCH, *Convergence of multi-grid iterations applied to difference equations*, Math. Comp., 34 (1980), pp. 425–440.
[51] K. STUDEN AND U. TROTTENBERG, *On the construction of fast solvers for elliptic equations*, Computational Fluid Dynamics, Rhode-Saint-Genese, 1982.
[52] M. R. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 409–436.
[53] J. K. REID, *On the method of conjugate gradients for the solution of large sparse systems of linear equations*, in Large Sparse Sets of Linear Equations, J. K. Reid, ed., Academic Press, New York, 1971, pp. 231–254.
[54] P. CONCUS, G. H. GOLUB AND D. P. O'LEARY, *A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations*, in Sparse Matrix Computations, D. J. Rose, ed., Academic Press, New York, 1976.
[55] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, John Hopkins Press, Baltimore, 1984.
[56] P. SWARZTRAUBER, *The methods of cyclic reduction, Fourier analysis and the* FACR *algorithm for the discrete solution of Poisson's equation on a rectangle*, SIAM Rev., 19 (1977), pp. 490–501.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 2, March 1987
© 1987 Society for Industrial and Applied Mathematics
007

THE USE OF LINEAR PROGRAMMING FOR THE SOLUTION OF SPARSE SETS OF NONLINEAR EQUATIONS*

I. S. DUFF†, J. NOCEDAL‡ AND J. K. REID†

Abstract. In this paper we propose a trust region algorithm for solving sparse sets of nonlinear equations. It is based on minimizing the l_1-norm of the linearized residual vector within an l_∞-norm trust region, thereby permitting linear programming techniques to be easily applied. The new algorithm has sparsity advantages over the Levenberg-Marquardt algorithm.

Key words. linear programming, sparse equations, nonlinear equations, trust regions, l_1-norm solution

AMS(MOS) subject classifications. 65H10, 65K99, 65F05

1. Introduction. We consider the solution of a system of n nonlinear algebraic equations

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \tag{1.1}$$

in the case where a large number of the derivatives $\partial f_i/\partial x_j$ are known to be always zero so that the Jacobian matrix

$$J = \left(\frac{\partial f_i}{\partial x_j}\right) \tag{1.2}$$

is sparse. Such systems arise, for example, in the solution of the algebraic equations produced by finite-difference or finite-element discretizations of boundary-value problems for ordinary and partial differential equations and they also arise in various network-flow problems.

Problems of this type are often successfully solved by a straightforward application of Newton's method. Given an initial approximation $\mathbf{x}^{(0)}$, successive iterates are found by solving the linear equations

$$J^{(k)}\boldsymbol{\nu}^{(k)} = -\mathbf{f}^{(k)}, \qquad k = 0, 1, 2, \cdots, \tag{1.3}$$

where $J^{(k)}$ is the Jacobian (1.2) at $\mathbf{x}^{(k)}$ and $\mathbf{f}^{(k)}$ is the function value at $\mathbf{x}^{(k)}$, to obtain a correction $\boldsymbol{\nu}^{(k)}$ which is used to construct the next iterate

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \boldsymbol{\nu}^{(k)}, \quad k = 0, 1, 2, \cdots. \tag{1.4}$$

Providing the initial iterate $\mathbf{x}^{(0)}$ is sufficiently close to a solution, and providing $J^{(k)}$, $k = 0, 1, 2, \cdots$ remain well away from singularity, this method is very successful. Our aim here is to retain these features in such well-behaved cases while also obtaining convergence for less easy problems.

Difficulties arise with Newton's method whenever the Newton step lies outside the region within which the linear model

$$\mathbf{f}^{(k)} + J^{(k)}\boldsymbol{\delta} \tag{1.5}$$

* Received by the editors January 23, 1985, and in revised form September 3, 1985.

† Computer Science and Systems Division, Atomic Energy Research Establishment, Harwell, Oxfordshire, England OX11 0RA.

‡ Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, Illinois 60201. The work of this author was supported in part by National Science Foundation grant DCR-8401903.

gives a good approximation to $\mathbf{f}(\mathbf{x}^{(k)}+\boldsymbol{\delta})$. In such cases a common practice is to restrict the step $\boldsymbol{\delta}$ so that this linear model is satisfactory. For instance, in the algorithm of Levenberg (1944) and Marquardt (1963) the step $\boldsymbol{\delta}^{(k)}$ is the solution of the problem

$$\begin{aligned} &\text{minimize} && \|\mathbf{f}^{(k)}+J^{(k)}\boldsymbol{\delta}\|_2 \\ &\text{subject to} && \|\boldsymbol{\delta}\|_2 \leqq \Delta^{(k)}, \end{aligned} \tag{1.6}$$

where $\Delta^{(k)}$ limits the size of the step and is called the trust region radius. The solution of (1.6) is obtained by solving a system of the form

$$(J^{(k)^T}J^{(k)}+\lambda^{(k)}I)\boldsymbol{\delta}^{(k)} = -J^{(k)^T}\mathbf{f}^{(k)}, \tag{1.7}$$

where $\lambda^{(k)} \geqq 0$ is to be determined. In the older implementations of the Levenberg–Marquardt method the size of the step is determined indirectly by varying $\lambda^{(k)}$. It is, however, preferable to control $\Delta^{(k)}$ and compute $\lambda^{(k)}$ by an iterative procedure (see Moré (1978)).

The Levenberg–Marquardt algorithm has proved to be robust, but, for large problems, it has the disadvantage that the linear least-squares problem (1.7) has to be solved at each iteration. In the sparse case only a single symbolic factorization of the structure of equation (1.7) needs to be performed, but this is very inadequate compensation for the great amount of fill-in. The purpose of this paper is to describe an algorithm where the sparsity of the problem is more fully exploited.

The alternative we consider here is to replace the spherical trust region $\|\boldsymbol{\delta}\|_2 \leqq \Delta^{(k)}$ by the cuboid $\|\boldsymbol{\delta}\|_\infty \leqq \Delta^{(k)}$ and the minimization of $\|\mathbf{f}^{(k)}+J^{(k)}\boldsymbol{\delta}\|_2$ by the minimization of $\|\mathbf{f}^{(k)}+J^{(k)}\boldsymbol{\delta}\|_1$. This permits linear programming to be employed, for which the storage requirements are often comparable with a simple Newton approach, unlike the case when the 2-norm is used. To maintain good sparsity, we use the sparse Bartels–Golub algorithm (Reid (1982)) to handle the linear programming bases, rather than any method based on orthogonal transformation.

Powell (1970) designed his "dog-leg" algorithm to avoid solving a problem of the form (1.7) at every iteration. This algorithm can also be viewed as a trust-region method, where the step $\boldsymbol{\delta}^{(k)}$ is the solution of the problem

$$\begin{aligned} &\text{minimize} && \|\mathbf{f}^{(k)}+J^{(k)}\boldsymbol{\delta}\|_2 \\ &\text{subject to} && \|\boldsymbol{\delta}\|_2 \leqq \Delta^{(k)}, \end{aligned} \tag{1.8}$$

where $\mathbf{x}^{(k)}+\boldsymbol{\delta}$ lies on the "dog-leg" consisting of the straight line from $\mathbf{x}^{(k)}$ in the direction of steepest-descent of $\|\mathbf{f}^{(k)}+J^{(k)}\boldsymbol{\delta}\|_2$ to the point minimizing $\|\mathbf{f}^{(k)}+J^{(k)}\boldsymbol{\delta}\|_2$ in this direction and a second straight line from there to the Newton point. By restricting the step to lie on this line, the trust region problem is made very simple and its solution can be found analytically. Powell's algorithm has proved to be successful for many problems. However, one cannot expect it to be as robust as the Levenberg–Marquardt method.

Reid (1973) adapted it to the sparse case, but reported finding examples for which it did not converge whereas the Levenberg–Marquardt algorithm did. An improved version of Reid's code has recently been placed in the Harwell Subroutine Library under the name NS02 (see Munksgaard and Reid (1983)). It extends the range of problems that Newton's method can handle and has the advantage of allowing any method for solving sparse linear equations to be used in conjunction with it (for example it would tolerate an out-of-core solver such as Duff's (1981) frontal code MA32). However it does not extend the range quite as far as the Levenberg–Marquardt algorithm does.

The algorithm proposed in this paper attempts to combine the best features of both methods. It is as robust as the Levenberg–Marquardt method, and its computational requirements are comparable with those of the sparse dog-leg method, provided that not many simplex steps are needed to solve the linear program. Our numerical tests, performed on a VAX 11/780 in double precision, indicate that very few steps are required in most problems. Trust-region methods based on norms other than l_2 have been proposed in other contexts. Fletcher (1972) developed an algorithm for the linearly constrained optimization problem where the trust region is defined in terms of the l_∞-norm, and Madsen (1975) described an algorithm for the minimax problem that also uses the l_∞-norm.

There are algorithms for nonlinear problems that use linear programming to solve the subproblems. Griffith and Stewart (1961) and Gomez-Palacios et al. (1982) describe such algorithms for linearly constrained optimization.

However, in the context of solving sets of nonlinear equations our choice of norms yields a different algorithm from all of these.

2. The linear programming step. We start the iteration with a trust-region radius $\Delta^{(k)}$, which specifies the size of the region about the current iterate $\mathbf{x}^{(k)}$ within which we believe the linear model (1.5) to be adequate. In the next section we will describe how to compute $\Delta^{(k)}$. The step $\boldsymbol{\delta}^{(k)}$ is a solution of the problem

$$\begin{aligned} &\text{minimize} && \|\mathbf{f}^{(k)}+J^{(k)}\boldsymbol{\delta}\|_1 \\ &\text{subject to} && \|\boldsymbol{\delta}\|_\infty \leqq \Delta^{(k)}. \end{aligned} \tag{2.1}$$

This is a linear program which can be expressed as

$$\begin{aligned} &\text{minimize} && \sum_i (p_i+q_i) \\ &\text{subject to} && J^{(k)}\boldsymbol{\delta}+\mathbf{p}-\mathbf{q}=-\mathbf{f}^{(k)}, \\ & && \left.\begin{aligned} &p_i \geqq 0,\ q_i \geqq 0 \\ &-\Delta^{(k)} \leqq \delta_i \leqq \Delta^{(k)} \end{aligned}\right\} i=1,2,\cdots,n. \end{aligned} \tag{2.2}$$

Note that the constraint matrix is

$$(J^{(k)} \quad I \quad -I) \tag{2.3}$$

so that a typical basis has a mixture of columns from $J^{(k)}$, I and $-I$ and its triangular factors will be sparser than those of $J^{(k)}$ if a good pivotal sequence is used. This linear program may be solved by the simplex method. In our code we use the subroutine LA05 of Reid (1982) to update efficiently the LU decomposition of the linear programming bases.

We have chosen to accept the step $\boldsymbol{\delta}^{(k)}$ obtained by solving the linear program (2.2) only if the inequality

$$\|\mathbf{f}(\mathbf{x}^{(k)}+\boldsymbol{\delta}^{(k)})\|_1 \leqq \|\mathbf{f}^{(k)}\|_1 - .0001\|J^{(k)}\boldsymbol{\delta}^{(k)}\|_1 \tag{2.4}$$

holds. Otherwise the step is rejected and we resolve (2.2) with $\mathbf{x}^{(k+1)}=\mathbf{x}^{(k)}$ and a reduced trust region $\Delta^{(k+1)}=\Delta^{(k)}/2$ (see § 3). Thus, what we define as an iteration in this section need not advance the solution and will only do so on satisfaction of (2.4).

One special feature of the linear program (2.2) is that $\mathbf{p}$ and $\mathbf{q}$ have been introduced only to represent the 1-norm, and so p_i and q_i will not be basis variables simultaneously (the basis would, in this case, be singular). If p_i is at its bound ($=0$) and $q_i > 0$, and the simplex step indicates that q_i should become nonbasic, it may be beneficial to

replace q_i in the basis by p_i (this is algebraically trivial and is equivalent to letting q_i become negative while p_i remains nonbasic with value zero). Barrodale and Roberts (1973) implemented this idea in their algorithm for l_1 approximation. There is another way of seeing this. Problem (2.1) can be written as the piecewise linear program

$$\begin{aligned} &\text{minimize} && \sum_{i=1}^{n} |s_i| \\ &\text{subject to} && \mathbf{s} = J^{(k)}\boldsymbol{\delta} + \mathbf{f}^{(k)}, \\ & && -\Delta^{(k)} \leqq \delta_i \leqq \Delta^{(k)}, \qquad i = 1, 2, \cdots, n, \end{aligned}$$

which can be solved by a slight variation of the simplex method. During each step of the simplex method one performs a line search on a piecewise linear objective function. A breakpoint in this objective function occurs where $s_i = 0$ (or, equivalently, $p_i = q_i = 0$ in (2.2)) for some i. Upon reaching a breakpoint, one decides either to continue in the present search direction if the reduced costs remain negative, or to conclude the iteration and drop s_i from the basis; see Fourer (1985) for further details.

Note that by solving the linear program we are finding the global minimum of problem (2.1). We show two distinctive properties of problem (2.1) in Figs. 1 and 2. In Fig. 1, if we vary $\Delta^{(k)}$, the solutions describe a piecewise-linear path, unlike the case when the l_2-norm is used, which gives rise to a smooth curve. Figure 2 shows that the solution to (2.1) is not necessarily unique since for some values of $\Delta^{(k)}$ there may be more than one solution. The fact that there may be more than one solution does not cause difficulties of any kind.

There are two natural choices for the starting basis in the linear program. The advantage of using $J^{(k)}$ is that if the Newton step lies within the trust region it will be the solution to the linear program and it will be found at once without any simplex steps. Thus the case where straightforward Newton iteration is adequate will be solved without any significant overhead.

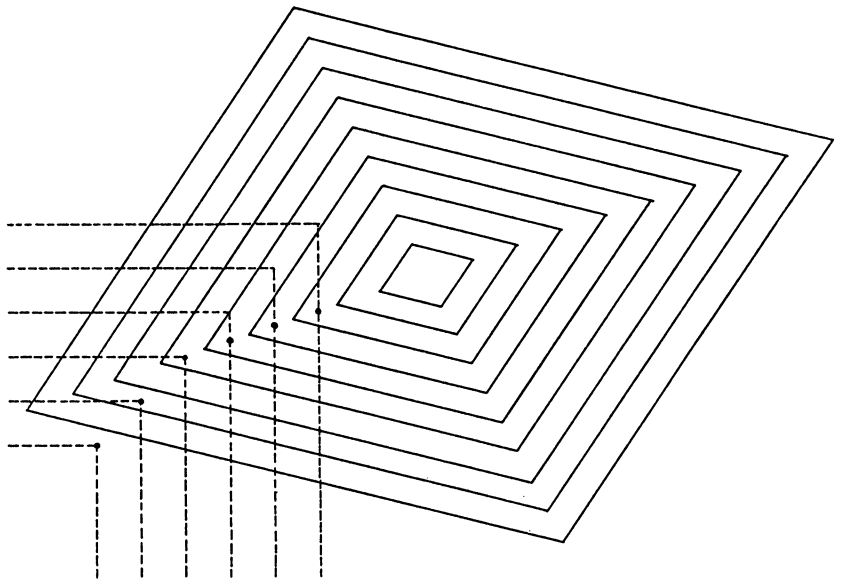


FIG. 1. *A two-dimensional problem, showing the minima following a piecewise straight line as* $\Delta^{(k)}$ *varies.* —— *contours of the objective function*; - - - *trust region boxes as* $\Delta^{(k)}$ *varies*; • • • • *constrained minima.*

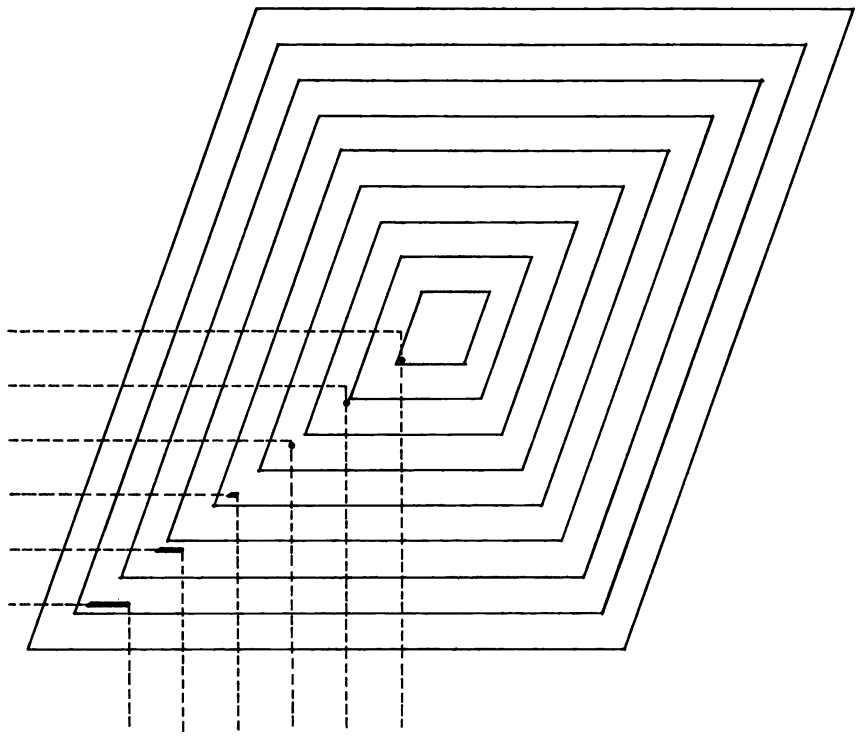


FIG. 2. *A two-dimensional problem, showing nonunique constrained minima.* —— *contours of the objective function*; --- *trust region boxes as* $\Delta^{(k)}$ *varies*; — – *constrained minima.*

The advantage of using the previous optimal basis over using $J^{(k)}$ arises when several components of the displacement $\boldsymbol{\delta}^{(k)}$ are at their bounds. In this case the use of $J^{(k)}$ as initial basis may require many simplex steps. On the other hand, we can assume that the optimal basis will not change much from one iteration to the next; therefore, starting with the previous optimal basis will result in a small average number of simplex steps per iteration.

Another advantage of using the previous basis is due to the particular form of our constraints (2.3). Suppose that the optimal basis is the same as in the previous iteration and that several of the components of $\boldsymbol{\delta}^{(k)}$ are at their bounds. Then the current basis will have the form

$$\begin{bmatrix} J_A & 0 \\ J_B & I_+ \end{bmatrix}$$

where $\begin{bmatrix} J_A \\ J_B \end{bmatrix}$ consists of a row permutation of a subset of columns of the Jacobian, and where I_+ is a diagonal matrix with elements $+1$ or -1. Reid's LA05 code only factors the matrix J_A, which will be small if many components of $\boldsymbol{\delta}^{(k)}$ are at their bounds. Therefore, if during an iteration we guess the optimal basis correctly, and if several components of $\boldsymbol{\delta}^{(k)}$ are at their bounds, the cost of the iteration will be even smaller than in the case of a simple Newton step.

To determine which of the two was the better choice for the initial basis, we performed some numerical tests. We illustrate the performance by the results in Table 1 on some highly nonlinear small problems from the collection of Moré et al. (1981). Often a less pronounced effect was apparent with other problems but the same trend was observed over a wide range of tests.

TABLE 1
Statistics on the use of the LP *approach.*

Problem	Order, n	Av. no. calls to LP subroutine per iteration: Previous basis	Av. no. calls to LP subroutine per iteration: Newton basis ($J^{(k)}$)	% of Newton steps
1. Watson	6	2.8	10.3	10%
2. Powell	2	1.3	2.3	38%
3. Helical valley	3	1.6	1.1	87%
4. Chebyquad	6	5.0	3.3	83%

Our linear programming code involves a subroutine call for each step of the simplex method. In Table 1 we report the average number of calls to this subroutine. This does not tell us directly how many simplex steps were performed. If both the initial and the optimal basis are $J^{(k)}$ then one call to the subroutine will suffice and no simplex steps will be performed. Also, if one simplex step is required to find the optimal basis, one call of the subroutine will suffice. We report the number of calls of subroutine LA05, instead of the number of simplex steps, because it is more indicative of the cost of solving the linear program.

In all runs the initial basis in the first iteration of the algorithm was $J^{(0)}$ and we terminate the algorithm when $\|\mathbf{f}\|_1 < 10^{-6}$. In the last column we report the percentage of Newton steps performed by the algorithm. Most of the problems from the Moré, Garbow and Hillstrom collection were solved mainly by Newton steps and both strategies performed very similarly. The remaining problems in their collection led us to the following conclusions:

(a) If most of the iterations were not Newton steps, using the previous basis was clearly preferable. See, for example, Problems 1 and 2 in Table 1.

(b) Some of the problems required many Newton steps interspersed with non-Newton steps. For these problems, starting with $J^{(k)}$ is advantageous, but the gain is not as great as for the problems described in (a). Examples of these problems appear in lines 3 and 4 of Table 1.

We have concluded that it is more effective to start with the previous basis, and this is the option used in our code.

It may seem that by having to solve a linear program at each iteration the algorithm will be as costly as the Levenberg-Marquardt method (if not more so). This is not the case. Our experience is that the average number of simplex steps per iteration is very low. In fact, it is much lower than the numbers reported in Table 1, which come from particularly difficult problems. We have observed that for most problems the average number of calls to the linear programming subroutine is fractionally over 1. Only in exceptional cases does the number of simplex steps exceed 3.

By using the Bartels-Golub decomposition, as implemented by Reid (1982), the cost of a simplex step will usually be moderate. When updating the LU decomposition of the basis, the algorithm of Reid (1982) uses row and column permutations to greatly reduce the number of eliminations. Once the new basis is obtained, the simplex iteration requires the solution of two linear systems, one involving the matrix and the other its transpose. It also requires a matrix-vector product in the pricing process. These back substitutions and the matrix-vector product will account for a large part of the computational cost of the simplex step.

In comparison, the Levenberg-Marquardt iteration requires at each step the solution of a problem of the form (1.7). Since we are then working with shifted normal equations, the coefficient matrix will be usually much denser than our linear programming bases and sometimes disastrously so (for example, if $J^{(k)}$ has a full row $J^{(k)T}J^{(k)}$ will be full). Thus there will generally be more work and storage required to solve (1.7) than (2.2). Additionally, the Levenberg-Marquardt iteration requires careful selection of $\lambda^{(k)}$ to prevent ill-conditioning of (1.7).

When we compare the algorithm with the sparse dog-leg method, we see that when an initial Newton step is acceptable the algorithms will have identical linear systems to solve and so, providing our initial basis for the linear program is $J^{(k)}$, computational requirements will be similar. When a Newton step cannot be taken, our approach will be better if only one or two simplex steps are required but will be poorer if many simplex steps are needed (and our numerical experience is that few steps are needed).

Note the delightful simplicity of the linear program (2.2). If we replace the l_1 norm in (2.1) by the l_∞ norm, we obtain the algorithm of Madsen (1975). At each step one solves

$$\begin{aligned} &\text{minimize} && \|\mathbf{f}^{(k)}+J^{(k)}\boldsymbol{\delta}\|_\infty \\ &\text{subject to} && \|\boldsymbol{\delta}\|_\infty \leqq \Delta^{(k)}. \end{aligned} \tag{2.5}$$

This can be reformulated in more than one way as a linear program. Madsen uses an extension of the exchange algorithm to solve this problem. Our first attempt at developing a new method was based on the feeling that there is really no need to solve (2.5) exactly, and so we experimented with holding an additional parameter μ which was the desirable size for $\|\mathbf{f}^{(k)}+J^{(k)}\boldsymbol{\delta}\|_\infty$ to give the feasibility linear program

$$\begin{aligned} &J^{(k)}\boldsymbol{\delta}+\mathbf{p}=-\mathbf{f}^{(k)}, \\ &\left.\begin{aligned} &-\mu \leqq p_i \leqq \mu \\ &-\Delta^{(k)} \leqq \delta_i \leqq \Delta^{(k)} \end{aligned}\right\} \quad i=1,2,\cdots,n. \end{aligned} \tag{2.6}$$

This is as simple as (2.2) (in fact it is slightly simpler) but the success depends critically on heuristics for μ and typically several programs with different μ values have to be solved at an iteration. We also decided to switch to the 1-norm because we felt that it is more suitable for our application.

3. The trust region. Our algorithm for choosing the trust region radius $\Delta^{(k)}$ is heuristic and is open to improvement, though we have not found that overall performance on large problems is very sensitive to changes to it. In detail our algorithm is as follows.

We begin with $\Delta^{(0)}$ equal to an upper limit set by the user as a bound on the distance to the solution. We do this because we wish to perform Newton steps if they are satisfactory. However, if this gives a point for which the sufficient descent condition (2.4), viz

$$\|\mathbf{f}(\mathbf{x}^{(0)}+\boldsymbol{\delta}^{(0)})\|_1 \leqq \|\mathbf{f}^{(0)}\|_1 - .0001\|J^{(0)}\boldsymbol{\delta}^{(0)}\|_1 \tag{3.1}$$

is not satisfied then we set $\Delta^{(1)}$ to the distance to the Cauchy point. This is the point that minimizes $\|\mathbf{f}^{(0)}+J^{(0)}\boldsymbol{\delta}\|_2$ in the direction $-\mathbf{g}^{(0)}=-J^{(0)T}\mathbf{f}^{(0)}$ of steepest descent of $\|\mathbf{f}^{(0)}+J^{(0)}\boldsymbol{\delta}\|_2$. The value of $\Delta^{(1)}$ in this case is

$$\|\mathbf{g}^{(0)}\|_2^3 / \|J^{(0)}\mathbf{g}^{(0)}\|_2^2, \tag{3.2}$$

and this is the only part of the algorithm that uses the l_2-norm. Throughout we do not permit $\Delta^{(k)}$ to be outside a user-supplied range.

For later iterations the step $\boldsymbol{\delta}^{(k)}$ is only accepted if (2.4) is satisfied. If not, $\mathbf{x}^{(k+1)}$ is set to $\mathbf{x}^{(k)}$, $\Delta^{(k+1)}$ to $\Delta^{(k)}/2$ and we continue. When (2.4) is satisfied, $\mathbf{x}^{(k+1)}$ is taken to be $\mathbf{x}^{(k)}+\boldsymbol{\delta}^{(k)}$ and the value of $\Delta^{(k)}$ is adjusted according to how well the value of $\|\mathbf{f}(\mathbf{x}^{(k)}+\boldsymbol{\delta}^{(k)})\|_1$ is approximated by the value $\|\mathbf{f}^{(k)}+J^{(k)}\boldsymbol{\delta}^{(k)}\|_1$ predicted by the linear model (1.5), as measured through the ratio

$$\rho^{(k)}=\frac{\|\mathbf{f}(\mathbf{x}^{(k)}+\boldsymbol{\delta}^{(k)})\|_1-\|\mathbf{f}^{(k)}\|_1}{\|\mathbf{f}^{(k)}+J^{(k)}\boldsymbol{\delta}^{(k)}\|_1-\|\mathbf{f}^{(k)}\|_1}. \tag{3.3}$$

If $\rho^{(k)}$ is close to 1, we consider that the linear model is good; if it is negative, then the norm of the residual vector increased; if $\rho^{(k)}$ is positive but small, we consider the prediction poor. If a Newton step is accepted, then $\Delta^{(k+1)}$ is set to $\|\boldsymbol{\delta}^{(k)}\|_\infty$, because successive Newton steps may be expected to reduce rapidly in length and evidently the linear model can be trusted in this region. Otherwise we distinguish three cases:

(a) $\rho^{(k)}\geqq 0.7$. Satisfactory agreement. Set $\Delta^{(k+1)}=2\Delta^{(k)}$ unless $\rho^{(k-1)}<0$, in which case set $\Delta^{(k+1)}=\Delta^{(k)}$.

(b) $0.1\leqq\rho^{(k)}\leqq 0.7$. Fair agreement. Set $\Delta^{(k+1)}=\Delta^{(k)}$.

(c) $\rho^{(k)}<0.1$. Unsatisfactory agreement. Set $\Delta^{(k+1)}=\sigma\Delta^{(k)}$, where $\sigma\in[\frac{1}{5},\frac{1}{2}]$.

The precise value of σ is determined by interpolating the values $\|\mathbf{f}(\mathbf{x}^{(k)}+\boldsymbol{\delta}^{(k)})\|_1$ and $\|\mathbf{f}(\mathbf{x}^{(k)})\|_1$ by a quadratic and finding its minimum. To do this, we need the value of the directional derivative at $\mathbf{x}^{(k)}$, which may not exist because we are using the l_1-norm. We follow Fletcher (1981) and set this directional derivative to $\|\mathbf{f}^{(k)}+J^{(k)}\boldsymbol{\delta}^{(k)}\|_1-\|\mathbf{f}^{(k)}\|_1$, which is the predicted reduction in the norm of the residual vector.

In (a) we have followed Powell (1970) in being cautious about increasing $\Delta^{(k)}$ because oscillations between reducing and expanding the trust-region size are clearly undesirable.

We have described in essence the algorithm for computing $\Delta^{(k)}$. There are some finer details that can improve its performance. These and other programming considerations are described in the next section.

4. Other details of the algorithm. In § 2 we concluded that it is convenient to use the optimal basis from the previous iteration as the initial basis in the simplex algorithm. Suppose that on optimality one of the components of $\boldsymbol{\delta}$, say δ_i, is at one of its bounds. Then when starting the next linear program it does not suffice to declare δ_i as a nonbasic variable; we should also set δ_i to its upper or lower bound depending on which bound it had attained previously. It is important to do this if we wish to perform few simplex steps.

If $\rho^{(k)}$ is very close to 1, say within 0.1, then it is a good idea to increase the value of $\Delta^{(k)}$ and solve the trust region problem again from the present iterate $\mathbf{x}^{(k)}$, before accepting the new iterate. The reason for this is that since the model (1.5) appears to be very good, and since evaluating the Jacobian is frequently very costly, it seems worthwhile to try a larger step. We expand $\Delta^{(k)}$ by a factor of 2 in this case. If by doing this the residuals increase, we reset $\Delta^{(k)}$ to its previous value and accept the original step. On the other hand, if the new step again gives $\rho^{(k+1)}$ close to 1 we repeat the process. This strategy is used, for example, by Dennis and Schnabel (1983, p. 144). We will not implement this strategy if either the step is a Newton step or if $\rho^{(k-1)}<0$ (i.e. if in the previous iteration the norm of the residuals increased).

In some of our tests we observed that for a long sequence of iterations the value of $\rho^{(k)}$ lay in (.2, .7), and therefore the value of $\Delta^{(k)}$ remained constant. However a

larger value of $\Delta^{(k)}$ would have been acceptable for all these iterations and the overall number of iterations for convergence would have been lower. To avoid this, we check if $\rho^{(k)} \in [.25, .65]$ and if $\rho^{(k-1)}$, $\rho^{(k-2)}$ and $\rho^{(k-3)}$ are all positive. If this is so, we increase $\Delta^{(k)}$ conservatively by a factor of 1.2.

5. Numerical experience. We ran our code on some discretized nonlinear partial differential equations, on the sparse trigonometric functions described by Reid (1973), on the problems from the Moré, Garbow and Hillstrom (1981) collection that allow an arbitrary increase in the dimension, and on some test problems associated with matrix square roots. We also ran the sparse Levenberg–Marquardt code NS03 (Reid (1973)) and the sparse dog-leg code NS02 (Munksgaard and Reid (1983)) and provided all codes with exact Jacobian matrices. We tested problems with dimensions between 10 and 625. Our tests suggest that the three codes behave similarly.

For the partial differential equations problems and the large problems of Moré, Garbow and Hillstrom the behaviour of the three algorithms was almost identical. This is due to the fact that all these problems were solved almost always by using pure Newton steps. For the sparse trigonometric functions the codes performed differently, but it is hard (and probably unreasonable) to draw any conclusion. We feel that these results are to some extent arbitrary and that the sparse trigonometric functions are not suitable for our purposes.

Although no clear winner emerged from the runs on matrix square roots, we will now discuss this class of problems further since the results at least show our new method to be competitive with the others. This class of test problems was suggested to us by Michael Saunders (SOL, Stanford). Given an $m \times m$ matrix A, we wish to find its square root, that is a matrix B such that $B^2 = A$. This problem represents a system of $n = m^2$ nonlinear equations, and the Jacobian matrix is sparse: it is block-diagonal with several outlying diagonals. For some A the problem can be solved rapidly by using only Newton steps, but for other choices of A the problem becomes very difficult and all three codes failed (or progressed too slowly). Here we will present the results for problems of an intermediate degree of difficulty.

We specify the matrix A indirectly by giving the solution B. For simplicity let us map B into the vector x^* as follows:

$$x^*(m(i-1)+j) = B(i,j), \qquad i,j = 1, \cdots, m.$$

We experimented with three choices of x^*:

$$\text{(S1)} \qquad x^*(i) = \sin(i^2), \qquad i = 1, \cdots, m^2,$$

$$\text{(S2)} \qquad x^*(i) = \begin{cases} \sin(i^2), & i = 1, \cdots, 2m, 2m+2, \cdots, m^2, \\ 0, & i = 2m+1, \end{cases}$$

$$\text{(S3)} \qquad x^*(i) = \begin{cases} 0, & i = 2m+1, \quad 3m+1, \cdots, (m-1)m+1, \\ \sin(i^2) & \text{otherwise.} \end{cases}$$

Two types of initial points x^0 were tried:

$$\text{(I1)} \qquad x^0(i) = x^*(i) - \beta \sin(i^2),$$

$$\text{(I2)} \qquad x^0(i) = x^*(i) - \beta i^2,$$

where β is a variable parameter. All the codes were stopped when $\|\mathbf{f}\|_2 < 10^{-6}$. The numbers in Table 2 indicate the number of iterations for convergence and LP stands for the new code.

These results show that our code is very competitive with the others and can succeed when other approaches fail.

TABLE 2
Number of iterations. The dog-leg code failed when overflow occurred.

n	x^*	x^0	β	Levenberg-Marquardt	Dog-leg	LP
36	S2	I1	0.8	10	9	8
36	S1	I1	0.8	11	11	16
49	S2	I1	0.6	9	8	17
64	S2	I1	5	7	6	5
16	S1	I2	0.1	>66	>66	40
49	S2	I2	10^{-4}	13	6	6
36	S3	I2	0.5	>70	F*	40
64	S3	I2	10^{-4}	11	11	11
64	S3	I2	10^{-5}	5	4	4

Acknowledgments. We would like to thank J. Moré and R. Fletcher for their comments and suggestions.

REFERENCES

I. BARRODALE AND F. D. K. ROBERTS (1973), *An improved algorithm for discrete l_1 linear approximation*, SIAM J. Numer. Anal., 10, pp. 839-848.

J. E. DENNIS AND R. SCHNABEL (1983), *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ.

I. S. DUFF (1981), MA32—*a package for solving sparse unsymmetric systems using the frontal method*, Report R.10079, AERE, Harwell.

R. FLETCHER (1972), *An algorithm for solving linearly constrained optimization problems*, Math. Programming, 2, pp. 133-165.

———, (1981), *Second order corrections for nondifferentiable optimization*, in Numerical Analysis Proceedings, G. A. Watson, ed., Dundee, 1981. Lecture Notes in Mathematics 912, Springer-Verlag, Berlin-Heidelberg-New York.

R. FOURER (1985), *A simplex algorithm for piecewise-linear programming*, Report 85-02, Dept. of Ind. Engng. and Mgmt. Sci., Northwestern Univ., to appear in Math. Programming.

F. GOMEZ-PALACIOS, L. LASDON AND M. ENGQUIST (1982), *Nonlinear optimization by successive linear programming*, Management Sci., 10, pp. 1106-1120.

R. E. GRIFFITH AND R. A. STEWART (1961), *A nonlinear programming technique for the optimization of continuous processing systems*, Management Sci., 7, pp. 379-392.

K. LEVENBERG (1944), *A method for the solution of certain nonlinear problems in least squares*, Quart. Appl. Math., 2, pp. 164-168.

K. MADSEN (1975), *An algorithm for minimax solution of overdetermined systems of nonlinear equations*, J. Inst. Math. Appl., 16, pp. 321-328.

D. W. MARQUARDT (1963), *An algorithm for least squares estimation of non-linear parameters*, J. Soc. Indust. Appl. Math., pp. 431-441.

J. J. MORÉ (1978), *The Levenberg-Marquardt algorithm: implementation and theory*, in Proc. Dundee Conference on Numerical Analysis, G. A. Watson, ed., Springer-Verlag, Berlin-Heidelberg-New York.

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

N. S. MUNKSGAARD AND J. K. REID (1983), NS02—*a Fortran subroutine for sparse sets of non-linear equations by Powell's dog-leg algorithm*, Report AERE R.11020, AERE, Harwell.

M. J. D. POWELL (1970), *A hybrid method for non-linear equations*, in Numerical Methods for Non-linear Algebraic Equations, P. Rabinowitz, ed., Gordon and Breach, New York, pp. 87-114.

J. K. REID (1973), *Least squares solution of sparse systems of non-linear equations by a modified Marquardt algorithm*, in Decomposition of Large-Scale Problems, D. M. Himmelblau, ed., North-Holland, Amsterdam, pp. 437-445.

——— (1982), *A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases*, Math. Programming 24, pp. 55-69.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 2, March 1987
© 1987 Society for Industrial and Applied Mathematics
008

LOCAL MESH REFINEMENT MULTILEVEL TECHNIQUES*

D. BAI† AND A. BRANDT†

Abstract. The purpose of this study is to provide criteria for optimizing meshsizes near singularities and develop fast and flexible multigrid methods for creating the nonuniform grids, their difference equations and their solutions. For simplicity, the Poisson problem is studied, with singularities introduced either in the forcing terms (algebraic singularities or sources) or in the shape of the boundaries (re-entrant corners). Local refinements are created by multigrid structures in which some extra finer levels cover increasingly narrower neighborhoods of the singularity, as proposed in [6]. The main innovations here are:

(1) Extra *local* relaxation sweeps near structural singularities (such as re-entrant corners) are employed to restore the asymptotic convergence rates to their values in regular (e.g. infinite) domains.

(2) An exchange-rate algorithm (λ-FMG) is introduced to maintain linear dependence of solution time on number of gridpoints. With these two algorithmic modifications, and with the optimal-mesh criteria from [6, § 8], it is found that singular problems are solved in the same efficiency as regular problems, i.e., the same accuracy in solving the *differential* equations is obtained in essentially the same computational work.

(3) In case of a source singularity it is shown that *no* local refinement is required to obtain a regular accuracy *far* from the source; if differencing is in conservation form, it is only important to represent correctly the strength of the source.

(4) A technique is developed for retaining conservation forms where local refinements are introduced.

Key words. numerical solution of partial differential equations, elliptic equations, multigrid, local mesh refinement, adaptive grids, mesh optimization, singularities, re-entrant corners, partial relaxation, λ-FMG, conservation form

AMS(MOS) subject classifications. 65N50, 65N20

1. Introduction.

1.1. Motivation and background. Nonuniform resolution is needed in the numerical solution of many, perhaps most, practical PDE problems. Progressively finer discretizations on progressively smaller subdomains are required in the neighborhood of various singular points, near re-entrant corners, at boundary layers, around captured shocks, etc. Increasingly coarser discretizations on increasingly larger subdomains are desired for exterior problems with unbounded domains, or, in solving some system-identification problems (inverse problems), on regions where the parameter to be identified can only crudely be defined, and so on. Quite often the increased resolution is needed not only for obtaining accurate solutions near the singularity itself, but also for increasing the accuracy far away: the details of the solution at the singularity may have far-reaching effects. Similarly, the details of the flow inside some boundary layers may be immaterial for the external flow, but in other cases that internal flow constitutes the very mechanism through which the entire flow is driven, and its accurate resolution is then crucial for the overall accuracy.

The present study is oriented toward steady-state boundary value problems. The simple Poisson equation will serve as our prime example, but the purpose is the development of general and flexible methods applicable in much more difficult situations. We will for example avoid using methods of "subtracting out the singularity" (see [21], [23, § 105] and [30]), which could nicely solve some of our model problems, but are not general enough. Also, our very focus here is on several-space dimensional aspect, so one-dimensional approaches, such as [10] (motivated by [11]), are excluded.

A general approach for effecting nonuniform resolution is through numerical coordinate transformations, designed so that a uniform grid in the new coordinate

* Received by the editors July 17, 1984, and in revised form August 5, 1985. The research reported here was partly supported by the Air Force Wright Aeronautical Laboratories, Air Force Systems Command, U.S. Air Force, under grant AFOSR 84-0070. It was also supported by U.S. Army contract DAJA 45-84-C-0036.

† Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot 76100, Israel.

system provides the required nonuniform resolution in terms of the old coordinates. This is an essential feature in "numerical grid generation" techniques, where, besides suitable resolution, another objective is to obtain boundary-fitted grids (see surveys [27], [26] and expository papers in [25]), and also in the "moving finite-element" method ([19], [20], [14]), in both of which the grid can be dynamically adapted. Although this approach has been very successful in many cases, it is not fully satisfactory in several respects: First, since the whole computational domain is transformed, the equations may become much more complicated, hence much more expensive to process, throughout the domain—whereas the *need* for transformation may in fact be only very *local.* Also, the obligation to use topologically rectangular grids may conflict with various local-refinements needs, hence strain the grids very much, and even introduce new singularities or new anisotropic properties into the system. Special extra equations should therefore be introduced, and solved, to control the grid generation, and re-solved many times over in case of time-dependent or other parameter-dependent problems: A new local need for changed resolution, due to a moving singularity for example, requires a new solution of the grid-generating equations, and substantial computational efforts must in particular be invested should a new singularity appear in a region where the solution has formerly been smooth. Even after all this effort, it can be proved in many cases that the transformed equations must still contain certain singularities, for purely geometrical reasons.

Furthermore, and perhaps most importantly, the transformed differential equations (and similarly the grid-generation equations, which are also solved in terms of the new coordinates) are not only more complicated than the original equations, but are usually also much more difficult to solve rapidly, by a fast multigrid solver for example, because they are less "natural": They are anisotropic where the original equations may be isotropic, or, more generally, their h-ellipticity measures (cf. [9, § 2]) are likely to be much weaker. Moreover, such weak measures in the *original* equations should not bother the fast multigrid solver (see [9, § 3.3, § 4.1, end of § 7, § 20.3.1]), but this is no longer true when the weak measures result from the coordinate transformation. Further complications arise when the desired resolution strongly depends on the solution itself: One may then have to solve simultaneously the given differential equations and the grid-generation equations, or to iteratively switch from one to the other. Finally, transformed equations often also require much larger computer storage than the original equations: The latter are most often autonomous equations (their coefficients vary only as functions of the solution itself); hence their fast multigrid solution basically requires storing only the solution itself (and even much less, if desired; cf. [9, § 8.7]); whereas the transformed equations are nonautonomous, and may require the storage of (sometimes many) location-dependent coefficients.

Part of the difficulties are due to the fact that the coordinate transformations attempt to accomplish at the same time two different tasks: the task of grid *curving* is generally independent from that of creating a desired *resolution.* Curving the gridlines is certainly important for aligning grid directions with boundaries, with interfaces or with characteristic directions, in order to allow much easier formulation of difference equations (especially higher-order ones) near and on the boundary, or to *allow* the use of meshsizes which are much smaller across than along a boundary layer, or in order to greatly reduce cross-stream viscosities, etc. But it can be done *separately* from the processes of effecting the desired resolution.

A method for creating local refinements as a natural part of multigrid fast solvers has been introduced in [6, §§ 7, 8] and further elaborated later (see [7, § 3] and [9, § 9], and a similar approach for the evolution problems in [8, § 3.9]. For an elementary

introduction to multigrid techniques see [9, § 1] or [24]). Basically, the method is to use the same sequence of uniform grids whose interaction produces the fast solution also to create nonuniform discretization patterns, simply by observing that the various grids need not all extend over the same domain: Some of the finest levels may be confined to progressively smaller subdomains, so as to provide higher resolution exactly where desired. No transformed equations are needed for that purpose. Furthermore, but separately, we may attach to each of these localized finer grids its own local coordinate system, to produce any required grid orientation or grid curving [7, § 3.5]. Unlike the global coordinate transformation, these local coordinates do not complicate the difference equations throughout the domain. Moreover, even in the small subdomain where a local transformation does take place, it is a much simpler one, first because of its completely local nature, and secondly because its only purpose is the grid *curving*, resolution being separately determined by the sequence of subdomains on which the sequence of grids is defined.

An important advantage of this special way of creating local refinements is that they come together with a very effective fast solver: As we will see in § 5, the usual multigrid efficiency can be obtained, i.e., the discrete system of equations can be solved to the level of truncation errors in just few work units, where a work unit is the number of operations appearing in the discrete system itself, and where, furthermore, these operations are highly vectorizable and convenient: they only involve simple difference equations on uniform grids, plus standard inter-grid transfers which can be programmed once for all. Also, the solution process and the generation of the nonuniform structure can form one integrated process, with no need to iteratively switch between them and without involving any grid-generation equations.

This method of local refinements is based on the FAS (Full Approximation Scheme) version of multigrid processing. In this version the usual multigrid processes can be interpreted as processes of fine-to-coarse defect corrections: the role of any fine grid, and the relaxation sweeps on it, is viewed as to supply corrections to the next-coarser-grid *equations*, so as to obtain the fine-grid accuracy on the coarser grid (see [9, § 8.2]). Since the only process being done on the finer grid is the local process of relaxation, it is clear from this point of view why (and how) can the finer grid be confined to a local subdomain. This viewpoint also shows that local refinements, in the form of finer local levels, can often be removed from the system once their defect correction is introduced to the equations of the next-coarser grid. Another very important advantage of this FAS viewpoint, in case of time-dependent problems with explicit time differencing, is that the size of stable time steps employed globally depend only on the meshsize of the global grid, not on that of any local refinement: Smaller time steps can be confined to the local-refinement grids.

This overall structure, where a nonuniform discretization is produced through a sequence of uniform grids, is highly flexible. New local refinements are easily introduced, and old ones are easily removed (keeping their defect corrections, if desired), expending negligible amounts of bookkeeping work and storage. Since only equidistance differencing is used, it becomes feasible to employ *high-order* difference approximations, despite the nonuniformity of the overall structure. The flexibility of the discretization allows its progressive improvement and adaptation. Also, the magnitude of the fine-to-coarse defect corrections (τ_h^{2h}) can be used in general criteria for adaptively deciding *where* further local refinements are needed.

Such general criteria, based on optimization consideration (e.g., minimizing the error in a given amount of computational work) were developed in [6, § 8], [9, § 9.5] and are briefly explained again in § 2 below. They can be used either *adaptively*,

determining the next levels of refinement in each stage of a λ-FMG algorithm (cf. §§ 2.3 and 3.2) based on current values of τ_h^{2h}, or a priori, determining those levels in advance based on some a priori knowledge of truncation errors.

Local mesh refinement techniques, especially for elliptic problems, appear of course extensively in the literature, starting perhaps with the "graded grids" in [1, § 14] and [23, §§ 92–93, 214–217], where local refinements are produced via rotated meshes (hence limited to rotatable operators). Later, finite elements became the main vehicle to effect such refinements. A posteriori criteria for deciding where further refinements are needed, based on local error estimator related to the finite element formulation, were introduced in [2] (and also, differently, in [29]), and were used in [22] and [3], for solving problems with re-entrant corners and boundary layers, employing multi-level (multigrid) solvers. Other works employed and analyzed multi-level methods for locally refined finite elements [28], [31], [13]. In all these, it is the finest level of the multi-level solver which is nonuniform, locally refined and adapted. The approach adopted here (and already in [6]), by contrast, is to produce the nonuniformity through the same sequence of levels used by the solver, where each level may itself be perfectly uniform, but covers a different subdomain. One advantage of this approach is that it is perfectly suitable for finite differences (with all their benefits in terms of speed and storage, even for high-order implementations) as well as for finite elements (with their superior analysis and sometimes greater convenience), and also for combinations of the two (e.g., finite differences distorted to finite elements near boundaries, as in [8, Fig. 2]). Moreover, this approach allows full integration of the grid adaptation process into the fast solver, whereas other approaches rely on switching back and forth between solving and adapting (hence their total work is *not* proportional, in principle, to the number n of points of the final grid, but to n times the number of adapatation stages, where the latter depends on the desired accuracy, hence increases with n).

Methods similar to those in [6] were employed, e.g., in [17], [18], [12]. They were also introduced to hyperbolic time-dependent schemes [15], [5]. In [5], the grid adaptation criterion is based on local function errors via Richardson extrapolation, which is similar to the τ_h^{2h} criterion of [6], and [8, § 3.9]. In both [5] and [15] the refinement is performed in both space and time (as also in [8, § 3.9]).

1.2. The present paper. The purpose of this article is to investigate several general questions which previously remained open concerning multigrid algorithms used with local *refinements*; grid *curving*, through local coordinate transformations, is *not* studied here. The general questions are investigated through simple examples with Dirichlet problems for the Poisson equation $\Delta U = F$, singularities being introduced either as singular right-hand sides ($F = r^\beta$ or a delta function), or as re-entrant corners in the shape of the domain. In these simple cases the behavior of the solution at the singularity is known; hence we have some a priori knowledge concerning the behavior of truncation errors, which we can introduce into our grid-optimization equations to determine optimal refinement patterns. For reason of clarity and methodicalness, in the present study we use such a priori determined refinements, so that the algorithmic questions examined here (speeds of convergence, algebraic as well as differential) are not confused with the detailed questions of self-adaptation. As mentioned, however, the developed λ-FMG algorithm are exactly such that they *could* as well work with self-adaptive grid refinements.

The first question we study is how to modify the usual full multigrid (FMG) algorithm to cases involving *many* levels of increasingly localized refinements. The

usual algorithm is inadequate in such cases because the number of times it uses any one of the coarser (global) grids increases at least linearly with the number of levels, so the total computational work can no longer be proportional to the total number of grid points. We replace this algorithm with the λ-FMG algorithm, based on the work-to-accuracy exchange rate λ (the Lagrange multiplier appearing in the grid-optimization equations). That is, the first approximation to the solution on all the levels created for a given λ is obtained by interpolating from a solution first obtained on the levels created for a larger λ (see §§ 3.2 and 3.3). Our experiments (see § 5) confirm that with such an algorithm, the usual multigrid efficiency is obtained, i.e., algebraic errors substantially smaller than the truncation errors are obtained, on all levels, in just few work units.

We also study the question of what interpolation orders should be used at the interior boundaries of any local-refinement level (§ 3.3). We find that that order can be the same as everywhere else, so the usual multigrid interpolation processes can uniformly be applied, with no special attention to those interior boundaries. This again is experimentally confirmed (§ 5).

Another general algorithmic question is the multigrid efficiency near "structural singularities," i.e., singularities *not* in the right-hand sides of the equations (interior equations or boundary conditions); re-entrant corners is the example we use for such singularities. Those singularities often cause serious degradation of multigrid convergence rates, especially *asymptotic* rates, i.e., rates obtained after many cycles. Although this degradation has nothing basically to do with local refinements, it only causes real troubles when *many* levels of local refinements are used. This is because the trouble is aggravated when cascading through many levels, and also because W cycles, which could cure the trouble, are prohibitively expensive in case of local refinements. Analyzing the source to this sort of trouble it becomes clear that it is necessarily related to the appearance of highly localized large residuals. We therefore propose to cure it by *local relaxation*: Before each usual (full) relaxation sweep, a small neighborhood of the singularity is relaxed (see § 4). This adds a negligible amount of computational work, but completely cures the trouble: The asymptotic convergence rates, and the performance of the FMG (or λ-FMG) algorithm, are identically the same as in regular domains (without re-entrant corners), or, for that matter, the same as in infinite domains. This strengthens our general belief that the ideal, "interior" rates can always be recovered by proper treatments of boundary troubles. We also believe that the local relaxation (also called *partial relaxation*—cf. [6, App. A.9]) is a general cure to *all* kinds of local structural singularities. It can also be useful in the first cycle of right-hand-side singularities (see § 6.1).

The numerical experiments (§ 5) also confirm the general validity of the grid-optimization equations taken from [6, § 8]. In all cases examined here we find that the resulting local refinements are such that *the differential error* (the error in approximating the solution to the *differential* equations) *decreases as a function of the total computational work as rapidly as in regular cases*, i.e., as in Poisson equations without singularities.

In certain cases, however, it was found that the optimization equations are inadequate, unless correctly re-interpreted. These are cases of *interior* singularities, such as source singularities (see § 6), in which the *signs* of truncation errors near the singularities should be taken into account, because these errors may very significantly cancel each other.

For the important case of source singularities (F is a delta function) it was found, somewhat surprisingly, that local refinements near the source are not at all significant in terms of improving the solution far away from the singularity; they are only effective

in reducing the errors right there, near the singularity. For the solution everywhere else, the only important feature is that the total strength of the singularity is properly represented, and this can automatically be accomplished by using conservative differencing.

This led us to another general, and quite central, question: *How can local refinements be done, in* FAS *framework, in a conservative way*? The approach developed and tested in § 6.2 below can be generalized to many other systems. Similar conservation at grid interfaces is more recently studied in [4], in the framework of hyperbolic systems of conservation laws, with more general grid geometries.

2. Mesh optimization equation. In solving singular differential problems by adaptable discretization, the choice of discretization parameters should be guided by our aim of minimizing the work invested to obtain a given accuracy, or, what turns out to be the same thing, minimizing the error obtained in a given amount of computational work. For the detailed derivation of optimization equations based on this principle, see [6], [9]. In [6] both the orders of approximation and meshsizes are optimized. Here only meshsize adaptation is discussed, starting with a brief review of the optimization equations for that case.

When a proper multigrid algorithm is used (see §§ 3-3.2), the amount of invested work, W, is proportional to the number of gridpoints in the domain, and can therefore be approximated by:

$$W \approx \int_{\Omega} \frac{w}{h^d}\, dx \tag{1}$$

where d is the dimension of the problem's domain Ω, $x = (x_1, \cdots, x_d)$, $h = h(x)$ is the meshsize, and $w = w(x)$ is the work per gridpoint, which is generally proportional to the complexity of the finite difference formulae.

The solution error E can always be approximated by a functional of the form

$$E \approx \int_{\Omega} G\tau\, dx, \tag{2}$$

where $\tau = \tau(x, h)$ is the local truncation error, i.e., $\tau(x, h) = |F^h - L^h U|$, U being the solution of the differential equation $LU = F$ and $L^h U^h = F^h$ is the discrete approximation used. The "error weighting function" $G = G(x)$ is chosen so as to express the goal of the entire calculation [6], [9]. Equation (2) simply states the assumption that the error made in the *equation* at one point x affects the solution error, and its measure E, independently of that made at another point y; we assume we cannot count on their canceling each other. This assumption is misleading in some special cases (see § 6.1).

The Euler differential equation for optimizing h by minimizing E under fixed W is

$$\frac{\partial E}{\partial h(x)} + \lambda \frac{\partial W}{\partial h(x)} = 0,$$

or, by (1) and (2)

$$G\frac{\partial \tau}{\partial h} = \lambda\, dw h^{-d-1} \quad \text{at all points } x, \tag{3}$$

where λ is a constant independent of x, expressing the marginal rate of exchanging optimal accuracy for work, i.e., $\lambda = -dE_{\min}/dW$. It will serve as our chief control parameter, called *the exchange rate*, specifying in fact the cost at which an additional unit of accuracy can be afforded.

Philosophical Comment A. Other grid adaptation schemes prefer to use criteria based on solution gradients and similar features. Such features may lead to implicating the solution process with grids which are refined due to purely local errors, i.e., errors which can be corrected as a local post-processing of the computed solution. The error estimator (2), by contrast, is expressed in terms of the *global* effects of the *local* truncation error, i.e., the error in the *equation.* This is exactly the type of errors that cannot be reduced by purely local post-processing, hence exactly the type that should be treated at the solution process itself.

Philosophical Comment B. Some grid adaptation schemes take no account of the work W. This must in principle lead to questionable situations, e.g., to an unbounded refinement sequence (cf. § 2.3).

The optimization equation (3) can be used in two different ways. First, the mesh adaptation procedures can be integrated into a multigrid solver, where at each stage an approximation to $\partial\tau/\partial h$ is known (from the quantity τ_h^{2h} appearing in the FAS multigrid processing; cf. [6] or [9]): Wherever the left-hand side of (3) is substantially larger than its right-hand side, a finer level should be added. This approach nicely suits the λ-FMG solver introduced below. In many cases, however, the process can be simplified since crude approximations to $\partial\tau/\partial h$ can be calculated in advance. Equation (3) can then be used to prescribe the set of grids (levels) ahead of time. Two examples of the later situation follow.

2.1. Example: right-hand side (RHS) singularity. As an example of a RHS singularity we take Poisson's equation $\Delta U = F$ with $U = r^\alpha$, $F = \alpha^2 r^{\alpha-2}$, where r is the distance to the origin and $\alpha > -2$. The domain of solution is the square $\{0 \leqq x_1 \leqq R_0, 0 \leqq x_2 \leqq R_0\}$, so that $r = 0$ is at the boundary corner point $(0, 0)$. The error weighting function for estimating the L_1 norm of the solution errors is $G(x) = d_1 d_2$, where d_1 and d_2 are the distances from x to the nearest vertical and horizontal boundaries, respectively. (A unit error made in the equation at the point x contributes an error in the solution, at any other point y, proportional to the Green function $G(x, y)$. Since our system is self-adjoint, $G(x, y) = G(y, x) \approx d_1 d_2 \bar{G}(y)$ for $|d_i| \ll |y|$ and a suitable function $\bar{G}(y)$. Thus, if E is any error norm not concentrated at the corners, the unit error in x will contribute to E proportionally to $d_1 d_2$.) Since we are only interested in the gross refinement patterns around the singularity at $(0, 0)$, for programming convenience the grids that cover only part of the domain will be taken to be squares of the form $\{0 \leqq x_1 \leqq R, 0 \leqq x_2 \leqq R\}$, where $0 < R = R(h) < R_0$ and the meshsize ratios between levels k and $k+1$ is $h_{k+1}/h_k = \frac{1}{2}$. If more precise refinement shapes or meshsize ratios were sought for better optimizations, an additional control and optimization work would be needed, which would be much greater than what can be gained by slightly improved optimization. Since we take only such square refinements, we want the meshsize only as a function of the distance to the origin. Hence, as a good approximation for our purposes we take $G = r^2$.

For second order approximations the truncation error is:

$$\tau(r) = O\left(h^2\left(\frac{\partial^4 U}{\partial x^4} + \frac{\partial^4 U}{\partial y^4}\right)\right) = O(h^2 r^{\alpha-4}). \tag{4}$$

Hence, if a numerical solution is obtained without local refinements, on a uniform grid with meshsize h, then any error norm E will roughly be

$$E \approx \int_h^{R_0} G(r)\tau(r) r\, dr = O(h^{2+\alpha} + h^2). \tag{5}$$

From (1) we have in this case $W = O(1/h^2)$. We therefore get for the RHS singularity with a second order approximation $WE = O(h^{\alpha}+1)$. For $-2<\alpha<0$ this behavior of WE is much worse than the regular case, in which $E = O(h^2)$ and hence $WE = O(1)$. We will show that by using local refinements the regular relation between W and E can be recovered.

For the truncation error given by (4), Euler optimization (3) gives:

$$h = O(\lambda^{1/4} r^{(2-\alpha)/4}). \tag{6}$$

Using this nonuniform meshsize, we get from (1)

$$W \approx \int_0^{R_0} \frac{1}{h(r)^2} r\,dr = O(\lambda^{-1/2}), \tag{7}$$

and from (2)

$$E \approx \int_0^{R_0} G(r)\tau(r) r\,dr = O(\lambda^{1/2}). \tag{8}$$

Thus we get $WE = O(1)$, *as in the regular case.* These estimates are confirmed by numerical experiments in § 5.1.

2.2. Example: structural singularity. As an example for structural singularity we take Laplace's equation $\Delta U = 0$ in the domain of Fig. 1.

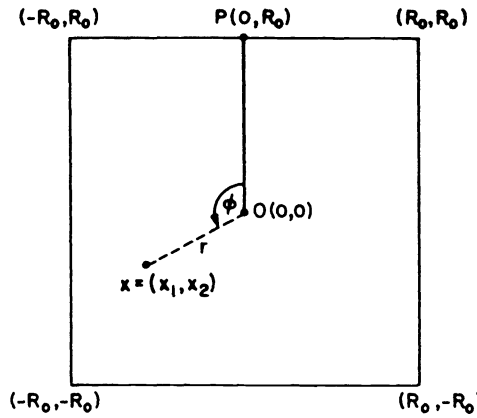


FIG. 1. *The domain for the structural singularity example. PO is the re-entrant boundary, with respect to which the radial coordinates* (r, ϕ) *are defined.*

The solution we take is $U = r^{1/2}\sin(\phi/2)$. This solution satisfies Laplace's equation $\Delta U = 0$ and the condition $U = 0$ on the re-entrant boundary so there is no right-hand side singularity, neither in the equation nor in the boundary conditions. Hence the singularity near the point (0, 0) result only from the structure of the boundary. (Singularities in the differential left-hand side operators should also be classified as "structural singularities," showing the same kinds of algorithmic troubles.)

The error weighting function around $r = 0$ is in this case $G \approx r^{1/2}\sin(\phi/2)$ (since in this case $G(x, y) = G(y, x) \approx r^{1/2}\sin(\phi/2)\bar{G}(y)$; cf. § 2.1), and as a good approximation for our purposes we take $G = r^{1/2}$ (cf. the justification for simplifying G in § 2.1). For programming convenience the grids that cover only part of the domain are on the

squares $\{|x_1| \leqq R, |x_2| \leqq R\}$, where $0 < R < R_0$. For second order approximation the truncation error is

$$\tau(r) \approx O\left(h^2\left(\frac{\partial^4 U}{\partial x_1^4}+\frac{\partial^4 U}{\partial x_2^4}\right)\right) = O(h^2 r^{-7/2}). \tag{9}$$

If a numerical solution is obtained with a uniform meshsize h, then the error norm is

$$E \approx \int_h^{R_0} G(r)\tau(r) r\, dr = O(h) \tag{10}$$

(as indeed shown in numerical tests; cf. § 4) and the invested work is $W = O(1/h^2)$, so that $WE = O(h^{-1})$, much worse than the regular case where $WE = O(1)$. On the other hand, for the truncation error (9) Euler optimization (3) gives

$$h = O(\lambda^{1/4} r^{3/4}), \tag{11}$$

for which

$$W \approx \int_0^{R_0} \frac{1}{h(r)^2} r\, dr = O(\lambda^{-1/2}), \tag{12}$$

$$E \approx \int_0^{R_0} G(r)\tau(r) r\, dr = O(\lambda^{1/2}), \tag{13}$$

so that $WE = O(1)$, as in the regular case. This is confirmed by numerical tests (§ 5.2).

Note that, more generally, if one chooses $h(r) = O(\lambda^{1/4} r^{\beta})$, (12) holds for any $\beta < 1$, while (13) holds for any $\frac{1}{2} < \beta$, hence $WE = O(1)$ for any $\frac{1}{2} < \beta < 1$. The significance of this result is that if in actual computations there is a small deviation from $\beta = \frac{3}{4}$, we still get $WE = O(1)$.

2.3. The sequence of grids. Once an exchange ratio λ is chosen, the optimization equation (3) determines the optimal meshsize $h(x)$ (whether a priori, as in the above examples, or adaptively, approximating $\partial\tau/\partial h$ from the evolving solutions). In the multigrid structure described here this means that subdomains $D_\nu = D_\nu(\lambda)$ are determined over which the corresponding grids G^ν extend, $(\nu = 0, 1, 2, \cdots)$, where each G^ν is a uniform grid with meshsize $h_\nu = 2^{-\nu} h_0$, with h_0 chosen so that G^0, which should of course cover the entire domain, contains few gridpoints.

It is important to note that, for any fixed λ, the sequence of grids $G^\nu(\lambda)$ thus chosen is always finite: As in the above two examples, a relation of the form $h = O(r^\beta)$, with $\beta < 1$, is always established near a singularity. Hence for h sufficiently small, the radius r of the grid becomes smaller than the meshsize h, which means that the grid is empty. (A grid G^h becomes meaningless when it contains no interior points belonging to G^{2h} and it therefore no longer contributes a τ_h^{2h} correction.) The smaller λ is, the longer is the sequence, larger is the number of complete grids (grids covering the entire domain), and larger is the subdomain covered by each incomplete grid.

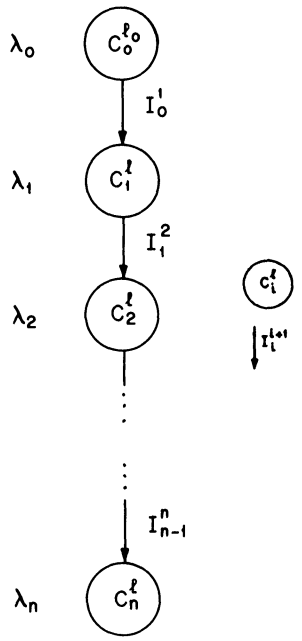
For programming convenience we take in all our numerical examples the boundary of G^ν $(\nu > 0)$ to be along gridlines of $G^{\nu-1}$.

3. Local refinement algorithmic considerations. Several aspects of the usual full multigrid (FMG) algorithm should be re-examined and possibly modified when local levels are used. They are discussed below.

3.1. The FMG algorithm in case of local refinements. When the FMG algorithm [9] is used for solving a regular problem with all grids covering the entire domain, the

work invested on coarse levels is usually smaller than the work invested at the finest one, despite the fact that the coarser grids are relaxed many more times. This is due to the fact that the coarser the grid, the fewer its points. As a result the total work is proportional to the number of gridpoints in the domain. On the other hand, when local levels of refinement are used, with finer levels covering much smaller subdomains, the number of points on coarser levels is not necessarily small in comparison. The regular FMG algorithms—which makes a cycle through all the coarser grids per each additional level of refinement, no matter how small is the refined domain—may easily end up investing work which is no longer proportional to the total number of gridpoints.

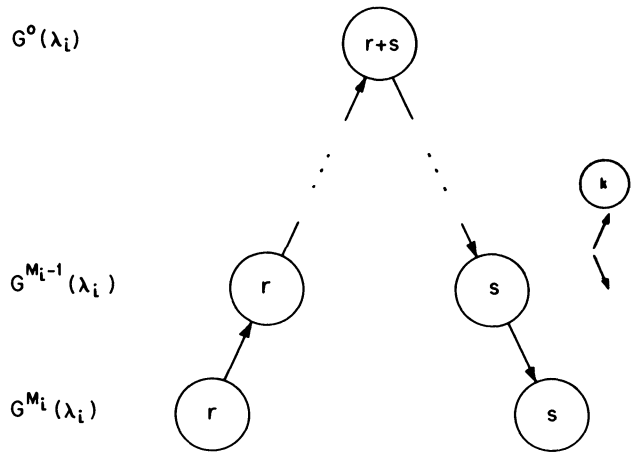
3.2. The exchange-rate FMG algorithm (λ-FMG). The λ-FMG algorithm is a modification designed to alleviate this difficulty. In this algorithm the problem is solved for a sequence $\lambda_0 > \lambda_1 > \lambda_2 > \cdots$ of exchange-rate values (λ in § 2), where λ_0 is some high value of λ for which a crude solution is obtained by investing a very small amount of work, and $\lambda_i = \alpha\lambda_{i-1}$ with a constant factor $\alpha < 1$. For each λ_i $(i \geqq 1)$ the first approximation is obtained by interpolating the λ_{i-1} solution to the grids $G^\nu(\lambda_i)$. Assuming λ_{i-1} to be close enough to λ_i, this first approximation is such that a multigrid cycling with only a small number of relaxation sweeps on each grid of λ_i is required to obtain the λ_i solution to within truncation errors: For regular elliptic equations one $V(r, s)$ cycle, usually with $r+s \leqq 3$, should suffice (see § 3.3). An entire such algorithm is shown in Fig. 2. A modification of the cycle near structural singularities is discussed below (§ 4).



C_i^l — l cycles of type C_i on the λ_i-grids $G^k(\lambda_i)$; for example the cycles can be $C_i = V_i(r, s)$ shown in Fig. 2b.

I_i^{i+1} — Interpolation from $G^k(\lambda_i)$, $k = 0, \cdots, M_i$ to $G^j(\lambda_{i+1})$, $j = 0, \cdots, M_{i+1}$. For gridpoints of $G^\nu(\lambda_{i+1})$ common to $G^\nu(\lambda_i)$ leave the same solution as on $G^\nu(\lambda_i)$. For other gridpoints of $G^\nu(\lambda_{i+1})$ – interpolate solution from $G^{\nu-1}(\lambda_{i+1})$.

FIG 2a. λ-FMG *algorithm.*



Ⓚ k relaxation sweeps (on the grid shown at left).
↗ FAS residual and solution transfer.
↘ FAS correction interpolation.

FIG. 2b. *$V_i(r, s)$ cycle. The grids participating in the cycle are shown in the left column.*

3.3. Choice of parameters. The exchange rate ratio $\alpha = \lambda_i/\lambda_{i-1}$ and some parameters of the multigrid cycling remained to be discussed. If the order of approximation is p, the truncation error is of the form $\tau(x, h) \approx c(x)h^p$. Denoting by N_i the total number of gridpoints in the λ_i levels, and by E_i the solution error (a norm of the difference between the exact discrete solution and the true differential solution) obtained for λ_i, the optimization eq. (3) yields $h = O(\lambda^{1/(p+d)})$ and hence

$$N_i = O\left(\int \frac{dx}{h(x)^d}\right) = O(\lambda_i^{-d/(p+d)}), \tag{14a}$$

$$E_i \approx \int h(x)^p c(x) G(x)\, dx = O(\lambda_i^{p/(p+d)}) \tag{14b}$$

so that

$$N_{i+1}/N_i = \alpha^{-d/(p+d)}, \qquad E_{i+1}/E_i = \alpha^{p/(p+d)}. \tag{14c, d}$$

To optimize α, detailed work-accuracy relations should be considered. For this purpose we denote:

s—the number of relaxation sweeps on each level per cycle.
μ_0—error reduction factor per sweep, i.e., the algebraic error is roughly reduced by μ_0^{ls} by l multigrid cycles with s sweeps on each level. (This usually can hold as long as μ_0^s is not too small, e.g., $\mu_0^s \geqq .02$ or so.)
w— the computational work per grid point in a relaxation pass.

For the l cycles done on the grids of each λ_i we thus have:

relaxation work $= N_i wsl$,
residual-transfer and correction-interpolation work $= N_i wl\beta$,

where β is some $O(1)$ factor which can be calculated: If for example the difference operators are much more complex than the I_h^{2h} and I_{2h}^h interpolation operators, then

$\beta \approx 2^{-d}$ for residual injection and $\beta \approx 1$ for full residual weighting.

$$\text{Initial cubic interpolation work} = (N_i - N_{i-1})w\gamma$$
$$= N_i(1-\alpha^{d/(p+d)})w\gamma,$$

where γ is another $O(1)$ factor. So the total work invested in the λ_i cycles is

$$W_i \approx wN_i[l(s+\beta)+\gamma(1-\alpha^{d/(p+d)})]. \tag{15}$$

Denoting by W^n the total work invested to produce the λ_n solution, we then have, from (14c),

$$W^n = W_n + W_{n-1} + \cdots \approx wN_n\left[\frac{l(s+\beta)}{1-\alpha^{d/(p+d)}}+\gamma\right]. \tag{16}$$

Denote by η the relative accuracy in solving the λ_i equations, i.e., the algebraic error (measured by a norm corresponding to that of E_i) is roughly reduced to ηE_i. Since in FMG algorithms algebraic errors tend to be in the direction of truncation errors, the algebraic error on the λ_{i+1} grids just after interpolation from the λ_i grids is about $(1+\eta)E_i - E_{i+1}$. This algebraic error should be reduced to ηE_{i+1}, hence

$$\mu_0^{st} \approx \eta E_{i+1}/[(1+\eta)E_i - E_{i+1}]. \tag{17}$$

The optimal value of η is such that the switch from cycling on the λ_i grids to cycling on the λ_{i+1} grids is done when the two have the same marginal error reduction per work unit (cf. [9, § 7.2]), i.e., when

$$\eta E_i/N_i \approx [(1+\eta)E_i - E_{i+1}]/N_{i+1}. \tag{18}$$

Hence, by (17) and (14),

$$\mu_0^{sl} \approx E_{i+1}N_i/(E_iN_{i+1}) = \alpha.$$

With this value of α substituted in (16), we obtain the total work W^n (excluding the fixed work of first interpolation, $\omega N_n\gamma$) in terms of p/d, s, l, β and μ_0. Table 1 gives the former for common ranges of the latter ($\mu_0 \approx .25$ is common for red-black relaxation, $\mu_0 \approx .5$ for lexicographic ordering). The table shows how insensitive is W^n to the exact choice of parameters. Any α in the range $.02 \leqq \alpha \leqq .25$ is clearly good, yielding W^n close to minimal. Usually one cycle ($l=1$) with 2 or 3 relaxation sweeps is close to optimal, although for larger p/d, or for a small choice of α, two cycles ($l=2$) may start to be attractive, especially since larger s may be less effective than shown by the table because the assumption of a reduction by μ_0^s per cycle no longer holds.

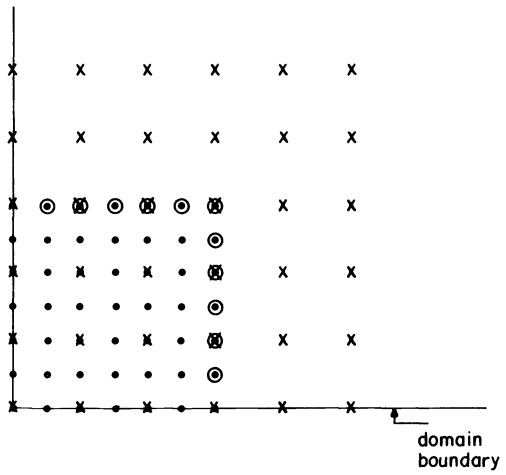
In our numerical experiments (§§ 5.1–5.2), where $p=d=2$ the value $\alpha = 1/16 = 0.0625$ was taken. This value corresponds to the value $\alpha = 2^{-p-d}$ used in usual FMG algorithms (without local refinements), where $N_{i+1} \approx 2^d N_i$.

3.4. Interpolation at refinement boundary. In Fig. 3 two grids are described: a fine grid with meshsize h and a coarse-grid with meshsize $H=2h$. The fine grid covers a part of the domain of the coarser grid. When a relaxation sweep is done on the fine grid, its interior boundary points, marked by $\odot$, are not relaxed. The solution values at these points are thus the accumulated result of the FAS interpolations from the coarse grid. Therefore the order and kind of interpolations at these points determine the numerical solution obtained at convergence. If the interpolation order is too low, the accuracy of the numerical solution will degrade.

TABLE 1

The computational work in a λ-FMG algorithm (in work units based on the total number of gridpoints, excluding the work of first interpolation: see (16)).

			$\mu_0 = .25$			$\mu_0 = .5$		
				$\frac{l(s+\beta)}{1-\alpha^{\alpha/(p+d)}}$			$\frac{l(s+\beta)}{1-\alpha^{\alpha/(p+d)}}$	
p/d	l	s	$\alpha = \mu_0^{sl}$	$\beta = .5$	$\beta = 1.2$	$\alpha = \mu_0^{sl}$	$\beta = .5$	$\beta = 1.2$
1	1	1	.25	3.0	4.4	.5	5.1	7.5
		2	.062	3.3	4.3	.25	5.0	6.4
		3	.016	4.0	4.8	.12	5.4	6.5
		4	.0039	4.8	5.5	.062	6.0	6.9
	2	1	.062	4.0	5.9	.25	6.0	8.8
		2	.0039	5.3	6.8	.062	6.7	8.5
		3	.00024	7.1	8.5	.016	8.0	9.6
2	1	1	.25	4.1	5.9	.5	7.3	10.7
		2	.062	4.1	5.3	.25	6.8	8.6
		3	.016	4.7	5.6	.12	7.0	8.4
		4	.0039	5.3	6.2	.062	7.5	8.6
	2	1	.062	5.0	7.3	.25	8.1	11.9
		2	.0039	5.9	7.6	.062	8.3	10.6
		3	.00024	7.5	9.0	.016	9.3	11.2



•, ⊙ Points of a grid with meshsize h.
× Points of a grid with meshsize $2h$.

FIG. 3. *Two noncoextensive grids.*

To be on the safe side, the solution interpolation order at the refinement boundary should be at least $m+p$, where m is the order of the differential equation and p is the approximation order. This is the interpolation which introduces the same order of local error as the local truncation error. Moreover, this error is introduced only at relatively few gridpoints. The area covered by these gridpoints is usually only $O(h)$, hence an interpolation of order $m+p-1$ could in fact be enough to keep the same *order* of errors; but the actual *size* of the error might then degrade, especially in cases where many intermediate grids participate so that the area associated with refinement boundaries is not so small.

Furthermore, the order $m+p$ is most convenient for interpolation since this is the order usually used anyway for interpolating the solution in the FMG algorithm (because one usually wishes to see *residuals* to within truncation error; cf. [9, § 7.1]).

There remains the question of the order that should be used for interpolating *corrections* at refinement boundaries, i.e., in the correction cycle. Here an $O(h^m)$ interpolation would do, since the correction size is only $O(h^p)$ relative to the size of the solution. This again is convenient, being the same order as usually used at all other points (see [9, § 4.3]).

The upshot of this discussion is that *nothing special need be done at the refinement boundary*: The interpolations there are simply parts of the usual multigrid interpolations performed throughout the fine grid.

For numerical experiments concerning interior boundary interpolation orders see § 5.1.

4. Multigrid convergence for re-entrant corners. When the FMG-algorithm is used for solving problems with structural singularities, convergence slower than usual is sometimes observed. As an example we give in Table 2 results of the numerical experiments done for the re-entrant corner problem discussed in § 2.2, with coextensive grids (i.e., no local refinements). The second column of the table shows the error $\|u^h - U\|_{L_1}$, where u^h is the numerical solution obtained by an FMG algorithm with one $V(2,1)$ cycle for each new fine level, and U is the differential solution. The third column shows the asymptotic convergence rate (ACR) per such cycle. (The ACR is $\lim_{i\to\infty} R_{i+1}/R_i$, where R_i is any fixed residual norm, measured after the ith cycle. It is independent of the particular norm chosen.) The fourth and fifth columns give the error $\|u^h - U\|_{L_1}$ and the ACR when $W(2,1)$ cycles are used. The sixth column gives the truncation error $\|U^h - U\|_{L_1}$, i.e., the error $\|u^h - U\|_{L_1}$ obtained after many cycles. In all cases we have used Gauss-Seidel relaxation in lexicographic ordering, cubic interpolation of solutions, linear interpolation of corrections and injection of residuals. Better results could be obtained by red-black ordering.

TABLE 2
The solution of the re-entrant corner problem with the usual V and W cycles.

	$V(2,1)$ cycles		$W(2,1)$ cycles		
# levels	$\|u^h - U\|_{L_1}$	ACR	$\|u^h - U\|_{L_1}$	ACR	$\|U^h - U\|_{L_1}$
4	1.60	0.10	1.50	0.070	1.44
5	0.93	0.15	0.78	0.083	0.74
6	0.53	0.19	0.39	0.094	0.37
7	0.31	0.23	0.19	0.096	0.19
8	0.23	0.26			0.095

From the last column of the table it can be seen that the truncation error is $O(h)$, in accordance with the discussion of § 2.2 (cf. (10)).

The V-cycles ACR shows significant dependence on the number of levels used, in contrast to regular problems, where similar cycles give ACR ≈ 0.08, independently of the number of levels. This even affects the accuracy obtained by one-cycle FMG (the second column): the error becomes appreciably larger than the truncation error as the number of levels is increased.

There are two reasons for the ACR degradation:

1) Error components slow to converge in relaxation are approximate solutions of the homogeneous differential equation. Near the singularity a component of this type is $r^{1/2} \sin(\phi/2)$, which has large second derivatives and hence large errors ($O(h^{1/2})$ instead of $O(h^2)$) in linear interpolation.

2) When transferring residuals from a fine grid to a coarse one, unusual weighting should be given to residuals near the singularity using the weight function $r^{1/2} \sin(\phi/2)$.

It can be seen from Table 2 that when W cycles are used the ACR is almost the regular one and as a result the one cycle FMG results are close to the truncation error. Thus the ACR degradation for V cycles is basically not a two-level feature, but results from the above errors cascading through many levels. Hence its dependence on the number of levels.

Our main concern is what should happen when local refinements are used. There the number of levels is much larger and the degradation, of both ACR and FMG accuracy, should become very significant. (This indeed is the case: cf. Table 5 below.) Moreover, W cycles in case of local refinements are extremely inefficient since they relax $O(2^{l-k})$ times through G^k, where l, the total number of grids, is large.

The slow convergence can presumably at least partly be corrected by using proper interpolations and residual weighting near the singularity. But such corrections are inconvenient to program and are too problem dependent.

A better way to obtain a good convergence with V cycles is to make special local relaxation passes over a small number of points near the singularity. This will eliminate the large interpolation errors (which are large only locally there) and will reduce large local residuals, hence making their correct weighting less important.

Typical results are shown in Table 3. The algorithm used for the second and third columns of Table 3 is identical to the corresponding columns of Table 2, but with an additional partial relaxation pass, only over the points marked by × in Fig. 4, performed before each full relaxation sweep.

As can be seen in Table 3, for 4 and 5 levels the ACR is the same as for regular problems. For more levels the ACR still slightly deteriorate. This deterioration can be eliminated by slowly increasing the number of points over which the partial relaxation

TABLE 3
The solution of the re-entrant corner problem with modified V cycles.

Modified V cycles			
# levels used	$\|u^h - U\|_{L_1}$	ACR	$\|U^h - U\|_{L_1}$
4	1.43	0.085	1.44
5	0.74	0.085	0.74
6	0.39	0.10	0.37
7	0.20	0.12	0.19
8	0.12	0.14	0.095

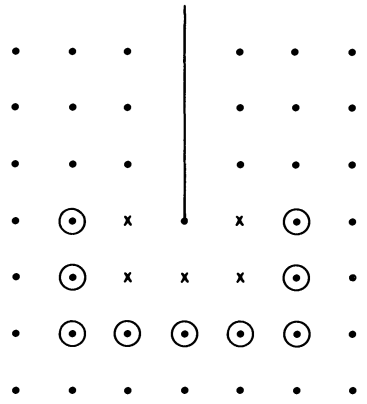


FIG. 4. *The gridpoints on which partial relaxation is performed. A portion of the grid near the re-entrant tip is shown.* ×—*Partial relaxation is performed on these points, if the grid is coarse.* ⊙—*Additional points for partial relaxation, in case of large fine grids.*

is performed as the level number increases. In a numerical experiment done for seven levels, the partial relaxation passed over the points marked by × in Fig. 4 and then, if the level number was $\geqq 4$, also on those marked by ⊙. The ACR obtained was 0.083, like the ACR of a regular problem.

Note that the work invested in the partial relaxation passes remains a negligible fraction of the work of the regular sweeps; the fraction in fact rapidly decreases when more levels are added.

5. Local refinement numerical experiments.

5.1. RHS singularity. Numerical experiments were performed for the problem discussed in § 2.1 with $\alpha = -1.5$, the domain $\{0 \leqq x_1 \leqq 10,\ 0 \leqq x_2 \leqq 10\}$, and $r = 0$ at the boundary point (0, 0). Levels of local refinement were created, governed by the relation (6) and the considerations of § 2.3. In the notation of § 2.1 this means that $R(h) = \min(10, [\lambda^{-2/7} h^{8/7}])$, where the brackets denote truncation to the nearest multiple of $2h$ and grids are defined for $h_i = 10/2^i$, $(i = 1, 2, \cdots, N)$, N being the last number for which $R(h_i) \geqq 4h_i$. Gauss-Seidel relaxation was used, with red-black ordering. Table 4 summarizes the work W and the error E obtained by a λ-FMG algorithm with $\lambda_{i+1} = \lambda_i/16$, with one $V(1, 1)$ cycle for each value of λ_i, and with cubic (i.e., fourth order) solution interpolation and linear (second order) interpolation of corrections at all refinement points (including the refinement boundary; cf. § 3.4). Also shown are asymptotic values of E for other interpolation strategies.

The table shows that the one-cycle FMG algorithm produces for each λ a solution well within the truncation error. (Comparison should be made with E_a^c, which uses the same solution interpolation order.) More importantly, it is clear from the results that as λ decreases *WE* remains approximately constant, as expected from § 2.1. Thus, *basically the same work-to-accuracy relation is obtained as for regular problems.* For high values of λ, where the number of levels is small, the numerical results are still somewhat affected by details of the grid structure.

For low values of λ the table shows that $E_a^c(\lambda_{i+1})/E_a^c(\lambda_i) \approx \frac{1}{4}$, in accordance with the discussion in § 3.4. The table shows, somewhat unexpectedly, that linear interpolation gives better results for large λ. This can be explained by the fact that *close enough to the singularity* cubic interpolation is worse than linear, especially at the point adjacent to the boundary, where the cubic interpolation is not central. Far from the singularity,

TABLE 4
Errors in the solution of the RHS *singularity problem with local mesh refinements.*

λ_i	M	N	E_a^l	E_a^c	λ-FMG, one $V(1,1)$ per λ_i		
					E	W	WE
6.25×10^{-4}	3	8	0.24	0.22	0.24	572	135
3.91×10^{-5}	4	16	0.028	0.039	0.029	3133	90
2.44×10^{-6}	5	24	0.0063	0.014	0.011	14766	162
1.53×10^{-7}	6	32	0.00087	0.0042	0.0040	64959	258
9.54×10^{-9}	7	40	0.00076	0.0011	0.0010	273952	274
5.96×10^{-10}	8	48	0.00059	0.00027	0.00023	1111473	257

M—number of levels covering the whole domain.
N—total number of levels.
$E = \|u^h - U\|_{L_1}$.
W—total work, measured as the total number of points passed in all relaxation passes.
E_a^l—asymptotic value of E after many cycles (i.e., at convergence), when only linear interpolation to refinement-boundary points was used.
E_a^c—asymptotic value of E with cubic interpolation at *all* stages.

however, linear interpolation is not satisfactory, for the reasons explained in § 3.4. Indeed, we see in Table 4 that for small λ the behavior of linear interpolation is unsatisfactory. A better general approach could presumably be to replace only the noncentral cubic interpolation, and only when close to the singularity, by linear interpolation.

In experiments when many cycles were employed per level, asymptotic convergence ratios were always the same as in regular problems. This is of course always true for RHS singularities, because the right-hand side never affects the iteration eigenvalues.

5.2. Structural singularity. Numerical experiments were performed for the problems discussed in § 2.2 with $R_0 = 5$. The results, obtained by one-cycle λ-FMG algorithms with exchange ratio 1:16 and with grids defined by $R(h) = \min(5, [\lambda^{-1/3}h^{4/3}])$, are summarized in Table 5. The cycles used were $V(2,1)$ cycles with Gauss-Seidel relaxation in lexicographic order. The table compares results obtained with and without partial relaxation passes. The latter were performed over the five points marked by × in Fig. 4, once before each regular sweep (cf. § 4). In all

TABLE 5
The solution of the re-entrant corner problem with local mesh refinement.

			with partial relaxation				without partial relaxation				
			λ-FMG				λ-FMG				
$10^5\lambda$	M	N	E	W	WE	ACR	E	W	WE	ACR	E_a
39.0	4	9	0.068	4809	329	0.095	0.29	4269	1244	0.23	0.070
2.4	5	13	0.010	18768	195	0.16	0.125	17868	2240	0.32	0.010
0.15	6	17	0.0019	78637	152	0.20	0.049	77319	3795	0.36	0.0020
0.0095	7	21	0.00061	316635	192	0.22	0.018	313503	5643	0.39	0.00048

M, N, E, W—the same meaning as in Table 4.
E_a—asymptotic values of E (i.e., truncation error).

cases the first interpolation (solution interpolation) to all points was cubic and the correction interpolations were linear (cf. § 3.4).

The table shows that in case of local refinements the partial relaxation is absolutely needed. Experiments described in § 4 indicate that the results could further be improved (especially for the lowest λ) by increasing the set of partial relaxation on some of the larger grids. But even with the limited set, the one-cycle λ-FMG results are satisfactory, giving errors well within truncation errors and basically constant values of *WE*. Thus, *the same efficiency in solving the differential problem is obtained as in cases without the structural singularity.*

6. Source singularity. As an example for a source singularity we take the domain Ω of Fig. 5 and the Poisson equation:

$$\Delta U = 2\pi\delta \tag{19a}$$

with Dirichlet boundary conditions that yield the solution

$$U(r) = \log\left(\frac{r}{R_0}\right) \tag{19b}$$

where r is the distance to the origin. The equation is approximated by the usual second order five-point approximation

$$\Delta^h U^h = S_{ij}^h = \begin{cases} 2\pi/h^2 & \text{if } (i,j) = (0,0), \\ 0 & \text{otherwise.} \end{cases} \tag{20}$$

This example of an RHS singularity is important, first because it appears as an essential component in many applications, and also because it clarifies a certain interpretation needed in using the optimization equation (3).

Indeed, if (3) is naively used, since $\tau = O(h^2 r^{-4})$ one gets $h = O(r)$, a very severe refinement since it requires an unbounded sequence of grids for any fixed λ (cf. § 2.3). However, results of numerical experiments (cf. § 6.1), show that *no* local refinement is needed: If the strength of the source is correctly represented, the error in the solution is $O(h^2/r^2)$. Moreover, local refinement may actually *impair* the accuracy, especially if nonconservative discretization at refinement boundaries is used.

6.1. Source singularities with coextensive grids. In Table 6 numerical results for coextensive grids are given. For each $5 \leqq l \leqq 7$ and $1 \leqq n \leqq 16$ the table gives $\|u^l - U\|_{S(n)}$, where $u^l = u^{h_l}$ is the solution obtained on grid l, whose meshsize is $h_l = 2^{1-l} R_0$, and $\|\cdot\|_{S(n)}$ denotes L_1 averaging (sum of absolute values, divided by the number of summed terms) along the square circumference

$$S(n) = \{(x_1, x_2): \max(|x_1|, |x_2|) = nh_5\}, \tag{21}$$

i.e., the boundary of a square with center at the origin and half-width nh_5. The domain is Ω of Fig. 5, with $R_0 = 5$. The FMG algorithm used one $V(1,1)$ cycle (cf. Fig. 2b) per level with *red-black* Gauss-Seidel relaxation (hence *half* injection should be used instead of the usual injection) and with cubic interpolation of solutions and linear interpolation of corrections. The results "with partial relaxation" were obtained by adding one relaxation pass on the points of the square

$$\{(x_1, x_2): \max(|x_1|, |x_2|) \leqq 3h_k\} \tag{22}$$

before any usual (full) relaxation sweep over any grid k. The table clearly shows that the error is generally $O(h^2/r^2)$; in fact, except very near the boundary, the error at (x_1, x_2) depends only on $(h/x_1, h/x_2)$.

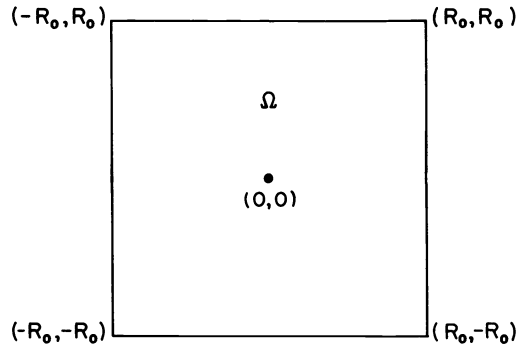


FIG. 5. *Domain of solution for the source singularity problem. The source is located at the center point* (0, 0).

As can be seen from the table, the results after one cycle with half-injection and without partial relaxations differ considerably from the asymptotic results, and in fact are much better. However, if full weighting of residuals is used, the results after one cycle are almost identical to the asymptotic results. This shows that after the first interpolation to a fine grid the residuals near the singularity changes rapidly and therefore the full weighting is needed to represent them correctly on coarser grids. We can therefore conclude that the better results obtained after one cycle with half-injection and no partial relaxations are really accidental: The misrepresentation of residuals immediately near the singularity happens to have such a symmetry that it is equivalent

TABLE 6
Discretization errors (times 10^3*) of source problem* (19).

	$10^3\|u^l - U\|_{S(n)}$											
	After one cycle FMG algorithm											
	half-injection						full weighting					
l	without partial relaxation			with partial relaxation			without partial relaxation			Asymptotic		
n	5	6	7	5	6	7	5	6	7	5	6	7
1	38.0	10.0	0.83	41.0	13.0	2.2	41.0	13.0	2.7	41.0	13.0	2.8
2	10.0	0.81	0.43	13.0	2.0	0.49	13.0	2.7	0.61	13.0	2.8	0.66
3	1.3	0.71	0.11	4.4	0.82	0.21	4.8	81.1	0.27	4.8	1.2	0.29
4	0.70	0.46	0.048	2.2	0.49	0.11	2.7	0.61	0.15	2.8	0.66	0.16
5	0.45	0.24	0.026	1.4	0.31	0.074	1.6	0.38	0.091	1.7	0.42	0.11
6	0.80	0.12	0.020	0.82	0.21	0.049	1.1	0.27	0.066	1.2	0.29	0.073
7	0.64	0.055	0.018	0.61	0.15	0.037	0.81	0.20	0.048	0.87	0.22	0.054
8	0.49	0.034	0.016	0.49	0.11	0.029	0.60	0.14	0.036	0.66	0.16	0.041
9	0.31	0.023	0.015	0.39	0.092	0.023	0.46	0.11	0.027	0.52	0.13	0.032
10	0.21	0.014	0.014	0.30	0.073	0.018	0.36	0.087	0.021	0.40	0.10	0.025
11	0.13	0.015	0.011	0.23	0.056	0.014	0.29	0.070	0.017	0.31	0.078	0.019
12	0.075	0.018	0.008	0.18	0.042	0.010	0.23	0.055	0.014	0.24	0.060	0.015
13	0.053	0.017	0.006	0.13	0.031	0.008	0.17	0.041	0.010	0.17	0.043	0.011
14	0.043	0.013	0.004	0.087	0.021	0.005	0.11	0.027	0.007	0.11	0.028	0.007
15	0.022	0.008	0.002	0.049	0.011	0.003	0.056	0.014	0.003	0.055	0.014	0.004
16	0	0	0	0	0	0	0	0	0	0	0	0

to adding a "quadruplet" to the "charge" represented by the source, hence it introduces an error with the same behavior in space as the discretization error. The two errors happen to have opposite signs, hence they partly cancel each other.

Note also that extra *partial* relaxation sweeps further smooth the residuals near the singularity so that they are better represented on coarser grids even with half-injection, hence yielding results closer to the asymptotic. This is useful to know since one may like to save the work of full weighting, which is equivalent to the work of another *full* cycle of relaxation.

In order to investigate the behavior of the truncation error in *asymmetric cases*, experiments with inhomogeneous mediums were performed, e.g., with the problem (see Fig. 6)

$$\begin{aligned} &\nabla \cdot (\varepsilon(\underline{x})\nabla U(\underline{x})) = 2\pi\delta(\underline{x}) \quad \text{for } \underline{x} \in \Omega, \\ &\varepsilon = 1 \quad \text{for } 0 < x_2 < R_0, \\ &\varepsilon = 2 \quad \text{for } -R_0 < x_2 < 0, \end{aligned} \tag{23}$$

and with the Dirichlet boundary conditions (19b). Results of numerical experiments, with $R_0 = 5$, central five-point approximations to (23) and the same algorithm as above, are given in Table 7. U^l in the table stands for U^{h_l}, the exact solution on grid h_l, and $D_n^{k,l}$ is the error on grid h_k relative to grid h_l, i.e.,

$$D_n^{k,l} = \| U^{h_k} - U^{h_l} \|_{S(n)}, \tag{24}$$

with $\| \cdot \|_{S(n)}$ defined as before. The results clearly show that the error is $O(h^2/r^2)$, similar to the homogeneous case.

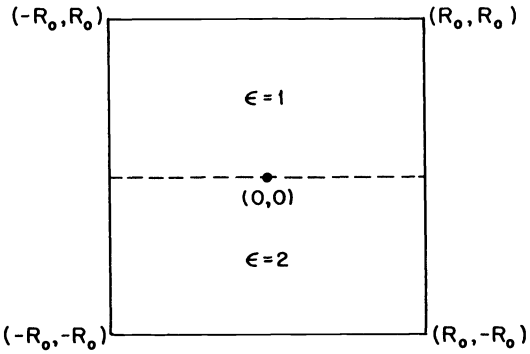


FIG. 6. *Domain with inhomogeneous medium. The source is located at* (0, 0).

In both cases we thus see that the optimization equation (3) fails to recommend a correct refinement pattern. The reason is that (2) fails in this case; it fails because it assumes that the local truncation errors τ at different points contribute independent contributions to E, while in the present cases these contributions nearly cancel each other, in a specific way.

Indeed, the (signed!) local truncation error $\tau^h(x) = F^h(x) - L^h U(x)$, which is the error in the *equation*, contributes to any error in the *solution*, such as $U(y) - U^h(y)$, some signed contribution, roughly equals to $h^2\tau^h(x)G(x, y)$, where G is the Green function—and different contributions from different points x may cancel each other. Usually these cancellations are very partial and accidental, hence they do not change the order of magnitude of the errors. We cannot count on them and we cannot, and

TABLE 7
Results for source problem in a discontinous medium (23).

n	$\|U^7\|_{S(n)}$	$10^3 n^2 D_n^{6,7}$	$D_n^{6,7}/D_n^{5,6}$
1	1.709	9.38	.62
2	1.266	5.98	.22
3	1.000	5.26	.24
4	0.809	5.39	.24
5	0.661	5.09	.25
6	0.539	5.29	.25
7	0.437	5.13	.25
8	0.347	5.18	.25
9	0.268	5.09	.25
10	0.197	4.93	.25
11	0.135	4.74	.25
12	0.095	4.34	.25
13	0.080	3.74	.25
14	0.082	2.89	.25
15	0.099	1.66	.25
16	0.132	0.00	

need not, account for them. Thus, (3) is generally used, where the simplifying assumption is that we want to minimize the sum of the *absolute values* of the different contributions. This assumption is very useful, except in the specific situation of an interior singularity, around which the large values of τ^h have nearly zero sum, so that their contributions very significantly cancel each other. By studying the exact nature of these cancellations, (3) can be modified for such specific situations.

In the present case, since the difference equations are conservative and represent the exact strength of the source, it can easily be seen that the sum of τ^h over the entire domain is $O(h^2)$. (It equals the discretization error in the total normal flux along the boundary, centrally differenced along a line $h/2$ from the boundary, where there is no singularity.) Hence the large values of τ^h near the singularity have zero sum. Since, for any point y far from the source, they all have nearly the same $G(x, y)$, their contributions to $U(y) - U^h(y)$ nearly cancel each other. This explains the accuracy of our solutions far from the source.

6.2. Local refinement near source singularity. As discussed in § 6.1, the error behavior is $O(h^2)$, as for regular problems, and *WE* is uniformly bounded even without local refinements (cf. § 2). However, the error *near* the singularity is large, and it is sometimes exactly there that accurate solutions are desired. Hence it is interesting to examine how local fine levels near the singularity affects the errors, both near and far.

In Table 8 numerical results with local refinements, for the problem posed in (20) and the domain of Fig. 5 with $R_0 = 5$, are given. In these experiments six full levels $G^1, G^2, \cdots, G^6$ and a seventh local level G_N^7 were used, and the results can therefore be compared with the results in the last two columns of Table 6.

As can be seen from the table, the error outside the region covered by G_N^7 is *worse* than the error obtained by dropping G_N^7 altogether; for low N (a small domain of local refinement) it is even *much* worse. On the other hand, well inside the refined region (if that region is large enough to show it), the error is reduced by the local refinement as effectively as by the full refinement.

TABLE 8

Solution errors $10^3\|U^l - U\|_{S(n)}$ with nonconservative refinements. Six full levels and a seventh local level were used. The local level G_N^7 covers the square $\{|x_1| \leqq Nh_7, |x_2| \leqq Nh_7\}$. Thus, the detached lines indicate the boundary of the local level. The last row gives $S^{6,5} = h_6^2 \sum_{i,j} \tau_{i,j}^{6,5}$, where $\tau_{i,j}^{l+1,l}$ is defined by (27), and the summation is over all the points of G^6.

	N					
n	4	8	12	16	20	24
1	68.0	6.5	2.6	2.6	2.7	2.7
2	51.0	5.9	2.1	0.93	0.65	0.63
3	41.0	4.9	1.9	0.89	0.47	0.30
4	34.0	4.0	1.6	0.82	0.45	0.26
5	28.0	3.4	1.4	0.71	0.41	0.24
6	23.0	2.8	1.1	0.60	0.36	0.22
7	19.0	2.3	0.66	0.51	0.30	0.19
8	16.0	1.9	0.79	0.42	0.25	0.16
9	13.0	1.6	0.65	0.34	0.21	0.14
10	11.0	1.3	0.52	0.27	0.16	0.11
11	8.4	1.0	0.41	0.21	0.13	0.089
12	6.3	0.75	0.31	0.16	0.096	0.069
13	4.5	0.53	0.22	0.11	0.068	0.050
14	2.8	0.33	0.13	0.071	0.043	0.033
15	1.3	0.16	0.063	0.033	0.021	0.016
16	0	0	0	0	0	0
$S^{6,5}$	0.159	0.019	0.0080	0.0043	0.0027	0.0019

The above results are clearly related to the conservation properties of Poisson's equation $\Delta U = F$. By using Gauss' theorem this equation can be written in the form of a flux equation

$$\int_{\partial D} \frac{\partial U}{\partial n}\, dS = \int_D F\, dV \tag{25}$$

where D is any domain where Poisson's equation is satisfied, ∂D is the boundary of D and $\partial U/\partial n$ is the derivative of U normal to the boundary in the outside direction. The left-hand side of (25) is the total flux leaving the domain ∂D, and the right-hand side is the total strength of the sources contained in D.

The second order approximation to Poisson's equation on a uniform grid is:

$$(U_{i+1}^h - U_{ij}^h) + (U_{i,j+1}^h - U_{ij}^h) - (U_{ij}^h - U_{i-1,j}^h) - (U_{ij}^h - U_{i,j-1}^h) = F_{ij}h^2 \tag{26}$$

where h is the meshsize and U_{ij}^h is the value of U^h are gridpoint (i, j), the point with cartesian coordinates (ih, jh). This equation can be regarded as a flux equation; the left-hand side terms approximate the flux through the four sides of an $h \times h$ square centered at (i, j). We denote this square by D_{ij}^h. The right-hand side of (26) gives the total source strength in D_{ij}^h.

If the discrete equations that correspond to all the points of a uniform grid in some domain R are summed up, most of the left-hand side terms in (26) are canceled out; the remaining terms approximate the flux through the boundaries of R. The right-hand side of the resulting equation is the total source strength in R.

For S_{ij} defined by (20), $h^2 S_{ij}$ represents exactly the physical source strength in D_{ij}^h for the problem (19). Suppose now that we have l coextensive grids G^k, $k = 1, \cdots, l$ and a local fine grid G^{l+1}. We denote by D_k the domain covered by G^k. In Fig. 7 G^l is described and the detached line denotes the boundary of G^{l+1}. The discrete equations for the gridpoints of G^l not interior to G^{l+1} (i.e., points marked by $\cdot$ or $\times$ in Fig. 7) are given by (20) ($h = h_l$). For the points marked by $\odot$

$$F_{ij}^h = S_{ij}^h + \tau_{ij}^{l+1,l} \tag{27}$$

where $\tau_{ij}^{l+1,l}$ is the truncation error of G^l relative to G^{l+1}. Since $\sum'_{ij} \tau_{ij}^{l+1,l} \neq 0$, where $\sum'$ denotes summation over the points marked by $\odot$ (cf. last row in Table 8), the total source strength on G^l is different from the physical source strength. The difference results in an $O(\sum' \tau_{ij}^{l+1,l})$ error at points far from the singularity, as clearly shown by Table 8. This error is much larger than the $O(h^2/r^2)$ error discussed in § 6.1, especially when the refined region (D_{l+1}) is small. Indeed, $\sum \tau_{i,j}^{l+1,l}$ equals to the difference between the flux along the boundary of D_{l+1} as measured on G^l and the same flux as measured on G^{l+1}, a difference which is $O(Rh^2 \partial^3 U/\partial r^3(R)) = O(h^2/R^2)$, where R is roughly the "radius" of D_{l+1}. This error is much larger than the $O(h^2/r^2)$ error discussed in § 6.1. For larger D_{l+1}, that is, for larger N in Table 8, $|\sum'_{ij} \tau_{ij}^{l+1,l}|$ is smaller and the effect of nonconservation of the physical source strength is proportionately smaller.

The conclusion is that it is better to do local refinements with conservative equations, i.e., equations that conserve the total source strength. For achieving this, one should modify the form of $\tau_{ij}^{l+1,l}$ at the refinement boundary. The desired form is obtained from the requirement that in $\sum^* w^* L^* U^* + \sum^B \tau_{i,j}^{l+1,l}$ all interior terms are canceled out, where $\sum^B$ is a summation over the points of the refinement boundary (marked by $\odot$ in Fig. 8), U^* is the discrete solution, L^* is the discrete operator (e.g., $L^* = \Delta^h$ at interior fine grid points and $L^* = \Delta^H$ at other points, $h = h_{l+1}$, $H = 2h = h_l$) and $\sum^* w^* \cdots$ is a discrete approximation to integrating over the domain. It is essential that this approximation is at least $O(h^2)$, otherwise the resulting conservative equations will be of lower orders, so for example we can take $\sum^*$ to be summation over all gridpoints shown in Fig. 8, with w^* having the weight h^2 at the points marked by $\times$,

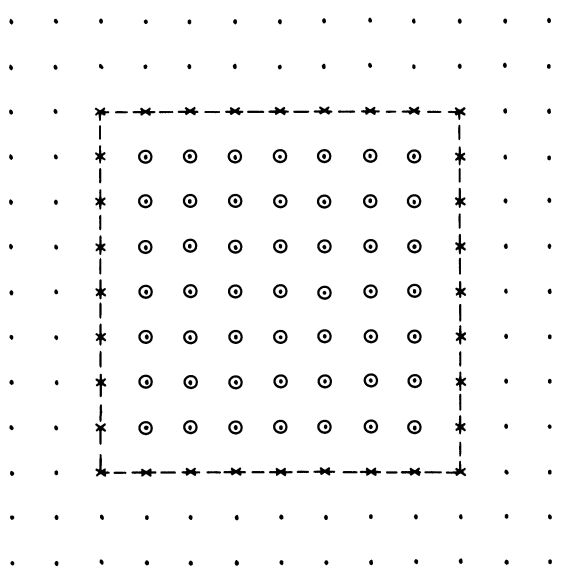


FIG. 7. *Local refinement near the source singularity. The gridpoints of a coarse grid are shown. The boundary of the locally refined grid is marked by the detached line.*

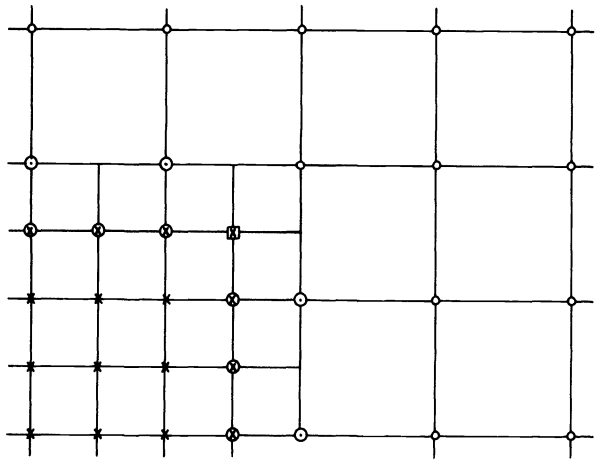


FIG. 8. *Portions of two noncoextensive grids.*

the weight $h^2/2$ at $\otimes$, $h^2/4$ at $\boxtimes$, and $(2h)^2$ at all other points. In our case (the five-point Laplacians Δ^h and Δ^H), for example, the resulting form of $\tau^{l+1,l} = \tau^{h,H}$ at a point (marked by $\odot$) on the *right* boundary of the fine grid will be

$$
\begin{aligned}
H^2\tau_{ij}^{h,H} &= \nabla^H u_{ij}^H - (\nabla^H u_{i,j+1}^h + 2\nabla^H u_{ij}^h + \nabla^H u_{i,j-1}^h)/4 \\
&= \tfrac{1}{2}(u_{ij}^h - u_{i-2,j}^h) - \tfrac{1}{4}(u_{i,j+1}^h + u_{i,j-1}^h - u_{i-2,j+1}^h - u_{i-2,j-1}^h)
\end{aligned} \tag{28}
$$

where the indices are fine-grid indices, $\nabla^H u_{ij} = u_{i,j} - u_{i-2,j}$, and the second equality is valid in case $u_{ij}^H = (\hat{I}_h^H u^h)_{ij} = u_{ij}^h$, i.e., in case the FAS fine-to-coarse solution transfer is an injection.

The implementation of these equations is easily done as a natural part of the FAS algorithm, where the coarse grid equations have the general form (cf. [6], [9])

$$
L^H U^H = F^H + \tau^{h,H} = I_h^H F^h + \tau^{h,H} \tag{29}
$$

with $\tau^{h,H}$ vanishing at points not covered by the fine grid, with

$$
\tau^{h,H} = L^H(\hat{I}_h^H u^h) - I_h^H(L^h u^h) \tag{30}
$$

at coarse gridpoints which are interior to the find grid, and with relations like (28) now added at fine-grid boundary points. Note that, from accuracy point of view, the added $\tau^{h,H}$ at each boundary point is $O(h)$, but since the area covered by such points is only $O(h)$, the overall accuracy is still $O(h^2)$.

In Table 9 results similar to Table 8, but with the conservative equations (28)-(30), are given. It is clearly seen from the table that except for a small region near the boundary of the seventh level, the results outside the refined area are about the same as the results obtained by using only six full levels, while inside the refined area the results are similar to the results obtained by using seven full levels.

We can therefore conclude that if one is interested in the solution far from the singularity it is best to solve with coextensive grids. However, to enhance accuracy *close* to the singularity, local refinements are very effective. These local refinements need not fully interact with the coarser grids that cover larger domains: The solution on the coarser grid G^l can as usual be used for obtaining a first approximation to the finer grid G^{l+1}, but the solution on G^{l+1} need not be used to improve the solution at points of G^l outside $D^l \cap D^{l+1}$. With this approach, since the solution of G^l determines the boundary values of G^{l+1}, the boundary of G^{l+1} should be far enough from the singularity, so that the error on the boundary will be small compared with the final

TABLE 9
Solution errors $10^2\|U^l - U\|_{S(n)}$ *with conservative local refinements. Tabulated as in Table* 8.

	N					
n	4	8	12	16	20	24
1	4.0	2.9	2.8	2.8	2.8	2.8
2	2.4	0.96	0.70	0.67	0.67	2.4
3	1.1	0.85	0.41	0.33	0.31	0.30
4	0.64	0.55	0.41	0.23	0.23	0.18
5	0.41	0.38	0.32	0.24	0.14	0.12
6	0.29	0.27	0.24	0.20	0.15	0.10
7	0.21	0.20	0.19	0.17	0.14	0.11
8	0.46	0.16	0.15	0.13	0.12	0.098
9	0.13	0.12	0.12	0.11	0.099	0.086
10	0.099	0.097	0.094	0.088	0.081	0.073
11	0.077	0.076	0.074	0.070	0.066	0.060
12	0.059	0.058	0.057	0.054	0.051	0.048
13	0.043	0.042	0.041	0.040	0.038	0.035
14	0.028	0.028	0.024	0.026	0.025	0.024
15	0.014	0.014	0.013	0.013	0.012	0.012
16	0	0	0	0	0	0

intended error near the source. This can be done if the error is known, as in the case of (20), where it is $O(h^2/r^2)$. As an example suppose that one is interested in a solution whose accuracy at $r = r_0$ is the same as that obtained with l coextensive grids, but with each D_k extending only over $\{|x_1|, |x_2| \leqq R_k\}$, where R_k is as small as possible. This means that the error on G^{l-1} at $r = R_l$ should be no larger than the error on G^l at $r = r_0$, i.e., $h_{l-1}^2/R_l^2 = C_l(h_l^2/r_0^2)$, where $0 < C_l \leqq 1$. Therefore for $h_{l-1}/h_l = 2$, $R_l = (2/\sqrt{C_l})r_0$. In order to obtain on G^{l-1} a good approximation at $r = R_l$ the error on G^{l-2} at $r = R_{l-1}$ should be smaller than the error on G^{l-1} at $r = R_l$. Therefore $h_{l-2}^2/R_{l-1}^2 = C_{l-1}(h_{l-1}^2/R_l^2)$. Hence $R_{l-1} = (2/\sqrt{C_{l-1}})R_l$. A similar relation exists for any other level k, $k < l-1$. A generally reasonable sequence will thus for example be $R_{k+1} \approx .4R_k$.

Note that since there is no full interaction between a local fine grid and coarser grids there is no need to care about the correction on the boundary of the fine grid given by (28).

If, however, interactions between the levels *is* allowed, the same accuracy may be obtained with smaller refinement zones (smaller R_l).

Acknowledgment. The authors are grateful for the hospitality given them by the Institute for Computational Studies at Colorado State University, and thank its director, Dr. Robert Hayman, for encouraging the research reported here.

REFERENCES

[1] D. N. DE G. ALLEN, R. V. SOUTHWELL AND G. VAISEY, *Relaxation methods applied to engineering problems*, XI. *Problems governed by the quasi-plane-potential equation*, Proc. Roy. Soc. A, 183 (1945), pp. 258-283.

[2] I. BABUSKA AND W. C. RHEINBOLDT, *Error estimates for adaptive finite element computations*, SIAM J. Numer. Anal., 15 (1978), pp. 736-754.

[3] R. E. BANK AND A. H. SHERMAN, *The use of adaptive grid refinement for badly behaved elliptic partial differential equations*, in Mathematics and Computers in Simulation XXII, North-Holland, Amsterdam, 1980, pp. 18-24.

[4] M. J. BERGER, *On conservation at grid interfaces*, ICASE Report No. 84-83, Hampton, VA, Sept. 1984.
[5] M. J. BERGER AND J. OLIGER, *Adaptive mesh refinement for hyperbolic partial differential equations*, J. Comp. Phys., 53 (1984), pp. 484-512.
[6] A. BRANDT, *Multi-level adaptive solutions to boundary-value problems*, Math. Comp., 31 (1977), pp. 333-390, ICASE Report 76-27, Hampton, VA.
[7] ———, *Multi-level adaptive techniques for singular-perturbation problems*, in Numerical Analysis of Singular Perturbation Problems, P. W. Hemker and J. J. H. Miller, eds., Academic Press, New York, 1979, pp. 53-142, ICASE Report 78-18, Hampton, VA.
[8] ———, *Multi-level adaptive finite-element methods*: I. *Variational problems*, in Special Topics in Applied Mathematics, J. Frehse, D. Pallaschke and U. Trottenberg, eds., North-Holland, Amsterdam, 1980, pp. 91-128, ICASE Report 79-8, Hampton, VA.
[9] ———, *Multigrid Techniques*: 1984 *Guide, with Applications to Fluid Dynamics*, monograph. Weizmann Institute of Science, Rehovot, Israel, February 1984. Now available as GMD-Studie No 85, from GMD-F1T, Postfach 1240, D-5205, St. Augustin, W. Germany. A preliminary version appeared in [16, pp. 220-312].
[10] S. F. DAVIS AND J. E. FLAHERTY, *An adaptive finite element method for initial-boundary value problems for partial differential equations*, this Journal, 3 (1982), pp. 6-27.
[11] C. DE BOOR, *Good approximation by splines with variable knots* II, Conference on the Numerical Solutions of Differential Equations, Lecture Notes in Mathematics 363, Springer-Verlag, New York, 1973.
[12] J. E. DENDY, *A priori local grid refinement in the multigrid method*, in Elliptic Problem Solvers II, G. Birkhoff and A. Schoenstadt, eds., Academic Press, New York, 1984, pp. 439-451.
[13] D. GANNON AND J. VAN ROSENDALE, *Partial architectures for iterative methods on adaptive block structures grids*, in Elliptic Problem Solvers II, G. Birkhoff and A. Schoenstadt, eds., Academic Press, New York, 1984, pp. 93-104.
[14] R. J. GELLINAS AND S. K. DOSS, *The moving finite element method: Application to general partial differential equations with multiple large gradients*, J. Comp. Phys., 40 (1981), pp. 202-249.
[15] W. D. GROPP, *A test of moving mesh refinement for* 2-D *scalar hyperbolic problems*, this Journal, 1 (1980), pp. 191-197.
[16] W. HACKBUSCH AND U. TROTTENBERG, eds., *Multigrid Methods*, Lecture Notes in Mathematics 960, Springer-Verlag, New York, 1982.
[17] D. R. MCCARTHY AND T. A. REYHNER, *Multigrid code for transonic potential flow about inlets*, AIAA J., 20 (1982), pp. 45-50.
[18] D. R. MCCARTHY AND R. C. SWANSON, *Embedded mesh multigrid treatment of two-dimensional transonic flows*, Applied Math. Comp., 13 (1983), pp. 399-418.
[19] K. MILLER AND R. N. MILLER, *Moving finite elements* I, SIAM J. Numer. Anal., 18 (1981), pp. 1019-1032.
[20] K. MILLER, *Moving finite elements* II, SIAM J. Numer. Anal., 18 (1981), pp. 1033-1057.
[21] H. MOTZ, *The treatment of singularities of partial differential equations by relaxation methods*, Quart. Appl. Math., 4 (1946), pp. 371-377.
[22] M. C. RIVARA, *Design and data structure of fully adaptive multigrid finite-element software*, ACM Trans. Math. Soft., 10 (1984), pp. 242-264.
[23] R. V. SOUTHWELL, *Relaxation Methods in Theoretical Physics*, Clarendon Press, Oxford, 1946.
[24] K. STÜBEN AND U. TROTTENBERG, *Multigrid methods: Fundamental algorithms, model problem analysis and applications*, in [16, pp. 1-176].
[25] J. F. THOMPSON, ed., *Numerical Grid Generation*, North-Holland, Amsterdam, 1982.
[26] ———, *Dynamically-adaptive grids in the numerical solution of partial differential equations*, in PDE Software: Modules Interfaces and Systems, B. Engquist and T. Smedsaas, eds., North-Holland, Amsterdam, 1984, pp. 411-427.
[27] U. F. THOMPSON, Z. U. A. WARZI AND C. W. MASTIN, *Boundary-fitted coordinate systems for numerical solution of partial differential equations—A review*, J. Comp. Phys., 47 (1982), pp. 1-108.
[28] J. R. VAN ROSENDALE, *Rapid solution of fluid element equations on locally refined grids by multilevel methods*,Ph.D. thesis, Univ. Illinois, Urbana-Champaign, IL, 1980.
[29] A. WEISER, *Local-mesh, local-order, adaptive finite element methods with a posteriori error estimation for elliptic partial differential equations*, Tech. Rep. 213, Dept. Computer Science, Yale Univ., New Haven, CT, 1981.
[30] L. C. WOODS, *The relaxation treatment of singular points in Poisson's equation*, Quart. J. Mech. Appl. Math., 6 (1953), pp. 163-185.
[31] H. YSERENTANT, *On the convergence of multilevel methods for strongly non-uniform families of grids and any number of smoothing steps per level*, Computing, 30 (1983), pp. 305-313.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 2, March 1987
© 1987 Society for Industrial and Applied Mathematics
009

A MODIFIED EQUATION APPROACH TO CONSTRUCTING FOURTH ORDER METHODS FOR ACOUSTIC WAVE PROPAGATION*

GREGORY R. SHUBIN† AND JOHN B. BELL†

Abstract. In this paper we use a modified equation analysis to develop formally fourth order accurate finite difference and pseudospectral methods for the one-dimensional wave equation. The difference scheme is constructed by performing a modified equation analysis of a centered, second-order conservative scheme to determine its dominant error term. Subtracting a centered discretization of this term from the scheme cancels the second order truncation errors. This technique yields a formally fourth order accurate explicit difference scheme that employs only three time levels. Similarly, the modified equation technique can be used to achieve fourth order time accuracy for the pseudospectral method with no increase in storage. The difference and pseudospectral schemes are fourth order convergent for constant coefficients even when a spatially singular forcing term is used for a source. Numerical results are given comparing the accuracy and efficiency of these methods for some model problems. Finally, we present a generalization of the fourth order finite difference scheme to two space dimensions.

Key words. acoustic wave propagation, finite difference methods, modified equation analysis

AMS(MOS) subject classifications. 65M05, 35L05, 76Q05

1. Introduction.

The most common numerical technique for solving the wave equation is to use second order accurate centered finite differences. A significant limitation of this method is the necessity to have about twenty points per (nominal) wavelength of the propagating pulse; when fewer points are used, oscillations caused by numerical dispersion mar the results. The use of formally higher order accurate interior schemes offers the promise of reducing the required number of grid points per wavelength, at a cost of increasing the computational work per grid point-time step.

One promising approach is the use of fourth order finite difference schemes (e.g. [1], [2]). The approach taken in [1] uses compact implicit difference approximations that require the solution of a tridiagonal linear system each time step. Recently, Brown [2] has used fourth order space differences and explicit second order time differences combined with Richardson extrapolation in time to obtain fourth order accuracy. He shows that fourth order schemes can provide significantly better accuracy than the standard second order scheme, even when the coefficients (material properties) are discontinuous. His results suggest that it is much more important to accurately model the phase velocity of the propagating wave than it is to accurately represent internal boundary conditions at material interfaces.

An alternate approach to higher order difference schemes for reducing the required number of grid points per wavelength is to calculate the spatial derivatives in the wave equation by Fourier transform techniques [3], [4], [5], while retaining second order centered time differences for the time derivative. The resulting pseudospectral (PS) method has been extensively developed for seismic modeling applications by Kosloff et al. [6]. In theory the PS method requires only 2 grid points per (highest frequency) wavelength, but in practice seems to require 4 or more. (There is considerable variability in the literature concerning the definition of "wavelength.") However, for the one-dimensional wave equation with variable coefficients, the PS method requires four discrete Fourier transforms per time step, which can be quite expensive. Whether the increased accuracy compensates for the expense will be investigated here.

* Received by the editors June 17, 1985, and in revised form October 1, 1985.
† Exxon Production Research Co., Houston, Texas 77001.

In this paper we construct a finite difference method for one-dimensional wave propagation that is fourth order accurate in space and time (when the medium is homogeneous). We also construct a fourth order accurate in time PS method. Our basic approach is to perform a modified equation analysis [7] of "underlying" second order schemes to identify the spatial and temporal truncation errors, and then to add a term to the second order schemes that cancels the second order truncation errors. The result is a formally fourth order accurate scheme. The modified equation technique uses a linear combination of derivatives of the truncation error expansion to express temporal truncation errors in terms of spatial errors. This allows us to obtain a scheme that is formally fourth order accurate in both space and time, but which uses only fourth order spatial differences or Fourier derivatives; fourth order time differences are not needed.

While there are some circumstances in which we want to model smoothly varying properties, the large bulk of wave equation modeling involves models whose properties are piecewise constant. Therefore, our basic strategy is to construct schemes that are formally fourth order accurate only in regions where the physical properties of the medium are constant. The schemes are constructed by starting with an "underlying" second order accurate, conservative scheme which effectively captures the behavior at material discontinuities. We then add a correction term that makes the schemes fourth order where the coefficients are constant. Since we cannot expect higher order accuracy where the coefficients are discontinuous, the extra complexity of a scheme that would be formally fourth order accurate for spatially varying coefficients is unjustified. Indeed, we have found that higher order corrections that account for spatially varying coefficients do not significantly improve the accuracy for problems with discontinuous properties.

Below, we review the simplest centered difference scheme for discretizing the one-dimensional wave equation. We also review the pseudospectral method. In § 2 we present the modified equation analysis and fourth order scheme development. Some results comparing the accuracy and efficiency of the finite difference and PS schemes are given in § 3. In § 4 we generalize the fourth order finite difference scheme to two space dimensions. In § 5 we conclude with some brief comments.

Mathematical formulation and scheme review. We are interested in solving

$$\begin{aligned} &p_{tt} - \rho c^2 \left(\frac{1}{\rho} p_x \right)_x - f = 0, \\ &p(x, t=0) = 0, \qquad p_t(x, t=0) = 0, \end{aligned} \tag{1.1}$$

which describes the propagation of the pressure perturbation p, where $\rho(x)$ is the density of the medium, $c(x)$ is the sound speed, and $f(x, t)$ represents a source. We consider only uniform grids with grid size $\Delta x = h$ and constant time step size Δt. Let p_j^n represent the approximate value of p at $x = jh$, $t = n\Delta t$. The following centered first difference operators will be useful now and in the development later:

$$D_x p_j^n = \frac{1}{h} (p_{j+1/2}^n - p_{j-1/2}^n),$$

$$D_t p_j^n = \frac{1}{\Delta t} (p_j^{n+1/2} - p_j^{n-1/2}).$$

Using this notation, the standard second order conservative difference scheme for (1.1)

is given by

$$\text{SCHEME FD2:}\quad D_t^2 p_j^n - (\rho c^2)_j \left[D_x\left(\frac{1}{\rho_j} D_x p_j^n\right)\right] - f_j^n = 0. \tag{1.2}$$

Here, $D_t^2 p_j^n$ is shorthand for $D_t D_t p_j^n$. This scheme has a stability limitation (for constant coefficients) of $\nu = c\Delta t/h \leqq 1$.

We next review the pseudospectral (PS) method. Let δ represent the operation of taking a discrete Fourier transform of p_j^n, $j = 1, \cdots, J$, multiplying each Fourier coefficient by its corresponding wave number, and taking an inverse Fourier transform; $(\delta p)_j^n$ therefore represents an approximation to p_x at $x = jh$, $t = n\Delta t$. The PS method is then given by

$$\text{SCHEME PS2:}\quad D_t^2 p_j^n - s_j^n - f_j^n = 0, \tag{1.3}$$

where

$$s_j^n = (\rho c^2)_j \left\{ \delta\left(\frac{1}{\rho_j} (\delta p)_j^n\right)\right\}_j^n.$$

This scheme requires two forward and two inverse discrete Fourier transforms, which are carried out using FFT's. While the scheme has a stability limitation (for constant coefficients) of $\nu = c\Delta t/h \leqq 2/\pi$, it has been found in practice that acceptable accuracy is obtained only when $\nu \leqq 0.2$ [6].

2. Development of the fourth order schemes. In this section we present the modified equation analysis that leads to the new schemes. As previously mentioned, we concentrate on the case of spatially constant material properties, but we include the source term. In this case equation (1.1) simplifies to

$$p_{tt} - c^2 p_{xx} - f = 0. \tag{2.1}$$

We first construct a fourth order difference scheme. The second order scheme (1.2) applied to (2.1) becomes

$$(D_t^2 - c^2 D_x^2) p_j^n - f_j^n = 0. \tag{2.2}$$

The modified equation for (2.2) is obtained by expanding all terms in Taylor series, and by then using the expanded equation itself (and derivatives of it) to eliminate all time terms except for the second derivative term. After performing these manipulations, and keeping only the second order error terms, we obtain

$$p_{tt} - c^2 p_{xx} - f - \frac{c^2 h^2}{12}((1-\nu^2) p_{xxxx}) + \frac{\Delta t^2}{12}(c^2 f_{xx} + f_{tt}) = 0 \tag{2.3}$$

where $\nu = c\Delta t/h$, the CFL number.

The new fourth order scheme is obtained by taking the standard centered scheme (1.2) and adding a correction term that cancels the error terms in (2.3). The resulting scheme is

$$\begin{aligned}\text{SCHEME FD4:}\quad & D_t^2 p_j^n - (\rho c^2)_j D_x\left(\frac{1}{\rho_j} D_x p_j^n\right) - f_j^n \\ & + \frac{c^2 h^2}{12}(1-\nu^2) D_x^4 p_j^n - \frac{\Delta t^2}{12}(c^2 D_x^2 + D_t^2) f_j^n = 0.\end{aligned} \tag{2.4}$$

A von Neumann analysis (see Appendix) shows that (2.4) possesses the same stability limit (for constant material properties) as the underlying second order scheme, namely $\nu \leqq 1$. We will also investigate the scheme obtained by including only the fourth order spatial correction

$$\text{(2.5)} \qquad \text{SCHEME FD24:} \quad D_t^2 p_j^n - (\rho c^2)_j D_x\left(\frac{1}{\rho_j} D_x p_j^n\right) - f_j^n + \frac{c^2 h^2}{12} D_x^4 p_j^n = 0.$$

We refer to this as the FD24 scheme since it is second order in time and fourth order in space. (Higher order source accuracy is unimportant with second order schemes.)

We use the same basic idea to obtain fourth order time accuracy with the PS method. Here, we ignore any spatial errors arising in the PS2 scheme (1.3) and simply use the modified equation technique to re-express temporal truncation errors as spatial errors and add a term to cancel them. The resulting scheme is

$$\text{(2.6)} \qquad \text{SCHEME PS4:} \quad D_t^2 p_j^n - s_j^n - f_j^n - \frac{\Delta t^2}{12}(c^4 \delta^4 p_j^n + (c^2 D_x^2 + D_t^2) f_j^n) = 0.$$

Here, δ^4 represents a fourth derivative obtained by one forward and one inverse FFT. This correction term could also be approximated much more inexpensively by $D_x^4 p_j^n$. The resulting scheme, referred to as PS4D, will also be included in the comparisons in § 3.

Corrections for varying material properties. It is straightforward to use the modified equation technique to obtain schemes that are higher order accurate for spatially varying coefficients. For example, if ρ is constant but c varies, the modified equation is given by

$$p_{tt} - c^2 p_{xx} - f - \frac{c^2}{12}(h^2 p_{xxxx} - \Delta t^2 (c^2 p_{xx})_{xx}) + \frac{\Delta t^2}{12}(c^2 f_{xx} + f_{tt}) = 0.$$

A fourth order accurate scheme is easily obtained by simply adding a correction term to cancel the errors. Indeed, we note that such a correction for the temporal part of the error (except for the forcing term) is naturally implemented in the PS4 scheme (2.6) by replacing $c^4 \delta^4 p_j^n$ by $c^2 \delta^2 s_j^n$. Since this correction is obtained at no additional cost, we shall actually use it in the PS4 and PS4D results quoted below.

A scheme that also accounts for spatially varying ρ can similarly be derived, but it is considerably more complicated.

Averaging of coefficients. When modeling problems with discontinuous coefficients it is important to properly interpret the coefficient values assigned to grid points. For the problems discussed here we view the grid point values of coefficients as representing the average values over a cell of size h centered at the grid point location. To obtain values of the velocity for each grid point we harmonically average over the cell volume, thereby ensuring the correct travel time for waves passing through the cell. The density is averaged arithmetically over each cell, which is the natural mass conserving averaging procedure. The values of $1/\rho$ required half-way between grid points in the FD schemes are obtained by harmonic averaging [2], [8] of the neighboring grid point values.

Source model. For seismic modeling applications we use a source function given by

$$\text{(2.7)} \qquad f(x, t) = \delta(x - x_0) R(t)$$

where $\delta(x-x_0)$ is a delta function and $R(t)$ is a smooth function giving the time variation of the source. Numerically, we model $\delta(x)$ as $1/h$ at a single grid point and zero elsewhere. To mimic seismic modeling we take the time function to be the third derivative Gaussian

$$-2\alpha^2 t(2\alpha^2 t^2-3)\, e^{-\alpha^2 t^2}$$

where $\alpha = \pi f_p$. This source produces a traveling pulse which is the second derivative of a Gaussian having its maximum Fourier amplitude at the frequency f_p. (In geophysics this type of pulse is referred to as a Ricker wavelet [9].) The computation begins at $t_0 = -4/\alpha$ so that $R(t)$ is very small relative to its peak value.

Grid size and time step selection. We will express our grid sizes in number of grid points per "nominal" wavelength. For this purpose, we define the nominal wavelength λ_n to be given by $c_{\min}/f_p$, where $c_{\min}$ is the smallest signal velocity in the problem, and f_p is the "peak frequency" defined above. We note that the amplitude spectrum (Fourier magnitude versus frequency) of the pulse decreases to about 3% of its peak value at the frequency $2.5^* f_p$; so that, roughly speaking, the number of grid points per "shortest" wavelength is about 0.4 times the number per nominal wavelength.

The choice of time step size is more problematic. For explicit finite difference schemes, the best results are obtained by taking the maximum allowable time step since the spatial and temporal truncation errors cancel one another. Indeed, in one dimension the exact solution can be obtained for homogeneous problems with $\nu = 1$. This latter phenomenon is an artifact of the one-dimensional problem, since in two dimensions the stability limit is $\nu \leqq \sqrt{2}/2$ and exact results are not obtained for the maximum Δt. To mimic this effect in one dimension, we will take $\text{CFL} \equiv \nu_{\max} = 0.6$ for all of the finite difference results reported later. This choice penalizes the finite difference results somewhat, but is more representative of the situation in higher dimensions.

For the pseudospectral method, the spatial and temporal errors add rather than cancel. On reasonably fine grids, the total error is usually dominated by the time truncation error arising from the centered time difference. It is therefore advantageous to take rather small time steps, and we have found that the recommended value for PS2 of $\text{CFL} = 0.2$ seems about right. We will, however, investigate whether the increased temporal accuracy of PS4 allows for larger time steps.

3. Results. In this section we compare the performance of the following methods:

FD2: the second order difference scheme (1.2),

FD4: the fourth order difference scheme (2.4),

FD24: scheme FD4 with time correction omitted, (2.5),

PS2: pseudospectral method (1.3),

PS4: fourth order pseudospectral method (2.6), with modification described in text,

PS4D: PS4 with fourth derivative approximated by finite differences instead of FFTs.

We apply these methods to two problems (see Fig. 1); propagation in a homogeneous medium and a problem with a single discontinuity with an impedance contrast of 2. (The impedance contrast $(\rho_1 c_1/\rho_2 c_2)$ determines the magnitude of reflected and transmitted waves.) In both cases the computational domain is $0 \leqq x \leqq 1$, the minimum value of c is 1, and the peak frequency f_p is 20. These data yield a minimum nominal wavelength of 0.05. (When scaled to the geophysical regime this corresponds to a 50 Hertz Ricker wavelet in a medium 2000 feet long with propagation velocity 5000 ft/sec.) We take the number of grid intervals to be 128, 256, and 512, corresponding

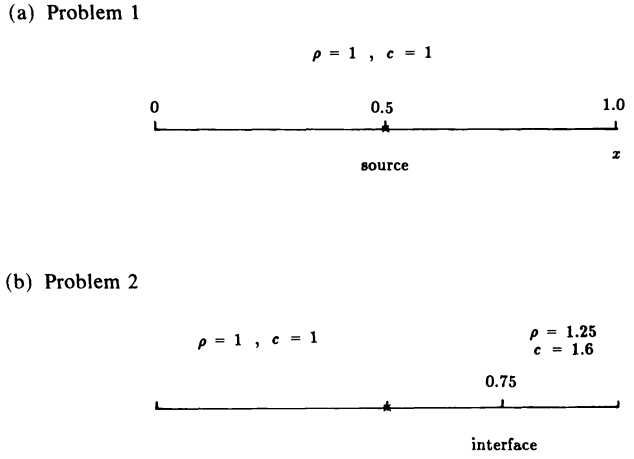


FIG. 1. *Model problems.*

to 6.4, 12.8, and 25.6 points per nominal wavelength, respectively. (None of the methods yields acceptable results with 3.2 points per wavelength.) The source is always located at $x = 0.5$. When not explicitly mentioned, the time step is chosen as discussed before, namely CFL of 0.2 for the PS methods and 0.6 for the FD methods. The final time is chosen so that no significant disturbances reach the boundaries. For all cases we compute an "exact" solution using a very fine grid (16,384 points), and then compute L^2 errors between this solution and those for the coarser grids. (Of course, for homogeneous media, an exact solution could also be analytically constructed.) For the error computation we use a discrete, relative L^2 error defined by

$$\left(\frac{1}{J}\sum_{j=1}^{J} e_j^2\right)^{1/2} \Big/ \left(\frac{1}{J_f}\sum_{j=1}^{J_f} u_j^2\right)^{1/2}$$

where e is the pointwise error and u is the fine grid solution. Convergence rates are also computed. We note that the L^2 error measure is more sensitive to phase errors than to amplitude errors. For this reason, visually acceptable results may be obtained even when the error measure is relatively large. We therefore also show plots of computed versus exact results for $0.5 \leqq x \leqq 1.0$ (the results to the left of the source corresponding to propagation in a homogeneous medium are redundant).

Problem 1: *Homogeneous medium.* The problem is shown in Fig. 1(a). Plots of the numerical solutions as functions of x at some fixed final time (0.47) are shown in Fig. 2. The numerical results are given by the dashed line and the "exact" solution by the solid line. The L^2 errors are given in Table 1.

In this table, the rate columns indicate the numerical convergence rate between the two grid sizes on either side.

Acceptable results are obtained with 6.4 ppw (points per nominal wavelength) with the PS methods, 12.8 ppw with FD4, and 25.6 ppw with the second order FD schemes. (The errors with PS2 and PS4 with 3.2 ppw are 0.630 and 0.252, with the latter perhaps providing marginally acceptable accuracy.) We observe that the fourth order methods seem truly fourth order convergent in a homogeneous medium. Also, the fourth order PS methods are extremely accurate. Furthermore, the finite difference fourth order correction in PS4D, while not as good as PS4, still yields much higher accuracy than PS2 at very little additional cost. The fourth order time correction in FD4 yields significantly greater accuracy than the second order time method FD24.

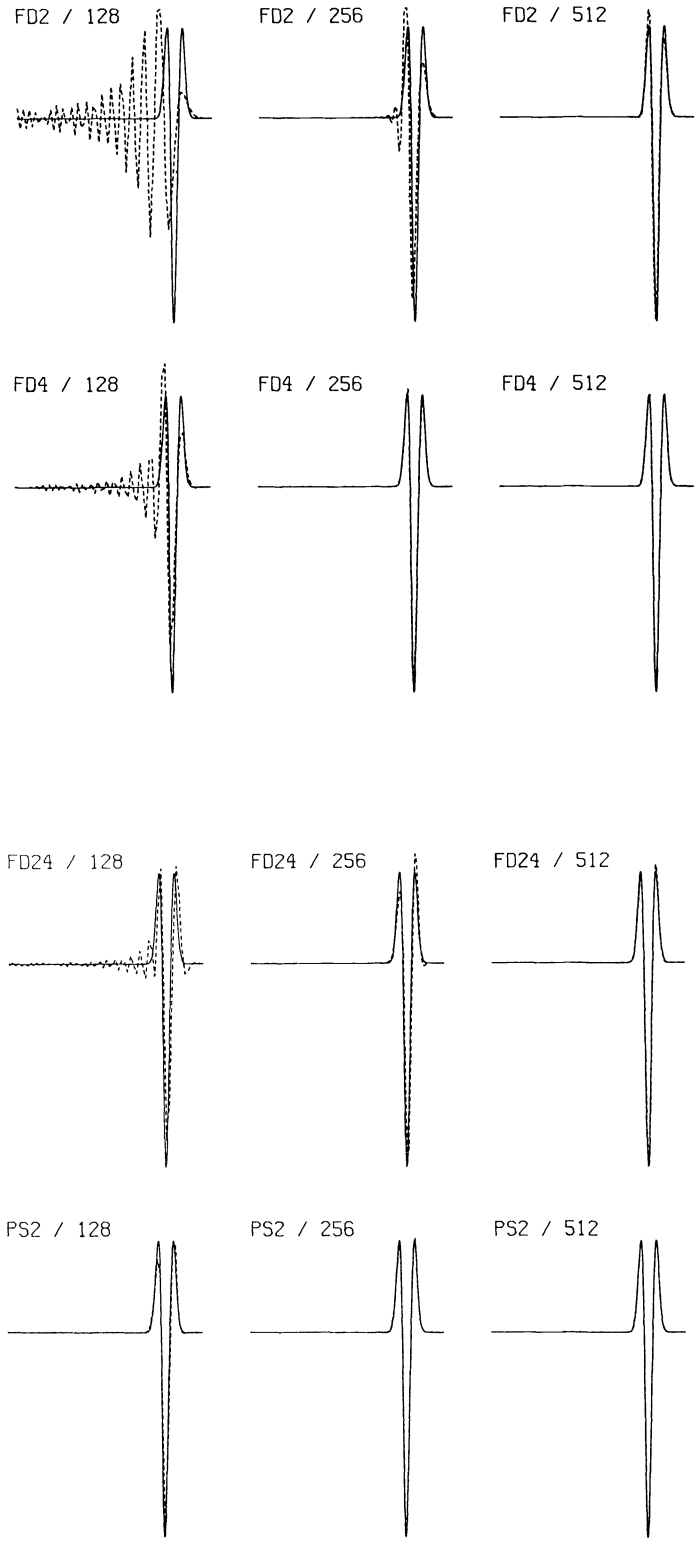


FIG. 2. *Results for Problem* 1.

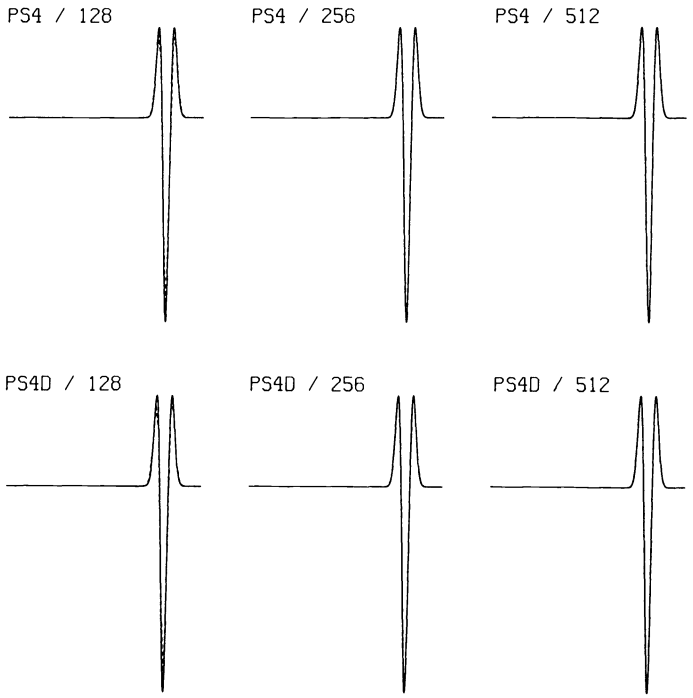


FIG. 2. *Results for Problem* 1 (*continued*).

TABLE 1
L^2 errors for Problem 1.

Scheme	Grid points (points per nominal wavelength)				
	128 (6.4)	Convg. rate	256 (12.8)	Convg. rate	512 (25.6)
FD2	1.45E0	1.09	6.80E−1	1.89	1.84E−1
FD4	6.32E−1	3.35	6.18E−2	3.97	3.93E−3
FD24	5.76E−1	0.87	3.15E−1	1.71	9.61E−2
PS2	1.82E−1	2.00	4.56E−2	2.00	1.14E−2
PS4	1.19E−3	4.00	7.43E−5	3.90	5.00E−6
PS4D	3.87E−2	3.88	2.62E−3	3.92	1.73E−5

Problem 2. *Single layer with impedance jump of* 2. The problem is shown in Fig. 1(b), the computational results at time 0.38 are displayed in Fig. 3, and the errors are given in Table 2.

We note that although we would expect the convergence rate to drop to first order due to the discontinuous coefficients, FD2 and FD4 appear to be second order convergent. Subsequent grid refinements show that the FD4 convergence deteriorates to first order; however, FD2 continues to appear second order convergent. To understand this phenomenon we plot in Fig. 4 the pointwise errors for FD2 and FD4 with 1024 grid points. (Note that we have plotted the entire solution on (0, 1) rather than the right half of the solution on (0.5, 1).) For FD4 the dominant part of the error is a phase shift occurring at the interface; the pulse propagating to the left is modeled very

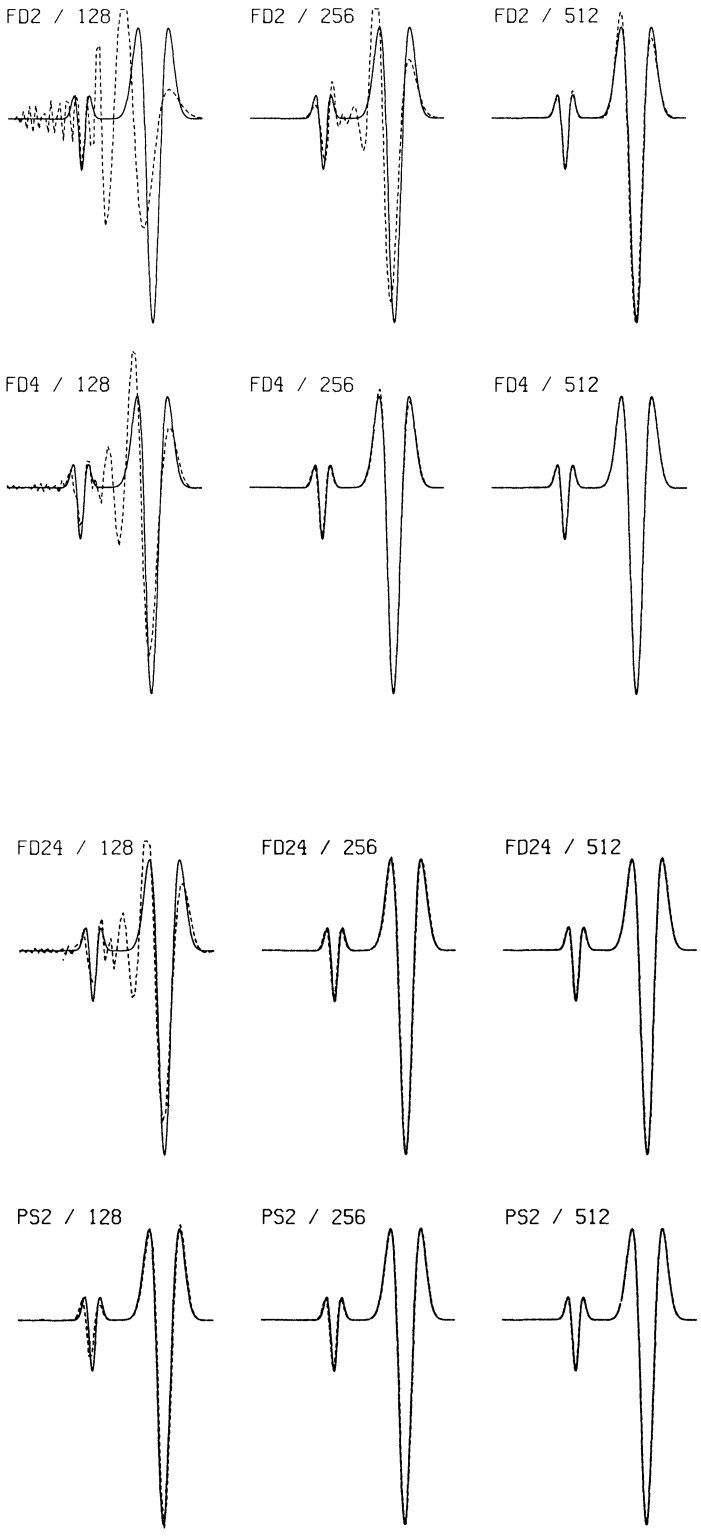


FIG. 3. *Results for Problem* 2.

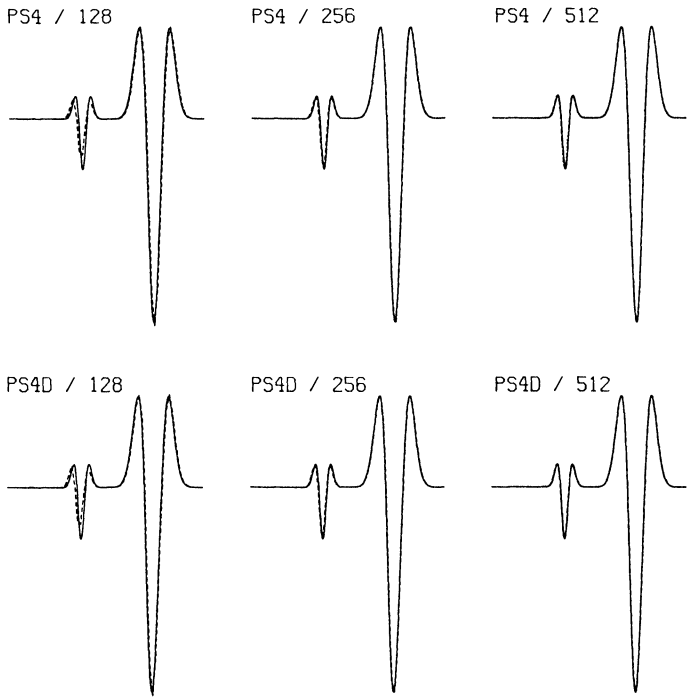


FIG. 3. *Results for Problem* 2 (*continued*).

TABLE 2
L^2 errors for Problem 2.

Scheme	Grid points (points per nominal wavelength) 128 (6.4)	Convg. rate	256 (12.8)	Convg. rate	512 (25.6)
FD2	1.30E0	1.06	6.24E−1	1.94	1.62E−1
FD4	5.80E−1	3.43	5.38E−2	2.01	1.33E−2
FD24	3.84E−1	2.15	8.65E−2	1.21	3.75E−2
PS2	1.05E−1	1.44	3.87E−2	1.28	1.60E−2
PS4	7.47E−2	1.36	2.91E−2	1.11	1.35E−2
PS4D	7.71E−2	1.39	2.94E−2	1.12	1.36E−2

accurately. The error for FD2 is dominated by dispersive errors in the propagation of the pulse through homogeneous subregions. Errors in propagating the wave through homogeneous regions exhibit the inherent second order convergence of the scheme; the phase shift at the interface is first order but much smaller than the propagation error on the grids considered here. Consequently, extremely fine grids would be needed before FD2 began to exhibit first order convergence.

For this problem, the PS methods provide acceptable accuracy (5-10%) with 6.4 ppw. For the PS methods the error is dominated by inaccuracies at the interface; hence, they appear to be first order convergent throughout. The fourth order PS results are essentially indistinguishable from the PS2 results. Finally, we note that the fourth order time correction in FD4 offers much less advantage over FD24 than for Problem 1.

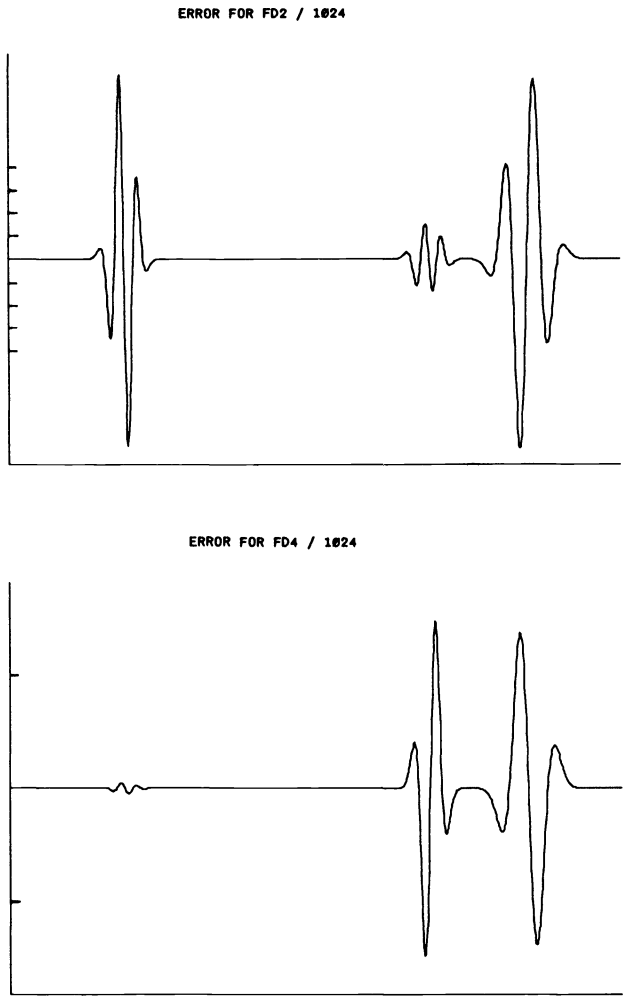


FIG. 4. *Pointwise errors for* FD2 *and* FD4.

Time step size for the PS methods. Next we investigate whether the increased formal time accuracy of the PS4 scheme allows for larger time steps. We consider problems 1 and 2 with time step sizes corresponding to CFL values of 0.2 (as before), 0.4, and 0.6. The relative L^2 errors are displayed in Table 3; results on the 128 grid for Problem 2 are shown in Fig. 5.

For problem 1, the errors decrease roughly like Δt^2 for the PS2 methods, and better than Δt^3 for the PS4 method (there is still some spatial error at 512 grid points). Turning to the results for Problem 2, we see that the PS4 results remain acceptable at fairly large time steps, while the PS2 results degrade markedly. We note that the PS4D scheme yields results quite similar to PS4 for problem 2 at the larger time steps.

Efficiency. We confine ourselves to comparing the efficiency of the FD4 and PS2 methods. Of course, the computational efficiency of each method is independent of the structure of the coefficients ρ and c, provided this structure is nontrivial. In our comparisons we assume that these methods can be implemented in efficient, vectorized code and therefore compare efficiency based on approximate operation counts. The actual computational times obtained on a CRAY-1S are in very nearly the same ratio

TABLE 3
L^2 errors for PS2, PS4 *as function of* CFL.

Scheme	CFL	Problem 1			Problem 2		
		128 points	256 points	512 points	128 points	256 points	512 points
PS2	0.2	1.82E−1	4.56E−2	1.14E−2	1.05E−1	3.87E−2	1.60E−2
	0.4	6.64E−1	1.82E−1	4.56E−2	2.52E−1	7.55E−2	2.47E−2
	0.6	1.13E0	4.00E−1	1.03E−1	5.06E−1	1.42E−1	4.09E−2
PS4	0.2	1.19E−3	7.43E−5	4.98E−6	7.47E−2	2.91E−2	1.35E−2
	0.4	1.31E−2	7.61E−4	5.18E−5	7.49E−2	2.91E−2	1.35E−2
	0.6	7.22E−2	3.77E−3	2.29E−4	7.52E−2	2.90E−2	1.35E−2

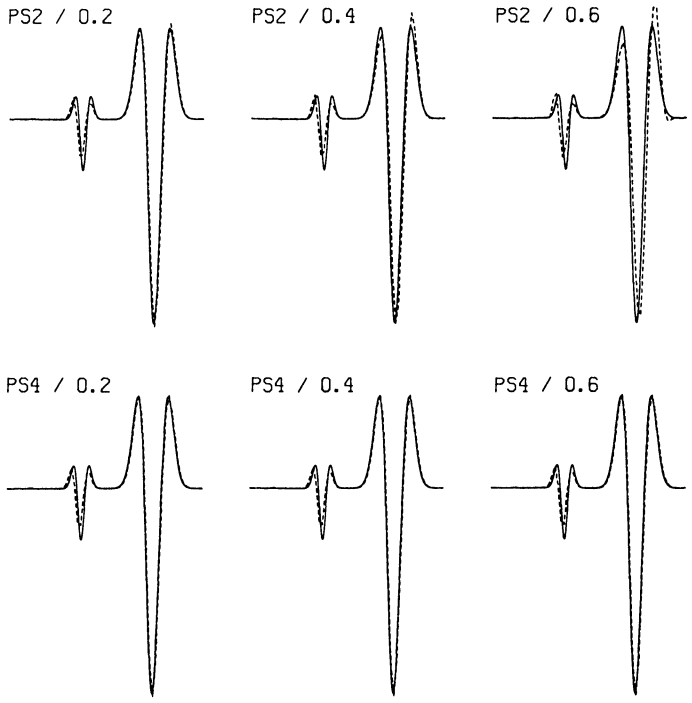


FIG. 5. *Results for Problem* 2, PS *methods, various* CFL *values.*

as the ratio of operations. (We assume that the necessary coefficients and averages of coefficients are stored and not recomputed each time step.)

Recalling that J is the number of spatial grid points, the FD4 algorithm requires $21\,J$ operations per time step. The PS2 method requires about $6J + 16J \log_2 J$ operations per step. Here, the first term represents the temporal advancement of the solution, and the second the evaluation of 4 FFTs taking 4 real operations per grid point. On a $J = 256$ grid, with a time step being 3 times larger for the FD4 method, the FD4 technique would be about 19 times faster than PS2. If the PS2 results with $J = 128$ are contrasted with FD4 with $J = 256$, the FD4 results are still about 4 times more economical. Even if comparable time steps are taken with the two methods (FD4/256, PS2/128), the FD4 method would be about 1.4 times faster. Such comparable times

steps produce acceptable results with the PS4 and PS4D schemes, with the latter being only slightly more expensive than PS2.

4. Two space dimensions. The comparisons and discussion of the previous section suggest that the fourth order finite difference method is the most efficient method for one-dimensional problems having nontrivial structure. A detailed comparison of whether finite difference or pseudospectral methods would be most effective for two-dimensional modeling is beyond the scope of this paper. While the pseudospectral method offers the possibility of reducing the required memory for a given calculation, it involves more computational work. In addition, the periodicity assumption underlying the pseudospectral method makes treatment of boundary conditions quite difficult. For this reason we focus our discussion in this section on the extension of FD4 to two space dimensions.

The two-dimensional analogue of (1.1) is given by

$$p_{tt} - \rho c^2\left[\left(\frac{1}{\rho} p_x\right)_x + \left(\frac{1}{\rho} p_z\right)_z\right] = f, \tag{4.1}$$

which we approximate using a standard conservative second order scheme given by

$$D_t^2 p_{jk}^n - (\rho c^2)_{jk}\left[D_x\left(\frac{1}{\rho_{jk}} D_x p_{jk}^n\right) + D_z\left(\frac{1}{\rho_{jk}} D_z p_{jk}^n\right)\right] = f_{jk}^n \tag{4.2}$$

where D_z is the analog of D_x defined in § 1. (We assume $\Delta x = \Delta z = h$.) As before we compute the correction term only for constant coefficients. In this case (4.1) collapses to

$$p_{tt} - c^2(p_{xx} + p_{zz}) = f. \tag{4.3}$$

We now compute the modified equation for the difference approximation (4.2) applied to (4.3) to obtain, to second order,

$$\begin{aligned} p_{tt} - c^2(p_{xx} + p_{zz}) - f - \frac{c^2 h^2}{12}((1-\nu^2)(p_{xxxx} + p_{zzzz}) - 2\nu^2 p_{xxzz}) \\ + \frac{\Delta t^2}{12}(f_{tt} + c^2(f_{xx} + f_{zz})) = 0. \end{aligned} \tag{4.4}$$

The two-dimensional analog of FD4 obtained by discretizing the correction term in (4.4) is then given by

$$\begin{aligned} D_t^2 p_{jk}^n - (\rho c^2)_{jk}\left[D_x\left(\frac{1}{\rho_{jk}} D_x p_{jk}^n\right) + D_z\left(\frac{1}{\rho_{jk}} D_z p_{jk}^n\right)\right] - f_{jk}^n \\ + \frac{c^2 h^2}{12}[(1-\nu^2)(D_x^4 + D_z^4) p_{jk}^n - 2\nu^2 D_x^2 D_z^2 p_{jk}^n] \\ - \frac{\Delta t^2}{12}(D_t^2 + c^2(D_x^2 + D_z^2)) f_{jk}^n = 0. \end{aligned} \tag{4.5}$$

As in the one-dimensional case the fourth order scheme satisfies the same stability criterion as the second order scheme; viz., $\nu \leqq \sqrt{2}/2$ (see Appendix).

To illustrate the improvement obtained with the fourth order scheme we consider a simple example arising in geophysics. The physical problem is depicted in Fig. 6. We have chosen a problem with plane layers so that a very accurate solution for comparison purposes can be obtained using Fourier analysis techniques. In spite of

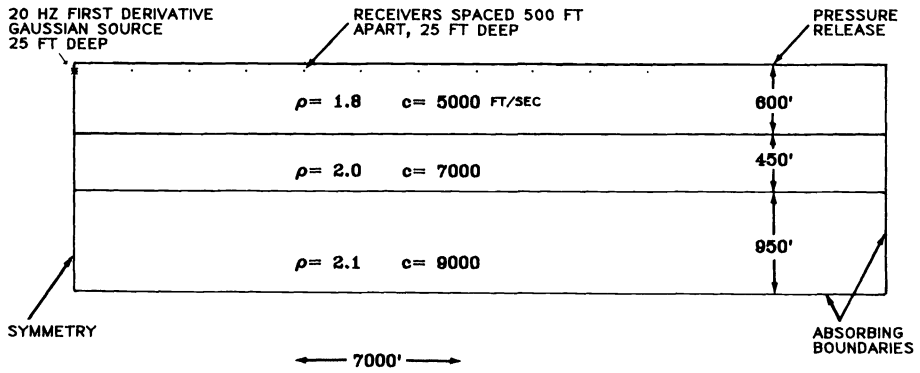


FIG. 6. *Acoustic model.*

its simplicity, this model provides a difficult test for modeling codes due to the presence of a free surface and due to critically refracted waves arising from oblique interactions at the material interfaces. For completeness we note that we treat the absorbing boundaries using conditions developed by Engquist and Majda [10]; however, reflections from the absorbing boundaries play no role in the results we present.

For geophysical modeling one does not measure the solution over space at a particular time; instead one measures the time histories at selected locations (receivers). In Fig. 7 we show time histories obtained using the second order scheme with 6.25 ft grid blocks. Noticeable errors are beginning to appear at the receivers located farthest from the source. In Fig. 8 we repeat the same experiment using the fourth order scheme (4.5) with 12.5 ft grid blocks. Although we are using only half the grid in each direction, the resolution of the fourth order scheme is somewhat better. In addition to reducing the required grid by a factor of 4, we have also reduced the computational time required by a factor of 3. Although there is considerable variability in the savings of time and

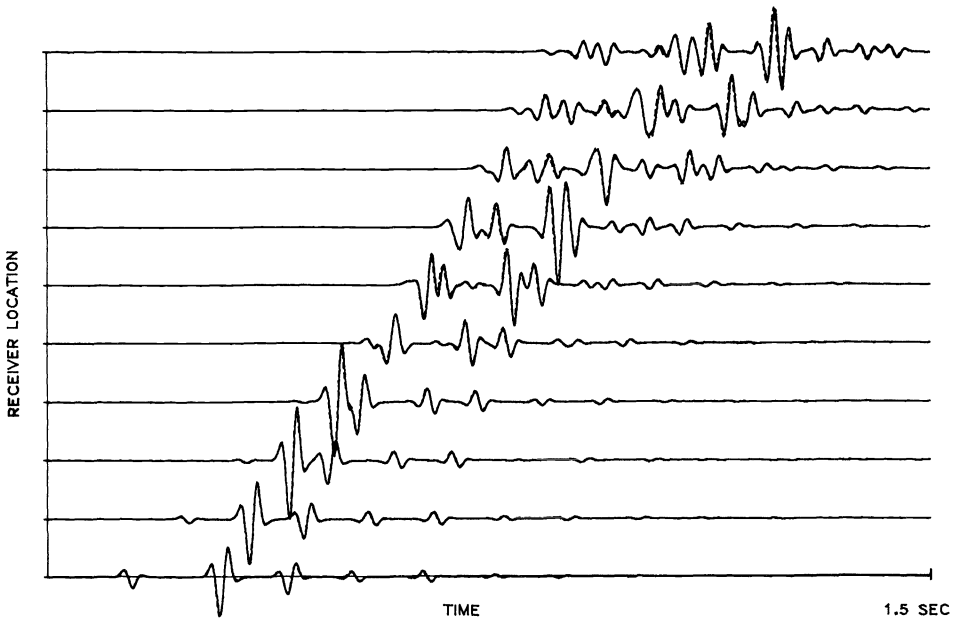


FIG. 7. *Second order scheme, 6.25 ft grid, (solid) vs. exact (dashed).*

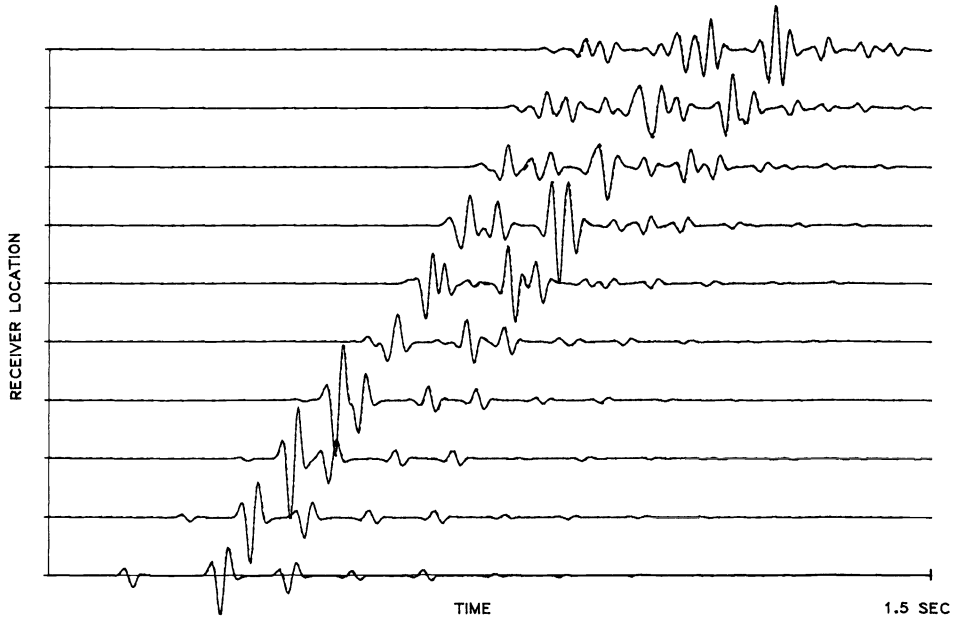


FIG. 8. *Fourth order scheme, 12.5 ft grid, (solid) vs. exact (dashed).*

memory for comparable accuracy (depending on model complexity and types of wave phenomena being modeled) the fourth order method does produce a considerable improvement over the second order method.

5. Concluding remarks. We have used a modified equation analysis to develop formally fourth order accurate finite difference and pseudospectral schemes for wave propagation in one space dimension. We have focused on developing schemes that attain this accuracy only when coefficients are constant. The resulting schemes are truly fourth order convergent, even when a delta function (in space) forcing function is used.

Even for practical problems with discontinuous coefficients, the fourth order difference scheme FD4 performs considerably better than the standard second order scheme. The fourth order time correction coming from the modified equation analysis generally improves the results compared to the second order time, fourth order space scheme FD24.

While the fourth order pseudospectral method PS4 is extremely accurate for homogeneous problems, the method offers less advantage over the second order method PS2 for problems with discontinuous coefficients, at least when a small time step is taken. However, PS4 allows larger time steps to be taken than PS2 with little degradation of accuracy. A modification of PS4, PS4D gives a similar advantage with little increased cost over PS2.

We have also illustrated, in the context of finite difference methods, that the modified equation technique can be used to derive higher order schemes in several space dimensions. The technique yields a stable scheme (sharing the stability limit of the second order scheme) that can offer considerable reduction in computer time and memory needed to obtain a given level of accuracy.

In comparing the relative merits of finite difference and pseudospectral methods, it is necessary to take into account both the accuracy obtained with a given number of grid points, and the computational cost. In general, the pseudospectral methods

provide an acceptable level of accuracy at about 6 points (or slightly more) per nominal wavelength, while the fourth order difference method FD4 requires 12. Even so, the cost of computing the FFTs and the need to take a smaller time step seems to put the PS methods at a slight disadvantage.

We note that in multiple dimensions, different considerations govern the efficiency tradeoffs between finite difference and pseudospectral methods than in 1D. Additionally, effective techniques for approximating free surfaces ($p=0$ boundaries) and absorbing or radiation conditions become crucial. Therefore, the efficiency tradeoffs for one dimensional wave propagation given here cannot be simply extrapolated to compare the promise of PS and higher order difference schemes for practical multidimensional problems.

Appendix: von Neumann analysis of scheme FD4. In this appendix we apply a standard von Neumann analysis [11] to show that the CFL condition $\nu = c\Delta t/h \leqq 1$ is necessary for stability of the one-dimensional fourth order finite difference scheme FD4 given in (2.4), and that $\nu \leqq \sqrt{2}/2$ is a necessary stability condition for the two-dimensional scheme (4.5).

We proceed first with the 2D scheme. Assuming constant coefficients and no forcing term ($f=0$), we substitute $p_{jk}^n = \lambda^n e^{i(j\theta_1 + k\theta_2)}$ into (4.5) to obtain

$$\lambda^2 + (B-2)\lambda + 1 = 0 \tag{A.1}$$

where

$$B(\mu, \tau_1, \tau_2) = 4\mu\left[\tau_1 + \tau_2 + \frac{1-\mu}{3}(\tau_1^2 + \tau_2^2) - \frac{2\mu}{3}\tau_1\tau_2\right] \tag{A.2}$$

and $\mu = \nu^2$, $\tau_1 = \sin^2(\theta_1/2)$, $\tau_2 = \sin^2(\theta_2/2)$. Since the product of the roots of (A.1) is 1, for stability we want the discriminant of (A.1) to be nonpositive, or

$$0 \leqq B \leqq 4, \tag{A.3}$$

for $0 \leqq \tau_1, \tau_2 \leqq 1$. We first observe that

$$B(\mu = 0, \tau_1, \tau_2) = 0, \qquad B(\mu = \tfrac{1}{2}, \tau_1, \tau_2) = 2[\tau_1 + \tau_2 + \tfrac{1}{6}(\tau_1 - \tau_2)^2],$$

and the maximum of the latter expression occurs at $\tau_1 = \tau_2 = 1$ where (A.3) is satisfied. It remains to show that $\partial B/\partial\mu$ is positive for $0 < \mu < \frac{1}{2}$. By direct computation

$$\frac{\partial B}{\partial \mu} = -\frac{4}{3}[2\mu(\tau_1 + \tau_2)^2 - \tau_2(\tau_2 + 3) - \tau_1(\tau_1 + 3)]$$

and the minimum occurs at $\mu = 1/2$ where the expression is positive.

For the one-dimensional problem we take $\theta_2 = 0$ and obtain

$$B(\mu = 0, \tau_1) = 0, \qquad B(\mu = 1, \tau_1) = 4\tau_1,$$

which satisfies (A.3). Again by direct computation,

$$\frac{\partial B}{\partial \mu} = 4\tau_1\left[1 + \frac{1-2\mu}{3}\tau_1\right]$$

which is positive.

Acknowledgments. The authors are indebted to A. Ngo for programming assistance, and to L. J. Baker and C. A. Rendleman for discussions that aided the development of this paper. We also thank C. A. Rendleman for providing the Fourier solution for the two-dimensional problem. Finally, we thank an anonymous reviewer for a number of constructive suggestions that greatly improved the paper.

REFERENCES

[1] M. CIMENT AND S. H. LEVENTHAL, *Higher order compact implicit schemes for the wave equation*, Math. Comput., 29 (1975), pp. 985-994.

[2] D. L. BROWN, *A note on the numerical solution of the wave equation with piecewise smooth coefficients*, Math. Comput., 42 (1984), pp. 369-391.

[3] H. O. KREISS AND J. OLIGER, *Comparison of accurate methods for the integration of hyperbolic equations*, Tellus, 24 (1972), pp. 199-215.

[4] B. FORNBERG, *On a Fourier method for the integration of hyperbolic equations*, SIAM J. Numer. Anal., 12 (1975), pp. 509-528.

[5] J. GAZDAG, *Modeling of the acoustic wave equation with transform methods*, Geophysics, 48 (1981), pp. 854-859.

[6] D. D. KOSLOFF AND E. BAYSAL, *Forward modeling by a Fourier method*, Geophysics, 47 (1982), pp. 1402-1412.

[7] R. F. WARMING AND B. J. HYETT, *The modified equation approach to the stability and accuracy analysis of finite-difference methods*, J. Comput. Phys., 14 (1974).

[8] A. A. SAMARSKII AND A. N. TIKHONOV, *Homogeneous Difference Schemes*, Zh. Vychisl. Mat. i Mat. Fiz., 1 (1961), pp. 5-63.

[9] N. RICKER, *The form and laws of propagation of seismic wavelets*, Geophysics, 18 (1953), pp. 10-40.

[10] B. ENGQUIST AND A. MAJDA, *Absorbing boundary conditions for the numerical simulation of waves*, Math. Comput., 31 (1977), pp. 629-651.

[11] E. ISAACSON AND H. B. KELLER, *Analysis of Numerical Methods*, John Wiley, New York, 1966.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 2, March 1987
© 1987 Society for Industrial and Applied Mathematics
010

STOCHASTIC FILTERING OF LINEAR SHALLOW WATER WAVE PROCESSES*

W. P. BUDGELL†

Abstract. An approach for the optimal combination of observations with dynamic models of the one-dimensional linearized shallow water equations has been developed. The dynamic-stochastic model includes the effects of errors in boundary conditions and momentum balances. The model, based on an incremental covariance formulation of the Kalman filter equations, describes the time-evolution of the spatially dependent error covariance field. Thus, error bounds can be placed on model predictions. The dynamic-stochastic model is applied to stochastic boundary forcing of a linearly varying channel and stochastic wind stress forcing of a one-dimensional closed channel and the results are compared to analytical solutions.

Key words. stochastic filtering, incremental covariance filter, optimal estimation, shallow water equations, dynamic-stochastic model, square root factorization

AMS(MOS) subject classifications. 60G35, 60H15, 62L12, 62M20, 65C20, 76B15

1. Introduction.

In the modelling of long wave processes, such as storm surges and tides, in coastal waters one can adopt deterministic or stochastic modelling approaches. The deterministic models [13]-[15], [21] include analytical or numerical solutions to hydrodynamic equations, whereas the stochastic models consist of impulse response methods [27], and other transfer function techniques [6]. Empirical tidal models such as the harmonic representation and response function model [25] rely upon the deterministic character of tidal time series for their structure. The deterministic analytical and numerical models possess no means by which measured data can be incorporated directly into the prediction process. Although the harmonic and Munk and Cartwright [25] response function models are derived directly from field data, these models cannot adapt to new data except through the inclusion of a stationary autogressive (Wiener) filter applied to the prediction residuals. The adaptive feature is desirable for real-time forecasting.

Even if the tidal signal can be described perfectly by existing models, meteorological forcing of coastal waters will often induce a significant response in the sea level and velocity fields. Since the prescribed wind stress and atmospheric pressure field used to drive most models are often subject to substantial error, the whole modelling process may be viewed as being stochastic in nature.

The stochastic impulse response and transfer function models, while accurate at a single point measurement location or at a small number of locations, are incapable of providing predictions over the whole spatial domain. The deterministic approaches, in particular the numerical models, tend to produce a more realistic picture of the spatially-varying, or distributed, nature of long wave propagation.

It can be seen that what is desirable is an approach that would combine the best features of the empirical and deterministic approaches to shallow water wave modelling. Stochastic filters, such as those described by Kalman and Bucy [18], have the capability of updating model predictions with new observations as they become available. In addition, it is possible to construct such filters around mechanistic models in order to preserve the correct spatial relationships between variables.

* Received by the editors March 4, 1985, and in final form October 15, 1985.
† Institute of Ocean Sciences, Sidney, British Columbia, Canada V8L 4B2.

The Kalman filter permits the segregation of a set of time series observations into deterministic, system noise and measurement noise components. The deterministic component is a slowly varying "mean" value about which perturbations of shorter time scales are taking place. The system noise component includes the effects of short-term variability of the natural system (e.g., turbulence) as well as errors in the underlying mechanistic model attributable to neglected nonlinearities, errors in boundary conditions, spatial averaging, etc. The measurement noise component incorporates all the uncertainty associated with the sampling process.

The Kalman filter approach and similar techniques have been successfully applied in a variety of fields. Applications of Kalman filter theory in such areas as aerospace, power systems and process control, have generated a considerable body of knowledge related to filter design and implementation [23]. Applications of Kalman filter techniques to air pollution problems [3], [9], [11], [20], advection-dispersion processes in estuaries [4], [8] and the prediction of oceanic density distributions [19] are relevant to the long wave prediction problem. While the Kalman filter technique is gaining acceptance in several areas of water resources [7], it has yet to be applied to the problem of shallow water wave propagation in coastal waters.

In this investigation, stochastic filters derived from the Kalman filter equations are constructed around the one-dimensional shallow water equations. The results of the study are presented in two parts. In the first part described herein, a dynamic-stochastic model based upon the linearized one-dimensional shallow water equations is described and is tested against analytical solutions for long wave propagation in channels with idealized topography. In the second part described in [5], the modelling approach is extended to nonlinear dynamics in branched channels.

2. Linearized shallow water equations. While the nonlinear, two-dimensional form of the shallow water equations is a more physically realistic description of long-wave dynamics in estuaries and coastal waters, the linearized one-dimensional equations will be employed here. This simple representation serves to illustrate most of the features of the combined dynamic-stochastic modelling approach and permits comparison with analytical results. Extensions to nonlinear dynamics and two-dimensionality are currently being addressed and will be described in forthcoming publications.

The linearized equations for long wave propagation in an open channel may be stated as follows:

$$\frac{\partial U}{\partial t}+g\frac{\partial \eta}{\partial x}+\frac{\lambda}{h}U=F(x,t), \tag{1}$$

$$b\frac{\partial \eta}{\partial t}+\frac{\partial (bhU)}{\partial x}=0, \tag{2}$$

where

U	is the cross-sectionally averaged velocity,
η	is the surface elevation,
x	is the distance along channel,
t	is time,
g	is gravitational acceleration,
λ	is the friction factor,
$h=h(x)$	is the time averaged depth,
$b=b(x)$	is the channel width,
$F(x,t)$	is a forcing term (e.g., a function of wind stress and/or atmospheric pressure).

The boundary conditions consist of open ocean $\eta(L, t) = \eta_0(t)$, upstream river discharge $U(0, t) = U_r(t)$, or solid boundary $U(0, t)$ or $U(L, t) = 0$, conditions.

Initial conditions consist of the prescribed velocity and elevation fields at $t = 0$:

$$U(x, 0) = U_0(x); \ \eta(x, 0) = \eta_0(x).$$

In order to solve the equations for channels of arbitrary geometry, the following four-point, weighted finite difference scheme is employed:

$$f(x, t) \simeq \frac{(1-\theta)}{2}[f_i^k + f_{i+1}^k] + \frac{\theta}{2}[f_i^{k+1} + f_{i+1}^{k+1}],$$

$$\frac{\partial f(x, t)}{\partial x} \simeq \frac{(1-\theta)}{\Delta x}[f_{i+1}^k - f_i^k] + \frac{\theta}{\Delta x}[f_{i+1}^{k+1} - f_i^{k+1}], \tag{3}$$

$$\frac{\partial f(x, t)}{\partial t} \simeq \frac{1}{2\Delta t}[f_i^{k+1} + f_{i+1}^{k+1}] - \frac{1}{2\Delta t}[f_i^k + f_{i+1}^k],$$

where $f(x, t)$ is some function continuous in space and time, θ is a weighting factor $0 \leqq \theta \leqq 1$, i and k are distance and time increments, respectively, Δx is the grid spacing, Δt is the time step. The scheme is stable for $1 \geqq \theta \geqq 0.5$. For $0 \leqq \theta < 0.5$ the Courant stability criterion $\Delta t \leqq \Delta x / \sqrt{gh}$ must be applied. For most applications a value of $\theta = 0.55$ has been found to produce accurate solutions while minimizing numerical noise [10].

Applying (3) to (1) and (2) and combining with boundary conditions, one obtains a set of $2n$ equations, where n is the number of grid points, which may be expressed in matrix form as

$$\mathbf{A}\mathbf{y}(k+1) = \mathbf{B}\mathbf{y}(k) + \mathbf{G}\mathbf{u}(k+1) \tag{4}$$

where **A** and **B** are coefficient matrices derived from the finite difference operations, $\mathbf{y}(k)$ is the solution vector of length $2n$ containing the discretized u and η values at time step k, $\mathbf{u}(k+1)$ is a vector of length p containing the boundary conditions and forcing functions. The $2n \times p$ matrix **G** prescribes the equations associated with the elements of $\mathbf{u}(k+1)$. Standard banded matrix decomposition algorithms can be used to solve (4).

3. The effect of model uncertainty. The numerical model described by (1)-(4) is a purely deterministic one. The initial, boundary and governing equations are assumed to be known with certainty. However, the prescribed initial conditions will be subject to error, the boundary conditions such as perfect reflection are simplifications of the true physical situation, the nonlinear terms are neglected, two- and three-dimensional effects are ignored and the prescribed atmospheric forcing function is uncertain in the momentum equation. These errors may be described as follows.

Initial error:

$$U(x, 0) = \hat{U}(x, 0) + U'(x, 0), \qquad \eta(x, 0) = \hat{\eta}(x, 0) + \eta'(x, 0); \tag{5}$$

Boundary error:

$$\eta(L, t) = \hat{\eta}(L, t) + \eta'(L, t), \qquad U(0, t) = \hat{U}(0, t) + U'(0, t); \tag{6}$$

Uncertain forcing:

$$F(x, t) = \hat{F}(x, t) + F'(x, t) \tag{7}$$

where the caret ($\hat{}$) denotes the prescribed value and the prime ($'$) denotes error in the prescribed value. The boundary error terms $\eta'(L, t)$ and $U'(0, t)$ in (6) contain the errors associated with the prescription of the equations (model error) as well as the errors in the actual specified values. Similarly, the forcing error term $F'(x, t)$ in (7) embodies all the errors associated with the momentum equation (1). Combining (5)-(7) with (1) and (2) and applying the finite difference approximation (3) yields the new matrix equation

$$\mathbf{A}\mathbf{y}(k+1) = \mathbf{B}\mathbf{y}(k) + \mathbf{G}\hat{\mathbf{u}}(k+1) + \mathbf{G}\mathbf{u}'(k+1), \tag{8}$$

where $\hat{\mathbf{u}}(k+1)$ contains the prescribed boundary conditions and forcing terms and $\mathbf{u}'(k+1)$ is the error associated with $\hat{\mathbf{u}}(k+1)$. The error vector $\mathbf{u}'(k+1)$ is also known as process (or system) noise and represents all the error associated with the modelling process. The series $\mathbf{u}'(k+1)$ is assumed to be a zero mean uncorrelated Gaussian process such that $E[\mathbf{u}'(k+1)] = 0$ and $E[\mathbf{u}'(k+1) \cdot \mathbf{u}'^T(k+1)] = \mathbf{Q}(k+1)$, where $E[\cdot]$ denotes expected value and $\mathbf{Q}(k+1)$ is the process noise covariance.

Letting $\mathbf{y}(k) = \hat{\mathbf{y}}(k) + \mathbf{y}'(k)$ where $\hat{\mathbf{y}}(k) = E[\mathbf{y}(k)]$, and taking the expected value of (8) yields

$$\mathbf{A}\hat{\mathbf{y}}(k+1) = \mathbf{B}\hat{\mathbf{y}}(k) + \mathbf{G}\hat{\mathbf{u}}(k+1). \tag{9}$$

Subtracting (9) from (8), squaring and taking expected values yields the error covariance propagation equation

$$\mathbf{A}\mathbf{P}(k+1)\mathbf{A}^T = \mathbf{B}\mathbf{P}(k)\mathbf{B}^T + \mathbf{G}\mathbf{Q}(k+1)\mathbf{G}^T \tag{10}$$

with the initial condition

$$\mathbf{P}(x, 0) = \mathbf{P}_0(x) = E[\mathbf{y}'(0) \cdot \mathbf{y}'^T(0)]. \tag{11}$$

Since the equations are linear and the noise sequence is Gaussian, the whole process must be Gaussian. Thus the first two moments of $\mathbf{y}(k)$ as given by (9)-(11) are sufficient to describe its probability distribution. Although it is now possible to provide confidence intervals for predictions of $\mathbf{y}(k)$, the formulation given by (9)-(11) have little more to offer than the original numerical model described by (1)-(4).

The major advantage of including the covariance equations (10) and (11) is that they help provide a mechanism for assimilation of observed data into the modelling process.

4. The Kalman filter. If discrete time series observations are collected at a finite number of locations along the channel, the measurements may be related to the system variables as follows:

$$\mathbf{z}(k) = \mathbf{H}\mathbf{y}(k) + \boldsymbol{\xi}(k) \tag{12}$$

where $\mathbf{z}(k)$ is a vector of length m containing the measured variables at time step k and $\mathbf{H}$ is the $m \times 2n$ observation control matrix specifying the elements of $\mathbf{y}(k)$ that have been observed. The measurement error vector of length m, $\boldsymbol{\xi}(k)$, represents all the uncertainty associated with the measurements $z(k)$. The measurement noise $\boldsymbol{\xi}(k)$ is a zero-mean uncorrelated Gaussian process with covariance $\mathbf{R}(k) = E[\boldsymbol{\xi}(k) \cdot \boldsymbol{\xi}^T(k)]$.

In order to incorporate observations into the modelling process it is necessary to develop update equations relating the estimates of $\mathbf{y}(k)$ to the measurements $\mathbf{z}(k)$. If it is assumed that the optimal estimate in the least square sense of $\mathbf{y}(k)$ is a linear combination of the most recent predictions with the corresponding observations, then one can proceed to develop an estimation algorithm based upon that of Kalman [17] and Kalman and Bucy [18], known as the Kalman filter. The Kalman equations will

not be derived here since a number of texts [12], [16] as well as the original papers [17], [18] provide suitable development of the theory. The Kalman filter equations may be summarized as follows:
Between observations (prediction):

$$\mathbf{A}\hat{\mathbf{y}}(k+1) = \mathbf{B}\tilde{\mathbf{y}}(k) + \mathbf{G}\hat{\mathbf{u}}(k+1), \tag{13}$$

$$\mathbf{A}\hat{\mathbf{P}}(k+1)\mathbf{A}^T = \mathbf{B}\tilde{\mathbf{P}}(k)\mathbf{B}^T + \mathbf{G}\mathbf{Q}(k+1)\mathbf{G}^T; \tag{14}$$

At observations (filtering):

$$\tilde{\mathbf{y}}(k+1) = \hat{\mathbf{y}}(k+1) + \mathbf{K}(k+1)[\mathbf{z}(k+1) - \mathbf{H}\hat{\mathbf{y}}(k+1)], \tag{15}$$

$$\mathbf{K}(k+1) = \hat{\mathbf{P}}(k+1)\mathbf{H}^T[\mathbf{R}^*(k+1)]^{-1}, \tag{16}$$

$$\mathbf{R}^*(k+1) = \mathbf{H}\hat{\mathbf{P}}(k+1)\mathbf{H}^T + \mathbf{R}(k+1), \tag{17}$$

$$\tilde{\boldsymbol{P}}(k+1) = \hat{\mathbf{P}}(k+1) - \hat{\mathbf{P}}(k+1)\mathbf{H}^T[\mathbf{R}^*(k+1)]^{-1}\mathbf{H}\hat{\mathbf{P}}(k+1); \tag{18}$$

With initial conditions:

$$\tilde{\mathbf{y}}(0) = \mathbf{y}_0 \qquad \tilde{\mathbf{P}}(0) = \mathbf{P}_0 \tag{19}$$

where a caret ($\hat{}$) now denotes a predicted value and a tilde ($\tilde{}$) denotes a filtered, or measurement-corrected, value. Thus $\hat{\mathbf{y}}(k+1)$ and $\hat{\mathbf{P}}(k+1)$ are estimates of time step $k+1$ based upon information up to and including time step k, whereas $\tilde{\mathbf{y}}(k+1)$ and $\tilde{\mathbf{P}}(k+1)$ are estimates based upon information up to and including that available at time step $k+1$.

The effect of introducing observations into the modelling process is seen in (18). The error covariance is reduced whenever observations are assimilated into the model, i.e., $\tilde{\mathbf{P}}(k+1) \leqq \hat{\mathbf{P}}(k+1)$. Thus, including observations increases the accuracy of the model.

The Kalman gain matrix $\mathbf{K}(k+1)$ describes the weighting applied to the model predictions relative to that applied to the observations during the filtering process. If the model is highly reliable and the measurements are questionable, then $\mathbf{Q}(k+1)$, the system noise covariance, becomes small and $\mathbf{R}(k+1)$, the measurement noise covariance becomes large causing $\mathbf{K}(k+1)$ to become small. In this case, the observations provide very little information on the behaviour of the system and the process may be viewed as being nearly deterministic. Conversely, if the model errors are large and the measurement errors are negligible, then $\mathbf{Q}(k+1)$ is large, $\mathbf{R}(k+1)$ is small, $\mathbf{K}(k+1)$ is large and the model is relatively unimportant in determining the filtered state estimate. The system under consideration is then prescribed by the observation field alone. In general, applications fall between those two extremes since there is usually uncertainty associated with both the model and the measured data.

The Kalman gain matrix, in addition to partitioning the innovation, or one step ahead prediction residual, $[\mathbf{z}(k+1) - \mathbf{H}\hat{\mathbf{y}}(k+1)]$ into prediction error and measurement error components, also acts as a spatial interpolation function for the measurement updates. Furthermore, the Kalman gain relates the velocity and surface elevation such that measurements of η are used to correct U.

A major difficulty associated with the solution of (13) to (18) is that the $2n \times 2n$ matrices $\hat{\mathbf{P}}(k)$, $\tilde{\mathbf{P}}(k)$, $\hat{\mathbf{P}}(k+1)$ and $\tilde{\mathbf{P}}(k+1)$ must be manipulated and stored in computer memory. Since n, the number of spatial grid points, is often on the order of 100 or more, using the Kalman filter for these problems may not be practical or even feasible. An approach adopted in air pollution applications [3], [11] to overcome this problem was to use the partitioning theory of Aoki [2] to subdivide the state vector $\mathbf{y}(k)$ into

subvectors corresponding to spatial regions surrounding measurement locations. Suboptimal filters of the form given by (13) to (18) were then run individually on these local subregions to provide filtered estimates of sulfur dioxide concentration in the vicinity of sensors and predictions of the concentration over a whole urban airshed. The main difficulty associated with this approach is that the relationships between variables in different subregions cannot be incorporated directly into the stochastic filter.

A second problem arises from the mathematical form of the filter. The filter equations constitute a solution to a nonlinear matrix Riccati equation, only one solution of which is positive definite. Nonpositive definite matrices can result from the subtraction of the two positive definite covariance matrices in (18). If covariance matrices become poorly conditioned in this manner, filter divergence and inaccurate predictions will be produced [16], [23]. In large scale distributed (spatially varying) problems, filter instability can result [20].

5. The Chandrasekhar-type square root filter. An alternate approach to the problem has been employed by Desalu, Gould and Schweppe [9]. If the process and measurement noise covariances can be assumed constant such that

$$\mathbf{Q}(k)=\mathbf{Q}, \qquad \mathbf{R}(k)=\mathbf{R}, \tag{20}$$

then an incremental covariance approach [24] can be used to develop a non-Riccatti Chandrasekhar-type square root filtering algorithm. If, in addition, it can be assumed that the initial conditions are known with certainty ($\mathbf{P}_0=0$) then the rank of the state covariance matrices becomes the rank of the system noise covariance $\mathbf{Q}$. Usually, the rank of $\mathbf{Q}$ is very much less than the dimension of the state covariance. Thus, the largest matrices that must be stored are of order $2n\times\alpha$, where α is the rank of $\mathbf{Q}$, $\alpha \ll 2n$.

Applying the incremental covariance equations results in the Chandrasekhar-type square root algorithm derived in the Appendix. The filter equations may be summarized as follows:

Between observations:

$$\mathbf{A}\hat{\mathbf{y}}(k+1)=\mathbf{B}\tilde{\mathbf{y}}(k)+\mathbf{G}\mathbf{u}(k+1), \tag{21}$$

$$\mathbf{AY}(k)=\mathbf{BS}(k); \tag{22}$$

At observations:

$$\tilde{\mathbf{y}}(k+1)=\hat{\mathbf{y}}(k+1)+\mathbf{K}(k+1)[\mathbf{z}(k+1)-\tilde{\mathbf{H}}\hat{\mathbf{y}}(k+1)], \tag{23}$$

$$\mathbf{K}(k+1)=\mathbf{C}(k+1)[\mathbf{R}^*(k+1)]^{-1}, \tag{24}$$

$$\mathbf{C}(k+1)=\mathbf{C}(k)+\mathbf{Y}(k)\mathbf{L}(k)\mathbf{Y}^T(k)\mathbf{H}^T, \tag{25}$$

$$\mathbf{R}^*(k+1)=\mathbf{R}(k)+\mathbf{HY}(k)\mathbf{L}(k)\mathbf{Y}^T(k)\mathbf{H}^T, \tag{26}$$

$$\mathbf{S}(k+1)=[\mathbf{I}-\mathbf{K}(k+1)\mathbf{H}]\mathbf{Y}(k), \tag{27}$$

$$\mathbf{L}(k+1)=\mathbf{L}(k)+\mathbf{L}(k)\mathbf{Y}^T(k)\mathbf{H}^T[\mathbf{R}^*(k)]^{-1}\mathbf{HY}(k)\mathbf{L}(k). \tag{28}$$

The initial conditions for the set of equations is given by

$$\mathbf{AY}(0)=\mathbf{G}, \qquad \mathbf{L}(0)=\mathbf{Q}, \qquad \mathbf{C}(0)=\mathbf{0}, \qquad \mathbf{R}^*(0)=\mathbf{R}. \tag{29}$$

The prediction and filter errors covariances may be constructed from the factor matrices in the following manner:

$$\hat{\mathbf{P}}(k) = \sum_{p=1}^{k} \mathbf{Y}(p)\mathbf{L}(p)\mathbf{Y}^T(p), \tag{30}$$

$$\tilde{\mathbf{P}}(k) = \sum_{p=1}^{k} \mathbf{S}(p)\mathbf{L}(p)\mathbf{S}^T(p). \tag{31}$$

The matrices $\mathbf{Y}(k)$ and $\mathbf{S}(k)$ are $2n \times \alpha$, $\mathbf{R}^*(k)$ and $\mathbf{R}$ are $m \times m$, and $\mathbf{L}(k)$ and $\mathbf{Q}$ are $\alpha \times \alpha$ matrices.

The assumption that there is no error in the prescribed initial conditions needs to be justified, since in practice the initial system state is rarely known with any degree of certainty. To determine the effects of this assumption we need to consider the e-folding time of the system. In the momentum equation (2), the friction coefficient λ can be determined from the expression $\lambda = g\bar{U}/C^2$ where $\bar{U}$ is the mean square velocity and C is the Chezy friction factor. For typical river and estuarine values of $\bar{U} = 0.5\ \mathrm{m\ sec^{-1}}$, $h = 10\ \mathrm{m}$, and $C = 100\ \mathrm{m^{-1/2}\ sec}$, the e-folding time (or time constant) of the system, as given by [21], is $\tau = 2h/\lambda = 11.3$ hours. Effectively, after 24 hours or so, the influence of errors in the initial conditions (and errors in the prescription of the initial error covariance) are negligible. In practice, then, the combined dynamic-stochastic model should be run for 2 or 3 days before results are interpreted. It should be noted that these guidelines are conservative, however, since using the filter will cause the model to converge to prototype conditions more rapidly.

6. A numerical experiment—stochastic boundary forcing. One of the major sources of error in the modelling of shallow water wave propagation is the prescription of boundary conditions. In order to examine the capability of the dynamic-stochastic numerical model to correctly simulate the effects of boundary errors, a test case was developed in which stochastic forcing was applied to both ends of a wedge-shaped estuary. An analytical solution for this problem has been developed by Robinson [26]. Setting $F(x, t) = 0$ in (1), the solution at a single forcing frequency σ may be expressed as

$$U = R_e\{V(x)\exp(i\sigma t)\}, \qquad \eta = R_e\{Z(x)\exp(i\sigma t)\}, \tag{32}$$

where $V(x)$ and $Z(x)$ are complex amplitude functions. If the channel width and depth are described as linear functions of distance such that $b = b_0 x$ and $h = h_0 x$, then the complex amplitude functions are determined from

$$V(x) = \frac{1}{k_1 x}[A_1 J_\nu(2\sqrt{k_1 x}) + A_2 J_{-\nu}(2\sqrt{k_1 x})], \tag{33}$$

$$\begin{aligned} Z(x) = \frac{ih_0}{2\sigma k_1 x}\{&2\sqrt{k_1 x}A_1 J_{\nu-1}(2\sqrt{k_1 x}) + (2-\nu)A_1 J_\nu(2\sqrt{k_1 x}) \\ &+ 2\sqrt{k_1 x}\,A_2 J_{-\nu-1}(2\sqrt{k_1 x}) + (2+\nu)A_2 J_{-\nu}(2\sqrt{k_1 x})\}. \end{aligned} \tag{34}$$

The complex order Bessel function $J_\nu(x)$, $\nu^2 = 4(1 + ik_2)$ is computed using the series expansion

$$J_\nu(x) = (\tfrac{1}{2}x)^\nu \sum_{k=0}^{p} \frac{(\frac{1}{4}x^2)^k}{k!\Gamma(\nu + k + 1)} \tag{35}$$

with $p = 8$ and the reciprocal of the complex gamma function was approximated using

$$\frac{1}{\Gamma(z)} = \sum_{k=1}^{26} c_k z^k$$

with the c_k defined by Abramowitz and Stegun [1, § 6.1.34]. The constants k_1 and k_2 are defined as

$$k_1 = \frac{\sigma^2}{h_0 g}, \qquad k_2 = \frac{\sigma\lambda}{h_0^2 g}.$$

The constants A_1 and A_2 are defined from the boundary conditions at $x = x_A$ and $x = x_B$. For $V(x_A) = V_A$ representing river inflow at the narrow end and a prescribed oceanic forcing by sea levels of $Z(x)_B = Z_B$,

$$A_1 = \frac{r_A q_B - r_B q_A}{p_A q_B - p_B q_A}, \qquad A_2 = \frac{r_A p_B - r_B p_A}{q_A p_B - q_B p_A},$$

$$p_A = J_\nu(y_A), \qquad p_B = y_B J_{\nu-1}(y_B) + (2-\nu) J_\nu(y_B),$$

$$q_A = J_{-\nu}(y_A), \qquad q_B = y_B J_{-\nu-1}(y_B) + (2+\nu) J_{-\nu}(y_B),$$

$$y_A = 2\sqrt{k_1 x_A}, \qquad y_B = 2\sqrt{k_1 x_B},$$

$$r_A = k_1 x_A V_A, \qquad r_B = -iZ_B 2gk_1^2 x_B/\sigma.$$

The test case was developed using values of $b_0 = 0.312$, $h_0 = 2.36 \times 10^{-4}$, $\lambda = 2.12 \times 10^{-3}$ m sec^{-1}, $x_A = 10$ km and $x_B = 210$ km. The boundary conditions were produced from the following time series

$$U_A(k) = 0.9986\, U_A(k-1) + a_A(k),$$

$$\eta_B(k) = 1.0157\, \eta_B(k+1) - 0.02007\, \eta_B(k-2) + a_B(k),$$

where the time increment was 360 sec and the Gaussian white series $a_A(k)$ and $a_B(k)$ with variances 6.9×10^{-5} m^2 sec^{-2} and 3.5×10^{-3} m^{-2}, respectively, were produced using the IMSL random number routine GGNML. The river inflow velocity $U_A(k)$ is a Markov process with an e-folding correlation decay scale of 36 hours. The series $\eta_B(k)$ has a pseudo-periodic character with a central frequency of 0.5 days^{-1} and an e-folding correlation decay time scale of one day. This structure was designed to emulate storm surges at the open boundary and river discharge fluctuations at the landward boundary. The dynamic-stochastic model equations (21)-(29) were augmented to incorporate the autocorrelation structure of the boundary conditions.

The two series of 4096 values were cosine tapered over half their lengths then discrete Fourier transformed to produce the complex amplitudes V_A and Z_B for each frequency component. The analytical model was forced with each of the 2048 frequencies. The resulting $V(x)$ and $Z(x)$ values at 101 grid points were inverse Fourier transformed to produce time series. Simulated observations were produced at grid points 21, 41, 61 and 81 by adding random sequences produced by IMSL routine GGNML with variance 2.5×10^{-3} m^2 to the analytical values of η at those grid points. The model time step, $\Delta t = 360$ sec, while the measurement interval and filter update interval, $\Delta t_m = 3600$ sec.

The 101 grid points used in both the analytical and dynamic-stochastic models were selected to preserve a constant Courant number over the length of the channel, e.g.,

$$\Delta x = C_r\, \Delta t \sqrt{gh}$$

where the Courant number $C_r = 2$. In this way, the grid resolution is enhanced in shallow regions where the phase propagation of the long wave is reduced.

Representative time series of velocity and surface elevation are shown in Figs. 1 and 2, respectively. The true values are the analytical solutions; the numerical model

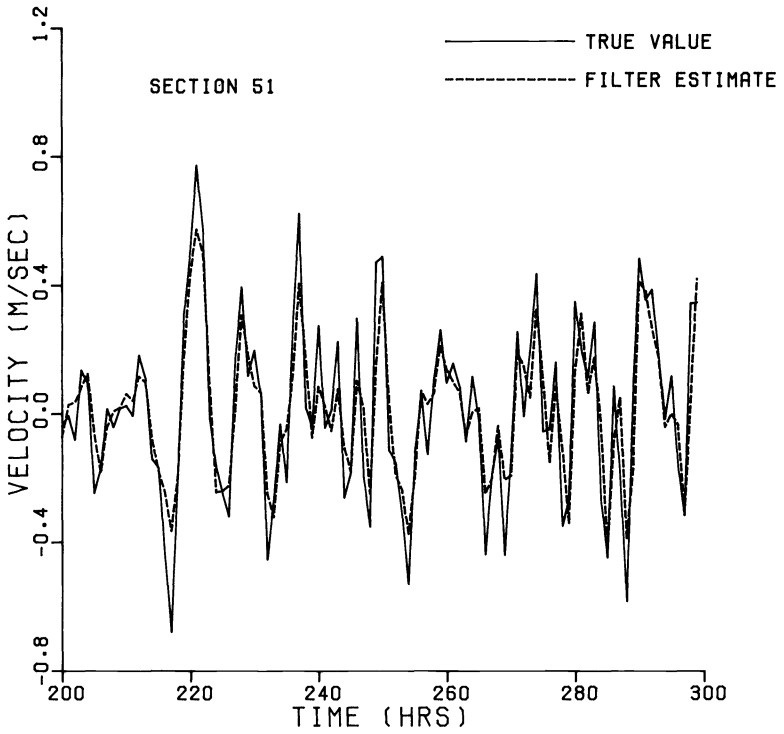


FIG. 1. *Representative time series of true (analytical) values and filter estimates of velocity at an interior grid point of the wedge-shaped estuary.*

values are computed using the deterministic numerical model (1)-(4) with the exact boundary conditions. The filter estimates are computed using the dynamic-stochastic model (21)-(28) augmented with equations describing the autocorrelation structure of $U_A(k)$ and $\eta_B(k)$. The models are effectively spun up over the first 103 hours and spun down over the last 103 hours because of the cosine tapering applied to η_B. Shown in the figures are the time series from the untapered central portion of the record. The filter estimates are generally in good agreement with the analytical solutions. The velocity series of Fig. 1 is considerably noisier than the elevation shown in Fig. 2. As a result, the filter is more successful in tracking elevation values.

The forcing boundary conditions are unknown variables in the dynamic-stochastic model. The landward and ocean boundary conditions of the system are estimated by the filter using only observations from the interior of the computational domain, a description of the autocorrelation functions of U_A and η_B and a numerical approximation of the governing dynamics. The filter estimates and true values of U_A are shown in Fig. 3. It can be seen that the filter is not capable of recovering that boundary condition to any significant degree. This lack of predictability is forecast by the filter variance, however. The actual filter error variance, in fact, is only half that predicted by the filter. The poor recovery of the landward boundary condition (the narrow end of the wedge) is attributable to the weak river forcing supplied there being swamped by the oceanic forcing at the wide end which has been amplified by a factor of 4 or so as it propagates down the funnel. As shown in Fig. 4, the filter is more successful in reproducing the ocean boundary condition. The filtered values, while successfully reconstructing η_B, lag the true values somewhat. Only past and present observed data

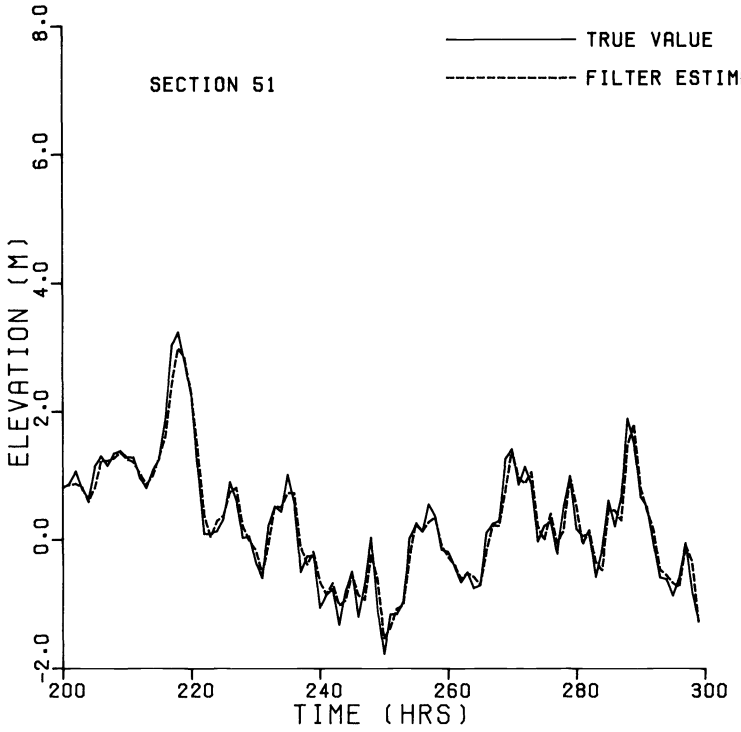


FIG. 2. *As for Fig.* 1, *but for surface elevation.*

are available to the filter. To eliminate the phase lag a full smoothing solution, in which observed data from both before and after a given time are incorporated into the estimation, must be implemented [12, Chap. 5]. Smoothing algorithms will be discussed in a subsequent publication.

The overall filter performance can be assessed through an examination of the mean square error (MSE) of the filter estimates, where

$$\text{MSE} = \frac{1}{N} \sum_{k=1}^{N} (y_{jk} - \tilde{y}_{jk})^2,$$

$\tilde{y}_{jk}$ is the jth variable at the kth time step. The MSE was computed over 204 hourly values from the middle of the time series. Shown in Figs. 5 and 6 are the MSE of the filter estimates, the MSE of numerical model values (with exact boundary conditions), the computed filter error variance $\tilde{\mathbf{P}}(k/k)$ at steady state (achieved to 6 significant digits after 100 hours) and the variance of the true values over the length of the channel for velocity and surface elevation, respectively. The variance of the true (analytical) values about their mean is included as a reference level. In both figures it can be seen that the filter MSE is effectively bounded by the numerical model MSE and the computed (theoretical) filter variance. This is as good as can be expected since numerical approximations will introduce an error component in the filter calculations as well as in the deterministic model. The filter MSE is diminished sharply in the vicinity of observations. The displacement of the filter MSE minimum from the measurement locations is due to the channel geometry. The long wave signal, the prediction errors and corresponding corrections are all amplified in the up-channel (negative x) direction.

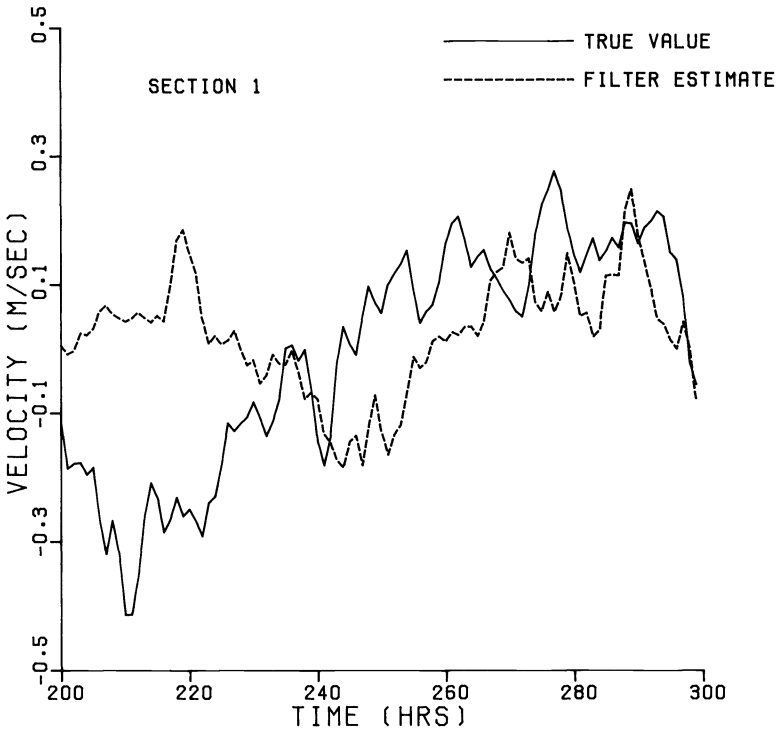


FIG. 3. *Representative time series of true values and filter estimates of the velocity boundary condition at the head of the wedge-shaped estuary.*

7. A numerical experiment—stochastic momentum forcing. In the modelling of long wave processes in coastal waters, neglected nonlinearities, spatial averaging and uncertain atmospheric forcing all produce errors in the momentum balance equation. These errors may be viewed as momentum noise or stochastic momentum forcing. In order to examine the performance of the stochastic filter under the condition of stochastic momentum forcing, the simple case of a constant depth channel, closed at both ends, in which zero-mean Gaussian white (in time) forcing has been imposed is developed. The governing equations (1) and (2) simplify to

$$\frac{\partial U}{\partial t}+g\frac{\partial \eta}{\partial x}+\lambda U=F(x,t), \tag{36}$$

$$\frac{\partial \eta}{\partial t}+h\frac{\partial U}{\partial x}=0. \tag{37}$$

The boundary conditions are simply $U(0, t)=0$ and $U(L, t)=0$ where L is the channel length. Expanding the velocity and surface elevation in Fourier series in frequency-wavenumber space and imposing the boundary conditions produces

$$U=R_e\left\{\sum_m\sum_n V_{mn}\,e^{im\sigma t}\sin n\omega t\right\}, \tag{38}$$

$$\eta=R_e\left\{\sum_m\sum_n Z_{mn}\,e^{im\sigma t}\cos n\omega t\right\}. \tag{39}$$

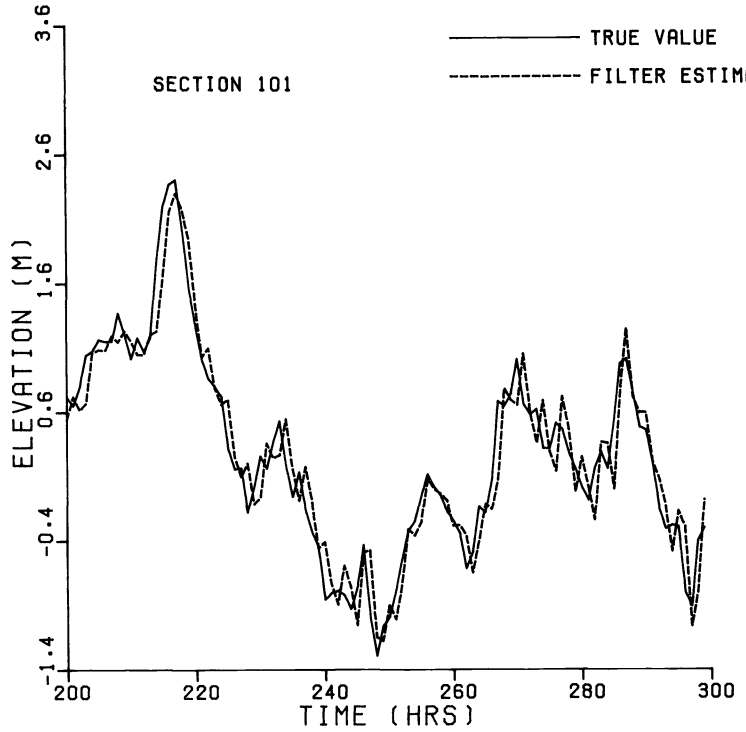


FIG. 4. *Representative time series of true values and filter estimates of the surface elevation boundary condition at the mouth of the wedge-shaped estuary.*

The forcing term is expanded as

$$F(x, t) = R_e\left\{\sum_m \sum_n F_{mn}\, e^{im\sigma t} \sin n\omega t\right\}. \tag{40}$$

Fourier transforming (36) and (37) yields

$$V_{mn} = \frac{\sigma F_{mn}}{\sigma\lambda + i[\sigma^2 - \omega^2 gh]}, \tag{41}$$

$$Z_{mn} = i\frac{\omega h}{\sigma} V_{mn}. \tag{42}$$

The fundamental wavenumber ω is defined in terms of the channel length, $\omega = \pi/L$. Four spatial modes are included in the forcing function, and thus in the solutions for η and U. The F_{mn} components for each spatial mode were obtained by discrete Fourier transforming cosine tapered series of 4096 random values produced by the IMSL routine GGNML as in the previous test case. The variances of the untapered time dependent coefficients of the first four spatial modes are 3.7×10^{-6}, 2.9×10^{-7}, 2.0×10^{-8} and 9.8×10^{-9} m^2 sec^{-4}, respectively. Observations of η at $x = 0$, $x = L/3$, $x = 2L/3$ and $x = L$ were produced by obtaining η from (39), (41) and (42), then adding zero-mean Gaussian white noise with variance 2.5×10^{-3} m^2 from GGNML. The dynamic stochastic model is run with a time step Δt of 600 sec, while the measurement and measurement update interval Δt_m is 3600 sec. The grid spacing, Δx, is 5 km and the channel length L is 300 km. The friction coefficient λ is 1.85×10^{-5} sec^{-1}. The only information available

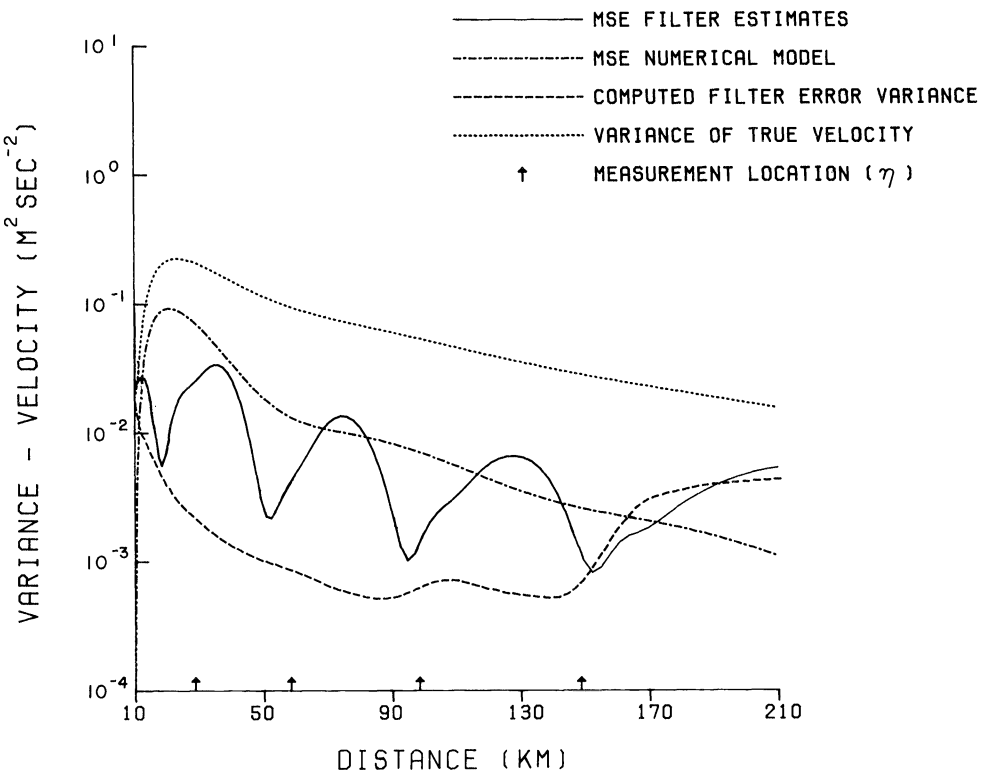


FIG. 5. *Mean square error of the filter estimates, mean square error of the numerical model with exact boundary conditions, computed filter error variance and variance of the true (analytical) values for velocity as a function of distance in the wedge-shaped estuary.*

to the dynamic-stochastic model concerning the forcing field $F(x, t)$ is the variance and spatial structure.

Typical time series of U and η at grid point 28, which is not a measurement location or nodal point, are shown in Figs. 7 and 8, respectively. It can be seen that the filter provides accurate estimates for both variables. The filter MSE, numerical model MSE, theoretical filter variance and variance of the true variables for U and η as a function of long channel distance are shown in Figs. 9 and 10. The MSE values are computed from 342 hourly samples from the untapered middle of the time series. In both cases, the filter MSE is less than the theoretical (computed) filter variance. The filter MSE of surface elevation is less than the MSE of the numerical model with exact prescribed forcing over nearly the whole channel (Fig. 10). The filter MSE of velocity exceeds the numerical model MSE somewhat but the filter still accounts for over 90 percent of the variance in the true velocity field. Thus, the dynamic-stochastic model successfully recovers the system state even with unknown stochastic forcing.

8. Concluding remarks. An approach has been presented for the optimal combination of linear shallow water long wave models with observed time series data. The basis of the data assimilation scheme is a non-Riccati square root filtering algorithm that ensures filter stability while reducing the rank of the error covariance matrices that must be stored and manipulated by an order of magnitude or more. The complete dynamic-stochastic model, which consists of the square root filter derived from and combined with a numerical approximation to the linearized one-dimensional shallow

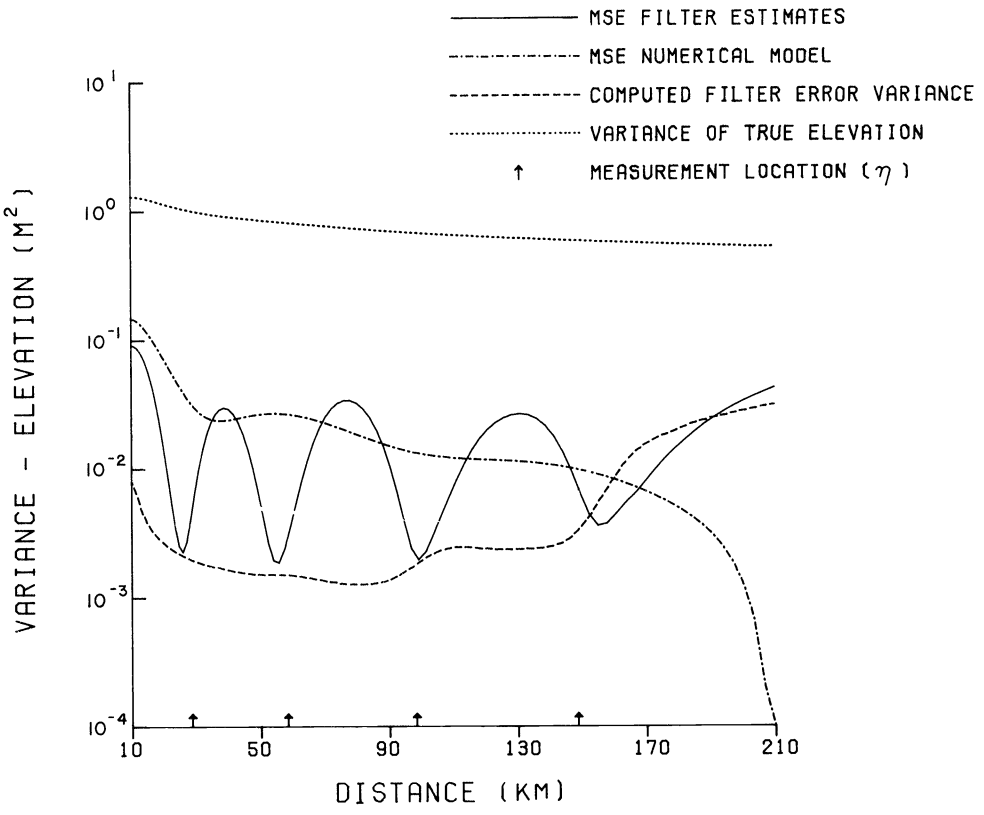


FIG. 6. *As for Fig. 5, but for surface elevation.*

water equations, embodies the best features of empirical prediction schemes and deterministic models based upon physical principles.

The dynamic-stochastic model was applied to two analytical test cases. In the first, a wedge-shaped estuary was forced with unknown first order autoregressive river discharge and unknown second order autoregressive broad band open boundary surface elevation. In the second, a channel closed at both ends (e.g., a long, narrow lake) was forced with an unknown broad band momentum source. In both cases, the dynamic-stochastic model recovered the true state to within the accuracy of the finite difference approximation of the governing equations.

Appendix. Chandrasekhar-type square root filter. A Chandrasekhar-type algorithm is developed by rewriting the Kalman filter equations (13)-(18) in terms of their incremental changes from one time step to the next. The incremental covariance $\delta\tilde{\mathbf{P}}(k+1)$ is defined as

$$\text{(A1)}\quad \delta\tilde{\mathbf{P}}(k+1)=\tilde{\mathbf{P}}(k+1)-\tilde{\mathbf{P}}(k)$$
$$=\delta\hat{\mathbf{P}}(k+1)-\mathbf{K}(k+1)\mathbf{R}^*(k+1)\mathbf{K}^T(k+1)+\mathbf{K}(k)\mathbf{R}^*(k)\mathbf{K}^T(k)$$

and

$$\text{(A2)}\qquad \delta\hat{\mathbf{P}}(k+1)=\hat{\mathbf{P}}(k+1)-\hat{\mathbf{P}}(k).$$

From (17) it is obtained that

$$\text{(A3)}\qquad \delta\mathbf{R}^*(k+1)=\mathbf{H}\delta\hat{\mathbf{P}}(k+1)\mathbf{H}^T.$$

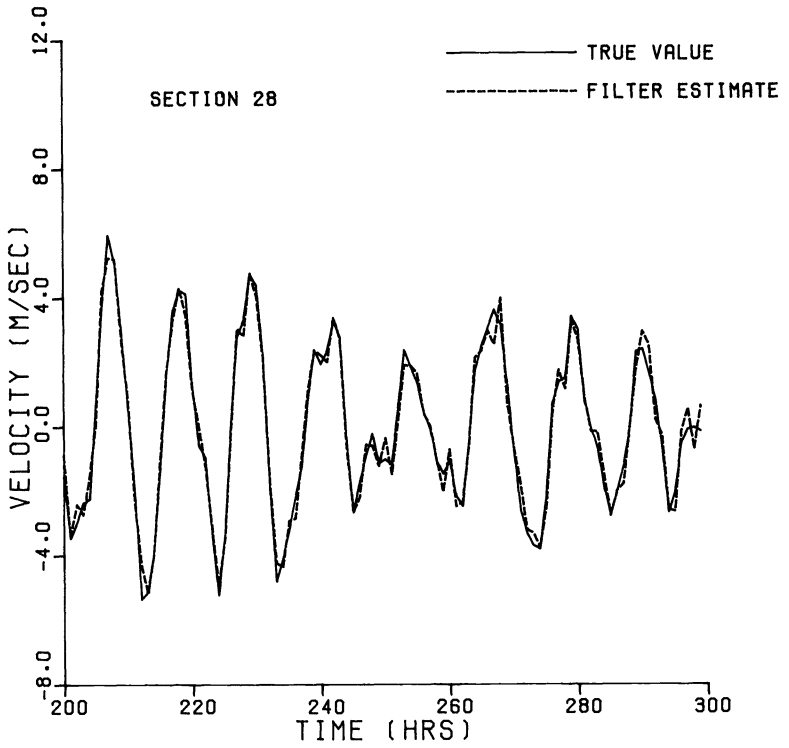


FIG. 7. *As for Fig.* 1, *but for the rectangular channel closed at both ends.*

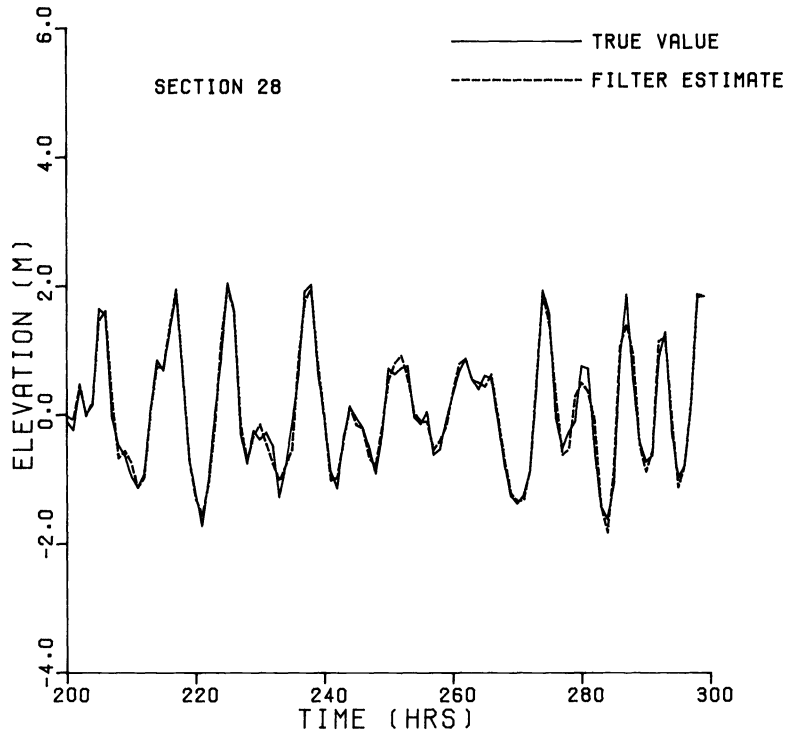


FIG. 8. *As for Fig.* 7, *but for surface elevation.*

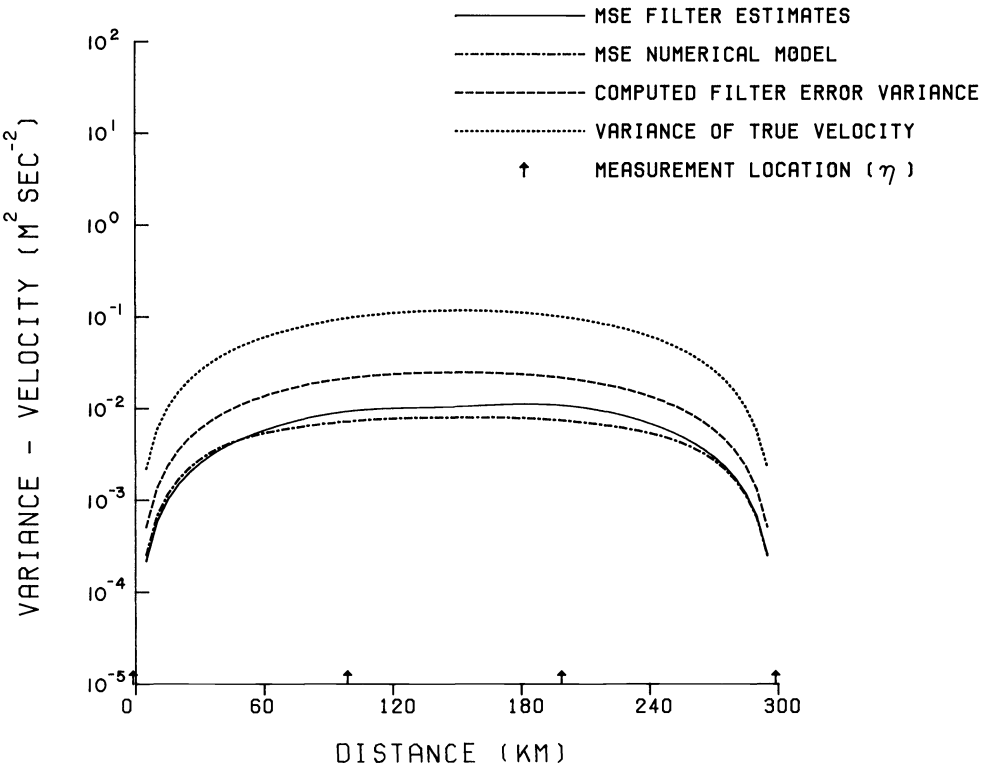


FIG. 9. *As for Fig. 5, but for the rectangular channel closed at both ends.*

Using (A3) and (15), one obtains

$$\delta\hat{\mathbf{P}}\mathbf{H}^T = \mathbf{K}(k+1)\mathbf{R}^*(k+1) - \mathbf{K}(k)\mathbf{R}^*(k) \tag{A4}$$

from which an expression for change in the Kalman gain may be written

$$\delta\mathbf{K}(k+1) = \mathbf{K}(k+1) - \mathbf{K}(k) = [\mathbf{I} - \mathbf{K}(k+1)\mathbf{H}]\delta\hat{\mathbf{P}}(k+1)\mathbf{H}^T[\mathbf{R}^*(k)]^{-1}. \tag{A5}$$

Substituting (A3) and (A5) into (A1):

$$\begin{aligned}\delta\tilde{\mathbf{P}}(k+1) = [\mathbf{I} - \mathbf{K}(k+1)\mathbf{H}][\delta\hat{\mathbf{P}}(k+1) + \delta\hat{\mathbf{P}}(k+1)\mathbf{H}^T \\ \cdot [\mathbf{R}^*(k)]^{-1}\mathbf{H}\delta\mathbf{P}(k+1)][\mathbf{I} - \mathbf{K}(k+1)\mathbf{H}]^T.\end{aligned} \tag{A6}$$

Equation (A6) has a quadratic form. Using (14) and imposing the condition that $\mathbf{Q}(k) = \mathbf{Q}$ is a constant to obtain the incremental covariance $\delta\hat{\mathbf{P}}(k+1)$,

$$\mathbf{A}\delta\hat{\mathbf{P}}\mathbf{A}^T = \mathbf{B}\delta\hat{\mathbf{P}}\mathbf{B}^T. \tag{A7}$$

It can be seen that (A7) also has a quadratic form. The incremental covariances $\delta\hat{\mathbf{P}}(k+1)$ and $\delta\tilde{\mathbf{P}}(k+1)$ may be factored as follows:

$$\delta\hat{\mathbf{P}}(k+1) = \mathbf{Y}(k+1)\mathbf{L}(k+1)\mathbf{Y}^T(k+1), \tag{A8}$$

$$\delta\tilde{\mathbf{P}}(k+1) = \mathbf{S}(k+1)\mathbf{L}(k+1)\mathbf{S}^T(k+1). \tag{A9}$$

The factor matrices $\mathbf{Y}(k)$, $\mathbf{L}(k)$ and $\mathbf{S}(k)$ have dimensions $2n \times \alpha$, $\alpha \times \alpha$ and $2n \times \alpha$, respectively. For zero initial error covariance, the rank of the incremental covariance reduces to that of the system noise covariance $\mathbf{Q}$ [24].

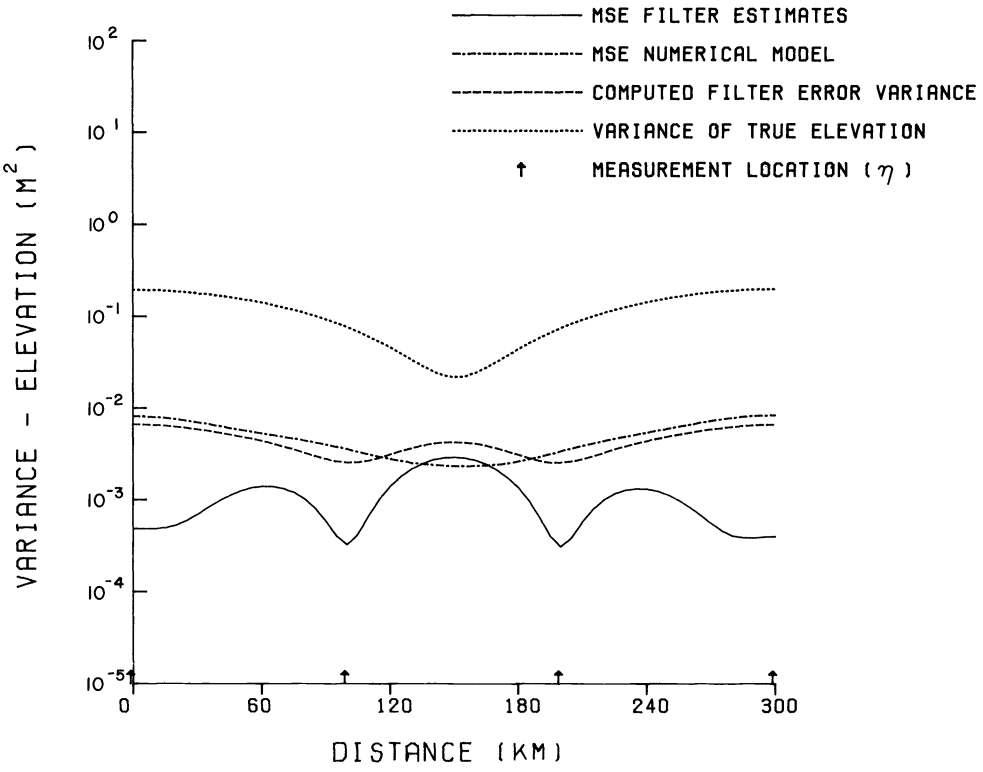


FIG. 10. *As for Fig. 9, but for surface elevation.*

Substituting (A8) and (A9) into (A7) produces

$$\mathbf{A}\mathbf{Y}(k)\mathbf{L}(k)\mathbf{Y}^T(k)\mathbf{A}^T = \mathbf{B}\mathbf{S}(k)\mathbf{L}(k)\mathbf{S}^T(k)\mathbf{B}^T. \tag{A10}$$

Thus

$$\mathbf{A}\mathbf{Y}(k) = \mathbf{B}\mathbf{S}(k). \tag{A11}$$

Substituting (A8) and (A9) into (A6),

$$\begin{aligned}\mathbf{S}(k+1)\mathbf{L}(k+1)\mathbf{S}^T(k+1) = {}& [\mathbf{I}-\mathbf{K}(k+1)\mathbf{H}]\mathbf{Y}(k) \\ & \cdot \{\mathbf{L}(k)+\mathbf{L}(k)\mathbf{Y}^T(k)\mathbf{H}^T[\mathbf{R}^*(k)]^{-1}\mathbf{H}\mathbf{Y}(k)\mathbf{L}(k)\} \\ & \cdot \mathbf{Y}^T(k)[\mathbf{I}-\mathbf{K}(k+1)\mathbf{H}]^T. \end{aligned} \tag{A12}$$

Therefore

$$\mathbf{S}(k+1) = [\mathbf{I}-\mathbf{K}(k+1)\mathbf{H}]\mathbf{Y}(k), \tag{A13}$$

$$\mathbf{L}(k+1) = \mathbf{L}(k)+\mathbf{L}(k)\mathbf{Y}^T(k)\mathbf{H}^T[\mathbf{R}^*(k)]^{-1}\mathbf{H}\mathbf{Y}(k)\mathbf{L}(k). \tag{A14}$$

In order to obtain the gain matrix let

$$\mathbf{C}(k) = \mathbf{K}(k)\mathbf{R}^*(k). \tag{A15}$$

Then from (16), (A9) and (A15)

$$\mathbf{C}(k+1) = \mathbf{C}(k)+\mathbf{Y}(k)\mathbf{L}(k)\mathbf{Y}^T(k)\mathbf{H}^T \tag{A16}$$

and combining (17) and (A8) yields

$$\mathbf{R}^*(k+1) = \mathbf{R}^*(k) + \mathbf{H}\mathbf{Y}(k)\mathbf{L}(k)\mathbf{Y}^T(k)\mathbf{H}^T. \tag{A17}$$

If the initial error covariance $\tilde{\mathbf{P}}(0)$ is constrained to be zero, then

$$\mathbf{A}\mathbf{Y}(0)\mathbf{A}^T = \mathbf{G}\mathbf{Q}\mathbf{G}^T. \tag{A18}$$

Thus

$$\mathbf{A}\mathbf{Y}(0) = \mathbf{G}, \tag{A19}$$

$$\mathbf{L}(0) = \mathbf{Q}. \tag{A20}$$

Similarly from (17)

$$\mathbf{R}^*(0) = \mathbf{R} \tag{A21}$$

and from (A16)

$$\mathbf{C}(0) = \mathbf{0}. \tag{A22}$$

These filter equations are summarized by (21) to (29) in the main body of the paper.

Acknowledgments. I would like to thank T. E. Unny, L. R. Muir and P. F. Hamblin for their valuable suggestions made throughout the course of this work.

REFERENCES

[1] M. ABRAMOWITZ AND J. A. STEGUN, *Handbook of Mathematical Functions*, Dover, New York, 1970.
[2] M. AOKI, *Optimization of Stochastic Systems*, Academic Press, New York, 1967.
[3] S. G. BANKOFF AND E. L. HANZEVAK, *The adaptive filtering transport model for prediction and control of pollutant concentration in an urban airshed*, Atmos. Environ., 9 (1975), pp. 793–808.
[4] W. P. BUDGELL, *A dynamic-stochastic approach for modelling advection-dispersion processes in open channels*, in Time Series Methods in Hydrosciences, A. H. El-Shaarawi and S. R. Esterby, eds., Elsevier, Amsterdam, 1982, pp. 244–263.
[5] W. P. BUDGELL, *Nonlinear data assimilation for shallow water equations in branched channels*, submitted to J. Geophys. Res., 1986.
[6] W. P. BUDGELL AND A. H. EL-SHAARAWI, *Time series modelling of storm surges on a medium-sized lake*, in Marine Forecasting, Predictability and Forecasting in Marine Science, J. C. J. Nihoul, ed., Elsevier, Amsterdam, 1979, pp. 197–218.
[7] C. L. CHIU, *Applications of Kalman filter theory to hydrology, hydraulics and water resources*, Univ. Pittsburgh, Pittsburgh, 1978.
[8] R. N. DEGUIDA, J. J. CONNOR AND R. R. PEARCE, *Application of estimation theory to design of sampling programs for verification of coastal dispersion predictions*, in Finite Elements in Water Resources, W. G. Gray, G. F. Pinder and C. A. Brebbia, eds., Pentech, London, 1977, pp. 4.303–4.334.
[9] A. A. DESALU, L. A. GOULD AND F. C. SCHWEPPE, *Dynamic estimation of air pollution*, IEEE Trans. Automat. Control, AC 19 (1974), pp. 904–910.
[10] D. L. FREAD, *Effects of time step size in implicit dynamic routing*, Wat. Resour. Bull., 9 (1973), pp. 918–926.
[11] G. FRONZA, A. SPIRITO AND A. TONIELLI, *Real-time forecast of air pollution episodes in the venetian region. Part 2: The Kalman predictor*, Appl. Math. Model., 3 (1979), pp. 409–415.
[12] A. GELB, *Applied Optimal Estimation*, MIT Press, Cambridge, MA, 1974.
[13] R. GORDON AND M. SPAULDING, *A bibliography of numerical models for tidal rivers*, Marine Technical Report No. 14, Univ. Rhode Island, Providence, 1974.
[14] J. B. HINWOOD AND I. G. WALLIS, *Classification of models of tidal waters*, J. Hydraul. Div. Amer. Soc. Civ. Engrs., 101 (1975), pp. 1315–1331.
[15] ———, *Review of models of tidal waters*, J. Hydraul. Div. Amer. Soc. Civ. Engrs., 101 (1975), pp. 1405–1421.
[16] A. H. JAZWINSKI, *Stochastic Processes and Filtering Theory*, Academic Press, New York, 1970.
[17] R. E. KALMAN, *A new approach to linear filter and prediction problems*, J. Bas. Engrg., 82 (1960), pp. 35–45.
[18] R. E. KALMAN AND R. S. BUCY, *New results in linear filtering and prediction theory*, J. Bas. Engrg., 83 (1961), pp. 95–108.

[19] V. V. KNYSH, B. A. NELEPO, A. S. SARKISYAN AND I. YC. TIMCHENKO, *A dynamic-stochastic approach to the analysis of density field observations in hydrophysical test areas*, Izv. Acad. Sci., USSR, Atmos. Oceanic Phys., 14 (1978), pp. 760-769.

[20] M. KODA AND J. H. SEINFELD, *Estimation of urban air pollution*, Automatica, 14 (1978), pp. 583-595.

[21] J. J. LEENDERTSE, *Aspects of a computational model for long-period water-wave propagation*, Rand. Corp. Mem. RM-5294-PR, San Francisco, 1967.

[22] S. K. LIU AND J. J. LEENDERTSE, *Multidimensional numerical modelling of estuaries and coastal seas*, Adv. Hydroscience, 13 (1978), pp. 95-164.

[23] M. K. MEHRA, *Practical aspects of designing Kalman filters*, in Applications of Kalman Filter Theory to Hydrology, Hydraulics and Water Resources, C. L. Chiu, ed., University of Pittsburgh, Pittsburgh, 1978, pp. 89-115.

[24] M. MORF, S. S. SIDHU AND T. KAILATH, *Some new algorithms for recursive estimation in constant, linear, discrete-time systems*, IEEE Trans. Automat. Control, AC 19 (1974), pp. 315-323.

[25] W. H. MUNK AND D. E. CARTWRIGHT, *Tidal spectroscopy and prediction*, Phil. Trans. Roy. Soc. Ser. A., 259 (1966), pp. 553-581.

[26] I. S. ROBINSON, *Tides in the Bristol Channel—an analytical wedge model with friction*, Geophys. J. Roy. Astr. Soc., 62 (1980), pp. 77-95.

[27] D. J. SCHWAB, *Analytical and empirical response functions for storm surges on Lake Erie*, in Proc. International Symposium on Long Waves in the Ocean, Manuscript Report Series, No. 53, Canadian Marine Sciences Directorate, Ottawa, 1979, pp. 140-144.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 2, March 1987
© 1987 Society for Industrial and Applied Mathemtics
011

NUMERICAL SPECIFICATION OF DISCRETE LEAST FAVORABLE PRIOR DISTRIBUTIONS*

PETER J. KEMPTHORNE†

Abstract. A broad class of statistical decision problems are solved by minimax procedures which are Bayes with respect to discrete least favorable prior distributions. A general algorithm for specifying such distributions is presented which exploits the statistical properties of minimax procedures. The algorithm is demonstrated by characterizing the procedure which simultaneously minimizes a Bayes risk and a maximum risk under different loss functions in a simple multi-objective decision problem.

Key words. Bayes risk, iterative optimization, least favorable distribution, minimax procedure, multi-objective decision problem

AMS(MOS) subject classifications. 65U05, 62C20, 62C25

1. Introduction. Many important problems in estimation, optimal design, robust statistics, and multi-objective decision making can be expressed as minimax problems. Typically, finding a minimax procedure which solves a given problem proceeds by considering the dual problem of finding the least favorable prior distribution for the corresponding Bayes problem. The associated duality theory ensures that if both problems have solutions, then the Bayes procedure with respect to the least favorable prior distribution solves the minimax problem. Explicit analytic solutions have been elusive in general. Still, closed form solutions have been found in many special cases; see, e.g., Hodges and Lehmann [1950], [1952], Wald [1950], Huber [1981], Kiefer [1974], and Casella and Strawderman [1981].

For a broad class of problems, the least favorable prior distributions are discrete with finite support. While explicit solutions may be intractable, numerical solutions are possible since such distributions are specified by a finite number of parameters. We present a general algorithm which solves for the least favorable prior distribution in such problems. In addition to proving that the algorithm converges, we demonstrate its success at solving a particular multi-objective decision problem. The next section begins with a description of the standard setup for minimax problems. We then review those results of classical minimax theory which form the basis for the algorithm.

2. Minimax problems: classical theory. Let X be a random variable whose distribution is in $\{P_\theta, \theta \in \Theta\}$, a family indexed by the parameter θ. Consider the problem of making an inference about the unknown parameter θ given an observation of X. A solution consists of a decision procedure δ which identifies a particular inference $\delta(x)$ for each value x which could be observed. Specifying the "optimal" decision procedure is a statistical decision problem. Its identification requires the ability to assess and compare the performance of different procedures.

A risk function $R(\delta, \theta)$ characterizes the performance of a decision procedure, δ, for each parameter $\theta \in \Theta$. Usually the risk function is defined in terms of an underlying loss function. Letting $\mathcal{A}$ denote the collection of all possible inferences, i.e. "actions" $a = \delta(x)$, a loss function maps $\mathcal{A} \times \Theta$ into $\mathcal{R}$, the space of real numbers. It defines the cost $L(a, \theta)$ of making inference a when θ is true. The risk of a procedure δ when θ is true is then the average loss incurred from using δ, i.e. $E_\theta L(\delta(X), \theta)$, where E_θ

* Received by the editors December 17, 1984, and in revised form June 25, 1985. This work was facilitated in part by National Science Foundation grants MCS-82-01820 and SES-8401422.

† Department of Statistics, Harvard University, Cambridge, Massachusetts 02138.

denotes expectation with respect to P_θ, the distribution of X. More generally, the risk function is any function whose values induce a preference ordering on the space of decisions for each $\theta \in \Theta$. There need not be an underlying loss function. In § 4 we consider a multi-objective decision problem for which this is the case.

Let $\mathcal{D}$ denote the collection of all decision procedures. By comparing risk functions one can assign preferences among different procedures in $\mathcal{D}$ and identify the optimal procedure. Under the minimax principle the optimal procedure δ_M minimizes the maximum risk. The minimax problem is to specify the minimax procedure.

Minimax problems are often solved by considering the corresponding Bayes problems. Under the Bayes principle, a distribution π is specified on the parameter space Θ, and the relative performance of a procedure δ is characterized by its Bayes risk: $r(\delta, \pi) = \int_\Theta R(\delta, \theta)\pi(d\theta)$. Given π, the "optimal" procedure δ_π minimizes the Bayes risk.

Define the Bayes risk $\underline{r}$ of a distribution π on Θ to be the Bayes risk of the Bayes procedure δ_π, i.e.

$$\underline{r}(\pi) = \underline{r}(\delta_\pi, \pi).$$

A distribution π^* is "least favorable" if its Bayes risk is no smaller than that of any other distribution, i.e.

$$\underline{r}(\pi^*) \geqq \underline{r}(\pi),$$

for any distribution π on Θ. Subject to the decision problem satisfying sufficient regularity conditions, a least favorable prior distribution exists and the corresponding Bayes procedure is minimax. We refer the reader to Wald [1950] and Brown [1976] for extensive discussions of the required conditions on the decision problem. They give results similar to Theorems 2.2 and 2.3 below. Due to the complexity and inaccessibility of their treatments, we provide short proofs for the completeness of the present paper.

Suppose that the decision problem satisfies the following assumptions:

A1. For every distribution π on Θ, the Bayes procedure δ_π is unique, almost everywhere for all θ.

A2. If $\{\pi_i, i = 1, 2, \cdots\}$ is any sequence of distributions on Θ which converges weakly to π, then the risk functions $\{R(\delta_{\pi_i}, \cdot), i = 1, 2, \cdots\}$ of the corresponding Bayes procedures converge uniformly on compacts to the risk function $R(\delta_\pi, \cdot)$ of the Bayes procedure with respect to π.

The first assumption ensures that the risk function of a Bayes procedure is unique so that the limits of such functions in A2 are well defined.

We first prove a theorem concerning the directional derivative of the Bayes risk on the space of probability distributions on Θ.

THEOREM 2.1. *Under assumptions* A1 *and* A2, *let* π_0 *be any distribution with Bayes procedure* δ_{π_0}. *For any other distribution* π_1, *the directional derivative of the Bayes risk* $\underline{r}(\cdot)$ *about* π_0 *in the direction* π_1 *is bounded from below by*

$$r(\delta_{\pi_0}, \pi_1) - \underline{r}(\pi_0).$$

Proof. Given π_0 and π_1, for $0 \leqq \alpha \leqq 1$, let π_α be the "α-mixture" of the distributions π_0 and π_1. If F_0, F_1, and F_α are the respective distribution functions then

$$F_\alpha(\theta) = (1-\alpha)F_0(\theta) + \alpha F_1(\theta),$$

for $\theta \in \Theta$. Let δ_α denote the Bayes procedure with respect to π_α, for $0 \leqq \alpha \leqq 1$. The

lower bound for the directional derivative of $\underline{r}(\cdot)$ about π_0 in the direction π_1 is

$$\nabla \underline{r}(\pi_0; \pi_1) = \liminf_{\alpha \to 0^+} \frac{\underline{r}(\pi_\alpha) - \underline{r}(\pi_0)}{\alpha}.$$

Since π_α is a linear mixture of π_0 and π_1, it follows that

$$\underline{r}(\pi_\alpha) = r(\delta_\alpha, \pi_\alpha) = (1-\alpha) r(\delta_\alpha, \pi_0) + \alpha r(\delta_\alpha, \pi_1)$$

and consequently

$$\frac{\underline{r}(\pi_\alpha) - \underline{r}(\pi_0)}{\alpha} = [r(\delta_\alpha, \pi_1) - \underline{r}(\pi_0)] + \frac{(1-\alpha)}{\alpha}[r(\delta_\alpha, \pi_0) - \underline{r}(\pi_0)].$$

Taking the limit as α converges to 0, the first term converges to $r(\delta_0, \pi_1) - \underline{r}(\pi_0)$, because $\lim_{\alpha \to 0} R(\delta_\alpha, \theta) = R(\delta_0, \theta)$ uniformly for θ in compacts by A2. The result follows upon noting that the second term is bounded below by 0 because $r(\delta_\alpha, \pi_0) \geqq \inf_{\delta \in \mathscr{D}} r(\delta, \pi_0) = \underline{r}(\pi_0)$, for all $\alpha > 0$. □

For the least favorable distribution π^*, the minimaxity of the Bayes procedure δ_{π^*} follows from a corollary to the following.

THEOREM 2.2. (a) *Under assumptions* A1 *and* A2 *a distribution* π^* *is least favorable only if*

$$\max_{\theta \in \Theta} R(\delta_{\pi^*}, \theta) = \underline{r}(\pi^*). \tag{2.1}$$

(b) *The converse is true in general.*

Proof. For (a) let π^* be least favorable and let δ_{π^*} be the corresponding Bayes procedure. Suppose that $\max_\theta R(\delta_{\pi^*}, \theta) \neq \underline{r}(\pi^*)$. Since $\underline{r}(\pi^*) = \int_\Theta R(\delta_{\pi^*}, \theta)\pi^*(d\theta)$, there must exist a θ_1 such that $R(\delta_{\pi^*}, \theta_1) > \underline{r}(\pi^*)$. By Theorem 2.1, the directional derivative of $\underline{r}$ about π^* in the direction of the distribution π_1 giving unit weight to θ_1 is bounded below by $R(\delta_{\pi^*}, \theta_1) - \underline{r}(\pi^*)$ which is positive. It follows that for some $\alpha > 0$ the α-mixture between π^* and π_1, say π^*_α satisfies $\underline{r}(\pi^*_\alpha) > \underline{r}(\pi^*)$. This contradicts the assumption that π^* is least favorable, hence (2.1) must be true. Conditions less restricted than A1, A2 for which (a) is true are given in Wald [1950] and Brown [1976].

To prove (b), suppose that π^* is not least favorable. Then, there exists a distribution π' such that $\underline{r}(\pi') > \underline{r}(\pi^*)$. It follows that $\max_\theta R(\delta_{\pi^*}, \theta) \geqq \underline{r}(\delta_{\pi^*}, \pi') \geqq \underline{r}(\pi') > \underline{r}(\pi^*)$, i.e. (2.1) is false, and the proof is complete. □

COROLLARY. *If* π^* *is least favorable then the Bayes procedure* δ_{π^*} *is minimax as well as an "equalizer," i.e.*

$$R(\delta_{\pi^*}, \theta') = \underline{r}(\pi^*)$$

for all θ' *in the support of* π^*.

Proof. The Bayes risk $\underline{r}(\pi^*)$ is the average risk of δ_{π^*} with respect to π^*. If π^* gives positive probability to any θ values for which the risk of δ_{π^*} is less than the maximum risk, then the Bayes risk will be smaller than the maximum risk, a contradiction of (2.1). □

Suppose that in addition to conditions A1 and A2 the decision problem satisfies:

A3. The parameter space Θ is a compact separable metric space.

A4. The risk function $R(\delta, \theta)$ for any (measurable) decision procedure, δ, is an upper-semicontinuous function of θ.

In the next theorem we prove the existence of a least favorable prior distribution. We construct a sequence of distributions which converges to a least favorable distribution. The logic of the proof is similar to that applied in the construction of optimal experimental designs. See, for example, Wynn [1970], [1972], Fedorov [1972]; also, Wu [1976]. It will provide the foundation for the algorithm described in § 3.

THEOREM 2.3. *Under assumptions* A1–A4, *a least favorable distribution* π^* *exists.*

Proof. Let π_1 be any distribution on Θ. Then, construct the sequence $\{\pi_i, i = 2, 3, \cdots\}$ by defining π_{i+1} in terms of π_i as follows. Given π_i with distribution function F_i, let θ_i maximize the risk of the Bayes procedure δ_{π_i}. The existence of θ_i is guaranteed by the continuity of $R(\delta_{\pi_i}, \theta)$ and the compactness of Θ. Let $\{\pi_{i,\alpha}, 0 \leqq \alpha \leqq 1\}$ be the family of mixture distributions between π_i and the distribution with unit weight at θ_i. If θ is real-valued then $\pi_{i,\alpha}$ has distribution function

$$F_{i,\alpha}(\theta) = (1-\alpha)F_i(\theta) + \alpha 1_{(-\infty,\theta_i]}(\theta)$$

for $\theta \in \Theta$. Let α_i specify the least favorable mixture distribution, i.e.

$$\underline{r}(\pi_{i,\alpha_i}) \geqq \underline{r}(\pi_{i,\alpha}),$$

for all $0 \leqq \alpha \leqq 1$. The existence of α_i is ensured by assumptions A2 and A4. Define the distribution π_{i+1} to be the least-favorable mixture π_{i,α_i}.

By A3, there exists a subsequence of the integers, $\{n_j\}$, for which $\{\pi_{n_j}\}$ converges weakly, say to π^*, and $\{\theta_{n_j}\}$ converges, say to θ^*. We now prove that π^* is least favorable. To simplify the notation in the discussion below the subscript j will refer to elements of the subsequence $\{n_j\}$.

Suppose that π^* is not least favorable. There must then exist a distribution π' for which $\underline{r}(\pi') > \underline{r}(\pi^*)$. Note that: $\underline{r}(\delta_{\pi^*}, \pi') \geqq \underline{r}(\delta_{\pi'}, \pi') = \underline{r}(\pi') > \underline{r}(\pi^*)$. This implies that $R(\delta_{\pi^*}, \theta^*) > \underline{r}(\pi^*)$ by the following reasoning. The limit of the sequence $\{\theta_j\}$ is θ^*. Each θ_j maximizes the risk of the Bayes procedure δ_{π_j} and the risk functions corresponding to the sequence of Bayes procedures converge to the risk function of δ_{π^*}. Consequently, it must be that θ^* maximizes the risk of δ_{π^*}, and so $R(\delta_{\pi^*}, \theta^*) \geqq \underline{r}(\delta_{\pi^*}, \pi')$.

Let π^*_α denote the α-mixture between π^* and π_{θ^*}, the distribution with unit weight at θ^*. Because the directional derivative of the Bayes risk about π^* in the direction of π_{θ^*} is positive, the least favorable mixture, specified by $\alpha^* > 0$, satisfies

$$\underline{r}(\pi^*_{\alpha^*}) > \underline{r}(\pi^*). \tag{2.2}$$

Consider the sequence of distributions $\{\pi_{j,\alpha^*}\}$, where π_{j,α^*} is the α^* mixture of π_i and π_{θ_i}, where $i = n_j$, the jth member of the subsequence. Since $\{\pi_j\}$ converges weakly to π^* and $\{\theta_j\}$ converges to θ^*, the sequence $\{\pi_{j,\alpha^*}\}$ converges weakly to $\pi^*_{\alpha^*}$ and so $\{r(\pi_{j,\alpha^*})\}$ converges to $r(\pi^*_{\alpha^*})$. But for each j, the distribution π_{n_j+1} is the least favorable mixture between π_{n_j} (i.e. π_j) and $\pi_{\theta_{n_j}}$ (i.e. π_{θ_j}). So $\underline{r}(\pi_{n_j+1}) > \underline{r}(\pi_{j,\alpha^*})$. Now because $\{\underline{r}(\pi_i)\}$ is monotone increasing, it must be that

$$\lim_i \underline{r}(\pi_i) \geqq \lim_j \underline{r}(\pi_{j,\alpha^*}),$$

i.e.

$$\underline{r}(\pi^*) \geqq \underline{r}(\pi^*_{\alpha^*}).$$

This contradicts (2.2), and thus π^* must indeed be least favorable. □

The proof of Theorem 2.3 demonstrates that any least favorable prior distribution can be approximated arbitrarily closely with a discrete distribution on Θ. In fact, for a large class of problems, this approximation is *exact*, i.e. the least favorable prior distribution is discrete. We state a fifth assumption for the decision problem.

A5. θ is real-valued and for any (measurable) procedure δ, the risk function $R(\delta, \theta)$ is an analytic function of θ.

THEOREM 2.4. *If the decision problem satisfies* A1–A5, *then a least-favorable distribution* π^* *has just one of the following properties*:

(a) *the risk of the Bayes procedure* δ_{π^*} *is constant on* Θ, *or*

(b) *the support of* π^* *is discrete and finite.*

Proof. By Theorem 2.3 the risk of the Bayes procedure δ_{π^*} is constant on the support of π^* and equals its maximum value. Since $R(\delta_{\pi^*}, \theta)$ is analytic in θ, if the support of π^* contains an accumulation point, the function must be constant on Θ, i.e. (a). Otherwise, the support of π^* must be finite, hence discrete. This analyticity argument was used by Bickel [1983] in a related problem. □

This result can be extended to special cases when Θ is multidimensional. Kempthorne [1985] discusses the general extension and Kempthorne [1986] applies the theory to characterize optimal minimax estimators of a multivariate normal mean under squared error loss.

In the context of particular minimax problems, it is sometimes the case that procedures with constant risk cannot be Bayes. If it exists, the least favorable prior distribution is then discrete. The algorithm in the proof of Theorem 2.3 might be applied to finding such a distribution. However, the support sets of the distributions in the sequence defined there are increasing with each successive member having one more point in its support together with the existing support points. For a discrete least favorable prior distribution with finite support, this algorithm might not converge in a finite number of steps. This is because the added support points, while converging to the support points may equal them only in the limit. In the next section we present an algorithm which constructs a sequence of discrete distributions whose successive support sets change by adjusting the existing points in the support as well as by adding new points to the support.

3. The algorithm. Suppose the decision problem satisfies assumptions A1–A5 given in § 2. The existence of a least favorable prior distribution π^* is guaranteed by Theorem 2.3. If in addition no decision procedure with constant risk can be Bayes, then π^* must be discrete with finite support by Theorem 2.4.

We describe an algorithm for specifying such a least favorable prior distribution numerically. Regarding computational considerations we assume only that given any discrete prior distribution with finite support, we can calculate $R(\delta_\pi, \theta)$, the risk of the Bayes procedure corresponding to π for any value $\theta \in \Theta$. The algorithm has four basic steps which iterate if necessary until the solution is achieved (to within acceptable tolerances).

Step 1: Specify π_1, an initial guess for the least favorable distribution.

Let $\{\theta_{i,1}, i = 1, 2, \cdots, n_1\}$ be the points of π_1's support and let $\{p_{i,1}, i = 1, 2, \cdots, n_1\}$ be the corresponding probabilities, i.e. $\pi_1(\theta_{i,1}) = p_{i,1}$, with $p_{i,1} > 0$ and $\sum_{i=1}^{n_1} p_{i,1} = 1$.

Step 2: Determine π_2, a local maximum of the Bayes risk $\underline{r}$.

With the size of the support fixed at n_1 let g be the function with $2n_1$ arguments: $(p_1, p_2, \cdots, p_n, \theta_1, \theta_2, \cdots, \theta_{n_1})$, which equals the Bayes risk of the distribution π for which $\pi(\theta_i) = p_i$, $i = 1, 2, \cdots, n_1$. With starting values given by the distribution π_1, find the distribution π_2 supported on n_1 points which maximizes (locally) g, the Bayes risk.

The maximization of g is a straightforward maximization of a function on $\mathcal{R}^{2n}$ subject to the constraints:

$$p_i \geqq 0, \qquad i = 1, 2, \cdots, n_1,$$

$$\sum_{i=1}^{n_1} p_i = 1,$$

$$\theta_i \in \Theta, \qquad i = 1, 2, \cdots, n_1.$$

Step 3: Check whether π_2 is least favorable, i.e. a global maximum of $\underline{r}(\cdot)$.

Find $\theta_{*,2}$, the parameter value(s) maximizing the risk function of the Bayes procedure δ_{π_2}. If $R(\delta_{\pi_2}, \theta_{*,2}) = \underline{r}(\pi_2)$ then stop; the distribution π_2 is least favorable and δ_{π_2} is minimax. Otherwise, proceed to the next step.

Step 4: Specify a new guess π_1' for the least favorable distribution and return to Step 2.

Define π_1' to be a mixture between the distribution π_1 and $\pi_{\theta_{*,2}}$, the point distribution at $\theta_{*,2}$. Specifically, for some $\alpha > 0$ (e.g., $\alpha = .1$), let

$$\pi_1'(\theta) = \begin{cases} (1-\alpha)p_{i,2}, & \theta = \theta_{i,2}, \quad i = 1, 2, \cdots, n_1, \\ \alpha, & \theta = \theta_{*,2}. \end{cases}$$

The size of the support of π_1' is $n_1' = n_1 + 1$.

Discussion of the algorithm. The success of the algorithm does not depend on the initial guess π_1. However, an intelligent guess can substantially reduce the extent of the computational effort. We discuss this further in § 4 in the context of a particular problem.

To implement the second step of the algorithm we express the constrained maximization on $\mathcal{R}^{2n}$ as an unconstrained maximization problem on $\mathcal{R}^{2n-1}$, and apply a quasi-Newton maximization procedure which uses differences to approximate derivatives[1]. This maximization procedure yields a distribution (π_2) which is assumed only to be a local maximum of the Bayes risk function. The reason for this is that the Bayes risk is not a concave function on the space of discrete distributions with support size fixed. This contrasts with the fact that the Bayes risk is a concave function on the space of all distributions (under linear mixtures). Interestingly, the Bayes risk is also concave on the space of distributions whose support sets are constrained to lie in any fixed subset of Θ (finite or infinite). Also, the Bayes procedure for the least favorable member of this space of distributions is an equalizer, i.e. its risk function is equal across all the points of that member's support set.

When implementing Step 2, we use this last fact to check that π_2 is indeed a local maximum. Specifically, we confirm that the Bayes procedure δ_{π_2} is an equalizer. If it is not (to within acceptable tolerances) then the convergence criteria specifying the maximization procedure were too weak. These are reset to more stringent values and the maximization procedure is reapplied.

The third step of the algorithm is based on Theorem 2.2. To check whether a distribution is least favorable, we calculate the maximum risk of its Bayes procedure. If the maximum risk equals the Bayes risk then the distribution is least favorable. Otherwise, the Bayes risk will increase by adding the parameter which maximizes the risk to the support of the distribution; i.e. Step 4.

[1] We use Fortran subroutine VA10 from the Harwell Subroutine Library (1984).

Determining the maximum risk is very intensive numerically. When the support set has more than one point, the risk function of the minimax procedure is multi-modal. We employ a two stage search for the global maximum of the risk function. First, a grid search identifies subintervals of the parameter space which branch local maxima. Second, on each subinterval a golden-section search determines the location and value of the local maximum.

Note that π_2, the distribution with support size n_1 determined in Step 2, is a local maximum among all distributions with support size less than or equal to n_1. Even if the parameter maximizing the risk of the Bayes procedure δ_{π_2} is not in the support, the true least favorable prior distribution may have support size equal to n_1. Step 4 adds this point to the support and in the next iteration, Step 2 finds a (local) maximum of the Bayes risk on the space of distributions with support size equal to n_1+1. The maximizing distribution in this step may have support size equal to n_1, however, because the locations of two support points may coincide or the probability of one support point may equal zero.

The computational algorithm described here is very similar to the method for constructing the sequence of distributions converging to the least favorable distribution in the proof of Theorem 2.3. In fact, we can prove the following.

THEOREM 3.1. *If the decision problem satisfies conditions* A1-A5, *the algorithm given in this section converges to the least favorable prior distribution.*

Proof. The convergence follows using the same basic argument as the proof of Theorem 2.3. The only difference is that the computational algorithm allows the locations of the support points to vary in the second step when maximizing the Bayes risk. Note that the Bayes risk for each distribution (π_2) at the end of Step 2 will thus be no smaller than if the support set had been fixed, as in the "algorithm" of the proof of Theorem 2.2. Consequently, the sequence of distributions produced by the computational algorithm converges as fast or faster than the sequence in the proof of Theorem 2.3. □

4. An example. We apply the algorithm in § 3 to solve a minimax problem which derives from a simple multi-objective decision problem. Let X be a normal random variable with unit variance and unknown mean $\theta \in \Theta$, a compact interval of $\mathcal{R}^1$. Consider the problem of making inferences about θ given an observation of X with simultaneous concern for two objectives: estimating θ accurately and testing the hypothesis H_0: $\theta = 0$ against H_1: $\theta \neq 0$.

Decision procedures which address both objectives are preliminary-test estimators. Let δ be such a procedure. When $X = x$ is observed, a (preliminary) test of H_0 against H_1 is performed. If H_0 is accepted then the only consistent estimate $\delta(x) = 0$ is made. Otherwise, some nonzero estimate, consistent with H_1 is used. For any nonrandomized preliminary test estimator δ, the acceptance region for the preliminary test of the null hypothesis, H_0: $\theta = 0$, is: $\{x: \delta(x) = 0\}$. If δ is randomized, then the chance that δ specifies a nonzero estimate of θ as a function of x defines the critical function for the preliminary test.

We seek to characterize those preliminary test estimators which simultaneously minimize the maximum squared error risk and the chance of a Type I error for testing H_0; $\theta = 0$. To assess performance with regard to the estimation and hypothesis-testing objectives we specify two separate loss functions and decision principles for comparing the corresponding risk functions. This problem is similar to one addressed by Cohen [1965]. However, he used a single "hybrid" loss function to judge performance with respect to the two objectives.

For estimation, let $L_1(a, \theta) = (a - \theta)^2$ be the loss when the true mean θ is estimated by a. For any procedure δ, the risk is

$$R_1(\delta, \theta) = E_\theta(\delta(X) - \theta)^2,$$

the mean-squared error from using δ to estimate θ. To compare the estimation performance of different procedures we apply the minimax principle and prefer those procedures δ with smaller maximum risk:

$$\bar{R}_1(\delta) = \max_{\theta \in \Theta} R_1(\delta, \theta).$$

Turning to the hypothesis testing objective, let $L_2(a, \theta)$ denote the usual zero-one loss function for testing H_0: $\theta = 0$ against H_1: $\theta \neq 0$, i.e. for estimating θ with a

$$L_2(a, \theta) = \begin{cases} 0 & \text{if } a = 0 \text{ and } \theta = 0, \\ & \text{or } a \neq 0 \text{ and } \theta \neq 0, \\ 1 & \text{if } a \neq 0 \text{ and } \theta = 0, \\ & \text{or } a = 0 \text{ and } \theta \neq 0. \end{cases}$$

For this loss, the risk function $R_2(\delta, \theta)$ of any procedure δ is the standard operating characteristic function for the preliminary test defining δ. At $\theta = 0$, its value is the size of the preliminary test, and for $\theta \neq 0$ it gives one minus the power of the test.

We apply the Bayes principle to compare the hypothesis-testing performance of procedures. Let π_2 be a distribution on Θ and let $r_2(\delta, \pi_2) = \int_\Theta R_2(\delta, \theta)\pi_2(d\theta)$ denote the Bayes risk of the procedure δ with respect to π_2. While the distribution π_2 would ordinarily model the statistician's beliefs about the unknown parameter θ, we shall assume that π_2 places all its probability at $\theta = 0$, the null hypothesis. The Bayes risk $r_2(\delta, \pi_2)$ then measures the size of the preliminary test defining a procedure δ, i.e. the chance that a Type I error is made using δ.

The multi-objective problem is to find those procedures δ which minimize both $\bar{R}_1(\delta)$ and $r_2(\delta, \pi_2)$ simultaneously. One approach is to minimize the maximum squared error risk subject to the constraint that the size of the preliminary test specifying a procedure not exceed some fixed level α, for $0 \leqq \alpha \leqq 1$. By allowing α to vary between zero and one, the collection of solution procedures comprise the pareto-optimal procedures for the multi-objective decision problem. For each α, $(0 \leqq \alpha \leqq 1)$, this multi-objective decision problem is a "minimax-Bayes compromise problem" (Kempthorne [1983]):

Find the procedure $\delta_0 \in \mathcal{D}$ which

$$\begin{aligned} &\text{minimizes } \max_{\theta \in \Theta} R_1(\delta, \theta), \\ &\text{subject to } r_2(\delta, \pi_2) \leqq \alpha. \end{aligned} \tag{4.1}$$

The solution procedure is the "minimax squared-error preliminary-test estimator of level α." We solve for δ_0 by considering an equivalent minimax problem for a generalized risk function.

Because $\bar{R}_1(\delta)$ is convex and $r_2(\delta, \pi_2)$ is linear on $\mathcal{D}$, the constrained optimization problem (4.1) can be re-expressed as an unconstrained optimization problem:

For some λ, $0 \leqq \lambda \leqq 1$ (depending on α) find the procedure $\delta^* \in \mathcal{D}$ which

$$\text{minimizes } \lambda \bar{R}_1(\delta) + (1 - \lambda) r_2(\delta, \pi_2). \tag{4.2}$$

Kempthorne [1983] proves this claim. Let $\lambda = \lambda_0$ be the particular value for which the problems (4.1) and (4.2) are equivalent. Define the generalized risk function R^* as

$$R^*(\delta, \theta) = \lambda_0 R_1(\delta, \theta) + (1 - \lambda_0) r_2(\delta, \pi_2), \tag{4.3}$$

i.e. a weighted average of the squared-error risk function R_1 and the Bayes risk function r_2. If a procedure δ^* is minimax with respect to R^*, then δ^* must solve (4.2). Thus, the original multi-objective problem (4.1) is equivalent to a minimax problem defined by a generalized risk function. Note, we call R^* a generalized risk function because it cannot be expressed as a simple risk with loss function depending only on the procedure's estimate and the parameter.

To specify δ^* we apply the classical minimax theory outlined in § 2, i.e. we characterize the least favorable prior distribution π^* and its Bayes procedure will be minimax for the risk R^*. First, however, we must be able to solve for Bayes procedures using the generalized risk R^*.

For the risk function R^* a Bayes procedure with respect to the prior distribution π_1 minimizes the Bayes risk

$$r^*(\delta, \pi_1) = \int_\Theta R^*(\delta, \theta)\pi_1(d\theta).$$

This Bayes risk satisfies

$$r^*(\delta, \pi_1) = \lambda_0 r_1(\delta, \pi_1) + (1-\lambda_0) r_2(\delta, \pi_2),$$

where r_1 is the Bayes risk function for the risk R_1. Assuming the order of the integrations with respect to x and θ in each Bayes risk can be reversed, it is easy to see that for each x the Bayes procedure δ_{π_1} uses that estimate a which minimizes

$$\lambda_0 p_1(x) R_1(a|x) + (1-\lambda_0) p_2(x) R_2(a|x),$$

where for $i = 1, 2$: $R_i(a|x)$ is the posterior risk of estimate a when the risk is R_i and the prior distribution is π_i; and $p_i(\cdot)$ is the marginal density for X under the prior distribution π_i.

Kempthorne [1983, p. 576] considers this example and shows that

$$\delta_{\pi_1}(x) = \begin{cases} 0, & x \in A, \\ E_1(\theta|x), & x \notin A, \end{cases} \tag{4.4}$$

where: $E_1(\theta|x)$ is the mean of the posterior distribution for θ given $X = x$ when the prior distribution is π_1; and A is the set

$$\{x: [E_1(\theta|x)]^2 \leqq [(1-\lambda_0)/\lambda_0][p_2(x)/p_1(x)][2\pi_2(\{\theta = 0\}|x)] - 1]\},$$

with $\pi_2(\cdot|x)$ denoting the posterior distribution for θ given $X = x$ when the prior distribution is π_2. If the prior distribution π_1 is symmetric about the origin and π_2 gives unit probability to $\{\theta = 0\}$, it can be shown that A is a closed interval, symmetric about the origin.

We turn to the problem of finding the least favorable prior distribution π^* and hence the minimax procedure δ^*. To apply the algorithm of § 3 we must confirm that assumptions A1-A5 are satisfied, and that no Bayes procedure for risk R^* has constant risk.

The results in Sacks [1963] prove that A1 and A2 are satisfied when the risk function is R_1, mean squared error. The proof that they also hold for the generalized risk R^* is straightforward given (4.3), the form of Bayes procedures for R^* in terms of the Bayes procedure $(E_1(\theta|x))$ for R_1. The parameter space Θ is a compact interval of $\mathscr{R}^1$ (A3) by assumption. Because the normal density is in the exponential family, the risk of any (measurable) decision procedure is an analytic function of θ, i.e. A5 (and A4) hold.

To prove the last condition suppose that δ is a procedure with constant R^* risk. This implies that its R_1 risk is constant also. By invariance arguments it can be shown that $\delta(X) \equiv X + C$ for some constant C; see, e.g., Lehmann [1983]. But no Bayes procedure for the risk R^* can have this form. This is proven using the fact that no proper prior distribution π_1 exists for which the mean of the posterior distribution satisfies $E_1(\theta|x) = x + C$, on a set of positive measure. Consequently, it is impossible to express $\delta(X) = X + C$ in the form of (4.4) for some prior distribution π_1.

We applied the algorithm in § 3 to solve the multi-objective decision problem (4.1) with level $\alpha = .0455$ and parameter space $\Theta = [-7, 7]$. By invariance arguments, the least favorable prior distribution π^* for the generalized risk R^* and the acceptance region A^* for the preliminary test used by the corresponding Bayes procedure are symmetric about $\{\theta = 0\}$. Since $P(X \notin A^* | \theta = 0) = .0455$, A^* must be $[-2.0, 2.0]$, and the numerical specification of π^* was found to be:

$$\pi^*(\theta) = \begin{cases} .4677, & \theta = \pm 2.177, \\ .0248, & \theta = \pm 4.744, \\ .0024, & \theta = \pm 7.000. \end{cases}$$

These values were obtained with an Apollo DN300 computer using a Fortran program of the algorithm in § 3 together with Fortran subroutines calculating the risk and Bayes risk of Bayes procedures with respect to (finite) discrete prior distributions. The risks are calculated by numerical integration using "QUANC 8" a subroutine given by Forsythe, Malcolm and Moler [1977] which uses an adaptive 8-panel quasi-Newton-Cotes method. Our calculations suggest that the probabilities (locations) of points in the support of π^* are accurate to within 10^{-4} (10^{-3}).

The numerical approximation to the minimax procedure δ^* is thus given by (4.4) with A^* and π^*, i.e.

$$\delta^*(X) = \begin{cases} 0, & |X| \leqq 2.0, \\ E^*(\theta|X), & |X| > 2.0, \end{cases}$$

where $E^*(\theta|X)$ is the mean of the posterior distribution for θ given $X = x$ when the prior distribution for θ is π^*. We plot the mean squared error risk of δ^* in Fig. 4.1. The risk of δ^* is symmetric about $\{\theta = 0\}$ so we only plot the risk on the interval $0 \leqq \theta \leqq 7.0$. Note that the maximum risk for $\theta \in [-7, 7]$ is achieved only at the points of support for π^*. Since the generalized risk R^* is maximized whenever the mean squared error risk is maximized, it follows (by Theorem 2.3) that π^* is least favorable and δ^* is minimax.

An alternative solution. An ad hoc solution to the multi-objective decision problem given by $\alpha = .0455$, and $\Theta = \mathcal{R}^1$ is the "classical" preliminary-test estimator:

$$\delta_c(x) = \begin{cases} 0, & |x| \leqq 2.0, \\ x, & |x| > 2.0. \end{cases}$$

This estimator uses a uniformly most powerful unbiased test of H_0: $\theta = 0$ against H_1: $\theta \neq 0$ and estimates θ by maximum likelihood under whichever hypothesis is accepted.

Comparing δ_c with δ^*, note first that both perform identically with regard to testing H_0 against H_1. They each commit a Type I error with probability .0455 and have identical power functions (risk function R_2). For estimation performance consider Fig. 4.2, a graph of each procedure's squared error risk on the same plot. The "optimal" preliminary test procedure δ^* has maximum squared error risk equal to 2.091 while

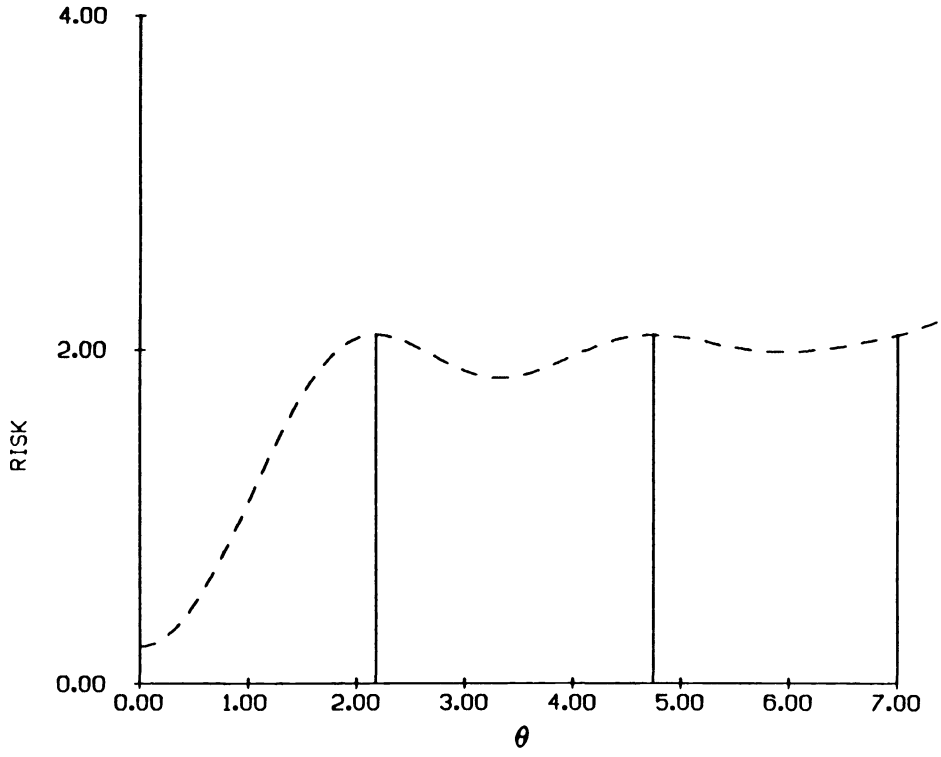


FIG. 4.1. *Mean squared error risk of the optimal preliminary test estimator of level* $\alpha = 0.0455$.

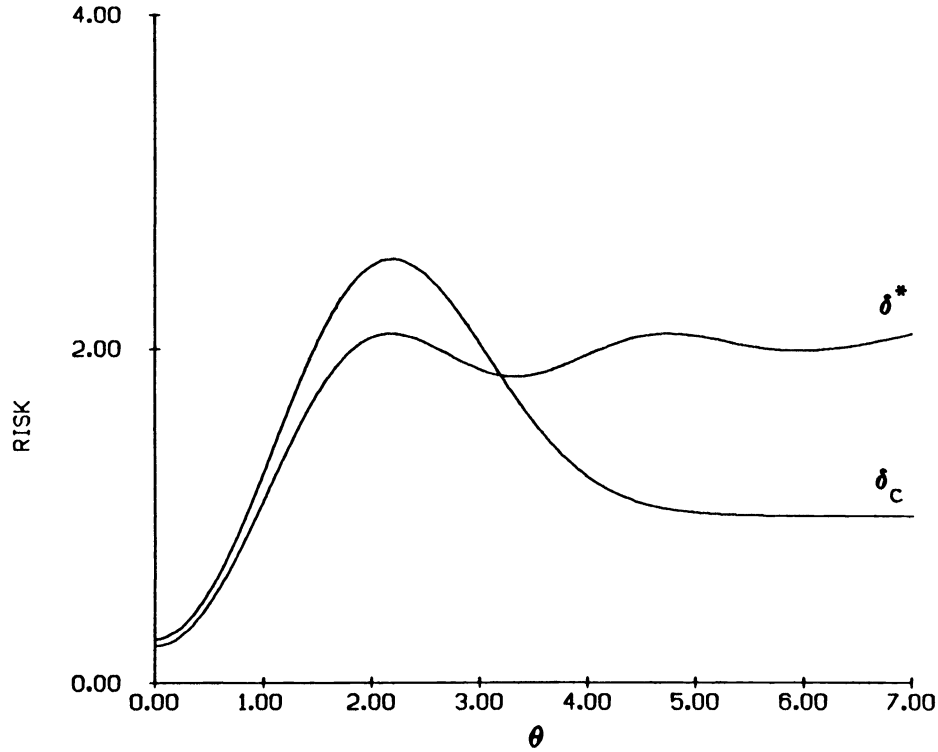


FIG. 4.2. *Mean squared error risks of the optimal and classical preliminary test estimators.*

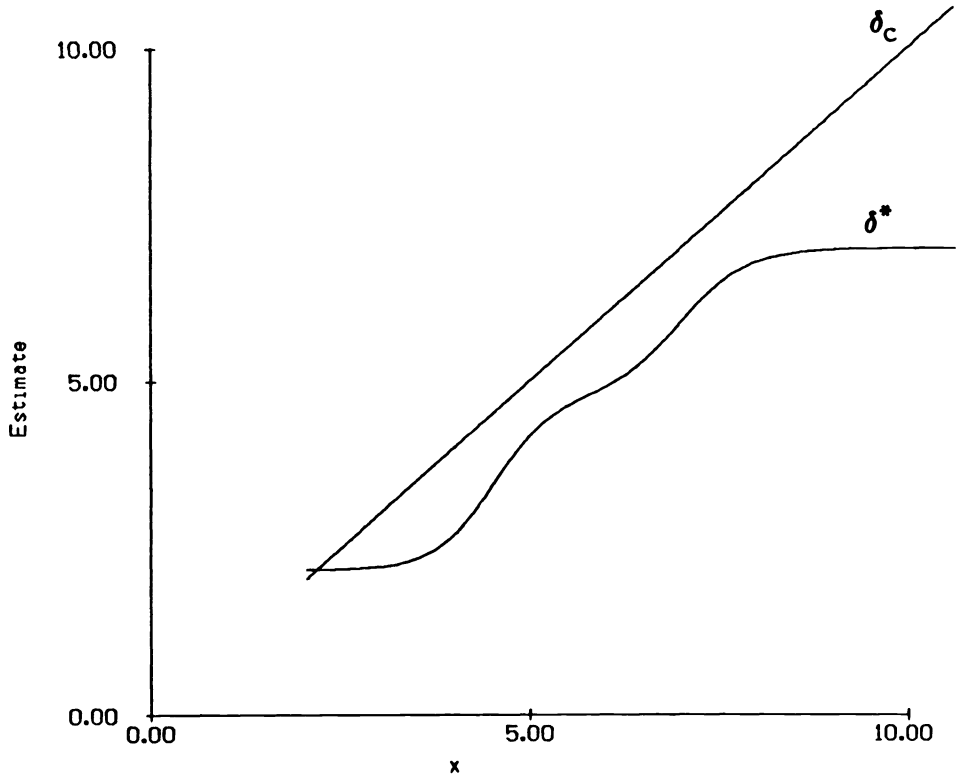


FIG. 4.3. *The optimal and classical preliminary test estimators.*

that of the classical preliminary test procedure δ_c is 2.538. Not only does δ^* have maximum squared error risk (on $\Theta = [-7, 7]$) which is about 20% below that of δ_c, its squared error risk is uniformly smaller than that of δ_c for values in a large neighborhood of $\{\theta = 0\}$.

To compare the actions of δ^* and δ_c, consider Fig. 4.3, a plot of their estimates of θ as functions of x on the interval $0 \leqq x \leqq 10$. Since they both use the same preliminary test, they coincide on $0 \leqq x \leqq 2$ (and $-2 \leqq x \leqq 0$) and estimate θ to be zero. For values of $x > 2$, δ^* estimates θ with values which are always closer to $\theta = 0$, except for values of x very close to $x = 2.0$, and the estimates never exceed the range of $\Theta = [-7, 7]$. Apparently, the rather flat nature of $\delta^*(x)$ for x values just greater than 2.0 is necessary to reduce the maximum risk below that of δ_c.

Solving a sequence of problems. To solve for the minimax squared error preliminary test estimator of level $\alpha = .0455$ with $\Theta = [-7, 7]$, we supplied the numerical algorithm with a guess at the least favorable prior distribution. For this guess, we used the least favorable prior distribution solving the same multi-objective decision problem with parameter space $\Theta = [-6.9, 6.9]$. We found that as the specification of the problem, e.g., the parameter space, changes in a continuous fashion, the changes in the solution are also continuous. In fact, we solved the sequence of problems defined by fixing $\alpha = .0455$ and varying the parameter space from $\Theta = [0, 0]$ to $[-7, 7]$.

Figure 4.4 plots the positive support points of the least favorable prior distributions as a function of the endpoint m of the parameter space $\Theta = [-m, m]$, for values of m between 0 and 7.0. Because the distributions are symmetric about $\{\theta = 0\}$, the negative

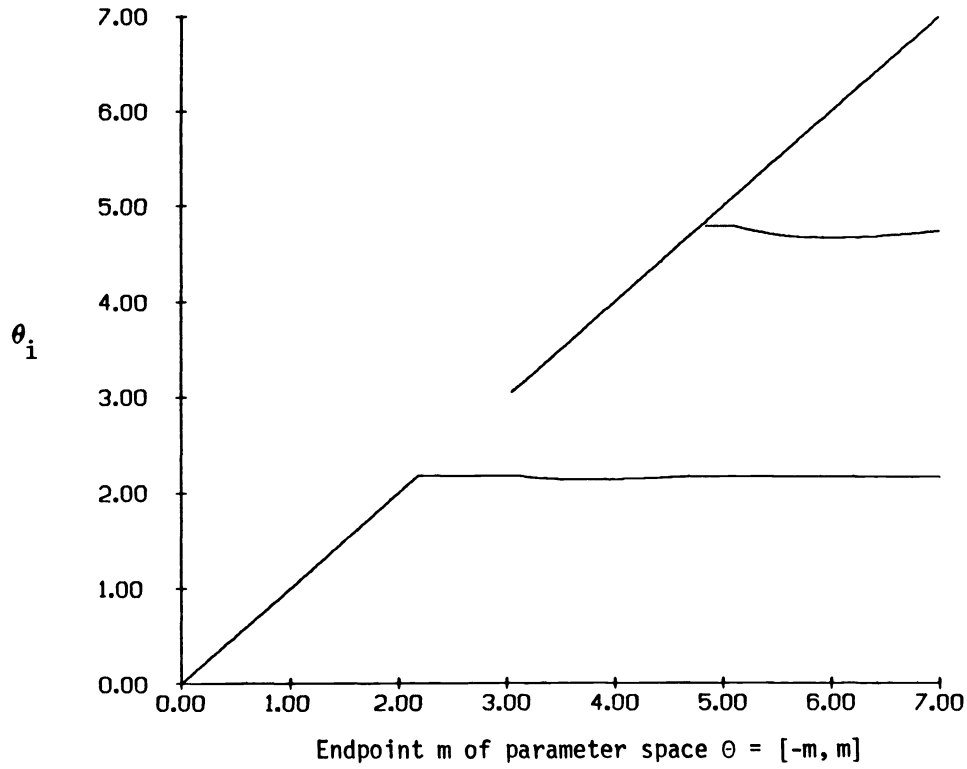


FIG. 4.4. *Positive support points for the least favorable prior distributions.*

support points are just the negatives of the positive points. When the parameter space is small, i.e. $\Theta = [-m, m]$, for $m < 2.182$, the least favorable prior distribution for the generalized risk R^* is particularly simple—it concentrates on the two endpoints of the parameter space. Then for values of m between 2.182 and ~3.10, the least favorable prior distribution does not change. For larger values of m, the two endpoints of the parameter space enter the support of the least favorable distribution. The size of the supports remains at four until m is about 5.15. Then a pair of points are added whose locations are symmetric about $\{\theta = 0\}$ in the interior of the parameter space.

Discussion. As the size of the parameter space increases, so does the size of the support of the least favorable prior distribution. Unfortunately, the least favorable prior distribution becomes increasingly difficult to specify numerically. When the prior distribution has more support points calculating the corresponding Bayes procedure is more expensive. More critically, the Bayes risk becomes very flat as a function of the locations and probabilities of the support points for the least favorable distribution. The maximizing values are thus harder to identify with the algorithm of § 3, especially when derivatives of the Bayes risk are approximated by differences.

Since the parameter space in many real problems is quite large, extending the methods of this report to solve such problems is an important direction for future research. Such advances will most likely require the development of an algorithm which specifies the least favorable prior distribution without directly maximizing the flat Bayes risk function.

Acknowledgment. I thank the referees for their comments and suggestions.

REFERENCES

P. J. BICKEL [1983], *Minimax estimation of the mean of a normal distribution subject to doing well at a point* in Recent Advances in Statistics, Festschrift for H. Chernoff, H. Rizui and D. Sigmund, eds., Academic Press, New York, pp. 511-528.

L. D. BROWN [1976], *Notes on statistical decision theory*, unpublished lecture notes, Cornell University, Ithaca, NY.

G. CASELLA AND W. STRAWDERMAN [1981], *Estimating a bounded normal mean*, Ann. Statist., 9, pp. 868-876.

A. COHEN [1965], *A hybrid problem on the exponential family*, Ann. Math. Statist., 36, pp. 1185-1206.

V. V. FEDOROV [1972], *Theory of Optimal Experiments*, Academic Press, New York.

G. E. FORSYTHE, M. A. MALCOLM AND C. B. MOLER [1977], *Computer Methods for Mathematical Computations*, Prentice-Hall, Englewood Cliffs, NJ.

Harwell Subroutine Library (1984), Atomic Energy Research Establishment, Oxfordshire, England.

J. L. HODGES AND E. L. LEHMANN [1950], *Some problems in minimax point estimation*, Ann. Math. Statist., 21, pp. 182-197.

——— [1952], *The use of previous experience in reaching statistical decisions*, Ann. Math. Statist., 23, pp. 396-407.

P. J. HUBER [1981], *Robust Statistics*, John Wiley, New York.

P. J. KEMPTHORNE [1983], *Minimax-Bayes compromise estimators*, 1983 Business and Economics Statistics Proceedings of the American Statistical Association, Washington, DC, pp. 568-573.

——— [1985], *Controlling risks under different loss functions*, Research Report NS-520, Department of Statistics, Harvard University.

——— [1986], *Optimal minimax squared error risk estimation of the mean of a multivariate normal distribution*, Research Report NS-521, Department of Statistics, Harvard University. *Stein-type multivariate estimation*, Commun. Statist., A15 (1986), special issue, to appear.

J. KIEFER [1974], *General equivalence theory for optimal designs (approximate theory)*, Ann. Statist., 2, pp. 849-879.

E. L. LEHMANN [1983], *Theory of Point Estimation*, John Wiley, New York.

J. SACKS [1963], *Generalized Bayes solutions in estimation problems*, Ann. Math. Statist., 34, pp. 751-768.

A. WALD [1950], *Statistical Decision Functions*, John Wiley, New York.

H. P. WYNN [1970], *The sequential generation of D-optimum experimental designs*, Ann. Math. Statist., 41, pp. 1655-1664.

——— [1972], *Results in the theory and construction of D-optimum experimental designs*, J. Roy. Statist. Soc., Ser. B, 34, pp. 133-147.

C. F. WU [1976], *Contributions to optimization theory with applications to optimal design of experiments*, Ph.D. dissertation, University of California, Berkeley.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 2, March 1987

© 1987 Society for Industrial and Applied Mathematics
012

A METHOD FOR COMPUTER GENERATION OF VARIATES FROM ARBITRARY CONTINUOUS DISTRIBUTIONS*

GARY ULRICH† AND LAYNE T. WATSON‡

Abstract. If X is a continuous random variable with cumulative distribution function (cdf) $F(x)$, then the most direct means of generating a variate from this distribution is the inverse cdf transform, $X = F^{-1}(U)$, where U is a random uniform variate. If $f(x)$ is the corresponding density function then X and U, as defined above, must satisfy the differential equation, $X' = 1/f(X)$. This differential equation is applied, first, in the high accuracy computation of distribution percentiles. The differential equation is then used to motivate an interpolatory Runge-Kutta algorithm to approximate the inverse cdf in the transform described above. Finally, the best discrete least squares cubic spline approximation to the inverse cdf is constructed for the approximation of the inverse cdf. Comparisons between inverse cdf approximations are made on the basis of accuracy, efficiency, and setup requirements for the normal, exponential, Cauchy and gamma distributions.

Key words. continuous random variables, inverse cumulative distribution function, differential equations, interpolatory Runge-Kutta, B-splines

AMS(MOS) subject classifications. 65C10, 65C05, 65D07, 65L05

1. Introduction.

Suppose that X is a continuous random variate with arbitrary density, $f(x)$, and cumulative distribution function, $F(x) = \int_{-\infty}^{x} f(t)\,dt$. The inverse cdf technique is the most rudimentary method for generating X on a computer; if U is a uniform pseudo-random variate on $(0, 1)$, then the variable defined by $X = F^{-1}(U)$ has density $f(x)$. In spite of the lack of elegance, this is still the best means of obtaining computer generated variates because variations in U are mapped directly to variations in X. This one-to-one mapping is an essential ingredient in many simulation applications such as variance reduction schemes, multivariate generation algorithms and the generation of order statistics. Unfortunately, the cdf (and inverse cdf) of an arbitrary continuous distribution is seldom available in a convenient or efficient numerical form. Consequently, work in the area of random number generation has concentrated on alternative sampling algorithms such as those based on special properties, on composition/acceptance-rejection schemes, and on ratios of uniforms (a survey is given in [18]). Highly specialized implementations usually result from these methods, although two notable attempts at developing methods appropriate for arbitrary continuous distributions are found in [10] and [16].

In many instances the accuracy of the random variate generator is not of critical importance and a minimal level of error in the generated variates can be tolerated. Indeed, "exactness" is a goal rendered impossible by the limitations of machine representation and fixed precision arithmetic. An obvious alternative in cases where small errors can be allowed is to construct an approximation to the inverse cdf and sample from this approximate distribution. A few studies have given modest attention to this approach (examples are found in [3] and [2]), but only recently has a satisfactory strategy been developed for this problem by Ahrens and Kohrt [2]. In [2] the unit interval is conveniently partitioned into 2^e subintervals with the two tail regions being further subdivided $b - e$ times (b is usually the word length of the computer). Because

* Received by the editors February 5, 1985; accepted for publication (in revised form) December 14, 1985.
† Systems Training Center, AT&T Bell Laboratories, Piscataway, New Jersey 08854.
‡ Computer Science Department, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24061.

this partition is able to deftly balance the somewhat conflicting requirements of increased sampling in the tails and sampling efficiency, we will use it in the proposals of §§ 3 and 4 and will refer to it simply as the AK partition. Ahrens and Kohrt proceed to fit the inverse cdf with an eighth degree Lagrange interpolating polynomial over each of the subintervals. To further improve sampling speed and also to avoid the oscillatory behavior exhibited by high degree Lagrange interpolation, they suggest a recipe for truncating insignificant high order polynomial terms. Ahrens and Kohrt did not assess the accuracy of their approximation, but comparisons in the present investigation reveal that their technique provides a generally satisfactory solution to the problem of approximating the inverse cdf, although it does exhibit some minor handicaps.

The primary goal of this work is to evaluate two new proposals for approximating the inverse cdf of a distribution, where both proposals consist of piecewise approximations to a target function. The first method develops a Runge–Kutta-type of interpolant which leads to an approximation requiring little setup and possibly one evaluation of the pdf. The second method uses an efficient cubic spline approximation which requires setup of large coefficient tables. These approximations will employ a sampling algorithm with two basic steps, the first step being the search for the appropriate subinterval and the second being the approximation over that subinterval. Our objective, therefore, is to achieve the greatest accuracy without using an excessive number of knots or intervals in the piecewise approximation. We will refer to these generation algorithms, along with the Ahrens–Kohrt procedure, as table sampling approximations. Past investigations have paid little attention to the characteristics of such approximations, so all of the proposals will be compared, along with the Ahrens–Kohrt algorithm, on the basis of accuracy and generation time.

We will assume throughout this treatise that we have an arbitrary continuous distribution for which only a density is available. This necessitates a routine for integrating the density in order to obtain the percentiles of the distribution, and in § 2 we digress briefly to describe a seldom acknowledged method for efficiently computing distribution percentiles. In § 3 we describe the Runge–Kutta-type approximation and in § 4 we discuss the cubic spline approximation to the inverse cdf. Section 5 provides timing comparisons of the algorithms and a summary of the results.

2. Computing distribution percentiles. The $100 \cdot U$ percentile of the distribution with density $f(x)$ is that value X such that $U = F(X) = \int_{-\infty}^{X} f(t)\,dt$. When numerical routines are not available for computing F or F^{-1} then the alternative most commonly advocated in the literature uses a high accuracy (double precision) integration routine imbedded in a Newton root finding algorithm (cf. [2] or [14, Chap. 5]). To implement such a method, the user must provide computer code which gives special attention to error control, step size selection and individual distribution characteristics such as bounded domains or unbounded densities. We describe a little acknowledged approach which has these qualities built in and requires no special coding. If X is the $100 \cdot U$ percentile of the distribution then X satisfies the differential equation,

$$X' = \frac{1}{f(X)} \tag{2.1}$$

with the initial condition, $X(U_0) = X_0$ for some X_0 which is the $100 \cdot U_0$ percentile of the distribution. If we restrict our attention to a subinterval of the domain, say (U^*, U^{**}), where $f(X(U)) > 0$, then we are assured that the solution to (2.1) exists and is unique. The user only needs to provide the initial conditions and a subroutine for computing the right-hand side of (2.1), and then can obtain satisfactory solutions

with almost any packaged ordinary differential equation solver. The packaged solver automatically controls error and step size selection, and can be directed not to overstep the boundaries of the domain. When the density has excessively heavy tails then a "stiff" ordinary differential equation solver can be used for added efficiency.

This proposal and the Newton search method were applied to the normal, exponential, Cauchy and gamma distributions. Five different shape parameters were used for the gamma, $\alpha = 0.2$, 0.5, 10, 100 and 1000. The adaptive Romberg integration routine from [5] was used in the root finding algorithm, while the ordinary differential equation algorithm of Shampine and Gordon [20] was used to solve (2.1). The integration was begun at the median in all cases and carried out in both directions (only one direction for symmetric densities) in increments of 0.002 over the entire open interval (0, 1). The experiment was executed twice, once with a requested absolute and relative accuracy of 1×10^{-8} and a second time with a requested absolute and relative accuracy of 1×10^{-14}. Let the exact value of the $100 \cdot U$ percentile be denoted by $X(U)$ while the estimated value is denoted by $\tilde{X}(U)$. To evaluate the accuracy of the two algorithms, we compute the maxnorm error

$$E_1 = \max_{U \in (0,1)} |X(U) - \tilde{X}(U)| \tag{2.2}$$

and the relative maxnorm error

$$E_2 = \max_{U \in (0,1)} |[X(U) - \tilde{X}(U)]/X(U)| \tag{2.3}$$

for each of the tested cases and report these results in Tables 1(a) and 1(b). It should be noted that the Newton search routine must call a subroutine to evaluate $f(x)$ while the ordinary differential equation routine calls a subroutine to evaluate $1/f(x)$. The average number of these function evaluations required for each percentile is a measure of the efficiency of the routines and this information is included in Table 1.

Before discussing the performance of these algorithms, we emphasize that the ordinary differential equation solver of Shampine and Gordon uses a procedure which balances efficiency (number of function calls) with accuracy and the resulting accuracy is sometimes relaxed in exchange for fewer calls. The routines were all implemented in double precision on an IBM 3081 and Table 1 shows that the differential equation approach holds up quite well overall. The Newton root finding method tends to be overaccurate when the requested accuracy is 1×10^{-8}, but both algorithms perform comparably when the requested accuracy is 1×10^{-14}. In this latter case, neither method attained the requested accuracy since the algorithms control local error, not global error. Both algorithms performed very poorly in the extreme case of the gamma ($\alpha = 0.2$), but this again is due to the fact that the global error is not controlled. It appears that in most situations, the user can anticipate at least single precision accuracy (10^{-8}) from the ODE approach by requesting an accuracy of 1×10^{-14}.

The most striking differences between the Newton search and ODE methods are in the average number of function calls. The Newton search algorithm required at least 30 times as many function calls as did the differential equation routine (and in a couple of cases it used 60 times as many function calls). This only serves to emphasize that the ordinary differential equation solver is designed to integrate this type of problem.

One additional advantage of the ODE approach is that it can be applied in multivariate settings. For example, the decomposition in [17] could be used with a system of differential equations to obtain percentile contours or other multivariate types of inverse integrations. Newton search algorithms become intensely cumbersome

TABLE 1

Comparison of Newton search and differential equation approaches for computing distribution percentiles.

(a) *Requested accuracy of* 1×10^{-8}.

	Newton search			ODE		
	E_1	E_2	Average # function calls	E_1	E_2	Average # function calls
Normal	3.2×10^{-11}	1.2×10^{-11}	14.87	1.0×10^{-7}	4.0×10^{-8}	0.54
Exponential	1.3×10^{-10}	2.2×10^{-11}	14.27	2.4×10^{-6}	4.1×10^{-7}	0.46
Cauchy	9.7×10^{-7}	9.1×10^{-9}	17.95	3.2×10^{-4}	3.0×10^{-6}	0.76
Gamma 0.2	2.1×10^{-9}	9.9×10^{0}	16.39	7.5×10^{-6}	6.5×10^{0}	0.45
Gamma 0.5	3.6×10^{-10}	2.5×10^{-7}	15.27	1.9×10^{-6}	9.6×10^{-6}	0.45
Gamma 10	1.1×10^{-10}	7.9×10^{-12}	16.09	2.8×10^{-6}	7.5×10^{-7}	0.55
Gamma 100	3.3×10^{-10}	3.8×10^{-12}	19.46	7.8×10^{-6}	9.4×10^{-8}	0.50
Gamma 1000	1.2×10^{-9}	1.3×10^{-12}	25.22	3.1×10^{-5}	3.4×10^{-8}	0.51

(b) *Requested accuracy of* 1×10^{-14}.

	Newton search			ODE		
	E_1	E_2	Average # function calls	E_1	E_2	Average # function calls
Normal	8.3×10^{-14}	3.0×10^{-14}	26.85	1.1×10^{-12}	4.1×10^{-13}	2.16
Exponential	1.4×10^{-12}	3.8×10^{-13}	26.32	2.8×10^{-12}	4.7×10^{-13}	1.30
Cauchy	3.5×10^{-11}	3.3×10^{-13}	31.13	1.3×10^{-9}	1.2×10^{-11}	2.06
Gamma 0.2	1.2×10^{-12}	1.5×10^{-5}	37.82	2.5×10^{-11}	2.9×10^{-1}	1.54
Gamma 0.5	1.6×10^{-12}	9.7×10^{-13}	29.30	5.4×10^{-12}	1.6×10^{-12}	1.31
Gamma 10	4.0×10^{-12}	1.9×10^{-13}	27.94	7.5×10^{-12}	2.2×10^{-12}	2.21
Gamma 100	3.9×10^{-11}	5.2×10^{-12}	40.50	2.7×10^{-10}	3.6×10^{-11}	1.83
Gamma 1000	9.4×10^{-10}	10×10^{-10}	102.98	5.9×10^{-9}	5.5×10^{-12}	1.63

when applied in a multivariate framework whereas the differential equation solver can handle a system of equations with ease.

3. Runge–Kutta approximation. In the proposals of this section and the following section we will divide the unit interval into n subintervals with endpoints $\{U_i\}_{i=0}^n$. It is assumed that $\{U_i\}_{i=0}^n$ provides a reasonable covering of the unit interval and, in the absence of any preferred interval, the AK partition can be used. Frequently, U_0 and U_n are chosen to be the smallest and largest machine representable numbers on the open interval $(0, 1)$. The percentiles corresponding to $\{U_i\}_{i=0}^n$ will be denoted by $\{X_i\}_{i=0}^n$. If $g(U, X)$ represents the right-hand side of the differential equation (2.1), then the percentile derivative will be denoted by $\{D_i\}_{i=0}^n$ where

$$D_i = g(U_i, X_i) = \frac{1}{f(X_i)}.$$

Runge-Kutta type interpolants will be used to construct approximations using only the U_i, X_i and D_i and possibly $g(u, x)$. The advantage here will be that the user need only supply the $\{U_i\}_{i=0}^n$ to obtain an approximation which requires modest setup.

To construct the approximation for $F^{-1}(U)$ on the interval (U_{i-1}, U_i), we first define the interval mean and radius as $\bar{U}_i = (U_{i-1} + U_i)/2$ and $h_i = (U_i - U_{i-1})/2$. Next, reparameterize the interval using $s = (U - \bar{U}_i)/h_i$ and rewrite the percentile in terms

of s, i.e.,

$$X(s) = F^{-1}(\bar{U}_i + sh_i). \tag{3.1}$$

In this section we will develop a fifth order Runge-Kutta type approximation to the inverse cdf and we motivate the formula by constructing the second, third and fourth order approximations as well. The second order interpolant for $X(s)$ is simply the linear approximation

$$\tilde{X}_2(s) = \frac{1-s}{2} X_{i-1} + \frac{1+s}{2} X_i, \tag{3.2}$$

which is essentially Butler's [7] proposal. This approximation can be improved by estimating F^{-1} at some point $(\bar{U}_i + th_i)$, where $t \in (-1, 1)$, and approximating the differential equation at that point. The second order solution can be adjusted in this way to yield the third order solution,

$$\tilde{X}_3(s, t) = \tilde{X}_2(s) + \frac{h_i(s^2-1)}{t}\left(g(\bar{U}_i + th_i, \tilde{X}_2(t)) - \frac{X_{i-1}+X_i}{2}\right). \tag{3.3}$$

The derivatives, D_{i-1} and D_i, can be utilized to obtain the fourth order (Hermite cubic) solution

$$\tilde{X}(s) = \tilde{X}_2(s) + \frac{h_i(s^2-1)}{2}\left(\left(\frac{1+s}{2}\right)D_i - \left(\frac{1-s}{2}\right)D_{i-1} - \frac{s(X_i - X_{i-1})}{2h_i}\right). \tag{3.4}$$

The differential equation can be estimated again at some arbitrary $t \in (-1, 1)$ to produce the fifth order solution

$$\begin{aligned}\tilde{X}_5(s, t) = \tilde{X}_4(s) + \frac{h_i(s^2-1)^2}{8t}\Bigg[&\frac{2}{t^2-1} g(\bar{U}_i + th_i, \tilde{X}_4(t)) + \frac{3(X_i - X_{i-1})}{2h_i} \\ &+ \frac{1}{2}\left(\left(\frac{3t-1}{1-t}\right)D_i - \left(\frac{3t+1}{1+t}\right)D_{i-1}\right)\Bigg].\end{aligned} \tag{3.5}$$

This process could be continued by using $(d/dX)g(U, X) = -f'(X)/f^2(X)$ to obtain more interpolation information but the focus of the present investigation precluded carrying the development any further. Related work in Runge-Kutta interpolation has been done by Watts [21]. Since formulas (3.3) and (3.5) are apparently new, we will demonstrate that the fifth order nature of $\tilde{X}_5(s, t)$; the orders of $X_2(s)$, $X_3(s, t)$ and $X_4(s)$ can be verified in a similar manner.

THEOREM. *Assume that $g(U, X) = 1/f(X)$ is four times continuously differentiable. The approximation $\tilde{X}_5(s, t)$ given in formula* (3.5) *satisfies*

$$|\tilde{X}_5(s, t) - X(s)| \sim O(h^5).$$

Proof. Without loss of generality, we will restrict our attention to the first subinterval, (U_0, U_1), and will drop the subscripts from $\bar{U}_i$ and h_i. Furthermore, we define $\bar{X} = X(\bar{U})$ and propose a solution to (2.1) of the form

$$\tilde{X}(s) = \rho X_0 + (1-\rho)X_1 + h(\alpha D_0 + \beta D_1 + \gamma G) \tag{3.6}$$

where

$$G = g(\bar{U} + th, \phi X_0 + (1-\phi)X_1 + h[\delta D_0 + \varepsilon D_1]).$$

Expanding $X(s)$ about $(\bar{U}, \bar{X})$ yields

$$X(s) = \bar{X} + sh\bar{X}' + \frac{(sh)^2}{2}\bar{X}'' + \frac{(sh)^3}{6}\bar{X}''' + \frac{(sh)^4}{24}\bar{X}'''' + O(h^5)$$

and matching this to the fifth order expansion of $\tilde{X}(s)$ (cf. [12]) requires that the parameters be defined as

$$\alpha = \frac{(s^2-1)(s-1)}{4}\left[1-\frac{(s+1)(3t+1)}{4t(1+t)}\right], \qquad \beta = \frac{(s^2-1)(s+1)}{4}\left[1+\frac{(s-1)(3t-1)}{4t(1-t)}\right],$$

$$\gamma = \frac{(s^2-1)^2}{4t(t^2-1)}, \quad \delta = \frac{1}{4}(t-1)^2(t+1), \quad \varepsilon = \frac{1}{4}(t-1)(t+1)^2,$$

$$\rho = \frac{1}{4}\left[(s-1)^2(s+2)-\frac{3(s+1)^2(s-1)^2}{4t}\right], \qquad \phi = \frac{1}{4}(t-1)^2(t+2).$$

Rearranging terms in (3.6) yields the formula in (3.5) and the error in the solution is therefore of order h^5.

The value of t in solution $\tilde{X}_3(s, t)$ and $\tilde{X}_5(s, t)$ is arbitrary and research by Shampine [19] and others on the existence of optimal Runge-Kutta formulas suggests there is no optimal choice of t. The adaptive choice,

$$t = \begin{cases} \dfrac{s+1}{2}, & s \geqq 0, \\ \dfrac{s-1}{2}, & s < 0, \end{cases} \tag{3.7}$$

proved quite satisfactory in testing. Another advantage of the above choice of t is that it avoids singularities that exist in (3.3) and (3.5).

These four approximations were implemented, using the AK partition with $b = 24$ and $c = 6$, for the normal, exponential, Cauchy and gamma ($\alpha = 0.2, 0.5, 10, 100, 1000$) distributions. The exact percentile, $X(s)$, was compared with each of the four approximations, $\tilde{X}_i$, $i = 2, 3, 4, 5$, using the maxnorm criteria, E_1, and the relative maxnorm criteria, E_2, as defined in (2.2) and (2.3). Looking at E_1 and E_2 alone can be deceptive because the regions of greatest error are generally those of lowest probability. A more reasonable criteria would compare the target density, $f(X(U))$, with the effective density of the solution, $\tilde{f}(\tilde{X}(U)) = 1/(d/dU)\tilde{X}(U)$, where we have simplified the notation by again expressing the approximations in terms of the parameter $U = \bar{U}_i + sh_i$. We compare these densities using the Integrated Absolute Deviation defined as

$$\text{IAD} = \int_{U_0}^{U_n} \frac{|f(X(U)) - \tilde{f}(\tilde{X}(U))|}{f(X(U))}\, dU. \tag{3.8}$$

The IAD can be interpreted as the total amount of probability which must be redeposited between f and $\tilde{f}$ to make them equal. This criterion is closely related to the total variation of [9] and several properties of total variation are outlined there. One property which is not mentioned in [9] is that IAD is location and scale invariant as long as $\tilde{X}$ is linear in location and scale translations.

The value of the three norms are given, in each case, in Table 2. As the order of the approximation increases, the value of each normed error decreases in a fairly consistent manner. Very satisfactory results are attained by the fifth order solution in every instance, even the extreme cases involving the Cauchy and gamma ($\alpha = 0.2$) distributions. If one recognizes that the fifth order solution is analogous to fitting a fourth degree polynomial over each of the subintervals, then these results compare directly to the Ahrens and Kohrt and cubic spline results of Table 3.

The second and fourth order Runge-Kutta approximations were monotone over all subintervals in all cases tested. The third and fifth order solutions were not completely

TABLE 2
Comparison of error in the Runge-Kutta approximations.

	approx.	E_1	E_2	IAD
Normal	$\tilde{X}_2$	2.1×10^{-2}	1.1×10^{-2}	2.3×10^{-2}
	$\tilde{X}_3$	4.3×10^{-3}	2.2×10^{-3}	3.6×10^{-3}
	$\tilde{X}_4$	1.2×10^{-3}	5.9×10^{-4}	6.2×10^{-4}
	$\tilde{X}_5$	2.0×10^{-4}	9.8×10^{-5}	8.4×10^{-5}
Exponential	$\tilde{X}_2$	6.0×10^{-2}	1.6×10^{-2}	1.9×10^{-2}
	$\tilde{X}_3$	1.4×10^{-2}	3.8×10^{-3}	1.8×10^{-3}
	$\tilde{X}_4$	3.6×10^{-3}	9.7×10^{-4}	4.0×10^{-4}
	$\tilde{X}_5$	6.7×10^{-4}	1.8×10^{-4}	6.0×10^{-5}
Cauchy	$\tilde{X}_2$	4.6×10^{5}	1.3×10^{-1}	5.8×10^{-2}
	$\tilde{X}_3$	2.7×10^{5}	7.5×10^{-2}	1.6×10^{-2}
	$\tilde{X}_4$	5.7×10^{4}	1.6×10^{-2}	4.0×10^{-3}
	$\tilde{X}_5$	2.0×10^{4}	5.7×10^{-3}	1.1×10^{-3}
Gamma 0.2	$\tilde{X}_2$	5.6×10^{-2}	1.9×10^{0}	8.5×10^{-2}
	$\tilde{X}_3$	1.4×10^{-2}	1.0×10^{0}	5.2×10^{-2}
	$\tilde{X}_4$	3.5×10^{-3}	8.7×10^{-2}	2.2×10^{-3}
	$\tilde{X}_5$	6.6×10^{-4}	3.7×10^{-2}	7.4×10^{-4}
Gamma 0.5	$\tilde{X}_2$	5.8×10^{-2}	1.3×10^{-1}	3.5×10^{-2}
	$\tilde{X}_3$	1.5×10^{-2}	3.2×10^{-2}	3.9×10^{-3}
	$\tilde{X}_4$	3.6×10^{-3}	1.3×10^{-3}	4.2×10^{-4}
	$\tilde{X}_5$	6.8×10^{-4}	2.6×10^{-4}	6.4×10^{-5}
Gamma 10	$\tilde{X}_2$	1.0×10^{-1}	7.3×10^{-3}	2.2×10^{-2}
	$\tilde{X}_3$	2.3×10^{-2}	1.4×10^{-3}	2.1×10^{-3}
	$\tilde{X}_4$	6.1×10^{-3}	3.7×10^{-4}	6.0×10^{-4}
	$\tilde{X}_5$	1.1×10^{-3}	6.2×10^{-5}	7.9×10^{-5}
Gamma 100	$\tilde{X}_2$	2.5×10^{-1}	2.2×10^{-3}	2.3×10^{-2}
	$\tilde{X}_3$	5.2×10^{-2}	4.3×10^{-4}	2.2×10^{-3}
	$\tilde{X}_4$	1.4×10^{-2}	1.2×10^{-4}	6.2×10^{-4}
	$\tilde{X}_5$	2.4×10^{-3}	2.0×10^{-5}	8.3×10^{-5}
Gamma 1000	$\tilde{X}_2$	7.0×10^{-1}	6.8×10^{-4}	2.3×10^{-2}
	$\tilde{X}_3$	1.5×10^{-1}	1.4×10^{-4}	2.2×10^{-3}
	$\tilde{X}_4$	4.0×10^{-2}	3.8×10^{-5}	6.2×10^{-4}
	$\tilde{X}_5$	6.6×10^{-3}	6.2×10^{-5}	8.3×10^{-5}

monotone in a few subintervals for the gamma ($\alpha = 0.2$). Monotonicity is not guaranteed by the third, fourth and fifth order solutions and if the user is concerned about a monotone cdf then the methods of [11] can be adapted to this application. In cases where the density may not be available, Fritsch and Carlson [11] suggest they be estimated using a three point difference formula. Finally, in cases where the percentiles, only, are contaminated with random error, the methods of [4] may be used to smooth the estimate of F^{-1} under the constraint of monotonicity.

4. Polynomial approximations. There are various ways of constructing piecewise polynomial approximations to a function, but the method of B-splines offers an elegant and accurate approach in the majority of applications. In this section, we will use

TABLE 3
Comparison of error in the piecewise polynomial approximation.

		E_1	E_2	IAD
Normal	AK	6.6×10^{-2}	1.3×10^{-2}	1.1×10^{-4}
	B/AK	7.5×10^{-4}	3.5×10^{-4}	7.2×10^{-4}
	B/1	9.2×10^{-5}	9.1×10^{-4}	6.8×10^{-4}
	B/0.9	1.9×10^{-5}	3.6×10^{-4}	1.2×10^{-4}
	B/0.8	2.7×10^{-2}	5.1×10^{-3}	4.9×10^{-5}
	B/0.7	2.1×10^{-1}	4.1×10^{-2}	4.8×10^{-5}
Exponential	AK	3.6×10^{-1}	4.8×10^{-1}	8.4×10^{-5}
	B/AK	2.3×10^{-3}	5.6×10^{-4}	4.8×10^{-4}
	B/1	5.0×10^{-4}	4.1×10^{0}	2.1×10^{-4}
	B/0.9	7.7×10^{-4}	3.7×10^{-1}	3.4×10^{-5}
	B/0.8	2.4×10^{-2}	4.0×10^{-2}	1.1×10^{-5}
	B/0.7	4.6×10^{-1}	2.8×10^{-2}	8.7×10^{-6}
Cauchy	AK	1.7×10^{6}	3.2×10^{-1}	6.9×10^{-4}
	B/AK	1.7×10^{4}	8.3×10^{-3}	3.9×10^{-3}
	B/1	1.5×10^{2}	6.5×10^{-1}	3.7×10^{-1}
	B/0.9	8.0×10^{2}	1.3×10^{-1}	9.3×10^{-2}
	B/0.8	9.1×10^{3}	7.1×10^{-3}	5.5×10^{-3}
	B/0.7	2.5×10^{5}	4.9×10^{-2}	8.1×10^{-4}
Gamma 0.2	AK	3.4×10^{-1}	9.9×10^{0}	7.1×10^{-2}
	B/AK	1.9×10^{-3}	1.1×10^{21}	1.8×10^{-1}
	B/1	1.2×10^{-4}	4.0×10^{18}	1.4×10^{0}
	B/0.9	6.2×10^{-4}	4.3×10^{17}	4.7×10^{-2}
	B/0.8	2.1×10^{-2}	8.1×10^{16}	2.7×10^{-2}
	B/0.7	4.1×10^{-1}	2.3×10^{16}	1.9×10^{-2}
Gamma 0.5	AK	3.5×10^{-1}	1.3×10^{0}	2.9×10^{-3}
	B/AK	2.1×10^{-3}	1.5×10^{0}	5.1×10^{-4}
	B/1	4.5×10^{-5}	4.3×10^{4}	6.1×10^{-4}
	B/0.9	6.9×10^{-4}	2.2×10^{3}	8.8×10^{-5}
	B/0.8	2.3×10^{-2}	2.4×10^{2}	2.4×10^{-5}
	B/0.7	4.3×10^{-1}	5.6×10^{1}	1.3×10^{-5}
Gamma 10	AK	4.8×10^{-1}	4.1×10^{-2}	5.9×10^{-5}
	B/AK	3.8×10^{-3}	2.5×10^{-4}	6.9×10^{-4}
	B/1	3.1×10^{-4}	3.3×10^{-4}	5.4×10^{-4}
	B/0.9	7.4×10^{-3}	2.2×10^{-3}	1.1×10^{-4}
	B/0.8	1.2×10^{-1}	3.0×10^{-2}	4.8×10^{-5}
	B/0.7	1.3×10^{0}	1.9×10^{-1}	4.7×10^{-5}
Gamma 100	AK	9.1×10^{-1}	8.0×10^{-3}	1.8×10^{-6}
	B/AK	8.9×10^{-3}	7.7×10^{-5}	7.2×10^{-4}
	B/1	9.4×10^{-4}	1.4×10^{-5}	5.9×10^{-4}
	B/0.9	2.1×10^{-2}	2.9×10^{-4}	1.2×10^{-4}
	B/0.8	3.0×10^{-1}	3.9×10^{-3}	4.9×10^{-5}
	B/0.7	2.7×10^{0}	2.9×10^{-2}	4.6×10^{-5}
Gamma 1000	AK	2.3×10^{0}	2.2×10^{-3}	3.6×10^{-6}
	B/AK	2.5×10^{-2}	2.4×10^{-5}	7.2×10^{-4}
	B/1	6.7×10^{-4}	2.9×10^{-3}	1.7×10^{-2}
	B/0.9	6.3×10^{-2}	6.8×10^{-5}	1.2×10^{-4}
	B/0.8	8.9×10^{-1}	9.5×10^{-4}	5.0×10^{-5}
	B/0.7	7.3×10^{0}	7.4×10^{-3}	4.7×10^{-5}

B-splines to obtain a smooth piecewise polynomial fit to the inverse cdf. For a review of the theory of B-splines, refer to [6]. The computer programs from de Boor [6] were used in obtaining the results of this section, but similar spline routines are available in many scientific software packages.

Fitting a spline approximation to the inverse cdf requires decisions about the degree of polynomials (order of the spline), the number of knots, and the placement of knots. These decisions must be made in such a way as to balance accuracy and sampling efficiency. In this study, splines of order $k=4$ will be implemented (cubic splines), but the user can easily fit higher order splines to obtain more accurate approximations. If n intervals are employed, with endpoints $\{U_i\}_{i=0}^{n}$, then the corresponding knot sequence $\{t_i\}_{j=1}^{n+2k-1}$ will be defined as

$$t_j = \begin{cases} U_0, & j \leqq k, \\ U_{j-k}, & k < j < n+k, \\ U_n, & j \geqq n+k, \end{cases} \tag{4.1}$$

where the end knots are repeated by convention. After the order of the spline and the number of subintervals is chosen, the knots need to be selected in a near optimal manner. To do this we must begin with an initial knot sequence and a set of percentiles, then fit an intermediate B-spline approximation which will in turn be used to choose an optimal set of knots and the final B-spline approximation.

For simplicity, we let the initial knot sequence be constructed from the AK partition with n intervals and the endpoint sequence $\{U_i\}_{i=0}^{n}$. Since we need a large set of points sampled from F^{-1}, we choose $(m-1)$ equally spaced abscissa on each subinterval to obtain a sequence $\{U_j^*\}_{j=0}^{mn}$, where the partition endpoints are included among these $M = mn+1$ points. Let X_j^*, $j=0, 1, \cdots, mn$, be the corresponding percentiles, i.e., $X_j^* = F^{-1}(U_j^*)$, giving M data points with which to construct a spline approximation. When splines of order $k=4$ are used then $m \geqq 10$ will provide satisfactory results. Once the percentiles are chosen then the best discrete least squares B-spline approximation can be found using the knot sequence based on the AK partition.

To find a near optimal knot placement, [6] shows that the knots should be placed so that the kth order variation of the target function, g, over each subinterval (ζ_{i-1}, ζ_i), $i=1, 2, \cdots, n$, is a constant:

$$\int_{\zeta_{i-1}}^{\zeta_i} |g^{(k)}(x)|^{1/k}\, dx = \text{constant} \tag{4.2}$$

where $g^{(k)}$ is the kth derivative of the target function. It is not computationally feasible to use the original function in (4.2) but the spline approximation (based on the AK partition) can easily be used as described below. Convert the B-spline approximation to its piecewise polynomial representation which has c_i, $i=1, \cdots, n$, as the coefficient of the highest degree term on each subinterval. Estimate the variation function in (4.2) with the piecewise linear approximation, $L(U) = a_i + b_i(U - U_{i-1})$, $U_{i-1} < U \leqq U_i$, having coefficients defined iteratively as

$$\begin{aligned} &a_1 = 0, \qquad b_1 = \left(2\frac{c_2 - c_1}{U_2 - U_0}\right)^{\beta/k}, \\ &a_i = a_{i-1} + b_{i-1}(U_{i-1} - U_{i-2}), \\ &b_i = \left(\frac{c_{i+1} - c_i}{U_{i+1} - U_{i-1}} + \frac{c_i - c_{i-1}}{U_i - U_{i-2}}\right)^{\beta/k}, \quad i = 2, \cdots, n-1, \end{aligned} \tag{4.3}$$

$$a_n = a_{n-1} + b_{n-1}(U_{n-1} - U_{n-2}), \qquad b_n = \left(2\frac{c_n - c_{n-1}}{U_n - U_{n-2}}\right)^{\beta/k}.$$

The new interval endpoints $\{\zeta_i\}_{i=0}^{n}$ are defined by choosing $\delta_0 = U_0$, $\delta_n = U_n$, and the middle points distributed so that $L(\zeta_i) = i \cdot s$ where $s = [a_n + b_n(U_n - U_{n-1})]/n$ is the constant in (4.2). De Boor [6] uses $\beta = 1$ but this tends to overemphasize the fit of the approximation in the tails of the distribution where the probability is low. Because the piecewise linear approximation from (4.3) has greatest increase when the inverse cdf has greatest increase, then an intuitively reasonable adjustment is to use $\beta < 1$ to deemphasize the tails and place more importance in fitting the body of the distribution. Numerical investigations revealed that $\beta = 0.8$ is a reasonable adjustment in all cases. It is clear that the knot selection algorithm could be used to select an arbitrary number of intervals, but for simplicity we continued to use n subintervals for the new knot sequence. Once the near optimal knots are selected they are used again to obtain a best discrete least squares B-spline approximation for the inverse cdf. We note in passing that this knot selection algorithm is just the subroutine NEWNOT in de Boor [6] with the simple adjustment of passing the variable BETA in the argument list and redefining the variable ONEOVK as

$$\text{ONEOVK} = \text{BETA}/\text{FLOAT(K)}.$$

The B-spline approximation was applied for the normal, exponential, Cauchy and gamma ($\alpha = 0.2, 0.5, 10, 100, 1000$) distributions. The AK partition with $e = 6$ and $b = 24$ was used, which led to $n = 99$ intervals. For the sake of comparison, the Ahrens-Kohrt proposal was implemented and the results for this approximation appear as the first entry in each subgroup of Table 3. The next approximation which was evaluated was the B-spline with the AK partition. Finally, the B-spline approximation with near optimal knots for $\beta = 1, 0.9, 0.8$ and 0.7 appear in Table 3. The E_1, E_2 and IAD norms were again used to compare these approximations.

The results of Table 3 suggest the following general conclusions. The Ahrens and Kohrt approximation did reasonably well in all the cases tested, at least with respect to IAD. The E_1 and E_2 norms indicate that the Ahrens-Kohrt approximation has some difficulty fitting the inverse cdf in the tails of the distribution. This probably occurs because the degrees of the Lagrange interpolants are higher in the tails and the oscillatory behavior of these polynomials is exaggerated there. The spline approximation using the AK partition does a good job of approximating the tails of the distribution but does not perform as well in the IAD norm. Comparing the spline approximation with optimal knots over the four choices of β, we find that the value of $\beta = 0.8$ gives a reasonable balance, in each case, between fitting the inverse cdf and the density of the distribution. Only for the Cauchy and the gamma ($\alpha = 100, 1000$) distributions will the Ahrens and Kohrt method provide a better IAD and this happens because the average degree of the polynomials in the Ahrens-Kohrt schemes is 3.5 or higher. Presumably, a quartic B-spline approximation ($k = 5$) would yield competitive accuracy here.

One of the appealing features of the B-spline approximation scheme is that the kth order B-spline will have a continuous $(k-2)$nd derivative and hence will provide a very smooth approximation to the inverse cdf. Another advantage to this B-spline approach is that the user can rely on packaged software for most of the setup. The values from Table 3 support the contention that B-splines offer a suitably accurate means of approximating the inverse cdf of an arbitrary distribution. The user will forfeit the simplicity and efficiency built into the AK partition by choosing the optimal

knot distribution, but timing comparisons in § 5 indicate that little sampling efficiency is actually lost in this trade.

5. Timing comparisons and summary. In the previous sections, we have developed several reasonable competitors for the Ahrens-Kohrt approximation: the fourth and fifth order Runge-Kutta approximations (henceforth denoted as RK4 and RK5) and the cubic spline approximation (SPLINE). We have made extensive comparisons of the accuracy of these approximations and summarize these results graphically in Fig. 1 where the IAD is plotted for the four competing proposals. Even though the performance of these four methods is uneven, we can still consider several patterns suggested by Fig. 1. First, the IAD of the normal and large parameter gamma distributions are approximately the same in the cases of the RK4, RK5 and SPLINE methods. This follows from the asymptotic normality of the gamma and the translation invariance of the IAD criteria. The AK approximation does not display the same property because the polynomial truncation recipe artificially overfits the large parameter gammas. In fact, this polynomial truncation scheme, coupled with the oscillatory behavior of high degree Lagrange polynomials, fosters relatively erratic AK approximations so that some distributions are underfit while others are overfit. The most consistent accuracy was achieved by the Runge-Kutta algorithms. The cubic spline approximation performed well overall but is so burdened with smoothness constraints that it is unable to accurately fit an inverse cdf with behavior as extreme as the Cauchy or gamma ($\alpha = 0.2$).

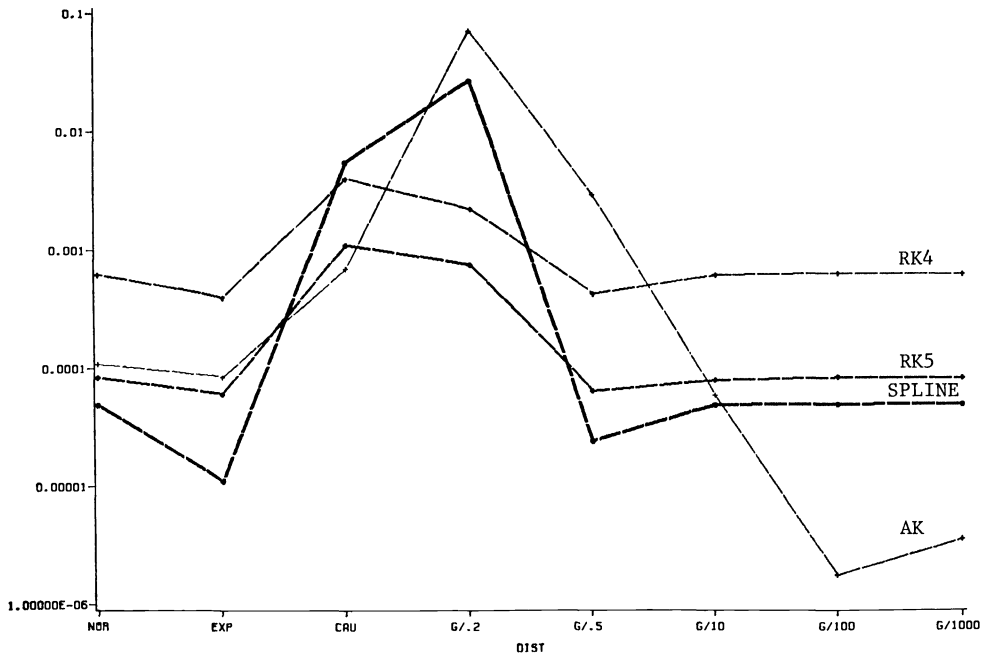


FIG. 1. IAD *comparisons for the* RK4, RK5, AK *and* SPLINE *approximations.*

Another basis on which to compare these approximation techniques is the monotonicity of the fitted approximation. Although none of these approximations are constrained to be monotone, it is reasonable to expect them to mirror properties of the target function and be, for the most part, monotone. Only the RK4 approximation was completely monotone over all subintervals and for each of the eight sampled

distributions. The other three methods were nonmonotone over the lower portion of the domain of the gamma ($\alpha = 0.2$) distribution. The AK method was also nonmonotone in various subintervals for the gamma ($\alpha = 0.5, 10$) distributions. Efforts could be made to constrain these approximations to be monotone, but this is hardly worth the work in most applications. The simpler approach is to choose a method which has proven to be monotone in a wide variety of situations and the RK4 method seems the best candidate in this respect.

Smoothness is also an important consideration when comparing approximations. In many applications, an approximation with continuous density might be desired and this would be equivalent to an inverse cdf approximation having continuous first derivative. The Runge-Kutta approximations both have continuous first derivatives while the kth order spline approximation has a continuous $(k-2)$nd derivative. We suspect that the spline approximation is actually too smooth; we could relax the smoothness constraints by increasing the number of knots with knots repeated in the middle line of definition (4.1). The Ahrens-Kohrt method does not provide a continuous first derivative and may therefore be unsuitable in some applications.

As a final comparison of the four approximation methods, the generation algorithms were timed using uniform variates from the Goodman, Lewis and Miller generator of [13]. The generation routines (subinterval sampling and approximation) were implemented in FORTRAN code on an IBM 3081 computer. Algorithm SGUI from [2] was used for the subinterval selection in the AK, RK4 and RK5 methods while algorithm KGUI from [2] was used for subinterval selection in the SPLINE approximation. The timings are based upon 50,000 generations of the random variates and the average time per variate is recorded in Table 4. In order to provide a frame of reference, the generation times of commonly used exact methods are also included in Table 4. These exact methods are the Kinderman and Ramage normal generator [15], the inverse cdf exponential algorithm, the synthetic tangent algorithm for the Cauchy distribution, the gamma algorithm GS [1] for $\alpha < 1$, and the gamma algorithm GB [8] for $\alpha > 1$. It is interesting to note in Table 4 that the RK4 algorithm is the fastest approximation and that abandoning the AK partition for the SPLINE approximation has made it only slightly less efficient than RK4. The inefficiency of the Ahrens-Kohrt method was due primarily to the overhead involved in looping to evaluate a variable degree polynomial. The RK5 method was the most costly, with generation times dependent on the complexity of the pdf. Comparison of RK5 with the exact algorithms for the gamma distribution indicates that the generation times of the RK5 method may still be competitive with some of the exact algorithms that appear in the literature.

TABLE 4
Average generation times in μsec.

	AK	SPLINE	RK4	RK5	Popular exact method	
Normal	16.8	15.2	13.2	36.4	10.2	Kinderman-Ramage
Exponential	16.6	15.2	13.2	35.2	9.0	Inverse CDF
Cauchy	18.2	14.8	13.2	29.0	8.2	Synthetic tangent
Gamma 0.2	16.2	15.0	13.2	42.8	39.4	GS
Gamma 0.5	16.6	15.2	13.2	43.0	44.2	GS
Gamma 10	17.4	15.2	13.2	42.6	34.4	GB
Gamma 100	18.0	15.2	13.2	43.0	33.8	GB
Gamma 1000	18.8	15.2	13.2	42.8	33.6	GB

REFERENCES

[1] J. H. AHRENS AND U. DIETER, *Computer methods for sampling from gamma, beta, Poisson and binomial distributions*, Computing, 12 (1974), pp. 228-246.

[2] J. H. AHRENS AND K. D. KOHRT, *Computer methods for efficient sampling from largely arbitrary statistical distribution*, Computing, 26 (1981), pp. 19-31.

[3] D. R. BARNARD AND M. N. CAWDERY, *A note on a new method of histogram sampling*, Oper. Res., 25 (1974), pp. 319-320.

[4] R. K. BEATSON, *Monotone and convex approximation by splines: Error estimates and a curve fitting algorithm*, SIAM J. Numer. Anal., 19 (1982), pp. 1278-1285.

[5] C. DE BOOR, CADRE: *An algorithm for numerical quadrature*, in Mathematical Software, John R. Rice, ed., Academic Press, New York.

[6] ———, *A Practical Guide to Splines*, Springer-Verlag, New York.

[7] E. L. BUTLER, *Algorithm* 370: *General random number generator*, Comm. ACM, 13 (1970), pp. 49-51.

[8] R. C. H. CHENG, *The generation of gamma variables with non-integer shape parameters*, Applied Statistics, 26 (1977), pp. 71-75.

[9] L. D. DEVROYE, *A note on approximations in random variate generations*, J. Statist. Comput. Simulation, 14 (1982), pp. 149-158.

[10] ———, *Random variate generation for unimodal and monotone densities*, Computing, 32 (1984), pp. 43-68.

[11] F. N. FRITSCH AND R. E. CARLSON, *Monotone piecewise cubic interpolation*, SIAM J. Numer. Anal., 17 (1980), pp. 238-246.

[12] C. H. GEAR, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1971.

[13] O. S. GOODMAN, P. A. W. LEWIS AND J. W. MILLER, *A pseudo-random number generator for the system* 360, IBM Systems J., 8 (1969), pp. 136-146.

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

[15] A. J. KINDERMAN AND J. G. RAMAGE, *Computer generation of normal random variables*, J. Amer. Statist. Assoc., 71 (1976), pp. 893-896.

[16] G. MARSAGLIA AND W. W. TSANG, *A fast, easily implemented method for sampling from decreasing or symmetric unimodal density functions*, this Journal, 5 (1984), pp. 349-359.

[17] M. ROSENBLATT, *Remarks on a multivariate transformation*, Amer. Math. Soc., 23 (1952), pp. 470-472.

[18] B. W. SCHMEISER, *Random variate generation: A survey*, in Simulation with Discrete Models: A State-of-the-Art-Survey, T. I. Oren, C. H. Shub, P. F. Roth, eds., IEEE, New York, 1980.

[19] L. F. SHAMPINE AND H. A. WATTS, *Practical solution of ordinary differential equations by Runge-Kutta methods*, Sandia Laboratories report SAND76-0585, Albuquerque, NH, 1976.

[20] L. F. SHAMPINE AND M. K. GORDON, *Computer Solution of Ordinary Differential Equations: The Initial Value Problem*, W. H. Freeman, San Francisco, 1975.

[21] H. A. WATTS, private communication, 1984.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 3, May 1987
© 1987 Society for Industrial and Applied Mathematics

APPROXIMATE CYCLIC REDUCTION FOR SOLVING POISSON'S EQUATION*

PAUL N. SWARZTRAUBER†

Abstract. A variant of the cyclic reduction algorithm for solving Poisson's equation is presented. The asymptotic operation count is proportional to $N^2 \log t$, where N is the number of grid points in one direction and t is the number of digits that are carried in the computation. It is shown that the rational matrix polynomials of relatively high degree that occur in cyclic reduction can be replaced by rational approximations of lower degree. The degree of the rational approximations depends only on t and is independent of grid size. On the CRAY-1 with $t \approx 15$ we estimate a maximum degree of 10 for any grid size and lower degrees are possible for computations in which 15 decimal digits do not need to be carried.

Key words. elliptic partial differential equations, cyclic reduction, Poisson equation

AMS(MOS) subject classifications. 65N20, 65F05

1. Introduction. A method called approximate cyclic reduction (ACR) will be developed for solving Poisson's equation

$$\Delta u = g \tag{1.1}$$

on a rectangle $a \leqq x \leqq b$ and $c \leqq y \leqq d$. In this paper it will be assumed that $u = 0$ is specified on the boundary; however, the results are applicable to any of the standard boundary conditions [5], [6]. More precisely, we will obtain an approximate solution of (1.1) in the form of a tabulation. Given the integers M and N, a grid (x_i, y_j) is defined by $x_i = a + i\Delta x$ and $y_j = c + j\Delta y$, where $\Delta x = (b-a)/(M+1)$ and $\Delta y = (d-c)/(N+1)$. We seek a tabulation $u_{i,j}$ that approximates the exact solution $u(x_i, y_j)$ at each of the grid points. To this end we require $u_{i,j}$ to satisfy the following finite difference approximation of (1.1):

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2} = g_{i,j}, \tag{1.2}$$

where $g_{i,j} = g(x_i, y_j)$. A convenient notation is obtained by introducing vectors $\mathbf{u}_j^T = (u_{1,j}, u_{2,j}, \cdots, u_{M,j})$ and $\mathbf{g}_j^T = \Delta y^2 (g_{1,j}, g_{2,j}, \cdots, g_{M,j})$. Then (1.2) can be written

$$\mathbf{u}_{j-1} + \mathbf{A}\mathbf{u}_j + \mathbf{u}_{j+1} = \mathbf{g}_j, \qquad j = 1, \cdots, N, \tag{1.3}$$

where

$$\mathbf{A} = \begin{bmatrix} -2\alpha & 1 & & & & & \\ 1 & -2\alpha & 1 & & & & \\ & 1 & & & & & \\ & & & \ddots & & & \\ & & & & 1 & & \\ & & & & 1 & -2\alpha & 1 \\ & & & & & 1 & -2\alpha \end{bmatrix} \tag{1.4}$$

and $\alpha = (\Delta y/\Delta x)^2 + 1$. With no loss of generality we can assume that $\Delta x = \Delta y$.

In contrast with standard cyclic reduction for solving (1.3), the accuracy of ACR can be varied and the operation count is proportional to $N^2 \log t$, where t is the number

* Received by the editors November 7, 1984; accepted for publication (in revised form) February 7, 1986.
† National Center for Atmospheric Research Computing Facility, Boulder, Colorado 80307, which is sponsored by the National Science Foundation.

of digits that are carried in the computation. The reduction in computation is achieved by replacing the rational polynomials of high degree that occur in cyclic reduction with rational approximations of substantially lower degree. It will be shown that the degree of these rational approximations depends only on t and not on the size N of the grid. For example, on the CRAY-1 with about 15 decimal digits, we estimate that rational approximations of degree less than 10 can be used for any grid size. Hence, for a 1024×1024 grid, a rational polynomial of degree 1024 could be replaced by a rational approximation of degree 10 without loss of accuracy.

When implemented in this manner, ACR is essentially a direct method. However, further economies are possible, since it is usually not necessary to compute with the full 15 digits. Indeed, an exact solution of (1.3) provides only an approximate solution of (1.1), which is likely accurate to about three decimal digits. Therefore 10 is likely the highest degree of a rational approximation on the CRAY-1, and in most cases it will be lower. For example, in time-dependent problems a particularly good initial estimate of the solution can be obtained by extrapolating the solution from previous time steps. For these problems one digit of accuracy is more than sufficient.

Cyclic reduction is reviewed in the next section, and ACR is developed in § 3. The rational approximations are discussed, and the accuracies of various rational approximations are presented for different grid sizes. The operation counts are developed in § 4.

2. Cyclic reduction. In this section we will review the method of cyclic reduction for solving the large sparse linear system (1.3) of finite difference equations. As mentioned previously, we have assumed that the solution is zero on the boundary. The detailed development for the other standard boundary conditions is given in either [5] or [6]. We will begin with a description of the *unstable* cyclic reduction algorithm, and consider the case $N = 7$, for which the system (1.3) has the form

$$
\begin{aligned}
\mathbf{A}\mathbf{u}_1 + \mathbf{u}_2 &= \mathbf{g}_1, \\
\mathbf{u}_1 + \mathbf{A}\mathbf{u}_2 + \mathbf{u}_3 &= \mathbf{g}_2, \\
\mathbf{u}_2 + \mathbf{A}\mathbf{u}_3 + \mathbf{u}_4 &= \mathbf{g}_3, \\
\mathbf{u}_3 + \mathbf{A}\mathbf{u}_4 + \mathbf{u}_5 &= \mathbf{g}_4, \\
\mathbf{u}_4 + \mathbf{A}\mathbf{u}_5 + \mathbf{u}_6 &= \mathbf{g}_5, \\
\mathbf{u}_5 + \mathbf{A}\mathbf{u}_6 + \mathbf{u}_7 &= \mathbf{g}_6, \\
\mathbf{u}_6 + \mathbf{A}\mathbf{u}_7 &= \mathbf{g}_7.
\end{aligned} \tag{2.1}
$$

If, beginning with the second equation, we multiply every other equation by $-\mathbf{A}$ and add the adjacent two equations, we obtain the following reduced system:

$$
\begin{aligned}
\mathbf{A}^{(1)}\mathbf{u}_2 + \mathbf{u}_4 &= \mathbf{g}_2^{(1)}, \\
\mathbf{u}_2 + \mathbf{A}^{(1)}\mathbf{u}_4 + \mathbf{u}_6 &= \mathbf{g}_4^{(1)}, \\
\mathbf{u}_4 + \mathbf{A}^{(1)}\mathbf{u}_6 &= \mathbf{g}_6^{(1)},
\end{aligned} \tag{2.2}
$$

where

$$
\mathbf{A}^{(1)} = 2\mathbf{I} - \mathbf{A}^2 \tag{2.3}
$$

and

$$
\mathbf{g}_j^{(1)} = \mathbf{g}_{j-1} - \mathbf{A}\mathbf{g}_j + \mathbf{g}_{j+1} \tag{2.4}
$$

for $j=2$, 4, and 6. But (2.2) has the same form as (2.1), and hence, we can repeat the process. If we multiply the second equation in (2.2) by $-\mathbf{A}^{(1)}$ and add the other two equations, we obtain

$$\mathbf{A}^{(2)}\mathbf{u}_4=\mathbf{g}_4^{(2)}, \tag{2.5}$$

where

$$\mathbf{A}^{(2)}=2\mathbf{I}-[\mathbf{A}^{(1)}]^2 \tag{2.6}$$

and

$$\mathbf{g}_4^{(2)}=\mathbf{g}_2^{(1)}-\mathbf{A}^{(1)}\mathbf{g}_4^{(1)}+\mathbf{g}_6^{(1)}. \tag{2.7}$$

The unknown vectors $\mathbf{u}_j$ can now be determined in the following sequence, which is called the backsubstitution phase. First, $\mathbf{u}_4$ is determined from (2.5). Next, $\mathbf{u}_4$ is substituted into the first and last equations in (2.2), which are then solved for $\mathbf{u}_2$ and $\mathbf{u}_6$, respectively. Finally, $\mathbf{u}_2$, $\mathbf{u}_4$, and $\mathbf{u}_6$ are substituted into (2.1), which then decouple, so that $\mathbf{u}_1$, $\mathbf{u}_3$, $\mathbf{u}_5$ and $\mathbf{u}_7$ can be determined from the first, third, fifth, and seventh equations, respectively.

Consider now the unstable cyclic reduction algorithm for the case of $N=2^k-1$ for some integer k. The algorithm divides naturally into three phases, namely, preprocessing, reduction, and backsubstitution.

2.1. Preprocessing. In this phase we compute the matrices $\mathbf{A}^{(r)}$. Define $\mathbf{A}^{(0)}=\mathbf{A}$; then

$$\mathbf{A}^{(r)}=2\mathbf{I}-[\mathbf{A}^{(r-1)}]^2, \qquad r=1,\cdots,k-1. \tag{2.8}$$

From (2.8) $\mathbf{A}^{(r)}$ is a polynomial of degree 2^r in $\mathbf{A}$, which will be denoted as $\mathbf{p}_{2^r}(\mathbf{A})$. Most of the properties of $\mathbf{p}_{2^r}(\mathbf{A})$ can be determined from its scalar analogue, $p_{2^r}(x)$, which is defined recursively beginning with $p_1(x)=x$ and continuing with

$$p_{2^r}(x)=2-[p_{2^{r-1}}(x)]^2. \tag{2.9}$$

If we define θ by $x=-2\cos\theta$, then for $-2\leqq x\leqq 2$

$$p_{2^r}(x)=-2\cos 2^r\theta \tag{2.10}$$

or

$$p_{2^r}(x)=-C_{2^r}(x), \tag{2.11}$$

where $C_{2^r}(x)$ represents the Chebyshev polynomials with zeros:

$$\theta_m^{(r)}=2\cos\left[(2m-1)\frac{\pi}{2^{r+1}}\right], \qquad m=1,\cdots,2^r. \tag{2.12}$$

Both $p_{2^r}(x)$ and $\mathbf{p}_{2^r}(\mathbf{A})$ can be expressed in factored form, in particular

$$\mathbf{p}_{2^r}(\mathbf{A})=\prod_{m=1}^{2^r}(\mathbf{A}-\theta_m^{(r)}\mathbf{I}). \tag{2.13}$$

All the matrix computations can be performed using this factored form. For example, consider the solution of (2.5), which can be written

$$(\mathbf{A}-\theta_1^{(2)}\mathbf{I})(\mathbf{A}-\theta_2^{(2)}\mathbf{I})(\mathbf{A}-\theta_3^{(2)}\mathbf{I})(\mathbf{A}-\theta_4^{(2)}\mathbf{I})\mathbf{u}_4=\mathbf{g}_4^{(2)}. \tag{2.14}$$

If we define $\mathbf{w}_0=\mathbf{g}_4^{(2)}$ and

$$\mathbf{w}_1=(\mathbf{A}-\theta_2^{(2)}\mathbf{I})(\mathbf{A}-\theta_3^{(2)}\mathbf{I})(\mathbf{A}-\theta_4^{(2)}\mathbf{I})\mathbf{u}_4 \tag{2.15}$$

then (2.14) becomes

$$(\mathbf{A}-\theta_1^{(2)}\mathbf{I})\mathbf{w}_1=\mathbf{w}_0, \tag{2.16}$$

which can be solved for $\mathbf{w}_1$ using the efficient tridiagonal form of the Gauss algorithm. If we then define

$$\mathbf{w}_2=(\mathbf{A}-\theta_3^{(2)}\mathbf{I})(\mathbf{A}-\theta_4^{(2)}\mathbf{I})\mathbf{u}_4, \tag{2.17}$$

(2.15) becomes

$$(\mathbf{A}-\theta_2^{(2)}\mathbf{I})\mathbf{w}_2=\mathbf{w}_1. \tag{2.18}$$

We can determine $\mathbf{w}_2$ by also using the efficient tridiagonal Gauss algorithm. If we continue this process by defining

$$\mathbf{w}_3=(\mathbf{A}-\theta_4^{(2)}\mathbf{I})\mathbf{u}_4, \tag{2.19}$$

then we can determine $\mathbf{w}_3$ in the same manner, and finally, from (2.19) we can determine the solution $\mathbf{u}_4$. All of the matrix computations can be performed in this manner using the factored form (2.13), and hence it is unnecessary to explicitly compute and store the $\mathbf{A}^{(r)}$.

2.2. Reduction. Define $\mathbf{g}_j^{(0)}=\mathbf{g}_j$; then for $r=1,2,\cdots,k-1$ and $j=2^r,\ 2\cdot 2^r,\ 3\cdot 2^r,\cdots,(2^{k-r}-1)2^r$ compute the right-hand sides:

$$\mathbf{g}_j^{(r)}=\mathbf{g}_{j-2^{r-1}}^{(r-1)}-\mathbf{A}^{(r-1)}\mathbf{g}_j^{(r-1)}+\mathbf{g}_{j+2^{r-1}}^{(r-1)}. \tag{2.20}$$

2.3. Backsubstitution. For $r=k-1,k-2,\cdots,0$ and $j=2^r,3\cdot 2^r,\cdots,(2^{k-r}-1)2^r$ compute the solution $\mathbf{u}_j$ by solving

$$\mathbf{A}^{(r)}\mathbf{u}_j=\mathbf{g}_j^{(r)}-\mathbf{u}_{j-2^r}-\mathbf{u}_{j+2^r}, \tag{2.21}$$

where $\mathbf{u}_j=0$ for j outside the range $1,\cdots,2^k-1$.

Buneman [1] stabilized this algorithm by introducing the quantities $\mathbf{p}_j^{(r)}$ and $\mathbf{q}_j^{(r)}$ such that

$$\mathbf{g}_j^{(r)}=\mathbf{A}^{(r)}\mathbf{p}_j^{(r)}+\mathbf{q}_j^{(r)}. \tag{2.22}$$

To compute $\mathbf{p}_j^{(r)}$ and $\mathbf{q}_j^{(r)}$ we substitute (2.22) into (2.20) and use (2.8) in the form $[\mathbf{A}^{(r-1)}]^2=2\mathbf{I}-\mathbf{A}^{(r)}$, and we obtain a stable reduction phase:

$$\begin{aligned}\mathbf{p}_j^{(r)}&=\mathbf{p}_j^{(r-1)}+[\mathbf{A}^{(r-1)}]^{-1}(\mathbf{q}_j^{(r-1)}-\mathbf{p}_{j-2^{r-1}}^{(r-1)}-\mathbf{p}_{j+2^{r-1}}^{(r-1)}),\\ \mathbf{q}_j^{(r)}&=\mathbf{q}_{j-2^{r-1}}^{(r-1)}+\mathbf{q}_{j+2^{r-1}}^{(r-1)}-2\mathbf{p}_j^{(r)}.\end{aligned} \tag{2.23}$$

Substituting (2.22) into (2.21), we obtain the stable backsubstitution phase:

$$\mathbf{u}_j=\mathbf{p}_j^{(r)}+[\mathbf{A}^{(r)}]^{-1}(\mathbf{q}_j^{(r)}-\mathbf{u}_{j-2^r}-\mathbf{u}_{j+2^r}). \tag{2.24}$$

In summary, the stable cyclic reduction algorithm begins with the reduction phase, which consists of computing $\mathbf{p}_j^{(r)}$ and $\mathbf{q}_j^{(r)}$ from (2.23), beginning with $\mathbf{p}_j^{(0)}=0$ and $\mathbf{q}_j^{(0)}=\mathbf{g}_j$. The second term in the first equation in (2.23) is computed using the factored form of $\mathbf{A}^{(r)}$ and the repeated solution of tridiagonal systems of equations. The solution $\mathbf{u}_j$ is then computed from (2.24), again using the factored form of $\mathbf{A}^{(r)}$ and the repeated solution of tridiagonal systems.

3. The approximation of $[\mathbf{A}^{(r)}]^{-1}$. It is evident from the results in the preceding section that most of the computation in cyclic reduction occurs in the repeated solution of the tridiagonal systems (2.23) and (2.24). The number of these systems is determined by the order of the rational matrix polynomials $[\mathbf{A}^{(r)}]^{-1}$, which can be as large as the

number of grid points in one direction. In this section we will show that the rational matrix polynomial $[\mathbf{A}^{(r)}]^{-1}$ can be replaced by rational approximations of lower degree. In particular, we show that for any $\varepsilon > 0$ there exists an integer $N(\varepsilon)$ and rational polynomial $\mathbf{R}(\mathbf{A})$ with degree lower than $N(\varepsilon)$, such that

$$\|[\mathbf{A}^{(r)}]^{-1} - \mathbf{R}(\mathbf{A})\|_2 < \varepsilon \tag{3.1}$$

for *all r.*

In practice the $[\mathbf{A}^{(r)}]^{-1}$ can be replaced by rational polynomials of substantially lower degree. For example, we estimate that $[\mathbf{A}^{(r)}]^{-1}$ with degree 256 can be approximated to 15 decimal digits by a rational polynomial of degree 10. Hence the amount of computation can be significantly reduced without a reduction in accuracy.

We begin now to develop these results. It will be sufficient to consider the approximation problem for the scalar analogue $p_n^{-1}(x)$ of the matrix polynomials $\mathbf{p}_n^{-1}(\mathbf{A})$. If $R(x)$ is a rational approximation of $p_n(x)$, such that

$$|p_n^{-1}(x) - R(x)| < \varepsilon \tag{3.2}$$

on an interval that contains the eigenvalues of $\mathbf{A}$, then

$$\|\mathbf{p}_n^{-1}(\mathbf{A}) - \mathbf{R}(\mathbf{A})\|_2 < \varepsilon. \tag{3.3}$$

Since the eigenvalues of $\mathbf{A}$ are

$$\lambda_i = -2\left(1 + 2\sin^2 i\frac{\pi}{2(N+1)}\right), \tag{3.4}$$

the expression (3.2) must be valid on the interval $-6 \leqq x \leqq -2$. The representation (2.10) is not valid on this interval and therefore we select the form

$$p_n(x) = -2\cosh\left[n\cosh^{-1}\left(-\frac{x}{2}\right)\right], \tag{3.5}$$

which is valid for $x \leqq -2$.

THEOREM. *For any $\varepsilon > 0$ and n, there exists a rational approximation $R(x)$ such that*

$$|p_n^{-1}(x) - R(x)| < \varepsilon, \qquad x < -2, \tag{3.6}$$

where the degree of $R(x)$ is less than some integer $N(\varepsilon)$, which depends only on ε.

Proof. For $x > 0$ define the polynomials

$$q_n(x) = -\frac{1}{2}p_n\left(-2 - \frac{x}{n^2}\right); \tag{3.7}$$

then we will first show that

$$\lim_{n\to\infty} q_n(x) = \cosh\sqrt{x}. \tag{3.8}$$

From (3.5) and (3.7)

$$q_n(x) = \cosh n\left[\cosh^{-1}\left(1 + \frac{x}{2n^2}\right)\right] \tag{3.9}$$

or

$$q_n(x) = \cosh n\left\{\ln\left[1 + \frac{x}{2n^2} \pm \left(\frac{x}{n^2} + \frac{x^2}{4n^4}\right)^{1/2}\right]\right\}. \tag{3.10}$$

But for small ε, $\ln(1+\varepsilon)\approx\varepsilon$, which leads to the desired result (3.8) which is essentially the same as that given in Appendix B of [2].

Now assume that ε is given; from (3.8) we can determine N such that

$$|q_n^{-1}(y)-\operatorname{sech}\sqrt{y}|<\frac{\varepsilon}{4} \tag{3.11}$$

for all $n>N$ and $y>0$. Next determine a rational polynomial $R(y)$ such that

$$|R(y)-\operatorname{sech}\sqrt{y}|<\frac{\varepsilon}{4}. \tag{3.12}$$

Then

$$\begin{aligned}|p_n^{-1}(x)+2R[-n^2(x+2)]| &= |-2q_n^{-1}[-n^2(x+2)]+2R[-n^2(x+2)]| \\ &= 2|q_n^{-1}(y)-R(y)|\end{aligned} \tag{3.13}$$

or

$$\begin{aligned}|p_n^{-1}(x)+2R[-n^2(x+2)]| &\leqq 2|q_n^{-1}(y)-\operatorname{sech}\sqrt{y}| \\ &\quad +2|\operatorname{sech}\sqrt{y}-R(y)|\leqq 2\left(\frac{\varepsilon}{4}+\frac{\varepsilon}{4}\right)=\varepsilon.\end{aligned} \tag{3.14}$$

Therefore, $-2R[-n^2(x+2)]$ provides an accurate approximation to *all* rational polynomials $p_n^{-1}(x)$ for $n>N$. To complete the proof, we must set $N(\varepsilon)$ equal to the maximum of the degrees of $R(x)$ and of the finite number of rational approximations for $n<N$, which in practice are found to be less than N. We note from its Taylor series that $\operatorname{sech}\sqrt{y}$ is an analytic function, and therefore, accurate rational approximations of relatively low degree are expected. Indeed, we observe the geometric convergence that has been proven [3] for rational approximations to e^{-z}.

Instead of selecting $R(x)$ as a rational approximation to $\operatorname{sech}\sqrt{y}$, it is somewhat more efficient to determine a rational approximation to $p_n^{-1}(x)$ directly. The error ε in such an approximation is tabulated in Table 1. The degree n of $p_n(x)$ changes from column to column. The rational approximations are chosen with the degree of the numerator m equal to the degree of the denominator. These approximations are designated by $R_{m,m}(x)$ in Table 1, where the degree m changes from row to row. The entries are the maximum of $|p_n^{-1}(x)-R_{m,m}(x)|$ on the interval $-6\leqq x\leqq -2$, where $R_{m,m}(x)$ is computed as the best fit in the Chebyshev sense.

TABLE 1

$$\varepsilon=\max_{-6\leqq x\leqq -2}|p_n^{-1}(x)-R_{m,m}(x)|$$

	n				
m	32	64	128	256	1024
1	1.501×10^{-2}	1.508×10^{-2}	1.509×10^{-2}	1.510×10^{-2}	1.510×10^{-2}
2	6.847×10^{-4}	6.984×10^{-4}	7.019×10^{-4}	7.028×10^{-4}	7.030×10^{-4}
3	2.907×10^{-5}	3.069×10^{-5}	3.111×10^{-5}	3.121×10^{-5}	3.125×10^{-5}
4	1.176×10^{-6}	1.320×10^{-6}	1.359×10^{-6}	1.368×10^{-6}	1.371×10^{-6}
5	4.513×10^{-8}	5.586×10^{-8}	5.885×10^{-8}	5.962×10^{-8}	5.986×10^{-8}
6	1.628×10^{-9}	2.322×10^{-9}	2.532×10^{-9}	2.587×10^{-9}	2.604×10^{-9}
7	5.463×10^{-11}	9.483×10^{-11}	1.083×10^{-10}	1.119×10^{-10}	1.131×10^{-10}

It is evident from Table 1 that approximate cyclic reduction can be used in several different modes. In the first mode we select rational approximations with accuracy that is comparable to machine accuracy. In this mode the result is the same as using cyclic reduction, but with a significant reduction in computing time. In the second mode we select rational approximations of still lower degree in order to take advantage of the fact that one does not need to compute with full machine precision. A third mode is described in § 4 in which rational approximations with linear factors are used together with iterative improvement in order to develop an iterative method.

The geometric convergence factor $\varepsilon^{1/m}$ of the rational polynomial $R(x)$ is given in Table 2 for $n = 1024$. The quantity $-(\log_2 \varepsilon)/m$, which is also given in Table 2, will be used in the next section to determine the operation count for ACR.

We can estimate from Table 1 and Table 2 that the rational polynomial $p_n^{-1}(x)$ of degree 1024 can be approximated to a machine precision of 15 decimal digits by a rational approximation of degree 10. Therefore, the application of $\mathbf{p}_{1024}^{-1}(\mathbf{A})$ can be computed with ten tridiagonal inversions instead of 1024. It will be shown that the tridiagonal matrix multiplications that result from the polynomial in the numerator of the rational approximation can be eliminated.

The rational approximations can also be made on a discrete set of points rather than on the continuous interval $-6 \leqq x \leqq -2$ that was used in Table 1. In fact, the approximation can be made on the set of points that consists of the eigenvalues λ_i of $\mathbf{A}$ that are defined in (3.4). For this case the approximations depend on the number of points or on the order of the matrix $\mathbf{A}$. The accuracies of linear rational approximations are given in Table 3 for the case in which the order of $\mathbf{A}$ is $M = 127$.

The entries in this table can be compared with those in the first row of Table 1. The overall error in ACR associated with using the approximations given in Table 3

TABLE 2

m	$\varepsilon^{1/m}$	$-\frac{1}{m}\log_2 \varepsilon$
1	0.0151	6.05
2	0.0265	5.24
3	0.0315	4.99
4	0.0342	4.87
5	0.0359	4.80
6	0.0371	4.75
7	0.0379	4.72

TABLE 3

$\varepsilon = \max_{\lambda_i} |p_n^{-1}(x) - R_{1,1}(x)|$

n	ε
2	2.960×10^{-3}
4	9.894×10^{-3}
8	1.349×10^{-2}
16	1.402×10^{-2}
32	1.208×10^{-2}
64	4.788×10^{-3}
128	3.195×10^{-4}

will be related to the maximum error, which is 1.402×10^{-2}, compared with 1.510×10^{-2} in Table 1. Hence the results would be comparable. On the other hand, there is a qualitative difference between the two sets of rational approximations; namely, the errors in Table 3 have a maximum at $n = 16$, after which they decline to 3.195×10^{-4} at $n = 128$. This is simply because the values of $p_n(\lambda_i)$ decrease with increasing n, compared with the values of $p_n(x)$ on the interval $-6 \leqq x \leqq -2$. For this reason it would be possible to use approximations of lower degree for values of n that are outside the region of maximum error which would further reduce the amount of computation. The effect that this might have on reducing the complexity of ACR will not be pursued in this paper.

4. Operation counts. We will begin this section with a derivation of the operation count for cyclic reduction. As in [5] an operation is defined as a multiplication or division plus an addition or a subtraction. This definition is consistent with current trends in large-scale computer architectures, in which it is often possible to "overlap" or "chain" these operations. However, the number of arithmetic operations is not an entirely accurate measure of actual computing time. Data access and operation sequences are also significant factors that are machine-dependent. Nevertheless, having stated these caveats, we will consider only the arithmetic operations for lack of a better measure. Furthermore, we consider only operations that contribute to the asymptotic operation count. This includes the solution of the tridiagonal systems in (2.23) and (2.24), but not the vector additions or subtraction.

The operations required for solving the tridiagonal systems are given in Table 4. The entries include the operations in both the reduction and the backsubstitution phases. For example, cycle 1 includes the first cycle in the reduction phase, or equation (2.23) with $r = 1$, and the last cycle of the backsubstitution phase, or (2.24) with $r = 0$. Similarly, cycle 2 includes the second cycle of the reduction phase and the next-to-the-last cycle of backsubstitution, and so forth. The first row contains the number of values of j in (2.23) and (2.24). The second row contains the degree of $\mathbf{A}^{(r-1)}$ or the number of tridiagonal systems that must be solved for each value of j. The third row contains the product of the first two rows multiplied by $3N$, which is the number of operations for the solution of a single tridiagonal system; that number is obtained by noting that the off-diagonal elements are 1 and assuming that the LU decompositions are recomputed and not stored.

TABLE 4
Operation count for cyclic reduction.

Cycle	1	2	3	···	$\log_2 N$
Number of systems in (2.23), (2.24)	N	$N/2$	$N/4$		1
Number of tridiagonal factors per system	1	2	4		N
Total operations per cycle	$3N^2$	$3N^2$	$3N^2$		$3N^2$
Total operations	$3N^2 \log_2 N$				

Next we consider the operation count for approximate cyclic reduction. To this end we consider the solution of a linear system in which the rational polynomial $[\mathbf{A}^{(r)}]^{-1}$ is replaced by its rational approximate $\mathbf{R}(\mathbf{A})$. In this event we wish to compute

$$\mathbf{x} = \delta \frac{\prod (\mathbf{A} - \alpha_i \mathbf{I})}{\prod (\mathbf{A} - \beta_i \mathbf{I})} \mathbf{b}, \tag{4.1}$$

where the vector $\mathbf{b}$ is given; α_i, β_i are the zeros of the polynomials in the rational approximation; and δ is the multiplicative factor. The development following equation (2.13) can be easily generalized to this case. If we define $\mathbf{w}_0 = \mathbf{b}$ and compute

$$\mathbf{w}_i = \frac{(\mathbf{A} - \alpha_i \mathbf{I})}{(\mathbf{A} - \beta_i \mathbf{I})} \mathbf{w}_{i-1}, \tag{4.2}$$

then $\mathbf{x} = \delta \mathbf{w}_n$. The vector $\mathbf{w}_i$ is computed from $\mathbf{w}_{i-1}$ by solving the system

$$(\mathbf{A} - \beta_i \mathbf{I})\mathbf{w}_i = (\mathbf{A} - \alpha_i \mathbf{I})\mathbf{w}_{i-1}. \tag{4.3}$$

The matrix multiplication on the right side can be eliminated following the development given in [4]. If we rewrite (4.3) in the form

$$(\mathbf{A} - \beta_i \mathbf{I})(\mathbf{w}_i - \mathbf{w}_{i-1}) = (\beta_i - \alpha_i)\mathbf{w}_{i-1} \tag{4.4}$$

and solve the following system for $\mathbf{z}$

$$(\mathbf{A} - \beta \mathbf{I})\mathbf{z} = (\beta_i - \alpha_i)\mathbf{w}_{i-1}, \tag{4.5}$$

then

$$\mathbf{w}_i = \mathbf{z} + \mathbf{w}_{i-1}. \tag{4.6}$$

The case in which the zeros β_i are complex can be handled by solving pentadiagonal systems with real coefficients that result by combining two linear factors into a quadratic factor. Following the LU decomposition of a pentadiagonal system, which can be included in the preprocessing, the cost of solving a pentadiagonal system is about double the cost of solving a tridiagonal system. However the number of pentadiagonal systems is half the number of tridiagonal systems and therefore the total amount of computation is about the same.

Given the number t of binary digits to be carried, then from Table 2 we can determine empirically that the maximum degree of any polynomial m satisfies

$$m \approx -\left(\frac{1}{5}\right) \log_2 \varepsilon = \left(\frac{1}{5}\right) t. \tag{4.7}$$

The operations required for solving the tridiagonal systems are given in Table 4. The operations required for ACR are given in Table 5. The algorithm is the same as cyclic reduction for the first $\log_2 m$ cycles, because the order of the polynomials is less than the upper limit m. After the $\log_2 m$ cycle, rational approximations with bounded degree m are used, and therefore the number of tridiagonal systems in the second row

TABLE 5
Operation count for approximate cyclic reduction.

Cycle	1	2	···	$\log_2 m$	$\log_2 m + 1$	···	$\log_2 N$
Number of systems in (2.23), (2.24)	N	$N/2$		N/m	$N/2m$		1
Number of tridiagonal factors per system	1	2		m	m		m
Total operations per cycle	$3N^2$	$3N^2$		$3N^2$	$1.5N^2$		$3mN$
Total operations	$3N^2 \log_2 t$						

remains constant at m. The total operation count ops (ACR) is the sum of the operations in the third row or

$$\text{ops (ACR)} = 3N^2 \log_2 m + \sum_{i=1}^{\log_2(N/m)} \frac{3n^2}{2^i} \leqq 3N^2(\log_2 m + 1) \tag{4.8}$$

for an asymptotic count of

$$\text{ops (ACR)} = 3N^2 \log_2 t. \tag{4.9}$$

It is interesting to compare this method with existing methods. In the limit the accuracy of the solution to the large linear system must be comparable to the truncation error, which is $\varepsilon = O(N^{-2})$. From (4.9)

$$\text{ops (ACR)} = 3(N^2 \log_2 \log_2 N), \tag{4.10}$$

which is comparable to the FACR algorithm [5].

Although the operation count for ACR is comparable to that of the FACR algorithm, it is nevertheless applicable to a larger class of problems:

(a) The amount of computation can be further reduced for applications such as time-dependent problems, in which possibly only a single digit of accuracy is needed.

(b) Cyclic reduction is applicable to a larger class of problems than the FACR method. Since the FACR method uses the Fourier method, there are certain mild restrictions on the number of grid points that are not present in cyclic reduction [7].

(c) ACR can be generalized to the separable elliptic equation. Generalized cyclic reduction [4] also makes use of high-degree rational matrix polynomials which could be replaced by rational approximations of lower degree.

With the exception of the discussion following (4.9), we have been careful to discuss error in the context of t, which is the number of digits that are carried in the computation. This of course differs from the error in the overall computation but it does not change the asymptotic complexity results given in (4.10). For example, if we use ACR to compute the solution on a 127×127 grid using the rational approximations $R_{1,1}$ with linear factors that were generated for Table 3, then we obtain the errors in the overall computation that are given in Table 6. Here we have developed an iterative method by combining the methods of iterative improvement and ACR. The exact solution to the finite difference equations is $u_{i,j}$, and the approximate solution at the kth iteration is $u_{i,j}^{(k)}$. The average convergence factor is $\rho = (3.015 \times 10^{-6})^{0.125} = 0.204$.

Finally we include an observation about ACR that was made by one of the referees. The solution of the tridiagonal systems using the recurrence (4.5), (4.6) is a sequential

TABLE 6
$\varepsilon = \max_{i,j} |u_{i,j}^{(k)} - u_{i,j}|$

k	ε
1	7.876×10^{-1}
2	1.034×10^{-1}
3	1.660×10^{-2}
4	2.838×10^{-3}
5	5.128×10^{-4}
6	8.938×10^{-5}
7	1.600×10^{-5}
8	3.015×10^{-6}

computation that makes cyclic reduction somewhat difficult to parallelize. On the other hand, if ACR is used with low order rational approximations, the sequential part of the computation is reduced or in fact eliminated in the event $R_{1,1}$ is used throughout.

Acknowledgments. I wish to acknowledge the valuable discussions about rational approximations with Professor Beresford N. Parlett and Professor Gerald D. Taylor. I also wish to acknowledge the comments made by one of the referees that led to the observation at the end of the manuscript.

REFERENCES

[1] O. BUNEMAN, *A compact non-iterative Poisson solver*, Report 294, Stanford University Institute for Plasma Research, Stanford, CA, 1969.

[2] B. N. PARLETT, *The Symmetric Eigenvalue Problem*, Prentice-Hall Series in Computational Mathematics, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[3] E. B. SAFF, A. SCHONHAGE AND R. S. VARGA, *Geometric convergence to e^{-z} by rational functions with real poles*, Numer. Math., 25 (1976), pp. 307–322.

[4] P. N. SWARZTRAUBER, *A direct method for the discrete solution of separable elliptic equations*, SIAM J. Numer. Anal., 11 (1974), pp. 1136–1150.

[5] ———, *The methods of cyclic reduction, Fourier analysis and the* FACR *algorithm for the discrete solution of Poisson's equation on a rectangle*, SIAM Rev., 19 (1977), pp. 490–501.

[6] ———, *Fast Poisson Solvers*, MAA Studies in Numer. Anal., Vol. 24, Mathematical Association of America, 1984.

[7] R. A. SWEET, *A cyclic reduction algorithm for block tridiagonal systems of arbitrary dimension*, SIAM J. Numer. Anal., 11 (1974), pp. 506–520.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 3, May 1987
© 1987 Society for Industrial and Applied Mathematics
002

A NOTE ON DOWNDATING THE CHOLESKY FACTORIZATION*

A. W. BOJANCZYK†¶, R. P. BRENT†, P. VAN DOOREN‡ AND F. R. DE HOOG§

Abstract. We analyse and compare three algorithms for "downdating" the Cholesky factorization of a positive definite matrix. Although the algorithms are closely related, their numerical properties differ. Two algorithms are stable in a certain "mixed" sense while the other is unstable. In addition to comparing the numerical properties of the algorithms, we compare their computational complexity and their suitability for implementation on parallel or vector computers.

Key words. Cholesky factorization, numerical stability, computational complexity, parallel computation, forward error analysis, backward error analysis, mixed stability, downdating, updating, rank-one correction, Sigma-unitary transformation, hyperbolic transformation, orthogonal transformation

AMS(MOS) subject classifications. 65F05, 65F30, 65G05

1. Introduction. The Cholesky downdating problem considered in this note is: given an upper triangular matrix $\mathbf{R} \in R^{n\times n}$ and a vector $\mathbf{x} \in R^n$ such that $\mathbf{R}^T\mathbf{R} - \mathbf{x}\mathbf{x}^T$ is positive definite, find an upper triangular matrix $\mathbf{U}$ such that

$$\mathbf{U}^T\mathbf{U} = \mathbf{R}^T\mathbf{R} - \mathbf{x}\mathbf{x}^T.$$

By our assumption of positive definiteness, $\mathbf{U}$ exists and is unique up to the signs of its diagonal elements (Golub and van Loan [7]).

The Cholesky downdating problem is closely related to that of downdating a QR factorization. To show this, let

$$\mathbf{A} = \begin{bmatrix} \mathbf{x}^T \\ \tilde{\mathbf{A}} \end{bmatrix} = \mathbf{Q}\begin{bmatrix} \mathbf{R} \\ \mathbf{O} \end{bmatrix},$$

where $\mathbf{A} \in R^{m\times n}$, $m > n$ and $\mathbf{Q} \in R^{m\times m}$ is an orthogonal matrix. The problem is to find an orthogonal matrix $\tilde{\mathbf{Q}} \in R^{(m-1)\times(m-1)}$ such that

$$\tilde{\mathbf{A}} = \tilde{\mathbf{Q}}\begin{bmatrix} \mathbf{U} \\ \mathbf{O} \end{bmatrix}.$$

Thus we have

$$\mathbf{R}^T\mathbf{R} = \mathbf{A}^T\mathbf{A} = \mathbf{x}\mathbf{x}^T + \tilde{\mathbf{A}}^T\tilde{\mathbf{A}} = \mathbf{x}\mathbf{x}^T + \mathbf{U}^T\mathbf{U}$$

which is precisely the Cholesky downdating problem described previously.

The problem of downdating a QR factorization occurs for example when an observation (such as an outlier) is deleted from a regression. An algorithm for computing $\tilde{\mathbf{Q}}$ and $\mathbf{U}$ has been described by Gill, Golub, Murray and Saunders [5] and it is not difficult to show that the algorithm is backward stable in the classical sense (Wilkinson [10]). This rather satisfactory state of affairs is due to the fact that we know precisely

* Received by the editors September 11, 1985; accepted for publication (in revised form) February 17, 1986.

† Centre for Mathematical Analysis, The Australian National University, GPO Box 4, Canberra ACT 2601, Australia.

‡ Philips Research Laboratory, Av. van Becelaere 2, Box 8, B-1170 Brussels, Belgium.

§ Division of Mathematics and Statistics, CSIRO, GPO Box 1965, Canberra ACT 2601, Australia.

¶ Present address, Department of Computer Science, Washington University, St. Louis, Missouri 63130.

which row (in our case, the first row $\mathbf{x}^T$) of $\mathbf{A}$ is to be deleted. However, this is not the case when downdating the Cholesky factorization. Here $\mathbf{x}^T$ may bear no relation to rows of $\mathbf{R}$ and the only requirement is that the matrix $\mathbf{R}^T\mathbf{R}-\mathbf{x}\mathbf{x}^T$ is positive definite. Thus, the best we can expect when computations are performed with finite precision is that the computed $\mathbf{U}$ is the exact Cholesky factor of $\tilde{\mathbf{R}}^T\tilde{\mathbf{R}}-\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T$ where $\tilde{\mathbf{R}}$ and $\tilde{\mathbf{x}}$ are close to $\mathbf{R}$ and $\mathbf{x}$, respectively. However, Stewart [9] has shown that under certain circumstances, small perturbations in $\mathbf{R}$ and $\mathbf{x}$ may lead to large perturbations in $\mathbf{U}$. Thus, we do not recommend that algorithms based on downdating Cholesky factors (as described in the present note) be used to downdate the triangular factor in the *QR* decomposition.

Nevertheless, the Cholesky downdating problem is important in its own right. Let $\mathbf{B}$ be a positive definite matrix for which a Cholesky factorization has been obtained, that is

$$\mathbf{B}=\mathbf{R}^T\mathbf{R}.$$

Now suppose we wish to compute the Cholesky factors of the positive definite matrix

$$\tilde{\mathbf{B}}:=\mathbf{B}-\mathbf{x}\mathbf{x}^T=:\mathbf{U}^T\mathbf{U}.$$

Then,

$$\mathbf{U}^T\mathbf{U}=\mathbf{R}^T\mathbf{R}-\mathbf{x}\mathbf{x}^T.$$

This occurs quite often as a subproblem when a positive definite matrix is perturbed by a matrix of low rank. Often the problem can be solved by a sequence of rank-1 updates followed by a sequence of rank-1 downdates. Such a situation occurs, for example, in structural problems when elements are added and deleted from a design (see for example, Argyris and Roy [1]). For another application of Cholesky downdating, see Bojanczyk, Brent and de Hoog [3].

In § 2a, we describe a method implemented in the LINPACK package, "Algorithm A," analysed by Stewart [9]. In § 2b we describe a recursive method, "Algorithm B," which has some computational advantages over Algorithm A but does not have comparable stability properties. The recursive algorithm is then modified in § 2c so that its stability properties are improved. The resulting algorithm will be called "Algorithm C."

The main results of this note are given in § 3, where we present an error analysis of Algorithms B and C. In particular, we show that Algorithm C is not stable in the classical sense but does satisfy a "mixed" error bound which shows that it gives forward errors of the same order as an algorithm which is backward stable in the classical sense. A similar result was obtained by Stewart [9] for Algorithm A. The best error bounds that could be obtained for Algorithm B depend on the data and can be arbitrarily large, from which we conclude that it is probably numerically unstable.

In § 4 we show that Algorithms B and C are preferable to Algorithm A from the point of view of computational complexity, as they require about 20 percent fewer floating point multiplications ($2n^2+O(n)$ versus $5n^2/2+O(n)$). We also show that Algorithms B and C are more readily implemented on parallel or vector computers than is Algorithm A. Finally, in § 5 some numerical results verifying our stability analysis are presented.

To summarize, Algorithm C appears the best overall, since it is faster than Algorithm A and more stable than Algorithm B.

2. Three different approaches. In this section we compare and relate three different approaches that can be used for solving the downdating problem.

2a. The LINPACK method (Algorithm A). This method is the one implemented in the LINPACK package and is described by Stewart in [9]. A sequence of $(n+1)\times(n+1)$ Givens rotations of the form

$$\mathbf{J}_k = \begin{bmatrix} \cos\Theta_k & 0 & \cdots & 0 & \sin\Theta_k & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\ -\sin\Theta_k & 0 & \cdots & 0 & \cos\Theta_k & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 \\ \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & 1 \end{bmatrix} \leftarrow (k+1)\text{st row}$$

$$\uparrow$$

$$(k+1)\text{st col}$$

with $k=1,\cdots,n$ are used to compute $\mathbf{U}$ via the relationship

$$\mathbf{J}_1\cdots\mathbf{J}_n\begin{bmatrix}\mathbf{O}^T\\ \mathbf{R}\end{bmatrix}=\begin{bmatrix}\mathbf{x}^T\\ \mathbf{U}\end{bmatrix}, \tag{2.1}$$

and since $\mathbf{J}_1,\cdots,\mathbf{J}_n$ are orthogonal matrices, it is easy to verify that

$$\mathbf{R}^T\mathbf{R}=\mathbf{x}\mathbf{x}^T+\mathbf{U}^T\mathbf{U}.$$

Although the right-hand side of (2.1) will always be an upper Hessenberg matrix, we clearly require some restrictions on $\mathbf{J}_1,\cdots,\mathbf{J}_n$ to ensure that the first row is indeed the required $\mathbf{x}^T$. Let

$$\mathbf{J}_n^T\cdots\mathbf{J}_1^T\mathbf{e}_1=\mathbf{q}=:\begin{bmatrix}\alpha\\ \mathbf{a}\end{bmatrix}, \tag{2.2}$$

where $\mathbf{e}_1=(1,0,\cdots,0)^T$. Then, from (2.1),

$$\mathbf{a}^T\mathbf{R}=\mathbf{e}_1^T\mathbf{J}_1\cdots\mathbf{J}_n\begin{bmatrix}\mathbf{O}^T\\ \mathbf{R}\end{bmatrix}=\mathbf{x}^T.$$

Thus we must have

$$\mathbf{R}^T\mathbf{a}=\mathbf{x} \tag{2.3}$$

and

$$\alpha^2=1-\mathbf{a}^T\mathbf{a}. \tag{2.4}$$

We note that a necessary and sufficient condition for $\mathbf{R}^T\mathbf{R}-\mathbf{x}\mathbf{x}^T$ to be positive definite is $\alpha^2>0$.

The basic steps in the algorithm are therefore as follows. First the n-vector $\mathbf{a}$ is obtained by forward substitution from (2.3) and α is calculated from (2.4) (the sign of α is immaterial and the positive root of (2.4) is usually taken). Givens rotations $\mathbf{J}_1,\cdots,\mathbf{J}_n$ satisfying

$$\mathbf{J}_1\cdots\mathbf{J}_n\mathbf{q}=\mathbf{e}_1$$

are constructed and **U** is then calculated from (2.1). Note that Algorithm A modifies the rows of **R** from bottom to top, whereas Algorithms B and C (described below) process them from top to bottom.

The algorithm has been analysed by Stewart [9] and his error analysis is summarised in § 3. Roughly speaking, the algorithm performs as well as can be expected given the potentially ill-conditioned nature of the downdating problem. However, as discussed in § 4, the algorithm is not particularly well suited to parallel implementation, the main problem being that the forward substitution (2.3) needs to be completed before the calculation of (2.1) can commence.

2b. A recursive method (Algorithm B). We now describe another standard algorithm for downdating which may be found for example in Lawson and Hanson [8]. Although we derive the method from the previous algorithm to show the close connection between them, the usual derivation is based on completely different ideas.

We begin by rewriting (2.1) as

$$\begin{bmatrix} \mathbf{O}^T \\ \mathbf{R} \end{bmatrix} = \mathbf{J}_n^T \cdots \mathbf{J}_1^T \begin{bmatrix} \mathbf{x}^T \\ \mathbf{U} \end{bmatrix} \tag{2.5}$$

and define

$$\begin{bmatrix} (\mathbf{x}^{(k)})^T \\ \mathbf{R}^{(k)} \end{bmatrix} := \mathbf{J}_k^T \cdots \mathbf{J}_1^T \begin{bmatrix} \mathbf{x}^T \\ \mathbf{U} \end{bmatrix}, \qquad k = 0, 1, \cdots, n, \tag{2.6}$$

where we have used the convention that $\mathbf{J}_k^T \cdots \mathbf{J}_1^T$ is the identity when $k = 0$. It is easy to verify that $\mathbf{x}^{(0)} = \mathbf{x}$, $\mathbf{R}^{(0)} = \mathbf{U}$, $\mathbf{x}^{(n)} = \mathbf{O}$ and $\mathbf{R}^{(n)} = \mathbf{R}$. Furthermore, the first k components of $\mathbf{x}^{(k)}$ are zeros and the first k rows of $\mathbf{R}^{(k)}$ and **R** are the same as are the last $n-k$ rows of $\mathbf{R}^{(k)}$ and **U**. From (2.5) and (2.6),

$$\begin{bmatrix} (\mathbf{x}^{(k)})^T \\ \mathbf{R}^{(k)} \end{bmatrix} = \mathbf{J}_k^T \begin{bmatrix} \mathbf{x}(^{(k-1)})^T \\ \mathbf{R}^{(k-1)} \end{bmatrix}, \qquad k = 1, \cdots, n, \tag{2.7}$$

which can be shown to be equivalent to

$$\begin{bmatrix} (\mathbf{x}^{(k)})^T \\ \mathbf{R}^{(k-1)} \end{bmatrix} = \mathbf{S}_k \begin{bmatrix} \mathbf{x}(^{(k-1)})^T \\ \mathbf{R}^{(k)} \end{bmatrix}, \qquad k = 1, \cdots, n, \tag{2.8}$$

where

$$\mathbf{S}_k = \begin{bmatrix} \sec\Theta_k & 0 & \cdots & 0 & -\tan\Theta_k & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\ -\tan\Theta_k & 0 & \cdots & 0 & \sec\Theta_k & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 \\ \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & 1 \end{bmatrix} \leftarrow (k+1)\text{st row.}$$

$$\uparrow$$

$$(k+1)\text{st col}$$

From (2.2), $\cos^2\Theta_k > \alpha^2$, $k = 1, \cdots, n$ and hence $\mathbf{S}_k$, $k = 1, \cdots, n$ are well defined.

To see that (2.8) forms the basis of a recursive algorithm to compute **U**, let us focus attention on the first and $(k+1)$st rows of (2.8). As noted previously, the kth

row of $\mathbf{R}^{(k)}$ is the same as the kth row of $\mathbf{R}$ and is therefore known. Let us suppose that in the kth step we also know $\mathbf{x}^{(k-1)}$. The role played by Θ_k in $\mathbf{S}_k$ (or equivalently $\mathbf{J}_k$) is to ensure that the kth component of $\mathbf{x}^{(k)}$ is zero. This can be achieved by setting

$$\cos\Theta_k = (r_{kk}^2 - (x_k^{(k-1)})^2)^{1/2}/r_{kk} \tag{2.9a}$$

and

$$\sin\Theta_k = x_k^{(k-1)}/r_{kk}. \tag{2.9b}$$

We now calculate $\mathbf{x}^{(k)}$ and the kth row of $\mathbf{U}$ (or equivalently the kth row of $\mathbf{R}^{(k-1)}$) from (2.8) as

$$x_j^{(k)} = (x_j^{(k-1)} - r_{kj}\sin\Theta_k)/\cos\Theta_k, \qquad j = k+1, \cdots, n \tag{2.10a}$$

and

$$u_{kk} = (r_{kk}^2 - (x_k^{(k-1)})^2)^{1/2} \tag{2.10b}$$

$$u_{kj} = (-x_j^{(k-1)}\sin\Theta_k + r_{kj})/\cos\Theta_k, \qquad j = k+1, \cdots, n. \tag{2.10c}$$

Since $\mathbf{x}^{(0)} = \mathbf{x}$ is known, the relations (2.9), (2.10) yield a recursive method for calculating $\mathbf{U}$.

Using (2.8), it can be shown that

$$\begin{bmatrix} \mathbf{O}^T \\ \mathbf{U} \end{bmatrix} = \mathbf{S}_n \cdots \mathbf{S}_1 \begin{bmatrix} \mathbf{x}^T \\ \mathbf{R} \end{bmatrix}$$

and it is easy to verify that $\mathbf{S}_k$, $k = 1, \cdots, n$ are Σ-unitary with

$$\Sigma = \begin{bmatrix} -1 & \mathbf{O}^T \\ \mathbf{O} & \mathbf{I} \end{bmatrix},$$

where a matrix $\mathbf{A}$ is Σ-unitary iff

$$\mathbf{A}^T\Sigma\mathbf{A} = \Sigma.$$

The product

$$\mathbf{S} := \mathbf{S}_n \cdots \mathbf{S}_1$$

is also Σ-unitary and hence

$$\mathbf{U}^T\mathbf{U} = [\mathbf{O}|\mathbf{U}^T]\Sigma\begin{bmatrix} \mathbf{O}^T \\ \mathbf{U} \end{bmatrix} = [\mathbf{x}|\mathbf{R}^T]\mathbf{S}^T\Sigma\mathbf{S}\begin{bmatrix} \mathbf{x}^T \\ \mathbf{R} \end{bmatrix},$$

which is an independent verification that $\mathbf{U}$ is the required Cholesky factor. Thus, the method can be regarded as that of finding a product of planar Σ-unitary transformations to reduce $|\mathbf{x}|\mathbf{R}^T]^T$ to $[\mathbf{O}|\mathbf{U}^T]^T$ in the same way that Givens rotations are used to reduce the matrix in the updating problem (see for, example Gill, Golub, Murray and Saunders [5]).

The recursive method has a number of advantages over Algorithm A. Since there is no forward substitution phase, the operation count is somewhat less. In addition the algorithm is quite well suited to parallel implementation. However, the error analysis in § 3 indicates that the stability properties of the method are substantially inferior to those of Algorithm A.

2c. A modified recursive algorithm (Algorithm C). It turns out that Algorithm B can be modified so that its stability properties are substantially improved.

At the kth step, consider the $(k+1)$st row of (2.7). In component form, this can be rewritten as

$$u_{kk} = (r_{kk}^2 - (x_k^{(k-1)})^2)^{1/2}, \tag{2.11a}$$

$$u_{kj} = (-x_j^{(k-1)} \sin \Theta_k + r_{kj})/\cos \Theta_k, \qquad j = k+1, \cdots, n, \tag{2.11b}$$

where $\cos \Theta_k$ and $\sin \Theta_k$ have been calculated using (2.9a) and (2.9b), respectively. Now, consider the first row of (2.7) and rewrite it as

$$x_j^{(k)} = \cos \Theta_k x_j^{(k-1)} - \sin \Theta_k u_{kj}, \qquad j = k+1, \cdots, n. \tag{2.11c}$$

This is clearly a well defined calculation as u_{kj}, $j = k+1, \cdots, n$ have already been calculated via (2.11b).

Thus, (2.9) and (2.11) form the basis of an algorithm for calculating $\mathbf{U}$ which differs only slightly from Algorithm B. However, as we shall see in § 3, this small modification enables us to establish stability estimates that are comparable to those for Algorithm A. In addition, we still retain all of the desirable features of Algorithm B.

3. Error analysis. We denote quantities stored in the computer with a tilde. In addition, ε is the relative precision of the machine considered and terms of order ε^2 are neglected.

3a. Algorithm A. An error analysis of this method is to be found in [9]. It is shown there that there exists an exactly orthogonal matrix $\hat{\mathbf{Q}}$ (which is not computed) such that

$$\hat{\mathbf{Q}} \begin{bmatrix} \mathbf{O}^T \\ \mathbf{R} \end{bmatrix} = \begin{bmatrix} \mathbf{x}^T + \Delta \mathbf{x}^T \\ \tilde{\mathbf{U}} + \Delta \tilde{\mathbf{U}} \end{bmatrix}, \tag{3.1}$$

where

$$\|[\Delta \tilde{\mathbf{U}}]_i\|_2 \leqq 6n\varepsilon \|[\mathbf{R}]_i\|_2, \tag{3.2}$$

$$|\Delta x_i| \leqq [(13n+5)/2 + (i+2)\sqrt{i}]\varepsilon \|[\mathbf{R}]_i\|_2. \tag{3.3}$$

Here we use $[\ ]_i$ to denote the ith column of a matrix. From this, one easily obtains a bound for the total error matrix,

$$\left\| \begin{bmatrix} \Delta \mathbf{x}^T \\ \Delta \tilde{\mathbf{U}} \end{bmatrix} \right\|_F \leqq [n^2/2 + 9n\sqrt{n} + O(n)]\varepsilon \|\mathbf{R}\|_F. \tag{3.4}$$

Notice that errors are not only superimposed on the data—$\mathbf{R}$ and $\mathbf{x}$ in this case—but also on the result $\tilde{\mathbf{U}}$. Hence, the bound (3.4) does not guarantee forward or backward stability of the algorithm, but rather proves what could be called "mixed" stability. As argued in [9], the forward part of the error, $\Delta \tilde{\mathbf{U}}$, is in fact unimportant since the "true" forward error $(\tilde{\mathbf{U}} - \mathbf{U})$ is usually much larger and mainly depends on $\Delta \mathbf{x}$, the backward part of the error in (3.1). The result of mixed stability is thus as satisfactory in practice as backward stability since error bounds in both cases are comparable.

3b. Algorithm B. The error analysis of this method depends on how it is implemented. We therefore write the order of computations for one step, which, without loss of generality, can be the first step as given in (2.9), (2.10).

Let us denote the elements of the rows involved in this step as follows:

$$1/c_1 \begin{bmatrix} 1 & -s_1 \\ -s_1 & 1 \end{bmatrix} \begin{bmatrix} x_1^{(0)} & x_2^{(0)} & \cdots & x_n^{(0)} \\ r_{11} & r_{12} & \cdots & r_{1n} \end{bmatrix} = \begin{bmatrix} 0 & x_2^{(1)} & \cdots & x_n^{(1)} \\ u_{11} & u_{12} & \cdots & u_{1n} \end{bmatrix}. \tag{3.5}$$

Now the operations are performed in the following order (where we use the fl(·) notation of Wilkinson [10]):

$$
\begin{aligned}
&\tilde{u}_{11} = \mathrm{fl}(\sqrt{(r_{11}-x_1^{(0)})(r_{11}+x_1^{(0)})}),\\
&\tilde{c}_1 = \mathrm{fl}(\tilde{u}_{11}/r_{11}),\\
&\tilde{s}_1 = \mathrm{fl}(x_1^{(0)}/r_{11}),\\
&\tilde{u}_{1i} = \mathrm{fl}((r_{1i}-\tilde{s}_1 x_i^{(0)})/\tilde{c}_1),\\
&\tilde{x}_i^{(1)} = \mathrm{fl}((x_i^{(0)}-\tilde{s}_1 r_{1i})/\tilde{c}_1) \quad \text{for } i = 2, \cdots, n.
\end{aligned} \tag{3.6}
$$

In the Appendix we show that for this implementation of (3.5), the following equality holds,

$$
1/c_1 \begin{bmatrix} 1 & -s_1 \\ -s_1 & 1 \end{bmatrix} \begin{bmatrix} x_i^{(0)} + \Delta x_i^{(0)} \\ r_{1i} + \Delta r_{1i} \end{bmatrix} = \begin{bmatrix} \tilde{x}_1^{(1)} \\ \tilde{u}_{1i} \end{bmatrix}, \tag{3.7}
$$

where

$$
\left\| \begin{bmatrix} \Delta x_i^{(0)} \\ \Delta r_{1i} \end{bmatrix} \right\|_2 \leqq (8\varepsilon/|c_1|) \left\| \begin{bmatrix} x_i^{(0)} \\ \tilde{u}_{1i} \end{bmatrix} \right\|_2. \tag{3.8}
$$

If we rewrite (3.7) as

$$
1/c_1 \begin{bmatrix} 1 & -s_1 \\ -s_1 & 1 \end{bmatrix} \begin{bmatrix} x_i^{(0)} + \Delta x_i^{(0)} \\ r_{1i} \end{bmatrix} = \begin{bmatrix} \tilde{x}_i^{(1)} \\ \tilde{u}_{1i} + \Delta \tilde{u}_{1i} \end{bmatrix}, \tag{3.9}
$$

then for the mixed error one obtains (see the Appendix),

$$
\left\| \begin{bmatrix} \Delta x_i^{(0)} \\ \Delta \tilde{u}_{1i} \end{bmatrix} \right\|_2 \leqq (8(1+|s_1|)\varepsilon/|c_1|) \left\| \begin{bmatrix} x_i^{(0)} \\ \tilde{u}_{1i} \end{bmatrix} \right\|_2. \tag{3.10}
$$

Since this 2-vector is now a subcolumn of the ith column of the matrix $[\mathbf{x}, \tilde{\mathbf{U}}^T]^T$, we have

$$
\mathbf{J}_1^T \begin{bmatrix} \mathbf{x}^T + \Delta^{(1)}\mathbf{x}^T \\ \tilde{\mathbf{U}} + \Delta^{(1)}\tilde{\mathbf{U}} \end{bmatrix} = \begin{bmatrix} (\tilde{\mathbf{x}}^{(1)})^T \\ \mathbf{R}^{(1)} \end{bmatrix} \tag{3.11}
$$

with $\mathbf{J}_1$ exactly orthogonal and

$$
\left\| \begin{bmatrix} \Delta^{(1)}\mathbf{x}^T \\ \Delta^{(1)}\tilde{\mathbf{U}} \end{bmatrix}_i \right\|_2 \leqq (8(1+|s_1|)\varepsilon/|c_1|)\|[\mathbf{R}]_i\|_2, \tag{3.12}
$$

since the ith columns of $\mathbf{R}$ and $[\mathbf{x}, \mathbf{U}^T]^T$ have the same norms.

Errors in the subsequent step are similarly bounded in terms of the deflated problem $[\tilde{\mathbf{x}}^{(1)}, (\mathbf{R}^{(1)})^T]^T$. Because corresponding columns in (3.11) are related by an orthogonal transformation, the errors of step 2 can be mapped backwards onto $[\mathbf{x}, \tilde{\mathbf{U}}^T]^T$ without altering their norms. (This would not have been possible if we had used backward errors as in (3.7), (3.8).) This now gives

$$
\mathbf{J}_2^T \mathbf{J}_1^T \begin{bmatrix} \mathbf{x}^T + \Delta^{(1)}\mathbf{x}^T + \Delta^{(2)}\mathbf{x}^T \\ \tilde{\mathbf{U}} + \Delta^{(1)}\tilde{\mathbf{U}} + \Delta^{(2)}\tilde{\mathbf{U}} \end{bmatrix} = \begin{bmatrix} (\tilde{\mathbf{x}}^{(2)})^T \\ \mathbf{R}^{(2)} \end{bmatrix}, \tag{3.13}
$$

where

$$
\left\| \begin{bmatrix} \Delta^{(1)}\mathbf{x}^T + \Delta^{(2)}\mathbf{x}^T \\ \Delta^{(1)}\tilde{\mathbf{U}} + \Delta^{(2)}\tilde{\mathbf{U}} \end{bmatrix}_i \right\|_2 \leqq 8\{(1+|s_1|)/|c_1| + (1+|s_2|)/|c_2|\}\varepsilon\|[\mathbf{R}]_i\|_2 \tag{3.14}
$$

for all columns i except the first one (since that one is not affected anymore). By induction we finally have

$$\mathbf{J}_n^T \cdots \mathbf{J}_1^T \begin{bmatrix} \mathbf{x}^T + \Delta\mathbf{x}^T \\ \tilde{\mathbf{U}} + \Delta\tilde{\mathbf{U}} \end{bmatrix} = \begin{bmatrix} \mathbf{O}^T \\ \mathbf{R} \end{bmatrix}, \tag{3.15}$$

where $\Delta\tilde{\mathbf{U}} = \sum_{i=1}^{n} \Delta^{(i)}\tilde{\mathbf{U}}$ and $\Delta\mathbf{x} = \sum_{i=1}^{n} \Delta^{(i)}\mathbf{x}$ are bounded columnwise by

$$\left\| \begin{bmatrix} \Delta\mathbf{x}^T \\ \Delta\tilde{\mathbf{U}} \end{bmatrix}_i \right\|_2 \leqq 8i \max_{1 \leqq j \leqq i} \{(1+|s_j|)/|c_j|\}\varepsilon \|[\mathbf{R}]_i\|_2 \tag{3.16}$$

which is significantly worse than (3.3) and problem dependent. For the total error we obtain

$$\left\| \begin{bmatrix} \Delta\mathbf{x}^T \\ \Delta\tilde{\mathbf{U}} \end{bmatrix} \right\|_F \leqq 5n\sqrt{n} \max_{1 \leqq i \leqq n} \{(1+|s_i|)/|c_i|\}\varepsilon \|\mathbf{R}\|_F. \tag{3.17}$$

3c. Algorithm C. Since this method is essentially a rearrangement of the previous method, one might expect similar bounds for the numerical errors, but this is not the case because operations are performed in a different order.

Let us again denote the elements of the rows involved in step 1 as

$$\begin{bmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{bmatrix} \begin{bmatrix} x_1^{(0)} & x_2^{(0)} & \cdots & x_n^{(0)} \\ u_{11} & u_{12} & \cdots & u_{1n} \end{bmatrix} = \begin{bmatrix} 0 & x_2^{(1)} & \cdots & x_n^{(1)} \\ r_{11} & r_{12} & \cdots & r_{1n} \end{bmatrix}; \tag{3.18}$$

then c_1, s_1 and u_{1j}, $x_j^{(i)}$ are constructed as follows:

$$\begin{aligned} \tilde{u}_{11} &= \mathrm{fl}(\sqrt{(r_{11}-x_1^{(0)})(r_{11}+x_1^{(0)})}), \\ \tilde{c}_1 &= \mathrm{fl}(\tilde{u}_{11}/r_{11}), \\ \tilde{s}_1 &= \mathrm{fl}(x_1^{(0)}/r_{11}), \\ \tilde{u}_{1i} &= \mathrm{fl}((r_{1i}-\tilde{s}_1 x_i^{(0)})/\tilde{c}_1), \\ \tilde{x}_i^{(1)} &= \mathrm{fl}(\tilde{c}_1 x_i^{(0)} - \tilde{s}_1 \tilde{u}_{1i}) \quad \text{for } i = 2, \cdots, n. \end{aligned} \tag{3.19}$$

The only difference between this algorithm and the previous one is in the computation of $\tilde{x}_i^{(1)}$. This single difference allows us to obtain the following bound for the mixed error vector (see the Appendix):

$$\left\| \begin{bmatrix} \Delta x_i^{(0)} \\ \Delta\tilde{u}_{1i} \end{bmatrix} \right\|_2 \leqq 6.25\varepsilon \left\| \begin{bmatrix} x_i^{(0)} \\ \tilde{u}_{1i} \end{bmatrix} \right\|_2. \tag{3.20}$$

Using this, we find by induction that

$$\left\| \begin{bmatrix} \Delta\mathbf{x}^T \\ \Delta\tilde{\mathbf{U}} \end{bmatrix} \right\|_2 \leqq 6.25i\varepsilon \|[\mathbf{R}]_i\|_2 \tag{3.21}$$

and for the total error matrix

$$\left\| \begin{bmatrix} \Delta\mathbf{x}^T \\ \Delta\tilde{\mathbf{U}} \end{bmatrix} \right\|_F \leqq 4n\sqrt{n}\varepsilon \|\mathbf{R}\|_F \tag{3.22}$$

which compares favourably with the two previous methods (see (3.4) and (3.16)).

4. Complexity. Here we compare the computational efficiency of the different methods and comment on the possibility of implementing them in parallel.

4a. The LINPACK method. The method first constructs the vector $\mathbf{q}$ which has length $n+1$ and is given by (2.2) and (2.3). This construction is a forward substitution

process for the triangular system (2.2) and requires $n^2/2+O(n)$ multiplications. Next the elementary rotations $\mathbf{J}_i$, $i=n, n-1, \cdots, 1$, are applied to $\mathbf{q}$ and the matrix $[\mathbf{O}, \mathbf{R}^T]^T$. Because of the triangular shape of $\mathbf{R}$, this requires $2n^2+O(n)$ multiplications giving a total operation count of

$$2.5n^2+O(n) \tag{4.1}$$

multiplications. This operation count can be reduced by using modified Givens rotations instead of plane rotations (see [4]), or by using an LDL^T version of the method (see [6]).

Both stages of the algorithm can be parallelized; the first stage, the back substitution, in $2n$ parallel steps using n processors and the second stage in $2(n+1)$ parallel steps using $n+1$ processors. The computation of α is performed while $\mathbf{a}$ is being constructed and thus requires only one additional step for the square root. It is this computation of α that separates both stages in the method and prevents any concurrency between both stages. In total $(n+1)$ processors and $4(n+1)$ parallel steps are required.

4b. The recursive Algorithms B and C. Here in each step one essentially constructs an elementary rotation $\mathbf{J}_k$ (or equivalently $\mathbf{S}_k$) and applies it to determine $\mathbf{x}^{(k)}$ and the kth row of $\mathbf{U}$ from $\mathbf{x}^{(k-1)}$ and the kth row of $\mathbf{R}$. This is easily seen to require $4(n-i+1)$ multiplications for each transformation $\mathbf{J}_i$ or $\mathbf{S}_i$, resulting in an operation count of

$$2n^2+O(n) \tag{4.2}$$

multiplications for the whole process.

Parallel implementation of these methods would require n processors and $2n$ steps which compares favourably with the LINPACK method. Notice also that by using modified elementary rotations [4] or modified Σ-orthogonal transformations, the work could be reduced and square roots could be avoided.

5. Numerical results. The results of Stewart [9] on the conditioning of the downdating problem show that we cannot expect that $\tilde{\mathbf{U}}$, the calculated factor, will be close to $\mathbf{U}$. However a mixed stability result of the form

$$\mathbf{R}^T\mathbf{R}-(\mathbf{x}+\Delta\mathbf{x})(\mathbf{x}+\Delta\mathbf{x})^T=(\tilde{\mathbf{U}}+\Delta\tilde{\mathbf{U}})(\tilde{\mathbf{U}}+\Delta\tilde{\mathbf{U}})^T$$

where $\Delta\mathbf{x}$ and $\Delta\tilde{\mathbf{U}}$ are small does ensure that

$$\mathbf{R}^T\mathbf{R}-\mathbf{x}\mathbf{x}^T=\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T+\mathbf{E}$$

where $\mathbf{E}$ is small. To demonstrate the superior stability properties of Algorithms A and C over Algorithm B, we have computed the quantity

$$\frac{\|\mathbf{R}^T\mathbf{R}-\mathbf{x}\mathbf{x}^T-\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T\|_F}{\|\mathbf{U}^T\mathbf{U}\|_F}$$

for the problem

$$\mathbf{R}=\begin{bmatrix}1 & \sin(\theta/2)\\ 0 & \sqrt{2}\cos(\theta/2)\end{bmatrix},\qquad \mathbf{x}=\begin{bmatrix}\sin\theta\\ \cos(\theta/2)\end{bmatrix},$$

which has the solution

$$\mathbf{U}=\begin{bmatrix}\cos\theta & -\sin(\theta/2)\\ 0 & \cos(\theta/2)\end{bmatrix}.$$

Results obtained on a Macintosh which has working accuracy of between 7 and 8 significant digits are tabulated in Table 1 for $\cos\theta=2^{-k}$, $k=3, 6, 9, 12$.

TABLE 1

$\cos\theta$	Alg A	Alg B	Alg C
2^{-3}	1.337E−8	2.367E−7	1.183E−7
2^{-6}	1.210E−7	1.073E−6	6.939E−8
2^{-9}	1.788E−7	3.218E−5	2.946E−8
2^{-12}	1.264E−7	1.011E−4	2.467E−8

The numerical results are consistent with the stability analysis of § 3 and clearly demonstrate the superior stability properties of Algorithms A and C.

Appendix. Here we derive the bounds (3.8), (3.10) and (3.20) for the numerical errors incurred in (3.6) and (3.19).

Let us denote by δ_i and ε_i quantities that are smaller in absolute value than ε, the relative precision of the machine used. Then we have, according to Wilkinson [10]:

$$\text{(A.1)}\quad \tilde{u}_{11}=\sqrt{\{[(r_{11}-x_1^{(0)})(1+\delta_1)(r_{11}+x_1^{(0)})(1+\delta_2)](1+\delta_3)\}}(1+\delta_4)=u_{11}(1+2.5\varepsilon_1),$$

$$\text{(A.2)}\quad \tilde{c}_1=(\tilde{u}_{11}/r_{11})(1+\delta_5)=(u_{11}/r_{11})(1+3.5\varepsilon_2)=c_1(1+3.5\varepsilon_2),$$

$$\text{(A.3)}\quad \tilde{s}_1=(x_1^{(0)}/r_{11})(1+\delta_6)=(x_1^{(0)}/r_{11})(1+\varepsilon_3)=s_1(1+\varepsilon_3),$$

and for each i,

$$\text{(A.4)}\quad \tilde{x}_i^{(1)}=[x_i^{(0)}-\tilde{s}_1r_{1i}(1+\delta_{10})](1+\delta_{11})/[\tilde{c}_1(1+\delta_{12})]$$
$$=[x_i^{(0)}-\tilde{s}_1r_{1i}(1+\varepsilon_6)]/[\tilde{c}_1(1+2\varepsilon_7)],$$

$$\text{(A.5)}\quad \tilde{u}_{1i}=[r_{1i}-\tilde{s}_1x_i^{(0)}(1+\delta_7)](1+\delta_8)/[\tilde{c}_1(1+\delta_9)]$$
$$=[r_{1i}-\tilde{s}_1x_i^{(0)}(1+\varepsilon_4)]/[\tilde{c}_1(1+2\varepsilon_5)].$$

Multiplying these with their respective denominators yields

$$\text{(A.6)}\quad \begin{aligned} x_i^{(0)}&=\tilde{s}_1r_{1i}(1+\varepsilon_6)+\tilde{x}_i^{(1)}\tilde{c}_1(1+2\varepsilon_7),\\ r_{1i}&=\tilde{s}_1x_i^{(0)}(1+\varepsilon_4)+\tilde{u}_{1i}\tilde{c}_1(1+2\varepsilon_5).\end{aligned}$$

Using (A.2) and (A.3), we finally obtain

$$\text{(A.7)}\quad \begin{aligned} x_i^{(0)}-s_1r_{1i}&=c_1\tilde{x}_i^{(1)}+2\varepsilon_8s_1r_{1i}+5.5\varepsilon_9\tilde{x}_i^{(1)}c_1,\\ r_{1i}-s_1x_i^{(0)}&=c_1\tilde{u}_{1i}+2\varepsilon_{10}s_1x_i^{(0)}+5.5\varepsilon_{11}c_1\tilde{u}_{1i},\end{aligned}$$

or

$$\text{(A.8)}\quad 1/c_1\begin{bmatrix}1 & -s_1\\ -s_1 & 1\end{bmatrix}\begin{bmatrix}x_i^{(0)}\\ r_{1i}\end{bmatrix}=\begin{bmatrix}\tilde{x}_i^{(1)}+\Delta\tilde{x}_i^{(1)}\\ \tilde{u}_{1i}^{(1)}+\Delta\tilde{u}_{1i}\end{bmatrix}=\begin{bmatrix}\tilde{x}_i^{(1)}+2\varepsilon_8(s_1/c_1)r_{1i}+5.5\varepsilon_9\tilde{x}_i^{(1)}\\ \tilde{u}_{1i}+2\varepsilon_{10}(s_1/c_1)x_i^{(0)}+5.5\varepsilon_{11}\tilde{u}_{1i}\end{bmatrix}.$$

Identifying the error terms in the right-hand side with the vector $[\Delta\tilde{x}_i^{(1)},\Delta\tilde{u}_{1i}]^T$ we then obtain from the Cauchy-Schwartz inequality the bounds

$$\text{(A.9)}\quad \begin{aligned} |\Delta\tilde{x}_i^{(1)}|&\leqq(1/|c_1|)\left\|\begin{bmatrix}5.5\varepsilon_9\tilde{x}_i^{(1)}\\ 2\varepsilon_8r_{1i}\end{bmatrix}\right\|_2\leqq(5.5\varepsilon/|c_1|)\left\|\begin{bmatrix}\tilde{x}_i^{(1)}\\ r_{1i}\end{bmatrix}\right\|_2\\ &=(5.5\varepsilon/|c_1|)\left\|\begin{bmatrix}x_i^{(0)}\\ \tilde{u}_{1i}\end{bmatrix}\right\|_2+O(\varepsilon^2),\\ |\Delta\tilde{u}_{1i}|&\leqq(1/|c_1|)\left\|\begin{bmatrix}2\varepsilon_{10}x_i^{(0)}\\ 5.5\varepsilon_{11}\tilde{u}_{1i}\end{bmatrix}\right\|_2\leqq(5.5\varepsilon/|c_1|)\left\|\begin{bmatrix}x_i^{(0)}\\ \tilde{u}_{1i}\end{bmatrix}\right\|_2,\end{aligned}$$

and subsequently,

$$\left\| \begin{bmatrix} \Delta \tilde{x}_i^{(1)} \\ \Delta \tilde{u}_{1i} \end{bmatrix} \right\|_2 \leq (5.5\sqrt{2}\varepsilon/|c_1|) \left\| \begin{bmatrix} x_i^{(0)} \\ \tilde{u}_{1i} \end{bmatrix} \right\|_2 \leq (8\varepsilon/|c_1|) \left\| \begin{bmatrix} x_i^{(0)} \\ \tilde{u}_{1i} \end{bmatrix} \right\|_2. \tag{A.10}$$

For the backward and mixed errors one obtains via similar techniques the following bounds:

$$\left\| \begin{bmatrix} \Delta x_i^{(0)} \\ \Delta r_{1i} \end{bmatrix} \right\|_2 \leq (8\varepsilon/|c_1|) \left\| \begin{bmatrix} x_i^{(0)} \\ \tilde{u}_{1i} \end{bmatrix} \right\|_2, \tag{A.11}$$

$$\left\| \begin{bmatrix} \Delta x_i^{(0)} \\ \Delta \tilde{u}_{1i} \end{bmatrix} \right\|_2 \leq [8\varepsilon(1+|s_1|)/|c_1|] \left\| \begin{bmatrix} x_i^{(0)} \\ \tilde{u}_{1i} \end{bmatrix} \right\|_2. \tag{A.12}$$

Notice that all three bounds depend on $1/|c_1|$ which is large when the condition number of $\mathbf{S}_1$,

$$\kappa(\mathbf{S}_1) = (1+|s_1|)/(1-|s_1|) = (1+|s_1|)^2/|c_1|^2, \tag{A.13}$$

is large.

Now we consider Algorithm C. A similar analysis to that used for Algorithm B now yields

$$\begin{aligned} \tilde{x}_i^{(1)} &= [c_1 x_i^{(0)}(1+\delta_{10}) - \tilde{s}_1 \tilde{u}_{1i}(1+\delta_{11})](1+\delta_{12}) \\ &= \tilde{c}_1 x_i^{(0)}(1+2\varepsilon_{12}) - \tilde{s}_1 \tilde{u}_{1i}(1+2\varepsilon_{13}). \end{aligned} \tag{A.14}$$

Together with the second equation of (A.6) and using (A.2), (A.3), this now leads to

$$\begin{bmatrix} \tilde{x}_i^{(1)} \\ r_{1i} \end{bmatrix} = \begin{bmatrix} c_{11} & -s_1 \\ s_1 & c_1 \end{bmatrix} \begin{bmatrix} x_i^{(0)} \\ \tilde{u}_{1i} \end{bmatrix} + \begin{bmatrix} -3\varepsilon_{14} s_1 \tilde{u}_{1i} + 5.5\varepsilon_{15} c_1 x_i^{(0)} \\ 5.5\varepsilon_{11} c_1 \tilde{u}_{1i} + 2\varepsilon_{10} s_1 x_i^{(0)} \end{bmatrix}. \tag{A.15}$$

Identifying the errors terms on the right-hand side with the vector $[-\Delta \tilde{x}_i^{(1)}, -\Delta r_{1i}]^T$ and using a similar argument as in the proof of (A.9) and (A.10), one now finds

$$\left\| \begin{bmatrix} \Delta \tilde{x}_i^{(1)} \\ \Delta r_{1i} \end{bmatrix} \right\|_2 \leq 6.25\varepsilon \left\| \begin{bmatrix} x_i^{(0)} \\ \tilde{u}_{1i} \end{bmatrix} \right\|_2. \tag{A.16}$$

From the relation

$$\begin{bmatrix} \Delta x_i^{(0)} \\ \Delta \tilde{u}_{1i} \end{bmatrix} = \begin{bmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{bmatrix} \begin{bmatrix} \Delta \tilde{x}_i^{(1)} \\ \Delta r_{1i} \end{bmatrix} \tag{A.17}$$

one then obtains a similar bound for the vector $[\Delta x_i^{(0)}, \Delta \tilde{u}_{1i}]^T$, which completes the proof.

Acknowledgments. We thank Professor G. H. Golub and Dr. M. A. Saunders for their helpful comments on a draft of this paper.

REFERENCES

[1] J. H. ARGYRIS AND J. R. ROY, *General treatments of structural modifications*, J. Structural Division, A.S.C.E., 98, ST2, Proc. Paper 8732 (1972), pp. 465–492.

[2] V. BELOVITCH, *Classical Network Theory*, Holden Day, San Francisco, CA, 1968.

[3] A. W. BOJANCZYK, R. P. BRENT AND F. R. DE HOOG, *QR factorization of Toeplitz matrices*, Report CMA-R07-1985, Centre for Math. Anal., The Australian National University, May 1985, Numerische Mathematik, to appear.

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

[5] P. E. GILL, G. H. GOLUB, W. MURRAY AND M. A. SAUNDERS, *Methods for modifying matrix factorizations*, Math. Comp., 28 (1974), pp. 505-535.

[6] P. E. GILL, W. MURRAY AND M. A. SAUNDERS, *Methods for computing and modifying the LDV factors of a matrix*, Math. Comp., 29 (1975), pp. 1051-1077, esp. p. 1063.

[7] G. H. GOLUB AND C. VAN LOAN, *Matrix Computations*, The Johns Hopkins Press, Baltimore, MD, 1983.

[8] C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[9] G. W. STEWART, *The effect of rounding errors on an algorithm for downdating a Cholesky factorization*, J. Inst. Math. Appl., 23 (1979), pp. 203-213.

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

[11] G. H. GOLUB, *Matrix decompositions and statistical calculations*, in Statistical Computation, R. C. Multon and J. A. Nelder, eds., Academic Press, New York, 1969, pp. 365-387.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 3, May 1987
© 1987 Society for Industrial and Applied Mathematics

COMPUTATION OF EIGENVALUES ASSOCIATED WITH FUNCTIONAL DIFFERENTIAL EQUATIONS*

A. MANITIUS†, H. TRAN‡, G. PAYRE§ AND R. ROY¶

Abstract. This paper describes a method for computing the eigenvalues associated with systems of linear retarded functional differential equations (RFDE's). The method finds the eigenvalues directly from a certain characteristic equation which is automatically determined from system matrices. The eigenvalues contained in some bounded region around the origin are approximately computed by a combinatorial algorithm suggested earlier by H. Kuhn [15] for approximations of zeros of ordinary polynomials. The eigenvalues of large modulus, which are distributed in some curvilinear strips, are computed from some asymptotic formulas obtained directly from the parameters of the characteristic equation. To verify that all the eigenvalues have been found, we use a highly reliable procedure proposed by Carpentier and Dos Santos, which evaluates the number of zeros of an analytic function in a given region. Numerical results are presented for several examples and compared with those obtained by a method based on finite-dimensional approximations of delay equations.

Key words. eigenvalues, functional differential equations, numerical computation

AMS(MOS) subject classifications. 34K, 65H, 65L

1. Introduction. The eigenvalues associated with a system of linear functional differential equations (FDE) play an important role in many problems such as stability theory, oscillations, bifurcation control and feedback stabilization. These eigenvalues coincide with zeros of a certain entire function, which in the case of differential-difference equations is called a quasi-polynomial. The distribution of zeros of these quasi-polynomials has been earlier investigated by Dickson [10], [11], Siewert and Burniston [20], Wright [22], Bellman and Cooke [4]. Efficient methods to compute these eigenvalues, however, have not been sufficiently developed.

In this paper we present an algorithm that enables one to compute directly these eigenvalues from a characteristic equation obtained automatically from system matrices. The present computer implementation of the algorithm is limited to retarded functional differential equations with commensurate delays; however, an extension to a case of noncommensurate delays and to neutral equations is possible. The existing computer program was first written by G. Payre at the Universite de Montreal in 1977 and modified by R. Roy (Montreal, 1979, 1980), S. Brierley (University of Minnesota, 1981–83) and substantially refined by H. Tran at RPI in 1983–84.

An alternative approach to the numerical computation of the eigenvalues associated with FDE's is to compute the eigenvalues of a large matrix arising in a finite-dimensional approximation of FDE's by one of the following methods: averaging, spline or spectral. Efforts in this direction have been reported by K. Ito [13], who has used the spline approximations. The advantage of those techniques is that they use

* Received by the editors September 4, 1984; accepted for publication (in revised form) February 28, 1986.

† Mathematical Science Department, Rensselaer Polytechnic Institute, Troy, New York 12180. Present address, Division of Mathematical Sciences, National Science Foundation, Washington, D.C. 20550. The work of this author was supported by National Science Foundation grant DMS-8201719 and in part by Air Force Office of Scientific Research grant AFOSR-ISSA-84-00052.

‡ Mathematical Science Department, Rensselaer Polytechnic Institute, Troy, New York 12180. Present address, Division of Applied Mathematics, Brown University, Providence, Rhode Island 02912. The work of this author was supported by National Science Foundation grant DMS-8201719 and in part by Air Force Office of Scientific Research grant AFOSR-ISSA-84-00052.

§ Universite de Sherbrooke, Sherbrooke, Quebec, Canada.

¶ College Militaire Royal, St. Jean, Quebec, Canada.

well developed algorithms of numerical linear algebra (e.g. the QR method). However, only some of the computed eigenvalues approximate the true eigenvalues, while there is also a large number of false eigenvalues which do not correspond to those of the original functional differential equation. Without having a characteristic equation, the user has no way of knowing which of the computed eigenvalues are close to the true ones.

Other efforts related to the computation of roots of the characteristic equations have been reported in [21] and, in a different context, in [1].

We emphasize that all the examples computed in [13] and by other authors were very simple, where the characteristic equation could be found by a hand calculation. The present method has the advantage of computing the characteristic equation automatically from system matrices. This makes it possible to solve the problem for systems of larger dimension.

The content of the paper is as follows. In § 2 we describe the type of equations considered and give a bound on the location of the spectrum. Section 3 contains the development of the algorithm to compute the characteristic equation. Section 4 contains the development of the algorithm to compute the roots of the characteristic equation. Section 5 gives a test to determine the number of roots inside a given region. Section 6 describes certain programming details. In § 7 we present numerical results for several examples. Section 8 is the conclusion.

Throughout the paper the following notation will be used. The dimensions n and the delay $h>0$ are fixed. For $[a, b]$ a subinterval of the real line and R^n euclidean n-space, we shall use $C([a, b]; R^n)$ to denote the usual space of R^n-valued continuous functions defined over $[a, b]$. For a linear operator A we use the standard notation $\sigma(A)$ to denote the spectrum of A. Given a measurable function x: $[-h, 0]\to R^n$ and $t\geqq 0$, the function segment x_t: $[-h, 0]\to R^n$ is defined by $x_t(\theta)=x(t+\theta)$, $\theta\in[-h, 0]$. Finally, the symbol $|x|$ will denote the norm of x, where the norm is to be understood to be on the space in which x lies.

2. Eigenvalues associated with retarded functional differential equations. Consider the following system described by the linear autonomous retarded functional differential equation

$$\dot{x}(t)=L(x_t) \tag{2.1}$$

where L is a continuous linear mapping from $C([-h, 0]; R^n)$ to R^n given by

$$L(x_t)=\int_{-h}^{0}[d\eta(\theta)]x(t+\theta) \tag{2.2}$$

where $\eta(\theta)$, $-h\leqq\theta\leqq 0$, is an $n\times n$ matrix whose elements are functions of bounded variation. The solution to (2.1) can be written as

$$x_t=T(t)x_0 \tag{2.3}$$

where $T(t)$ is a solution operator T: $C([-h, 0], R^n)\to C([-h, 0], R^n)$. As is shown in [12] the family $\{T(t), t\geqq 0\}$ forms a strongly continuous semigroup on $[0, \infty]$. If A denotes the infinitesimal generator of $T(t)$, $t\geqq 0$, then

$$A\phi(\theta)=\begin{cases} d\phi(\theta)/d\theta, & -h\leqq\theta<0,\\ L(\phi), & \theta=0.\end{cases} \tag{2.4}$$

The eigenfunctions of A is any nonzero $\phi\in D(A)$ satisfying the equation

$$(\lambda I-A)\phi=0, \tag{2.5}$$

where I is the identity operator in C.

Following [12] define the $n \times n$ matrix dependent on λ.

$$\Delta(\lambda) = \lambda I - \int_{-h}^{0} [d\eta(\theta)]\, e^{\lambda\theta}. \tag{2.6}$$

Then the spectrum of A, $\sigma(A)$, is identical with the set of eigenvalues of A, which are the solutions of the characteristic equation

$$\det \Delta(\lambda) = 0, \tag{2.7}$$

while the eigenfunction ϕ_i corresponding to the eigenvalue λ_i is given by

$$\phi_i(\theta) = e^{\lambda\theta}\zeta_i, \qquad -h \leqq \theta < 0, \tag{2.8}$$

where $\zeta_i \in R^n$ satisfies the equation $\Delta(\lambda_i)\zeta_i = 0$. It is known that the eigenvalues of A have the following properties [12]:

(i) they are of finite multiplicity;
(ii) they have real parts bounded above;
(iii) for η that is piecewise constant on $[-h, 0]$, the eigenvalues are asymptotically distributed in curvilinear strips of type

$$|\mathrm{Re}\,(\tau + \mu_i \ln \tau)| \leqq \rho, \qquad i = 1, \cdots, k \quad [4].$$

Let $v_\eta(\theta)$ denote the variation function of η, for $\theta \in [-h, 0]$ and let Var η denote the total variation of η on $[-h, 0]$.

PROPOSITION 2.1. *Let λ be the eigenvalue of A. Then*

$$|\lambda| \leqq \int_{-h}^{0} e^{\mathrm{Re}\,\lambda\theta}\, dv_\eta(\theta). \tag{2.9}$$

Proof. Since λ satisfies (2.7), we have

$$\lambda w = \left(\int_{-h}^{0} d\eta(\theta)\, e^{\lambda\theta} \right) w \tag{2.10}$$

where w is some nonzero element of R^n. Taking the norm of both sides of (2.10), we have

$$\begin{aligned} |\lambda||w| &\leqq \left| \int_{-h}^{0} d\eta(\theta)\, e^{\lambda\theta} \right| |w|, \\ \left| \int_{-h}^{0} d\eta(\theta)\, e^{\lambda\theta} \right| &\leqq \int_{-h}^{0} |e^{\lambda\theta}|\, dv_\eta(\theta) \leqq \int_{-h}^{0} e^{\mathrm{Re}\,\lambda\theta}\, dv_\eta(\theta). \end{aligned} \tag{2.11}$$

Since $w \neq 0$, by dividing both sides of (2.11) by $|w|$ we obtain result (2.9).

Remark 2.1. Since $\mathrm{Re}\,\lambda \leqq |\lambda|$, formula (2.9) gives us a maximum bound on the real part of eigenvalues of A. This bound also serves as an estimate on the location of the right-hand boundary of the region needed to start the Kuhn algorithm. Furthermore, since

$$\int_{-h}^{0} e^{\mathrm{Re}\,\lambda\theta}\, dv_\eta(\theta) \leqq \mathrm{Var}\ \eta \sup_{-h \leqq \theta \leqq 0} e^{\mathrm{Re}\,\lambda\theta}, \tag{2.12}$$

formula (2.9) yields the following statement.

The spectrum $\sigma(A)$ of A is contained in the set of λ satisfying

$$|\lambda| \leqq \mathrm{Var}\ \eta \qquad \text{for } \mathrm{Re}\,\lambda \geqq 0, \tag{2.13}$$

$$|\lambda| \leqq \mathrm{Var}\ \eta\, e^{-\mathrm{Re}\,\lambda h} \qquad \text{for } \mathrm{Re}\,\lambda \leqq 0. \tag{2.14}$$

3. Algorithm to compute det $\Delta(\lambda)$. Let us assume that the matrix function $\eta(\cdot)$ is piecewise constant, so that the functional differential equation has the form

$$\dot{x}(t) = \sum_{i=0}^{N} A_i x(t-h_i), \tag{3.1}$$

where the delays h_i are commensurate. Without loss of generality, we can assume that $h_i = ih$, $i = 0, \cdots, N$, since the delays ih which are not represented in the equation will have corresponding matrices A_i equal to zero. Let

$$\hat{A}(\mu) = \sum_{i=0}^{N} A_i \mu^i, \quad \text{where } \mu = e^{-\lambda h}. \tag{3.2}$$

The characteristic equation is

$$\det \Delta(\lambda) = \det (I\lambda - \hat{A}(\mu)) = \lambda^n + \theta_1(\mu)\lambda^{n-1} + \cdots + \theta_n(\mu). \tag{3.3}$$

The coefficients $\theta_i(\mu)$ can be determined from the following algorithm generalizing the well-known Faddeev algorithm. $\theta_i(\mu)$ are the polynomials in μ and are given by the recursive formulas

$$F_1 = I,$$

$$\theta_i(\mu) = -\frac{1}{i} \operatorname{tr} (\hat{A}(\mu) F_i(\mu)), \qquad i = 1, 2, \cdots, n, \tag{3.4}$$

$$F_{i+1}(\mu) = \hat{A}(\mu) F_i(\mu) + \theta_i(\mu) I, \qquad i = 1, 2, \cdots, n-1. \tag{3.5}$$

For a proof of validity of these formulas see Buslowicz [6]. It follows from (3.4) and (3.5) that $F_i(\mu)$ are $n \times n$ matrix polynomials in μ of degree $(i-1)N$, and $\theta_i(\mu)$ are of degree iN. By writing

$$\theta_i(\mu) = \sum_{k=0}^{iN} \theta_{i,k} \mu^k, \tag{3.6}$$

$$F_i(\mu) = \sum_{k=0}^{(i-1)N} \Phi_{i,k} \mu^k, \tag{3.7}$$

and defining

$$\theta_{i,k} = 0, \quad k < 0 \text{ or } k > iN; \quad i \geqq 1,$$

$$\Phi_{i,k} = 0, \quad k < 0 \text{ or } k > (i-1)N; \quad i \geqq 1,$$

we have the recursive algorithm

$$\Phi_{1,0} = I,$$

$$\theta_{i,k} = \frac{1}{i} \operatorname{tr} \left(\sum_{j=0}^{N} A_j \Phi_{i,k-j} \right), \qquad i = 1, \cdots, n, \quad k = 0, \cdots, iN. \tag{3.8}$$

$$\Phi_{i+1,k} = \sum_{j=0}^{N} A_j \Phi_{i,k-j} + \theta_{i,k} I, \qquad i = 1, \cdots, n-1, \quad k = 0, \cdots, iN. \tag{3.9}$$

Remark 3.1. The number of coefficients $\theta_{i,k}$ to be stored increases with the number of delays, N, and with the dimension of the system, n. Furthermore, the number of elements of the matrices $\Phi_{i,k}$ involved in the algorithm increases as $(n^3(n-1))/2$. If one wants to compute the matrix $\operatorname{adj} (\lambda I - \hat{A}(\mu))$, all of these matrices need to be stored; otherwise, some of them can be erased. The coefficients $\theta_{i,k}$ can be stored in

a quasi-triangular matrix as shown in Fig. 1. The part of the matrix containing all the nonzero coefficients has a shape of a pyramid. Note that the coefficient $\theta_{0,0}$ of λ^n is always 1.

Remark 3.2. Once the coefficients $\theta_{i,k}$ are computed, we can use the following formula to compute the derivative $(d/d\lambda)\det\Delta(\lambda)$ at a given point λ:

$$\frac{d}{d\lambda}\det(I\lambda-\hat{A}(\lambda))=n\lambda^{n-1}+\sum_{i=1}^{n}(n-i)\theta_i(\mu)\lambda^{n-i-1}-h\sum_{i=1}^{n}\sum_{k=0}^{iN}k\theta_{i,k}\mu^k\lambda^{n-i}.$$

Remark 3.3. The present implementation of the algorithm to compute $\det\Delta(\lambda)$ is restricted to retarded systems with commensurate delays. However, an extension to noncommensurate delays is possible, albeit more cumbersome in coding. Also, an extension to neutral systems with commensurate delays is possible. Systems with an integral on the right-hand side can be approximated by systems with η piecewise polynomial. For the latter, the computation of the characteristic quasi-polynomial can be done by the same methods as in the case of a differential difference equation with commensurate delays.

Example 3.1 [17]. Consider the following scalar equation:

$$\dot{x}(t)=-\frac{1}{2}x(t)-\int_{-1/2}^{0}x(t+\theta)\,d\theta-\int_{-1}^{-1/2}\left(2\theta+\frac{3}{2}\right)x(t+\theta)\,d\theta. \tag{3.10}$$

It is possible to express (3.10) in form (2.1) and (2.2), that is

$$\dot{x}(t)=\int_{-1}^{0}[d\eta(\theta)]x(t+\theta),$$

where

$$\eta(\theta)=\sum_{i=0}^{2}A_i(\theta)\chi_{[-h_i,0]}(\theta),$$

$$A_0(\theta)=-\tfrac{1}{2},\quad A_1(\theta)=\theta(\theta+\tfrac{1}{2}),\quad A_2(\theta)=-(\theta+1)(\theta+\tfrac{1}{2}),$$

$$h_0=0,\quad h_1=\tfrac{1}{2},\quad h_2=1,$$

and $\chi_I(\theta)$ denotes a characteristic function of the interval I (i.e. equals one for $\theta\in I$ and zero otherwise).

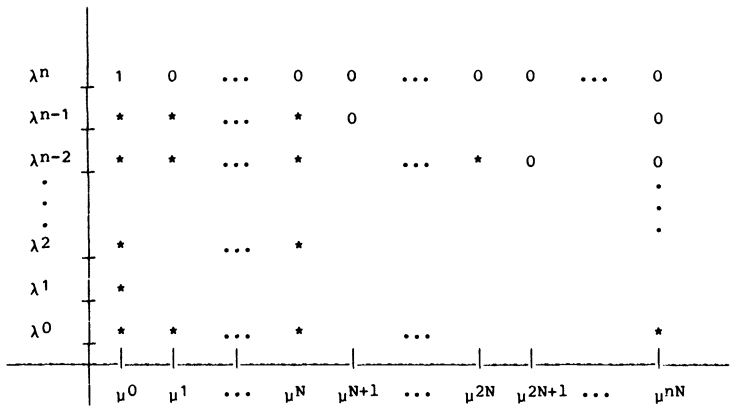


FIG. 1. *Coefficients $\theta_{i,k}$ stored as a quasi-triangular matrix. The asterisks * denote the (possibly) nonzero elements of the matrix.*

The characteristic equation is given by

$$\det \Delta(\lambda) = \lambda + \frac{1}{2} + \int_{-1/2}^{0} e^{\lambda\theta}\, d\theta + \int_{-1}^{-1/2} \left(2\theta + \frac{3}{2}\right) e^{\lambda\theta}\, d\theta$$
$$= \lambda + 0.5 + \frac{(1 - 0.5\, e^{-\lambda/2} + 0.5\, e^{-\lambda})}{\lambda} - \frac{(2e^{-\lambda/2} - 2e^{-\lambda})}{\lambda^2}.$$

Let

$$G(\lambda) = \lambda^2 \det(\Delta(\lambda)) = \lambda^3 + 0.5\lambda^2 + \lambda - 0.5\lambda\, e^{-\lambda/2} + 0.5\lambda\, e^{-\lambda} - 2e^{-\lambda/2} + 2e^{-\lambda}.$$

The function $G(\lambda)$ and $\det(\Delta(\lambda))$ have the same roots (except that $G(\lambda)$ also has a root of multiplicity 2 at the origin). However, $G(\lambda)$ has a similar form to that of (3.3) (and thus in principle one can use formulas (3.8), (3.9) modified to account for a polynomial dependence of A_i on λ to compute its coefficients). The following theorem quoted from [17] tells us that this is true in general.

THEOREM 3.1. *Let $\eta(\theta)$ be a piecewise polynomial of the form*

$$\eta(\theta) = \sum_{i=1}^{N} A_i(\theta)\chi_{[-h_i, -h_{i-1}]}(\theta), \qquad h_i = ih$$

where $A_i(\theta)$ is an $n \times n$ matrix of polynomials in θ of degree d_i, and χ is as in Example 3.1. Let d be the largest degree d_i. Then there exists an integer γ, $0 \leqq \gamma \leqq nd$ such that if

$$G(\lambda) = \lambda^{\gamma} \det\left(\lambda I - \int_{-h}^{0} d\eta(\theta)\, e^{\lambda\theta}\right)$$

then

$$G(\lambda) = q(\lambda, e^{-\lambda h})$$

where $q(\lambda, \mu)$ is a polynomial in two indeterminates.

4. Algorithms to compute the roots of $\det \Delta(\lambda)$.

4.1. A combinatorial algorithm to compute roots located in a bounded region of the complex plane (Kuhn's method). In a 1974 paper [15], H. W. Kuhn proposed a combinatorial algorithm to find approximations to roots of polynomials. This algorithm adapts easily to computation of zeros of entire functions. It has been used to compute eigenvalues of retarded functional differential equations [17] and eigenvalues of a Schrödinger operator.

We include a brief description of the algorithm.

Let $f(\lambda)$ denote an entire function of λ. Define the index

$$N_f(\lambda) = \begin{cases} 1 & \text{if } -\frac{\pi}{3} \leqq \arg f(\lambda) \leqq \frac{\pi}{3} \text{ or if } f(\lambda) = 0, \\ 2 & \text{if } \frac{\pi}{3} < \arg f(\lambda) \leqq \pi, \\ 3 & \text{if } -\pi < \arg f(\lambda) < -\frac{\pi}{3}. \end{cases}$$

Let $\{\lambda_1, \lambda_2, \lambda_3\}$ be a triple of distinct points. The triangle with vertices at λ_i, $i = 1, 2, 3$ is said to be *saturated* if $N_f(\lambda_j) = j$ for $j = 1, 2, 3$.

It can be shown that ε-small saturated triangles have the property that $|f(\lambda_j)|$ is small for $j = 1, 2, 3$. By subdividing a saturated triangle into smaller triangles and

finding the saturated ones, one can construct a sequence of shrinking triangles whose centers converge to a zero of the function $f(\lambda)$. Hence, the problem of finding the zeros of $f(\lambda)$ in a given bounded region can be attacked by constructing a subdivision of the region into triangles and finding the saturated ones.

In practice, one takes a rectangular region R and constructs a grid. The meshpoints of the grid are then searched for possible saturated triples. To avoid searching all the points of the grid, the algorithm proceeds as follows:

Step 1. A region R, usually a rectangular one, and a mesh size are chosen. A starting point on the boundary of R is chosen.

Step 2. The algorithm advances along the boundary of R, computing $N_f(\lambda)$ at each meshpoint of the boundary. The pairs of points at which the index change occurs are stored along with a direction n pointing toward the interior of R. This continues until the algorithm comes back to the starting point.

Step 3. For each triple $(\lambda_1, \lambda_2, n)$ with λ_1, λ_2 being consecutive meshpoints on the boundary of R, such that $N_f(\lambda_1) \neq N_f(\lambda_2)$, the algorithm finds two other meshpoints λ_3, λ_4 inside R obtained by moving one step from each λ_1 and λ_2 in the direction n. The values of $N_f(\lambda_3)$, $N_f(\lambda_4)$ are computed.

Step 4. If one of $N_f(\lambda_i)$, for $i=3, 4$ is different from both $N_f(\lambda_1)$ and $N_f(\lambda_2)$, a saturated triangle has been found. Otherwise, the algorithm determines a new triple $(\lambda_i, \lambda_j, m)$ with $i, j \in \{1, 2, 3, 4\}$, at least one of i, j is in $\{3, 4\}$, and the direction m points outwards of the rectangle $\{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}$, orthogonally to the segment $[\lambda_i, \lambda_j]$.

In this way the algorithm will proceed along the boundary of two zones of $N_f(\lambda)$ until either a saturated triangle has been found, or a boundary of R is reached again. In the latter case, the next triple $\{\lambda_1, \lambda_2, n\}$ with λ_1, λ_2 on the boundary is picked and the search continues until all the boundaries of zones have been traversed.

The moves of the algorithm on the grid are like those of a rook in a chess game.

In the next stage the algorithm proceeds to a subdivision of rectangles with saturated triples of vertices, by decreasing the meshsize tenfold. This subdivision is performed several times, until a desired accuracy of approximation is reached. The centers of saturated rectangles then serve as a starting point for a refinement via Newton's method. The speed of the algorithm depends on the size of the region and the number of roots in the region, so it is worthwhile to use it for relatively small regions around 0. The roots of sufficiently large modulus are more quickly computed by the asymptotic formulas.

Certain difficulties such as (1) repeated obtaining of the same saturated triangles or (2) a cluster of several saturated triangles in case of multiple eigenvalues have been overcome by a careful coding. That is, in case (1), before adding each newly obtained saturated triangle to the list, a check is made whether or not this triangle is already included. In case (2), if the cluster persists after several mesh refinements, each saturated triangle is used as a starting point for the Newton's algorithm. The final iterates are then compared. Those closer to each other than a prescribed tolerance are considered identical. Redundant repetitions of the same eigenvalue are then deleted from the list.

Remark 4.1. Kuhn's algorithm can also be used to compute zeros of other transcendental functions, which do not necessarily arise as a characteristic function associated with a delay equation. However, if such a function has poles very close to zeros, the Kuhn algorithm may miss such zeros.

For example, for

$$f(\lambda) = \frac{1 - e^{-\lambda}}{\lambda + \varepsilon}$$

the following phenomena were observed. The initial rectangle to start Kuhn's algorithm was chosen to have vertices at $(-2, 0), (2, 0), (-2, 20), (2, 20)$ in the complex plane with mesh size equal to 0.2. With $\varepsilon = \sqrt{2}$, the computed roots are $\lambda = 0$, $i6.283185307$ $(\simeq 2\pi)$, $i12.566370614$ $(\simeq 4\pi)$, and $i18.849555922$ $(\simeq 6\pi)$. For $\varepsilon = \sqrt{2} \times 10^{-6}$, the computed roots are $\lambda = i6.283185307$, $i12.566370614$ and $i18.849555922$. In this case, when ε is small, the function $f(\lambda)$ has a pole near a zero which is at the origin. Kuhn's method missed this zero.

We emphasize, however, that this situation *does not* arise in the context of eigenvalues associated with the retarded differential-difference equations.

4.2. Computation of approximate roots of large modulus. It is well known [4, Thm. 12.8] that large roots of the function

$$f(\lambda) = \lambda - \omega e^{-\lambda h} \tag{4.1}$$

can be approximated by the formulas

$$\begin{aligned} \lambda_k = &\frac{1}{h}\left[\ln|\omega| - \ln\frac{1}{h}|\arg(\omega) \mp \pi/2 + 2k\pi| + \varepsilon(k)\right] \\ &+ i\frac{1}{h}[\arg(\omega) \mp \pi/2 + 2k\pi + \varepsilon(k)] \end{aligned} \tag{4.2}$$

for $k = 0, \pm 1, \pm 2, \cdots$ where $\varepsilon(k) \to 0$ when $|k| \to \infty$. The upper signs correspond to $k > 0$ while the lower signs correspond to $k < 0$. This formula shows that the roots of (4.1) are asymptotically located on the curve

$$\operatorname{Re}\left(\lambda + \frac{1}{h}\log\lambda\right) = \frac{1}{h}\log|\omega| \tag{4.3}$$

and are asymptotically separated from each other by the distance $2\pi/h$. Computed examples show that the approximations obtained for $k = 1$ and 2 are already quite good. For example, for $h = 1$, $\omega = -\pi/2$, we have the following comparison for $k = 1$:

approximation given by (4.2): $-1.609438120 + 7.853981634i$,

root computed with $|f(\lambda)| < 10^{-15}$: $-1.604291124 + 7.647192249i$.

The following fact is crucial:

Remark 4.2. An arbitrary polynomial in two variables, $p(\lambda, \mu)$, monic in λ, in general *is not factorable* into a product of linear terms $\lambda - \omega_i\mu$. However, if $\mu = e^{-\lambda h}$, we have some sort of asymptotic $(|\lambda| \to \infty)$ factorizability into a product of terms of type $\lambda - \omega_i e^{-\lambda h/m}$. That is, if $|\lambda| \to \infty$ along certain curves in the complex plane, the zeros of $p(\lambda, e^{-\lambda h})$ are close to those of some comparison functions *which are factorable.* A more detailed explanation is given below in items (i)-(iv). Here $p(\lambda, \mu)$ will denote a polynomial in two variables λ, μ such that $p(\lambda, e^{-\lambda h}) = \det \Delta(\lambda)$.

(i) Let P denote the matrix of coefficients p_{ij} of terms $\lambda^i\mu^j$ in the polynomial $p(\lambda, \mu)$, with p_{00} in the lower left corner of P, and with j increasing in the horizontal direction, i in the vertical. See Fig. 1.

(ii) Let $L_1, L_2, \cdots, L_s$ denote a concave polygonal graph connecting the uppermost nonzero entries of matrix P, with straight line segments L_k having slope $-m_k$, where $m_k > 0$.

(iii) By the results of [4, Chap. 12], the roots of large modulus of $p(\lambda, e^{-\lambda h})$ are situated in the curvilinear strips

$$V_k = \{\lambda \in C \,|\, |\operatorname{Re}(\lambda + m_k \ln\lambda)| \leqq \rho\} \tag{4.4}$$

where $\rho > 0$ is some constant. In each strip the roots are asymptotically (as $|\lambda| \to \infty$) close to the zeros of the following "comparison function:"

$$f_k(\lambda) = \sum_{L_k} p_{ij} \lambda^i e^{-j\lambda h} \tag{4.5}$$

where the sum is taken over terms corresponding to those nonzero entries of the matrix P which are located on the segment L_k.

(iv) Since on L_k one has $i - \tau_k = -m_k j$ for some real τ_k, by making the substitution $z = \lambda e^{\lambda h/m_k}$, and dividing $f_k(\lambda)$ by the nonzero function $e^{-\lambda \tau_k h/m_k}$, we see that $f_k(\lambda)$ is *replaced by the polynomial*

$$\phi_k(z) = \sum_{L_k} p_{ij} z^i. \tag{4.6}$$

We now find the roots z_l of the polynomial and solve the equation $\lambda e^{\lambda h/m_k} = z_l$. The latter is equivalent to $\lambda - z_l e^{-\lambda h/m_k} = 0$. Hence, this corresponds to a factorization of (4.5) into a product of linear terms of type (4.1), with appropriate constants ω_i, h_k.

Example 4.1.

$$p(\lambda, \mu) = \lambda^{10} + \lambda^9 + \lambda^5 \mu + (-\lambda^8 + 2\lambda^3)\mu^2 - 4\lambda^6 \mu^3 + (-4\lambda^4 + 7\lambda)\mu^4 + 4\mu^5.$$

The concave polygonal graph has three segments L_1, L_2, L_3 with respective slopes $-m_k = -1, -2, -4$. The comparison functions are (for $\mu = e^{-\lambda\pi}$)

$$f_1(\lambda) = \lambda^{10} - \lambda^8 e^{-2\pi\lambda},$$
$$f_2(\lambda) = -\lambda^8 e^{-2\pi\lambda} - 4\lambda^6 e^{-3\pi\lambda} - 4\lambda^4 e^{-4\pi\lambda},$$
$$f_3(\lambda) = -4\lambda^4 e^{-4\pi\lambda} + 4 e^{-5\pi\lambda}.$$

Now $f_1(\lambda) = \lambda^8 e^{-2\pi\lambda}[(\lambda e^{\lambda\pi})^2 - 1]$, and the zeros of $f_1(\lambda)$ coincide with the union of zeros of functions $\lambda - z_l e^{-\lambda\pi}$ for $l = 1, 2$, z_l being roots of $z^2 - 1$. Those of $f_2(\lambda)$ are solutions of $\lambda - z e^{-\lambda\pi/2} = 0$, with z satisfying $z^4 + 4z^2 + 4 = 0$; those of $f_3(\lambda)$ are solutions of $\lambda - z e^{-\lambda\pi/4}$, with z satisfying $z^4 - 1 = 0$.

The algorithm for finding zeros of large modulus can be thus summarized as follows:

Step 1. Construct the matrix P and the segments L_k; compute the slopes $-m_k$ of L_k, $k = 1, \cdots, s$. (This is done by a certain systematic search procedure described in § 6.)

Step 2. Construct the polynomials $\phi_k(z)$ given by (4.6) and compute their roots z_l (by using a standard IMSL routine ZRPOLY).

Step 3. Find the approximate roots of "comparison functions" (4.5) by finding zeros of functions (4.1) by using formulas (4.2) with $\omega = z_l$, and h replaced by h/m_k.

Step 4. Use the roots of "comparison functions" as initial guesses for the roots of $p(\lambda, e^{-\lambda h})$ and iterate by Newton's method.

4.3. Refinement of the approximations. The approximations to the roots of the equation $f(\lambda) = \det \Delta(\lambda) = 0$ computed by the algorithms given in §§ 4.1 and 4.2 are then used as initial guesses to start Newton's method for improvement, which is defined by the following well-known iteration formula:

$$\lambda_{m+1} = \lambda_m - \frac{f(\lambda_m)}{f'(\lambda_m)}. \tag{4.7}$$

The formula requires the evaluations of the function $f(\cdot)$ and its derivative $f'(\cdot)$. In § 3, we proposed a procedure for the calculations of the coefficients of the characteristic equation $f(\lambda)$. The knowledge of these coefficients enables us to obtain exactly the derivative $f'(\lambda)$. The latter is useful in checking whether a computed eigenvalue is single or multiple.

It can be easily shown that the following formula is an estimate of the error after a finite number of iterations. This estimate also takes into account the fact that the calculated value of $\phi(\lambda_m)=\lambda_m-f(\lambda_m)/f'(\lambda_m)$ is usually afflicted with error; for example, round-off errors (also see [8]) are

$$|\bar{\lambda}_{m+1}-\lambda|<\frac{\beta}{1-\beta}|\bar{\lambda}_{m+1}-\bar{\lambda}_m|+\frac{1}{1-\beta}\sigma \tag{4.8}$$

where $\bar{\lambda}_i$ are computed roots from (4.7), λ is the exact root of $f(\lambda)=0$, β is the upper bound of the absolute value of $\phi'(\cdot)$, and σ is the largest absolute value of the error in the computation of $\phi(\bar{\lambda}_m)$. For large values of m, the computational error is the dominant contribution to $|\bar{\lambda}_{m+1}-\lambda|$. Thus the limiting factor for the accuracy which can be achieved in calculating the eigenvalues is the accuracy of computation of $f(\lambda_m)$. Therefore, in actual computations, instead of solving iteratively $f(\lambda)=0$, we solve $e^{k\lambda h}f(\lambda)=0$, where k is the largest j such that at least one of coefficients $\theta_{i,j}$, $i=1,\cdots,n$, is nonzero. This is done just for the purpose of scaling and improving numerical accuracy of iterations.

To terminate the iterations, we accept $\bar{\lambda}_m$ as an eigenvalue when the following two conditions are satisfied simultaneously:

$$|\bar{\lambda}_{m+1}-\bar{\lambda}_m|\geqq|\bar{\lambda}_m-\bar{\lambda}_{m-1}| \quad \text{and} \quad |\bar{\lambda}_m-\bar{\lambda}_{m-1}|<\varepsilon. \tag{4.9}$$

Here ε is some coarse tolerance, used only to prevent the iterations from being terminated before $\bar{\lambda}_m$ has even come close to λ.

According to [8, § 6.6.2] this termination criterion reduces the risk of not terminating the iterations for ill-conditioned eigenvalues (that is such eigenvalues λ for which $|f'(\lambda)|$ is small) or the risk of giving up the attainable accuracy for well conditioned eigenvalues.

With the IBM 3081D using double precision and $\varepsilon=10^{-8}$, the Newton procedure usually produces single eigenvalues within 4 to 8 iterations. For double eigenvalues, the convergence is asymptotically the same as with the bisection method, and it requires about 30 iterations.

5. Test on the number of eigenvalues. The improved eigenvalues are arranged in one listing in the order of increasing modulus. The number of eigenvalues within a given disc D of prescribed radius is then counted. Next, to verify that all the eigenvalues have been found, the index of the function $f(\lambda)=\det\Delta(\lambda)$ in the disc D is numerically computed by using the Carpentier-Dos Santos algorithm [7].

Using the argument principle, the index of $f(\lambda)$ in D is given by

$$C_0=\frac{1}{2\pi i}\int_D\frac{f'(z)}{f(z)}\,dz, \tag{5.1}$$

or

$$C_0=\frac{1}{2\pi}\int_0^{2\pi}\frac{-i\phi'(\theta)}{\phi(\theta)}\,d\theta, \tag{5.2}$$

where $\phi(\theta)=f(\rho e^{i\theta})$ and ρ is the radius of D. However, since the primitive of $\phi'(\theta)/\phi(\theta)$ is known, we can get a formula for C_0 that does not require knowledge of f. That is, (5.2) can be written in the form

$$C_0=\frac{1}{2\pi i}\frac{M}{2\pi}\int_0^{2\pi}g(\theta)\,d\theta, \tag{5.3}$$

where $g(\theta)=\ln[\phi(\theta)/\phi(\theta-(2\pi/M))]$ and $M+1$ is the total number of mesh points in the subdivision of $[0, 2\pi]$. The evaluation of these integrals by the trapezoidal rule yields the formula

$$C_0=\frac{1}{2\pi i}\sum_{l=1}^{M} g(\theta_l)=\frac{1}{2\pi}\sum_{l=1}^{M}\operatorname{Im}[g(\theta_l)], \tag{5.4}$$

where θ_l, $l=1, \cdots, M$, are mesh points of $[0, 2\pi]$.

Formula (5.4) is exact, providing $g(\theta)$ is the branch of $\ln[f(z)/f(z\,e^{-i2\pi/M})]$ that is analytic in an annulus containing the contour and vanishes for $N=\infty$. However, the range of $\operatorname{Im}[g(\theta)]$ may be larger than $[-\pi, \pi]$ in which case the computer may give erroneous values for some points θ_l, as it can only obtain the principal value of $\arg[f(z)/f(z\,e^{-i2\pi/M})]$. Carpentier and Dos Santos in [7] proposed the following test on the computed values of ϕ_l/ϕ_{l-1}:

Test. If for all l, ϕ_l is such that

(i) $|\arg(\phi_l/\phi_{l-1})|<3\pi/4$,

(ii) $1/6.1<|\phi_l/\phi_{l-1}|<6.1$,

the computed value of the index is accepted. If any of the conditions (i) or (ii) is not satisfied the index is recalculated with M replaced by $2M$.

In all our computations the test was very reliable, usually providing an acceptable value of the index with $M=128$ for discs of radius 8 and with $M=256$ for discs of radius 20.

6. Programming details.

6.1. Algorithm implementation. One of the main steps in the method for finding eigenvalues of large modulus is the search for the concave polygonal graph, consisting of straight line segments L_k, connecting the uppermost nonzero elements of matrix P. An algorithm for obtaining L_k will now be described as follows:

Step 1. Set $m=2$, $s=1$.

Step 2. Find any $p_{ij}\neq 0$, $i\in\{m, \cdots, n+1\}$ and $j\in\{(i-1)N, (i-1)N-1, \cdots, m\}$. If none, stop (characteristic equation is a polynomial and asymptotic formulas are not applicable here).

Step 3. Compute and store slope connecting p_{11} and p_{ij}.

Step 4. Find any $p_{kl}\neq 0$, $k\in\{i+1, \cdots, n+1\}$ and $l\in\{(i-1)N, \cdots, j+1\}$. If none, go to step 7.

Step 5. Compute slope connecting p_{11} and p_{kl}. If this slope is equal to the slope in step 3 then p_{ij} and p_{kl} are stored; they are lying on a straight line connecting p_{11}, p_{ij} and p_{kl}. Otherwise, set $p_{ij}=p_{kl}$. (This means that the old p_{ij} was not on the concave polygonal envelope of the set of points p_{lr}.)

Step 6. Set $m=k+1$ and go to step 2.

Step 7. A slope connecting the uppermost nonzero elements has been found, denoted by L_s.

Step 8. Set $s=s+1$, $p_{11}=p_{ij}$ and $m=i+1$. Go to step 2.

The algorithm terminates when the line L_k has endpoint $p_{n+1,j}$ or there is no other line L_i with smaller slope than L_k.

Another important point is the starting value for M in the Carpentier-Dos Santos algorithm discussed in § 5. If M is chosen too small, it may happen that the method will produce a wrong answer while satisfying conditions (i), (ii) in § 5. To correct that, one should double the value of M when both conditions (i) and (ii) are satisfied for the first time, and check if the count of roots remains the same. While it is difficult to state a general rule of thumb for the starting value of M, in our examples $M=16$ was always a good starting value.

6.2. CPU time and storage. Table 1 shows CPU times obtained for three different examples of a system of dimension $n = 10$ with one delay; these systems are shown in the Appendix. The systems with $n = 2, 4, 6, 8$ are obtained from the system with $n = 10$ by deleting the last rows and columns of matrices A_0, A_1. The CPU time reported was obtained by using system subroutine TIME on IBM 3081D. All the results were obtained while using Newton's refinement with $\varepsilon = 10^{-8}$.

For $n \geqq 12$ Kuhn's algorithm still works, but the Newton refinement for the asymptotic eigenvalues of large modulus did not converge (for the test example), because of round-off errors arising in the numerical evaluation of $\det \Delta(\lambda)$ (the latter expansion is a sum of a large number of complex-valued terms of large modulus).

With regard to storage considerations the algorithms require $(n^3(n-1))/2 + cn^2 + dn$ storage locations, where c and d are small constants. Although this figure is not a factor on the IBM 3081D, it might limit the applicability of the algorithm to RFDE of order 10 or less in some smaller computing systems. The storage requirements of order n^4 are not large when compared with the storage requirement of $n(N+1) \times n(N+1)$ arrays in the finite-dimensional approximation schemes.

7. Numerical results. In this section we present numerical results obtained by implementing the methods described in the previous sections. For the sake of comparison, we also exhibit results obtained by approximating the FDE's by a finite-dimensional system. The following approximation schemes are used: averaging, spline (linear spline based) and spectral (Legendre polynomial based) which are discussed in [2], [3], [14], respectively. The eigenvalues of the matrices arising in the finite-dimensional approximations of the FDE's by the above methods are computed using the standard algorithm, the QR-method. All of the examples which follow were carried out on an IBM 3081D at Rensselaer Polytechnic Institute using software packages written in Fortran.

TABLE 1
CPU *time versus equation order for three different examples.*

		CPU time (sec.)		
Example	Equation order $n =$	Kuhn's method with two mesh refinements	Asymptotic formulas of § 4.2 for 100 eigenvalues	Total
A	2	0.2	1.0	1.2
	4	0.8	6.4	7.2
	6	4.0	19.8	23.8
	8	23.2	44.4	67.6
	10	126.8	117.8	244.6
B	2	0.2	0.6	0.8
	4	0.5	3.6	4.1
	6	1.2	11.3	12.5
	8	9.4	26.4	35.8
	10	28.8	51.1	79.9
C	2	0.1	0.9	1.0
	4	0.5	2.0	2.5
	6	3.4	3.3	6.7
	8	19.7	4.9	24.6
	10	49.6	7.1	56.7

In order to start Kuhn's algorithm, we need to specify a rectangular region around the origin inside which the eigenvalues are to be approximated and a mesh size. In our examples, the rectangle is chosen to have vertices at $(-7, 10)$, $(7, 10)$, $(7, -10)$ and $(-7, -10)$ in the complex plane and the mesh size is 0.2. As discussed in § 4.2, the eigenvalues of large modulus are distributed in some curvilinear strips. In each strip, the eigenvalues can be approximated by formula (4.2) which is valid for all integers k. In our results, k is chosen to have values from 1 to 100.

The entries λ^0 and λ_k^0 in the tables which follow represent the approximate eigenvalues by using Kuhn's method and the asymptotic formulas, respectively. The symbols λ and λ_k denote the roots found by using Newton's method with λ^0 and λ_k^0 used as initial guesses, respectively. The λ_{SPL}^N, λ_{SPC}^N and λ_{AVE}^N represent the computed eigenvalues by using a finite-dimensional approximation, by a linear spline, Legendre polynomial and averaging based methods (where N characterizes the dimension of the approximating system). Only those eigenvalues that truly approximate the actual eigenvalues λ are listed in the tables. $\det \Delta(\lambda)$ or $\det \Delta(\lambda_k)$ is the evaluation of the characteristic equation (3.3) at λ or λ_k with positive imaginary parts.

Example 7.1 (Ito [13, Ex. 3]). We consider the equation of a harmonic oscillator with an additional delayed velocity term.

$$\ddot{x}(t)+\dot{x}(t-1)+x(t)=0.$$

The characteristic equation is

$$\lambda^2+\lambda e^{-\lambda}+1=0.$$

For this example, the matrix P described in § 4.2 is

$$P=\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 1 & 0 & 0\end{bmatrix}$$

(this was obtained by rewriting the differential equation as a 2×2 system of first order equations).

Table 2 shows the eigenvalues found by Kuhn's method in the rectangle $|\mathrm{Re}\,\lambda|\leqq 7$, $|\mathrm{Im}\,\lambda|\leqq 10$. Table 3 shows the eigenvalues found by the asymptotic formulas. Only a selected number of the first 100 eigenvalues generated by the asymptotic formulas are shown. The first of these coincides with the third found by Kuhn's method. The index k denotes the ordering number in the asymptotic chain and it corresponds to the index k in formula (4.2). A complete set of 100 eigenvalues found by both methods is plotted on Fig. 2. Table 4 lists some of the eigenvalues computed by using averaging, spline and spectral methods, with $N=16$. Note that only some of the computed eigenvalues are listed; those are the ones that approximate the true eigenvalues. However, among

TABLE 2

Eigenvalues for Example 7.1 *found by Kuhn's method in the rectangle* $|\mathrm{Re}\,\lambda|\leqq 7$, $|\mathrm{Im}\,\lambda|\leqq 10$.

No.	λ^0	λ	$\det \Delta(\lambda)$
1	-0.73800	-0.7384324	$8\times 10^{-17}+i4\times 10^{-40}$
2	$+0.02200\pm i1.60200$	$+0.0219316\pm i1.6019533$	$2\times 10^{-15}+i9\times 10^{-17}$
3	$-2.04600\pm i7.58200$	$-2.0468788\pm i7.5820276$	$-4\times 10^{-15}+i2\times 10^{-15}$

TABLE 3

Eigenvalues for Example 7.1 found by the asymptotic formulas. The index k denotes the ordering number in the asymptotic chain as in formula (4.2). Eigenvalues 1 and 2 are not in any chain.

No.	k	λ_k^0	λ_k	No. Iter.	$\det \Delta(\lambda_k)$
3	1	$-2.061021 \pm i7.853982$	$-2.0468788 \pm i7.5820276$	6	$-1\times10^{-15} - i6\times10^{-16}$
4	2	$-2.648807 \pm i14.137167$	$-2.6483989 \pm i13.9476930$	6	$2\times10^{-14} + i6\times10^{-15}$
5	3	$-3.016532 \pm i20.420352$	$-3.0179149 \pm i20.2718706$	6	$-4\times10^{-14} - i6\times10^{-15}$
6	4	$-3.284796 \pm i26.703538$	$-3.2863986 \pm i26.5801809$	5	$-9\times10^{-14} - i9\times10^{-15}$
7	5	$-3.496105 \pm i32.986723$	$-3.4976128 \pm i32.8805555$	6	$-7\times10^{-14} - i6\times10^{-15}$
8	6	$-3.670459 \pm i39.269908$	$-3.6718111 \pm i39.1763359$	6	$-1\times10^{-13} - i1\times10^{-14}$
9	7	$-3.818879 \pm i45.553093$	$-3.8200782 \pm i45.4691955$	5	$-1\times10^{-13} - i1\times10^{-14}$
⋮	⋮	⋮	⋮	⋮	⋮
52	50	$-5.754888 \pm i315.730062$	$-5.7549850 \pm i315.7118347$	5	$-1\times10^{-11} - i2\times10^{-12}$
⋮	⋮	⋮	⋮	⋮	⋮
101	99	$-6.435519 \pm i623.606142$	$-6.4355532 \pm i623.5958220$	5	$-2\times10^{-11} - i2\times10^{-13}$
102	100	$-6.445544 \pm i629.889327$	$-6.4455777 \pm i629.8790943$	4	$-2\times10^{-11} - i3\times10^{-13}$

the eigenvalues computed by these methods there is also a large number of false eigenvalues which do not correspond to those of the original FDE. This is illustrated by Figs. 3, 4 and 5 where the correct approximations to the true eigenvalues are circled. Figures 3 and 4 show the computed eigenvalues obtained by the averaging and the spline method, respectively. Symbols □, △, ↑, * indicate the location of computed eigenvalues obtained with $N = 4, 8, 16, 32$, respectively. Figure 5 shows the computed eigenvalues found by the Legendre-tau spectral method, where symbols □, △, ↑, * correspond to $N = 4, 8, 12, 16$, respectively. Of these three methods, the spectral method produced the largest number of correct approximations.

Remark 7.1. In Figs. 3 and 5 the solid curve is the plot for the function $2e^{-\mathrm{Re}\,\lambda}$. Lasiecka and Manitius in [16] proved that the semigroups corresponding to the averaging approximations of RFDE are differentiable in the space $R^n \times L_2$, *uniformly with respect to N.* A crucial feature of the differentiable semigroups is that the spectrum of the generator is contained in some logarithmic sector in the left-half plane. For this example, the spectrum has to be contained by the function $2e^{-\mathrm{Re}\,\lambda}$. Figures 3 and 5 show that averaging and spectral approximations have this property; however, spline approximations do not have it.

To make sure that all the roots have been found by Kuhn and asymptotic methods, we use the algorithm discussed in § 5 to compute the index of the function $\det \Delta(\lambda)$ in the disc D. For this example, the disc D is a circle centered at origin with radius equals to 10. The algorithm produces an index equal to 5 with $M = 64$, $\max |\arg(\phi_l/\phi_{l-1})| = 0.374$ and $\max |\phi_l/\phi_{l-1}| = 2.606$. The number of eigenvalues in Table 2 and Table 3 having absolute values less than 10 is also equal to 5. One can

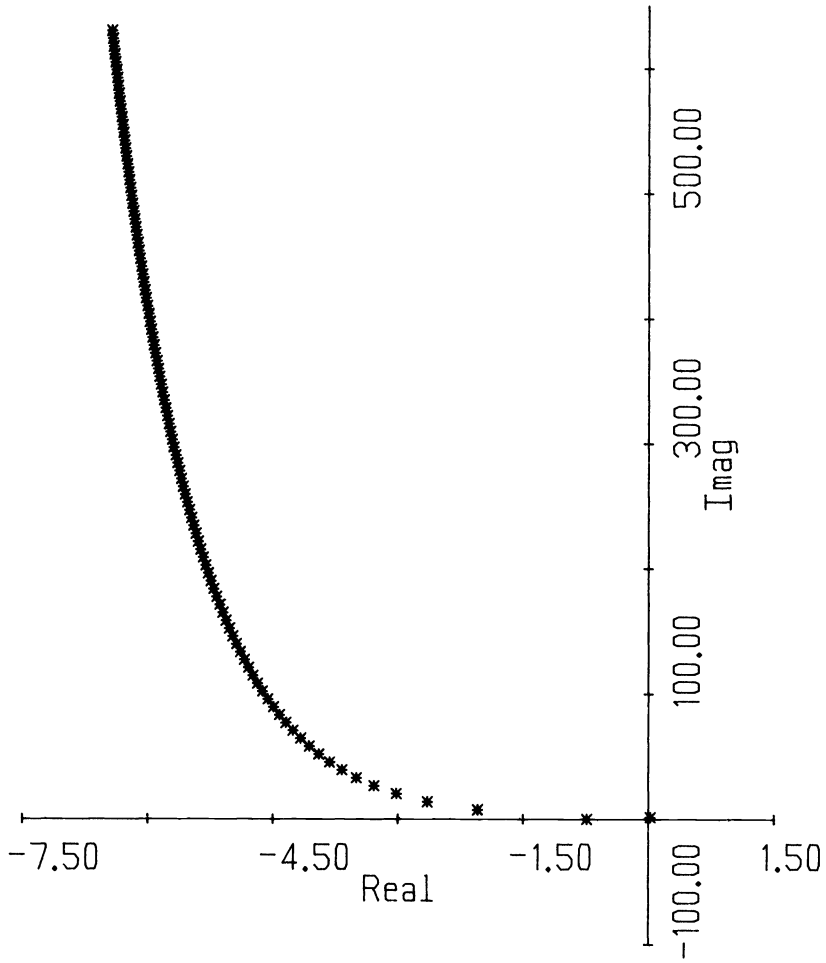


FIG. 2. *Location of the first* 100 *eigenvalues in Example* 7.1.

TABLE 4

Eigenvalues for Example 7.1 *computed by using the finite-dimensional approximations, i.e. averaging* (AVE), *spline* (SPL) *or Legendre-tau spectral method* (SPC), *with* $N = 16$. *Only the correct approximations to the eigenvalues are listed in the table.*

No.	λ_{AVE}^{16}	λ_{SPL}^{16}	λ_{SPC}^{16}
1	−0.7263888	−0.7385605	−0.7384324
2	$-0.0046729 \pm i1.5682735$	$+0.0229392 \pm i1.6009948$	$+0.0219316 \pm i1.6019533$
3	⋮	$-2.5198010 \pm i7.3326441$	$-2.0468788 \pm i7.5820276$
4		⋮	$-2.6483989 \pm i13.9476930$
5			$-3.0178133 \pm i20.2717593$
6			$-3.3347344 \pm i26.5255116$
⋮			⋮

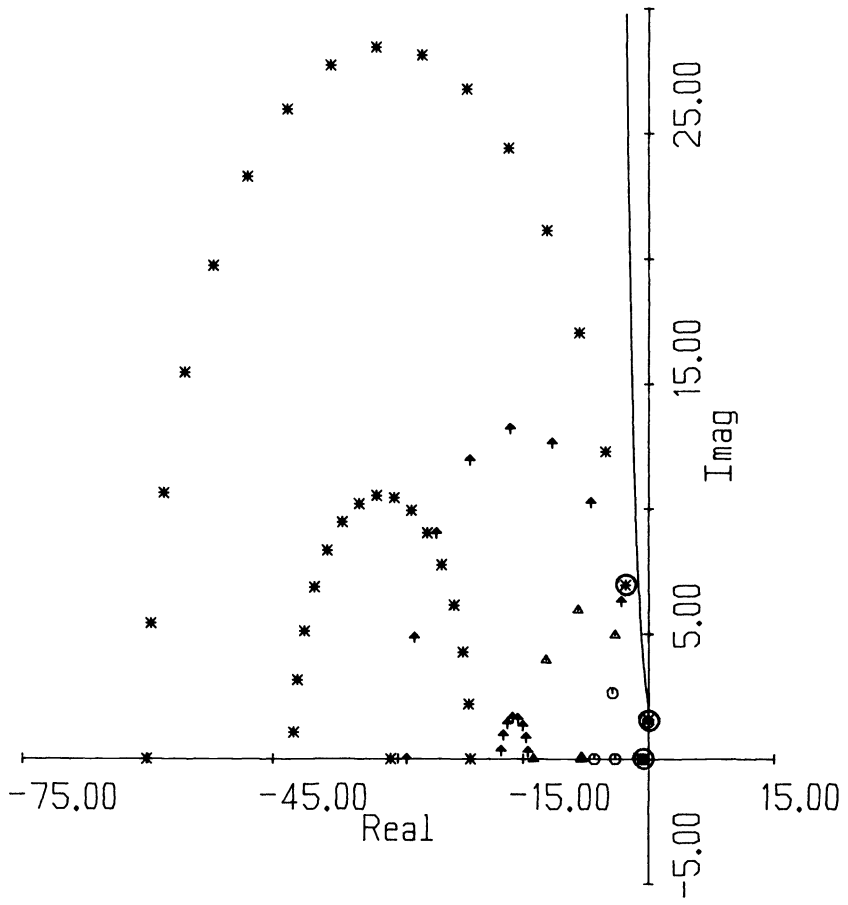


FIG. 3. *The correct (circled) and the false eigenvalues (uncircled) obtained by* AVE *approximation of equation in Example* 7.1 *with* $N = 4$ (□), 8 (△), 16 (↑) *and* 32 (*).

increase the radius of the circle to check on the number of roots in a larger region. In all of our examples, the index produced by the method discussed in § 5 and the count on the number of eigenvalues produced by Kuhn's method and asymptotic formulas always match.

Example 7.2. For the next example we consider the following system of three first-order RFDE's

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_3(t) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1(t-1) \\ x_2(t-1) \\ x_3(t-1) \end{bmatrix}$$

with the matrix P being

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

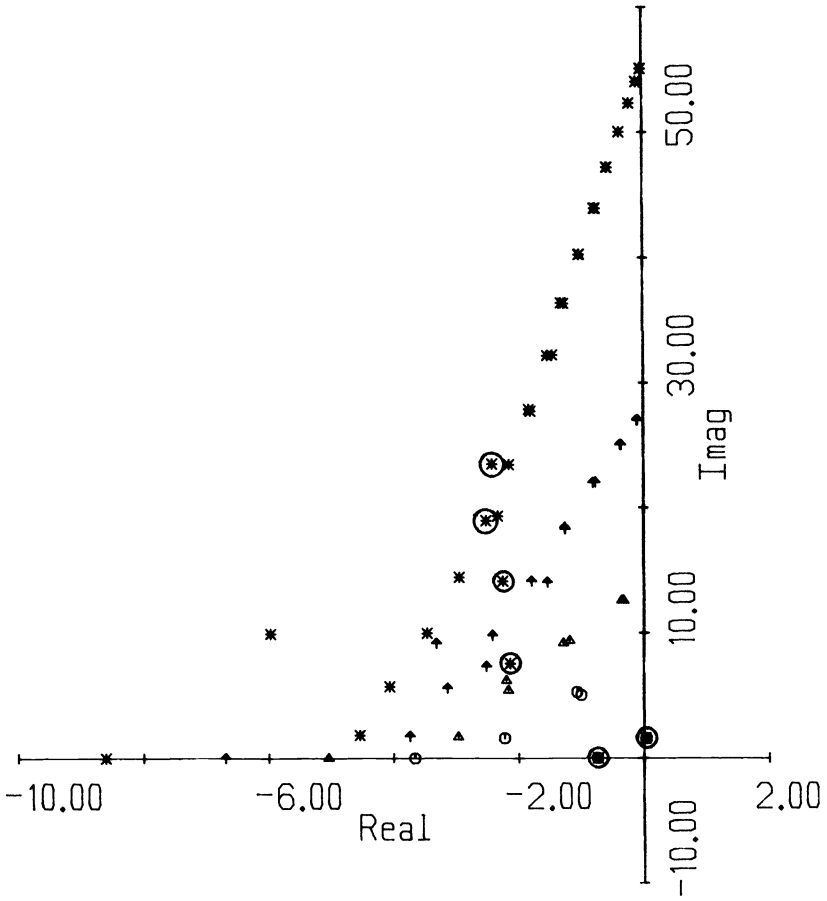


FIG. 4. *The correct (circled) and the false eigenvalues (uncircled) obtained by the linear spline approximation of equation in Example* 7.1 *with* $N=4$ (□), 8 (△), 16 (↑) *and* 32 (*).

and the characteristic equation given by

$$\det \Delta(\lambda) = \lambda^3 - \lambda^2 e^{-\lambda} - e^{-2\lambda}.$$

From matrix P, one can see that the roots of large modulus are situated in two curvilinear strips (see Fig. 6). In each strip the roots are asymptotically close to the zeros of the comparison functions $\lambda^3 - \lambda^2 e^{-\lambda}$ and $-\lambda^2 e^{-\lambda} - e^{-2\lambda}$. Table 5 and Table 6 list the eigenvalues found by Kuhn's method and by asymptotic formulas, respectively. Again only a selected number of eigenvalues is listed in Table 6, while a complete set of eigenvalues found by Kuhn's method and by asymptotic formulas is plotted in Fig. 6. Table 7 shows the true eigenvalues computed by using averaging, spline and spectral methods. To check on the number of roots, a circle of radius equals to 10 and centered at origin is given. The algorithm produced an index equal to 7 with $M=128$, $\max |\arg(\phi_l/\phi_{l-1})| = 0.359$ and $\max |\phi_l/\phi_{l-1}| = 2.813$. A count on number of eigenvalues in Table 5 and Table 6 having absolute values less than 10 is also 7.

Example 7.3. Here we consider the following system of six first-order RFDE's with two delays:

$$\dot{x}(t) = A_0 x(t) + A_1 x(t-1) + A_2 x(t-2),$$

where

$$A_0 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix},$$

$$A_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$A_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Here the matrix P is

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -3 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

and the characteristic equation is given by

$$\det \Delta(\lambda) = \lambda^6 - \lambda^5 e^{-2\lambda} - 3\lambda^4 e^{-\lambda} - 2\lambda^4 e^{-4\lambda} + 3\lambda^3 e^{-3\lambda} + 2\lambda^3 e^{-6\lambda} + 6\lambda^2 e^{-5\lambda} - 6\lambda e^{-7\lambda}.$$

TABLE 5

Eigenvalues for Example 7.2 found by Kuhn's method in the rectangle $|\mathrm{Re}\,\lambda| \leqq 7$, $|\mathrm{Im}\,\lambda| \leqq 10$.

No.	λ^0	λ	$\det \Delta(\lambda)$
1	$+0.78800$	$+0.7879311$	$4 \times 10^{-17} - i8 \times 10^{-29}$
2	$+0.11000 \pm i0.73600$	$+0.1094469 \pm i0.7365426$	$-3 \times 10^{-17} - i8 \times 10^{-17}$
3	$-1.48200 \pm i4.14000$	$-1.4814767 \pm i4.1408032$	$-1 \times 10^{-14} - i1 \times 10^{-14}$
4	$-3.64200 \pm i5.22600$	$-3.6413171 \pm i5.2271866$	$2 \times 10^{-13} + i1 \times 10^{-13}$

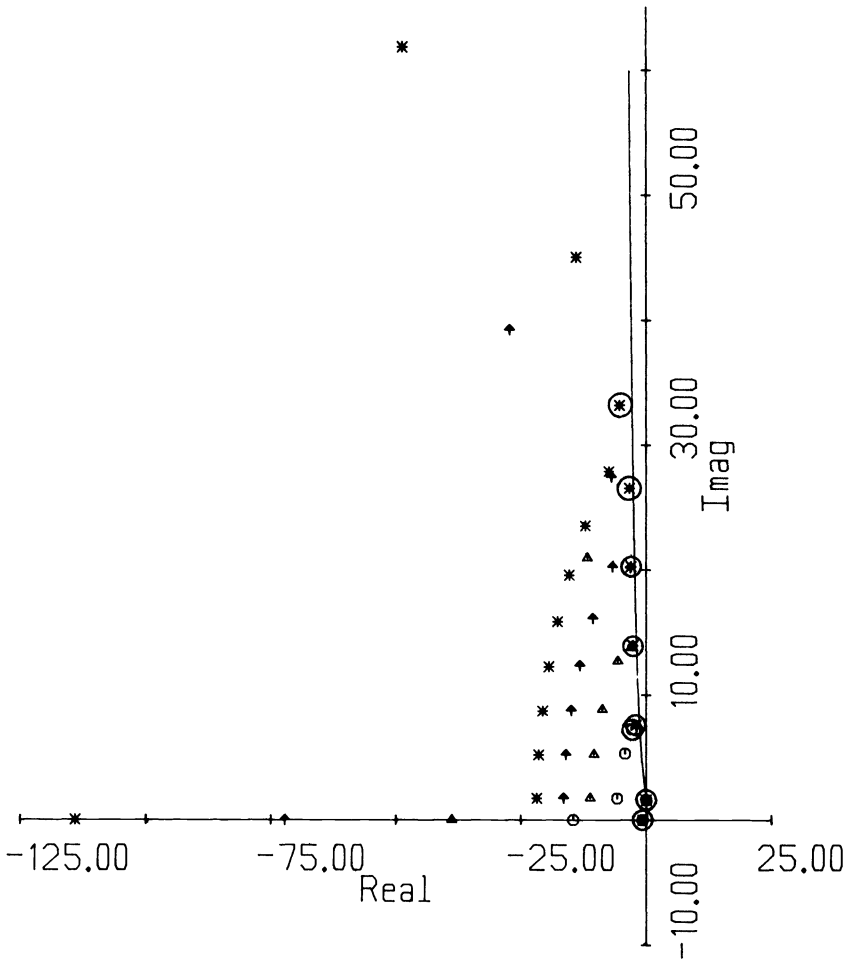


FIG. 5. *The correct (circled) and the false eigenvalues (uncircled) obtained by the Legendre-tau approximation of equation in Example* 7.1 *with* $N = 4$ (□), 8 (△), 16 (↑) *and* 32 (*).

For this example, the roots of large modulus are situated in two curvilinear strips (see Fig. 7) and the comparison functions are $\lambda^6 - \lambda^5 e^{-2\lambda} - 2\lambda^4 e^{-4\lambda} + 2\lambda^3 e^{-6\lambda}$ and $2\lambda^3 e^{-6\lambda} - 6\lambda e^{-7\lambda}$. Table 8 and Table 9 list the selected eigenvalues computed by Kuhn's method and asymptotic formulas, respectively. Figure 7 plots the eigenvalues found by the two methods. The algorithm that computes the number of roots inside a circle centered at origin with radius equal to 10 produced an index of 26 with $M = 256$, $\max |\arg(\phi_l/\phi_{l-1})| = 0.498$ and $\max |\phi_l/\phi_{l-1}| = 5.732$. The number of eigenvalues in Table 8 and Table 9 having absolute value less than 10 is also 26.

8. Concluding remarks. We have described a method to compute directly the eigenvalues associated with FDE from a characteristic equation obtained automatically from system matrices. A number of numerical issues and an algorithm to numerically compute the index of a function were addressed and several examples were given. The method works well and performs reliably even on relatively high order systems of

TABLE 6

Eigenvalues for Example 7.2 found by asymptotic formulas. These eigenvalues are located in two curvilinear strips (see Fig. 6). The first strip has one chain and in the second strip there are two separate chains. The index k denotes the ordering number in the chain. Eigenvalues 4, 2 and 1 (Table 5) are not in any chain.

No.	Strips	Chains	k	λ_k^0	λ_k	No. Iter.	$\lvert\det \Delta(\lambda_k)\rvert$
3			1	$-1.5501950 \pm i4.7123890$	$-1.4814767 \pm i4.1408032$	10	3×10^{-16}
5			2	$-2.3964929 \pm i10.9955743$	$-2.4008970 \pm i10.6823916$	8	1×10^{-14}
6			3	$-2.8494780 \pm i17.2787596$	$-2.8548357 \pm i17.0548496$	7	4×10^{-14}
7			4	$-3.1596329 \pm i23.5619449$	$-3.1641774 \pm i23.3849889$	7	1×10^{-13}
8	1	1	5	$-3.3960217 \pm i29.8451302$	$-3.3997161 \pm i29.6976537$	6	5×10^{-14}
9			6	$-3.5870769 \pm i36.1283155$	$-3.5901051 \pm i36.0012618$	7	9×10^{-14}
10			7	$-3.7474196 \pm i42.4115008$	$-3.7499418 \pm i42.2995197$	8	6×10^{-14}
⋮			⋮	⋮	⋮	⋮	⋮
103			100	$-6.4405441 \pm i62.7477344$	$-6.4405906 \pm i626.7358629$	5	1×10^{-11}
104			1	$-5.0620485 \pm i12.5663706$	$-5.0881172 \pm i11.8310895$	8	2×10^{-16}
105			2	$-6.4483429 \pm i25.1327412$	$-6.4689887 \pm i24.6585330$	6	2×10^{-15}
106			3	$-7.2592731 \pm i37.6991118$	$-7.2733607 \pm i37.3404972$	7	7×10^{-16}
107			4	$-7.8346372 \pm i50.2654825$	$-7.8448423 \pm i49.9737505$	6	2×10^{-15}
108		1	5	$-8.2809243 \pm i62.8318531$	$-8.2887054 \pm i62.5842929$	6	3×10^{-15}
109			6	$-8.6455674 \pm i75.3982237$	$-8.6517331 \pm i75.1822574$	6	1×10^{-15}
110			7	$-8.9538688 \pm i87.9645943$	$-8.9588992 \pm i87.7724695$	5	1×10^{-15}
⋮			⋮	⋮	⋮	⋮	⋮
203	2		100	$-14.2723889 \pm i1256.6370614$	$-14.2724749 \pm i1256.6151424$	5	4×10^{-14}
204			1	$-5.8729787 \pm i18.8495559$	$-5.8979914 \pm i18.2768428$	7	3×10^{-15}
205			2	$-6.8946300 \pm i31.4159265$	$-6.9115690 \pm i31.0085775$	6	2×10^{-15}
206			3	$-7.5675744 \pm i43.9822972$	$-7.5794774 \pm i43.6609828$	6	2×10^{-15}
207			4	$-8.0702033 \pm i56.5486678$	$-8.0790652 \pm i56.2810413$	6	1×10^{-15}
208		2	5	$-8.4715447 \pm i69.1150384$	$-8.4784424 \pm i68.8844726$	6	9×10^{-16}
209			6	$-8.8056528 \pm i81.6814090$	$-8.8112043 \pm i81.4781356$	5	9×10^{-16}
210			7	$-9.0918545 \pm i94.2477796$	$-9.0964388 \pm i94.0655335$	6	3×10^{-15}
⋮			⋮	⋮	⋮	⋮	⋮
303			100	$-14.2823639 \pm i1262.9202467$	$-14.2824493 \pm i1262.8984209$	5	4×10^{-14}

TABLE 7

Eigenvalues for Example 7.2 computed by using the finite dimensional approximations, as in Table 4.

No.	λ_{AVE}^{16}	λ_{SPL}^{16}	λ_{SPC}^{16}
1	+0.7951816	+0.7878990	+0.7879311
2	+0.1044013 ± *i*0.7326004	+0.1095807 ± *i*0.7365189	+0.1094469 ± *i*0.7365426
3	−1.8110187 ± *i*3.6165802	−1.5008392 ± *i*4.1930237	−1.4814767 ± *i*4.1408032
4	−3.6002633 ± *i*4.0183907	−3.2163304 ± *i*5.542026	−3.6413171 ± *i*5.2271860
5	⋮	−2.4559083 ± *i*9.9251154	−2.4008970 ± *i*10.6823916
6		⋮	−2.8548342 ± *i*17.0548497
7			−3.1641490 ± *i*23.3800679
8			−5.0881172 ± *i*11.8310895
9			−6.4335948 ± *i*24.7067641
10			−5.8980228 ± *i*18.2768119
⋮			⋮

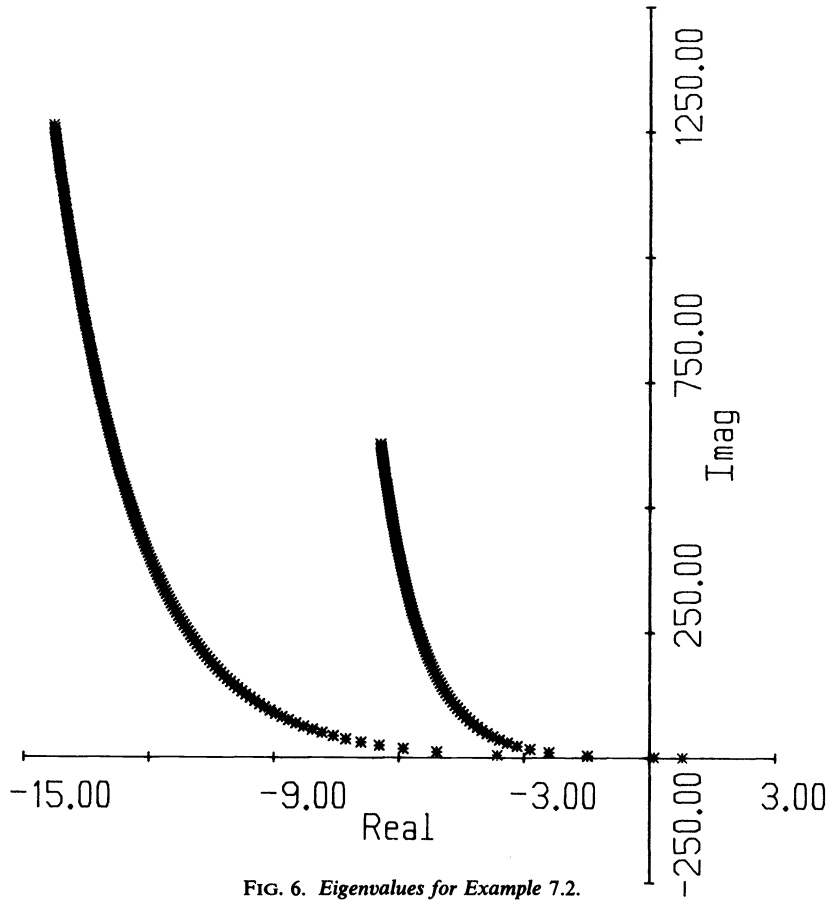


FIG. 6. *Eigenvalues for Example 7.2.*

TABLE 8

Eigenvalues for Example 7.3 computed by Kuhn's method in the rectangle $|\mathrm{Re}\,\lambda| \leqq 7$, $|\mathrm{Im}\,\lambda| \leqq 10$.

No.	λ^0	λ	$\det \Delta(\lambda)$
1	$+0.00000$	0	$-1\times10^{-16}+i1\times10^{-18}$
2	$+0.42600$	$+0.4263028$	$-2\times10^{-17}-i2\times10^{-34}$
3	$+0.51000$	$+0.5099793$	$+7\times10^{-18}+i6\times10^{-43}$
4	$+0.2100\pm i0.90000$	$+0.2119759\pm i0.9009384$	$-2\times10^{-15}+i4\times10^{-16}$
5	$+1.03400$	$+1.0332309$	$4\times10^{-17}+i7\times10^{-48}$
6	$-0.83600\pm i2.49400$	$-0.8362345\pm i2.4947335$	$3\times10^{-13}+i1\times10^{-13}$
7	$-0.50600\pm i3.86200$	$-0.5065419\pm i3.8617791$	$-2\times10^{-11}+i7\times10^{-14}$
8	$-0.41800\pm i2.26400$	-0.4171552 ± 2.2651330	$-6\times10^{-14}-i5\times10^{-14}$
9	$-0.24600\pm i2.30400$	$-0.2466377\pm i2.3028472$	$2\times10^{-14}+i3\times10^{-14}$
10	$-0.85200\pm i5.42000$	$-0.8511295\pm i5.4199043$	$2\times10^{-12}+i5\times10^{-12}$
11	$-0.67600\pm i5.43600$	$-0.6770679\pm i5.4358280$	$-5\times10^{-12}+i3\times10^{-12}$
12	$-0.80400\pm i7.01200$	$-0.8037540\pm i7.0115159$	$-2\times10^{-11}+i1\times10^{-10}$
13	$-0.90400\pm i8.58600$	$-0.9045913\pm i8.5869006$	$-5\times10^{-11}+i7\times10^{-11}$
14	$-1.07800\pm i8.57800$	$-1.0784544\pm i8.5768379$	$9\times10^{-11}+i1\times10^{-11}$
15	$-3.64000\pm i8.68600$	$-3.3645385\pm i8.6856389$	$2\times10^{-4}+i2\times10^{-6}$
16	$-0.98800\pm i10.16200$	$-0.9883799\pm i10.1616900$	$2\times10^{-10}+i7\times10^{-10}$

RFDE with commensurate delays.[1] An extension of this method to the case of noncommensurate delays and to neutral systems is possible. Also, one can adapt the method to systems with distributed delays, if they satisfy conditions of Theorem 3.1. Finally, the method can be modified to compute the eigenvalues associated with the optimal control problem involving RFDE's [17].

Limitations of the method (in its present version) are: (1) the automatic computation of the characteristic equation has been coded only for equations of retarded type with commensurate delays; this limits the possibility of computing automatically the asymptotic eigenvalues only to those equations. (2) Kuhn's algorithm can be applied to any characteristic equation whose left-hand side is analytic on a given bounded domain. Kuhn's method will find all the eigenvalues in a given bounded region, but it does not automatically generate the asymptotic chains. In the event the function has in the region a pole close to a zero (this situation does not arise in differential-difference equations), Kuhn's algorithm may miss the zero. (3) Storage and CPU time factors

[1] A Fortran code is available at request from the authors at Rensselaer Polytechnic Institute.

TABLE 9

Eigenvalues for Example 7.3 computed by the asymptotic formulas. There are 5 chains of eigenvalues, located in 2 curvilinear strips (Fig. 7). The first strip has 3 separate chains and the second strip has 2 separate chains. The eigenvalues are ordered by chains and by their location in the chain. Eigenvalues 1, 2, 3, 4, 5 and 6 (Table 8) are not in any chain.

No.	Strips	Chains	k	λ_k^0	λ_k	No. Iter.	$\lvert\det \Delta(\lambda_k)\rvert$
8			1	$-0.4285239 \pm i2.3561945$	$-0.4171552 \pm i2.2651330$	8	7×10^{-15}
10			2	$-0.8521728 \pm i5.4977871$	$-0.8511295 \pm i5.4199043$	7	3×10^{-13}
14			3	$-1.0781654 \pm i8.6393798$	$-1.0784544 \pm i8.5768379$	7	6×10^{-13}
17		1	4	$-1.2332429 \pm i11.7809725$	$-1.2337650 \pm i11.7285686$	7	2×10^{-12}
18			5	$-1.3514373 \pm i14.9225651$	$-1.3519728 \pm i14.8772520$	7	5×10^{-13}
⋮			⋮	⋮	⋮	⋮	⋮
113			100	$-2.8736985 \pm i313.3738672$	$-2.8737122 \pm i313.3692821$	5	4×10^{-10}
7			1	$-0.5106499 \pm i3.9269908$	$-0.5065419 \pm i3.8617791$	7	7×10^{-14}
12			2	$-0.8045433 \pm i7.0685835$	$-0.8037540 \pm i7.0115159$	8	1×10^{-12}
16			3	$-0.9884056 \pm i10.2101761$	$-0.9883799 \pm i10.1616960$	6	1×10^{-12}
114	1	2	4	$-1.1225376 \pm i13.3517688$	$-1.1227320 \pm i13.3096912$	7	1×10^{-12}
115			5	$-1.2281922 \pm i16.4933614$	$-1.2284508 \pm i16.4561055$	6	8×10^{-11}
⋮			⋮	⋮	⋮	⋮	⋮
210			100	$-2.7029117 \pm i314.9446635$	$-2.7029233 \pm i314.9403725$	5	1×10^{-14}
9			1	$-0.2552371 \pm i2.3561945$	$-0.02466377 \pm i2.3028472$	8	6×10^{-14}
11			2	$-0.6788860 \pm i5.4977871$	$-0.6770679 \pm i5.4358280$	9	1×10^{-13}
13			3	$-0.9048786 \pm i8.6393798$	$-0.9045913 \pm i8.5869006$	8	3×10^{-12}
211		3	4	$-1.0599561 \pm i11.7808725$	$-1.0600723 \pm i11.7359311$	10	5×10^{-12}
212			5	$-1.1781505 \pm i14.9225651$	$-1.1783873 \pm i14.8830593$	10	$3\times10^{-1}2$
⋮			⋮	⋮	⋮	⋮	⋮
307			100	$-2.7004117 \pm i313.3738672$	$-2.7004234 \pm i313.3695586$	7	5×10^{-7}
15			1	$-3.3880721 \pm i9.4247780$	$-3.3645385 \pm i8.6856389$	7	1×10^{-14}
308			2	$-5.0826678 \pm i21.9911486$	$-5.0944418 \pm i21.5263788$	8	6×10^{-13}
309			3	$-5.9866380 \pm i34.5575192$	$-5.9967146 \pm i34.2104673$	6	6×10^{-13}
310		1	4	$-6.6069479 \pm i47.1238898$	$-6.6147488 \pm i46.8433247$	7	5×10^{-13}
311			5	$-7.0797254 \pm i59.6902604$	$-7.0858652 \pm i59.4530110$	6	4×10^{-13}
406	2		100	$-13.1687703 \pm i1253.4954688$	$-13.1688472 \pm i1253.4744568$	5	2×10^{-10}
407			1	$-4.4097233 \pm i15.7079633$	$-4.4177991 \pm i15.1401407$	7	2×10^{-14}
408			2	$-5.5852966 \pm i28.2743339$	$-5.5965808 \pm i27.8780970$	6	6×10^{-14}

TABLE 9 (*cont.*).

No.	Strips	Chains	k	λ_k^0	λ_k	No. Iter.	$\lvert\det \Delta(\lambda_k)\rvert$
409			3	$-6.3207462 \pm i40.8407045$	$-6.3296113 \pm i40.5308718$	6	2×10^{-13}
410			4	$-6.8572742 \pm i53.4070751$	$-6.8641729 \pm i53.1502035$	7	1×10^{-13}
411		2	5	$-7.2798924 \pm i65.9734457$	$-7.2853924 \pm i65.7527463$	6	6×10^{-13}
⋮			⋮	⋮	⋮	⋮	
506			100	$-13.1787703 \pm i1259.7786541$	$-13.1788466 \pm i1259.7577320$	5	1×10^{-11}

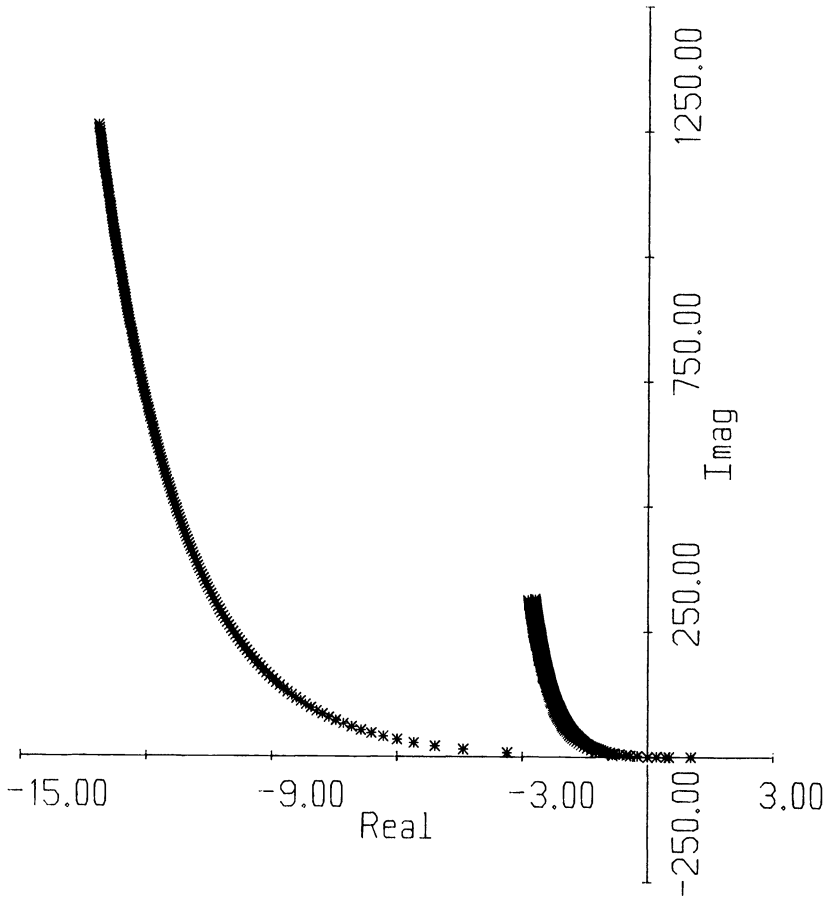


FIG. 7. *Eigenvalues for Example* 7.3.

may limit the dimension of a system. The largest systems tried here were of dimension $n = 10$.

Appendix. In § 6.2 we computed the CPU times for the following three examples. The matrices P of coefficients p_{ij} of terms $\lambda^i\mu^j$ in the polynomial $p(\lambda, \mu)$ are lower triangular matrices of dimension $n+1$. As the number of dimension n increases, the

nonzero elements of P also increase in magnitudes. In these examples, the asymptotic eigenvalues are located in one curvilinear strip.

Example A. We consider the following system of 10 first-order RFDE's with one delay

$$\dot{x}(t) = A_0 x(t) + A_1 x(t-1)$$

where

$$A_0 = 10 \times 10 \text{ identity matrix},$$

$$A_1 = \text{diag}\,(1, 2, 3, 4, 5, 6, 7, 8, 9, 10).$$

The matrix P is a full lower triangular matrix.

Example B. Here we consider the same system as in example A but with system matrix

$$A_1 = \begin{bmatrix} 0 & 1 & & & & & \\ 1 & 0 & 1 & & & & \\ & 1 & 0 & 1 & & & \\ & & 1 & 0 & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & \cdot & \cdot & 1 \\ & & & & & 1 & 0 \end{bmatrix}.$$

The matrix P has all zeros in the even columns and is full of nonzero elements in the odd columns.

Example C. Again we consider the same system as in example A but with system matrix

$$A_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & 0 & & & \cdots & 0 \\ 1 & \vdots & & & & \vdots \\ \vdots & \vdots & & & & \\ 1 & 0 & & & \cdots & 0 \end{bmatrix}.$$

Here the matrix P has nonzero elements in the first three columns and zero elements in the remaining columns for systems with $n = 2, 4, 6, 8$ and 10. The eigenvalues cluster around the origin as n increases. Also, asymptotically the eigenvalues are located in two chains independently of n. Hence, it takes longer to compute the roots by Kuhn's method than by asymptotic formulas as the system dimension n increases.

REFERENCES

[1] E. BAHAR, *Numerical technique to trace the loci of the complex roots of characteristic equations in mathematical physics*, this Journal, 2 (1981), pp. 389-403.

[2] H. T. BANKS AND J. A. BURNS, *Hereditary control problems: numerical methods based on averaging approximations*, SIAM J. Control Optim., 16 (1978), pp. 169-208.

[3] H. T. BANKS AND F. KAPPEL, *Spline approximations for functional differential equations*, J. Differential Equations, 34 (1979), pp. 496-522.

[4] R. BELLMAN AND K. COOKE, *Differential-Difference Equations*, Academic Press, New York, 1963.

[5] S. BRIERLEY, *A method for determining the asymptotic stability range as a function of the delay duration h for linear time delay systems*, Proc. 21st IEEE Conference on Decision and Control, 1982.

[6] M. BUSLOWICZ, *Inversion of polynomial matrices*, Internat. J. Control, 33 (1980), pp. 977-984.

[7] M. CARPENTIER AND DOS SANTOS, *Solution of equations involving analytic functions*, J. Comput. Phys., 45 (1982), pp. 210-220.

[8] G. DAHLQUIST AND A. BJÖRCK, *Numerical Methods*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[9] M. DELFOUR AND S. MITTER, *Controllability, observability, and optimal feedback control of hereditary differential equations*, SIAM J. Control Optim., 10 (1972), pp. 298-328.

[10] D. G. DICKSON, *Asymptotic distribution of zeros of exponential sums*, Publ. Math. Debrecen, 11 (1964), pp. 297-300.

[11] ———, *Zeros of exponential sums*, Proc. Amer. Math. Soc., 16 (1965), pp. 84-89.

[12] J. HALE, *Functional Differential Equations*, Springer-Verlag, New York, 1972.

[13] K. ITO, *On the approximation of eigenvalues associated with functional differential equations*, ICASE Report 82-29, NASA Langley Research Center.

[14] K. ITO AND R. TEGLAS, *Legendre-Tau approximations for functional differential equations*, ICASE Report 172155, NASA Langley Research Center, 1983.

[15] H. KUHN, *A new proof of the fundamental theorem of algebra*, Math. Programming Stud., 1 (1974), pp. 148-158.

[16] I. LASIECKA AND A. MANITIUS, *Differentiability and convergence rates of approximating semigroups for retarded functional differential equations*, Report, Mathematical Sciences, Rensselaer Polytechnic Institute, May 1984.

[17] A. MANITIUS AND R. ROY, *Calcul du spectre du generateur infinitesimal associe aux equation lineaires a retard*, Report CRMA 1002, Centre de Recherche de Math. Appl., Universite de Montreal, Montreal, Canada, 1980.

[18] A. MANITIUS AND H. TRAN, *Computation of closed loop eigenvalues associated with the optimal regulator problem for functional differential equations*, Proc. 22nd IEEE Conference on Decision and Control, 1983, pp. 38-45; IEEE Transactions on Automatic Control, AC-30 (1985), pp. 1245-1248.

[19] W. RUDIN, *Principles of Mathematical Analysis*, McGraw-Hill, New York, 1964.

[20] C. E. SIEWERT AND E. E. BURNISTON, *Exact analytical solution of* $ze^z = a$, J. Math. Anal. Appl., 43 (1973), pp. 626-632.

[21] A. SIKORA AND M. KOCIECKI, *Numerical evaluation of roots of quasipolynomials*, Abstracts Internat. Conference on Functional Differential Systems and Related Topics, Blazejewko, Poland, 1979. Institute of Mathematics, Polish Academy of Sciences, 1979.

[22] E. W. WRIGHT, *The linear difference-differential equation with constant coefficients*, Proc. Roy. Soc. Edinburgh, Sec. A 62 (1949), pp. 387-393.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 3, May 1987
© 1987 Society for Industrial and Applied Mathematics
004

SYMBOLIC GIVENS REDUCTION AND ROW-ORDERING IN LARGE SPARSE LEAST SQUARES PROBLEMS*

GEORGE OSTROUCHOV†

Abstract. In the solution of large sparse least squares problems by Givens factorization, a preliminary symbolic step that determines a good processing order and a data structure for the matrix factor is used. In this paper, it is shown that a processing order equivalent to sequential processing by rows can be as good as any processing order using a single pivot row in each column. A notion of local acceptability in row-ordering is introduced and shown to reduce fill globally. Row-orderings satisfying this notion are essentially equivalent to sequential processing by rows and the nature of intermediate fill produced makes implicit representation of fill possible. This forms the basis for a symbolic Givens reduction algorithm that operates in a fixed data structure.

Key words. sparse least squares, symbolic Givens factorization, row-ordering, column-ordering, bipartite graph

AMS(MOS) subject classifications. 65F50, 05C50

1. Introduction.

A popular method for the solution of large sparse least squares problems is orthogonal factorization by Givens rotations. Givens factorization of an $n \times p$ matrix $\mathbf{X}$ of rank r is of the form

$$\mathbf{X} = \mathbf{Q}\begin{bmatrix}\mathbf{R}\\ \mathbf{0}\end{bmatrix}, \tag{1.1}$$

where $\mathbf{R}$ is upper triangular of order p with $p-r$ zero rows and $\mathbf{Q}$ is a product of orthogonal Givens rotation matrices of order n. The reduction of $\mathbf{X}$ into $\mathbf{R}$ by elimination of nonzeros with Givens rotations can proceed sequentially by rows [6], by columns [3], or in any order that does not fill previously created zeros [5]. It is well known that sparsity of $\mathbf{R}$ varies with the column order of $\mathbf{X}$ and is independent of its row order (see for example [8]). However, the intermediate fill and thus the number of Givens rotations or operations required to produce $\mathbf{R}$ varies with both row order and column order.

Because the upper triangular factor from Cholesky factorization of $\mathbf{X}'\mathbf{X}$ is equivalent to the $\mathbf{R}$ factor, methods for sparsity preservation in Cholesky factorization as well as symbolic Cholesky factorization are applicable here. A number of heuristic ordering strategies (symmetric permutations of $\mathbf{X}'\mathbf{X}$) and symbolic Cholesky factorization are discussed in [7]. George and Heath [6] use these methods for a symbolic phase that computes a good column-ordering and the nonzero structure of $\mathbf{R}$ and then proceed with numerical Givens reduction sequentially by rows.

Other column-ordering strategies are discussed in [3] and [4], where the ordering is determined directly during numerical factorization of $\mathbf{X}$, and in [15], where a symbolic reduction algorithm that operates directly on $\mathbf{X}$ is described. This algorithm can be

* Received by the editors March 20, 1985; accepted for publication (in revised form) February 19, 1986. This research was supported in part 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.

viewed as symbolic reduction by Householder reflections, because of the way it represents intermediate fill in the active portion of the matrix.

Row-ordering can significantly affect the amount of computation required for reduction. A symbolic Cholesky factorization operates on $\mathbf{X}'\mathbf{X}$ and thus it cannot be used to perform any row-ordering of $\mathbf{X}$. In this case, a row-ordering may be obtained from the initial structure of $\mathbf{X}$ and the computed column-ordering. Such row-ordering strategies are discussed in [6], [8] and [10]. A symbolic Householder reduction also cannot be used to perform any row ordering, because its intermediate fill is independent of the row order, so the same strategies apply here. Strategies that make local decisions during Givens reduction are discussed in [3], where these were used directly during numerical reduction. In order to use these strategies in a symbolic phase, a symbolic Givens reduction algorithm needs to be developed.

The main focus of this paper is on row-ordering and the resulting intermediate fill in the active portion of the matrix during Givens reduction. The paper is organized as follows. In § 2 we introduce some graph-theoretic notation and a bipartite graph model of fill that occurs during reduction by Givens rotations. Section 3 starts out with some discussion of column-ordering, but the majority of the section deals with row-ordering. Several classes of row-orderings are defined and discussed. In particular, the concept of local acceptability in row-ordering is defined and locally acceptable single-pivot row-orderings are shown to be globally better than arbitrary single-pivot row-orderings. In § 4, some of the § 3 results are used to develop a symbolic Givens reduction algorithm. The main aspect of this algorithm is that intermediate fill is represented implicitly through the concept of a reduction forest and the symbolic reduction takes place in a fixed data structure. Finally, § 5 contains comments on performance of the symbolic algorithm.

2. Givens reduction on a bipartite graph. It is assumed that the reader is familiar with basic graph-theoretic concepts of symmetric graphs, nodes, edges, adjacent sets, degrees, paths, trees, subtrees, and forests as well as the use of some of these in sparse matrix computations. Formal definitions of these can be found in [7].

DEFINITION 2.1. A bipartite graph (bigraph) associated with an $n \times p$ matrix $\mathbf{X}$ is denoted by $B(\mathbf{X}) = (R(\mathbf{X}), C(\mathbf{X}); E(\mathbf{X}))$. $R(\mathbf{X})$ is a set of n nodes corresponding to rows of $\mathbf{X}$ and labeled r_1 to r_n. $C(\mathbf{X})$ is a set of p nodes corresponding to columns of $\mathbf{X}$ and labeled c_1 to c_p. An edge $(r_i, c_j) \in E(\mathbf{X})$ iff $x_{ij} \neq 0$.

When the associated matrix $\mathbf{X}$ is clear from the context, B, R, C, and E will be used in place of $B(\mathbf{X})$, $R(\mathbf{X})$, $C(\mathbf{X})$, and $E(\mathbf{X})$. Figure 2.1 shows an example of a matrix and its associated bipartite graph.

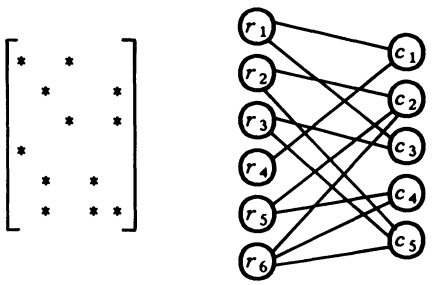


FIG. 2.1. *A matrix and its associated bipartite graph.*

Each Givens transformation involves only two rows. If rows i and j of $\mathbf{X}$ are the two rows involved, and the first element of row j is to be annihilated, then the transformation takes the form

$$(2.1)\qquad \begin{bmatrix} c & s \\ -s & c \end{bmatrix}\begin{bmatrix} x_{i1} & x_{i2} & \cdots & x_{ip} \\ x_{j1} & x_{j2} & \cdots & x_{jp} \end{bmatrix} = \begin{bmatrix} cx_{i1}+sx_{j1} & cx_{i2}+sx_{j2} & \cdots & cx_{ip}+sx_{jp} \\ 0 & -sx_{i2}+cx_{j2} & \cdots & -sx_{ip}+cx_{jp} \end{bmatrix},$$

where $c = x_{i1}/S$, $s = x_{j1}/S$, and $S = (x_{i1}^2 + x_{j1}^2)^{1/2}$. In this transformation, x_{i1} is called the pivot element and x_{j1} is called the target element.

When processing sequentially by rows, as is done in the George-Heath algorithm [6], the pivots used in each rotation are fixed, as is their order, since any other sequence may result in filling previously annihilated positions. This sequence is illustrated in Fig. 2.2(a). An advantage of such processing is that rows can be read one at a time from auxiliary storage, and only the partially formed factor $\mathbf{R}$ needs to be accessed. A more general type of row elimination is reported in [5] and [14], where several $\mathbf{R}$ factors are formed from blocks of rows and then combined into a single factor.

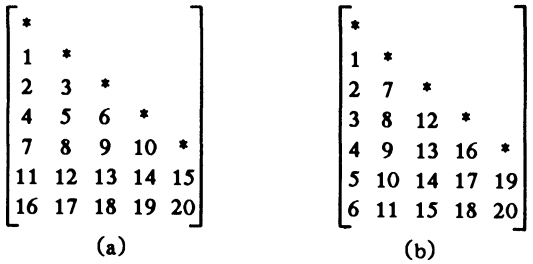


FIG. 2.2. *Elimination order in Givens reduction by rows* (a), *and an equivalent elimination order by columns* (b).

When processing by columns, each rotation can use any eligible row as a pivot, and rows can be processed in any order. During reduction, the completed rows of $\mathbf{R}$ do not need to be accessed; however, the entire active portion of $\mathbf{X}$ needs to be accessed. Processing by columns is equivalent to sequential processing by rows, when a particular elimination order is taken within each column. All eliminations within a column must use the same pivot row, and rows must be processed in the same order within each column. Figure 2.2(b) illustrates this order. Because of the equivalence and the added flexibility described above, the following will discuss the reduction process in terms of processing by columns. The processing of an entire column shall be referred to as a major step of the reduction, and the annihilation of a single nonzero by a rotation will be referred to as a minor step of the reduction.

Assuming no cancellation in (2.1), the nonzero structure of each of the two rows involved in a single Givens transformation becomes the union of their nonzero structures before the transformation less the target element:



Let B be the bigraph associated with $\mathbf{X}$. Suppose a Givens transformation is applied to rows i and j of $\mathbf{X}$, with x_{ik} as the pivot element and x_{jk} as the target element.

Both x_{ik} and x_{jk} must be nonzero. In terms of the bigraph B it is said that a Givens transformation is applied to nodes r_i and r_j of B, with $(r_i, c_k) \in E$ as the pivot edge and $(r_j, c_k) \in E$ as the target edge. A *row node* or a *column node* of a bigraph will usually be referred to as *row* and *column*, respectively, to simplify the presentation as well as emphasize the equivalence. Similarly, the terms *row* (*column*) *node adjacency set* and *row* (*column*) *nonzero structure* are equivalent. If B' is the bigraph after the Givens transformation, then the structure of B' is given by adjacency sets

$$\begin{aligned} &\text{Adj}\,(r_i, B') = \text{Adj}\,(r_i, B) \cup \text{Adj}\,(r_j, B), \\ &\text{Adj}\,(r_j, B') = \text{Adj}\,(r_i, B) \cup \text{Adj}\,(r_j, B) - \{c_k\}, \\ &\text{Adj}(r, B') = \text{Adj}\,(r, B) \quad \text{for all other } r. \end{aligned} \tag{2.2}$$

Consider now a complete major step sequence of Givens transformations with a fixed pivot row. Let B_j be the bigraph associated with $\mathbf{X}$ after the jth minor step. Suppose column c_i is being processed, and $\text{Adj}\,(c_i, B_0) = \{r_{j_1}, r_{j_2}, \cdots, r_{j_k}\}$. Let r_{j_1} be the pivot row and let $j_2, j_3, \cdots, j_k$ be the order in which rows are processed. After completing the $k-1$ minor steps, thus completing the major step, the structure of B_{k-1} is given by:

$$\begin{aligned} &\text{Adj}\,(r_{j_1}, B_{k-1}) = \bigcup_{s=1}^{k} \text{Adj}\,(r_{j_s}, B_0), \\ &\text{Adj}\,(r_{j_\lambda}, B_{k-1}) = \bigcup_{s=1}^{\lambda} \text{Adj}\,(r_{j_s}, B_0) - \{c_i\} \quad \text{for } \lambda = 2, 3, \cdots, k, \\ &\text{Adj}\,(r, B_{k-1}) = \text{Adj}\,(r, B_0) \quad \text{for all other } r. \end{aligned} \tag{2.3}$$

At this point, row r_{j_1} becomes the next row of $\mathbf{R}$, and only the remaining rows require further processing. The above thus gives the means of updating the bigraph for each major step of the reduction when a fixed pivot row is used. As it stands, however, this symbolic Givens reduction requires a dynamic data structure to represent the bigraph, since the adjacency sets may be growing.

3. Ordering. Column-ordering strategies can be divided into two classes. Strategies in one class use only the initial nonzero structure of $\mathbf{X}$ or $\mathbf{X}'\mathbf{X}$ to determine the column order. These include band and envelope methods and dissection methods. Strategies in the other class make local decisions during factorization about which column to choose next. These include the minimum degree algorithm and other variations or generalizations of the Markowitz [16] scheme. Strategies in the latter class are of particular interest here, since they require the partially factored matrix structure.

The most popular algorithm based on local decisions during Cholesky factorization is the minimum degree algorithm [7], [18]. This algorithm can be adapted to symbolic Givens reduction by noting the relationship between the bigraph of $\mathbf{X}$ and the symmetric graph of $\mathbf{X}'\mathbf{X}$:

$$\text{Adj}\,(c, G(\mathbf{X}'\mathbf{X})) = \text{Adj}\,(\text{Adj}\,(c, B(\mathbf{X})), B(\mathbf{X})) - \{c\}.$$

This was characterized in various forms in [2], [10] and [17]. Thus the degree of a column c of $B(\mathbf{X})$ is the number of other columns that can be reached by a path of length exactly two from c.

Usually, in a large sparse matrix, many columns have the same degree. When operating on the graph $G(\mathbf{X}'\mathbf{X})$, there is no information that can be meaningfully used to break the minimum degree ties. The bigraph $B(\mathbf{X})$, however, has information on row nonzero structure, which can be used for tiebreaking. An example of a simple tiebreaking strategy is to take the tied minimum degree column with the least nonzeros (minimum column-count). This will have the effect of processing columns involving fewer rows early. The number of operations needed to produce $\mathbf{R}$ should decrease, but the number of nonzeros in $\mathbf{R}$ will probably not be greatly affected. The minimum column-count can of course be used for column ordering by itself. This and other column-ordering strategies applicable during symbolic Givens reduction are given in [3].

The comparison of two row-orderings is meaningful only if the same column order is used for both. The column order, therefore, is assumed fixed in the discussion of row-orderings in the remainder of this section. Like column-ordering strategies, row-ordering strategies can be divided into two classes. One contains those based on the initial structure of the matrix, and the other contains those based on local decisions during factorization. Row-ordering strategies discussed in [6], [8] and [10] belong to the former class whereas the strategies discussed here belong to the latter class.

DEFINITION 3.1. When processing by columns, a row-ordering is a sequence of ordered triples (s, t, c) each corresponding to a minor step of the reduction and specifying the pivot row, the target row, and the pivot column, respectively, where the pivot column entry is constant within major steps.

There are two important restricted row-orderings that will be considered. The first restriction is using only a single pivot row within each major step of the reduction.

DEFINITION 3.2. A single-pivot row-ordering is a row-ordering in which the pivot row entry of each ordered triple is constant within major steps.

Each major step of a single-pivot row-ordering induces a partial ordering on the rows of $\mathbf{X}$.

DEFINITION 3.3. Let $\{(s, t_1, i), (s, t_2, i), \cdots, (s, t_k, i)\}$ be a subsequence of a single-pivot row-ordering α corresponding to a major step i. The partial ordering induced by this subsequence on the set of n rows is $s < t_1 < t_2 < \cdots < t_k$. Define Θ_α^i to be the ordered set $\{s, t_1, t_2, \cdots, t_k\}$.

A sequence of ordered sets Θ_α^i completely describes a single-pivot row-ordering α. The next restriction requires in addition that the partial orderings of the n rows induced by all major steps agree.

DEFINITION 3.4. A compatible row-ordering is a single-pivot row-ordering in which the partial orderings induced within each major step on rows of the matrix have no conflicts.

A compatible row-ordering corresponds to the elimination order of Fig. 2.2(b). With this type of row-ordering, a linear order of the n rows is produced, in which case processing by columns is equivalent to sequential processing by rows.

Each of the successive definitions puts more restrictions on the row-ordering. Thus, the class of all row-orderings contains the class of single-pivot row-orderings, which contains the class of compatible row-orderings. This section contains results on the latter two classes of row-orderings.

Suppose Givens reduction by columns with a single-pivot row-ordering is performed on an $n \times p$ matrix $\mathbf{X}$. Let $B^i = (R^i, C^i; E^i)$ be the bigraph associated with the active portion of $\mathbf{X}$ after the ith major step. Thus B^0 is associated with the original matrix $\mathbf{X}$. The set Θ_α^i is the ordered set of rows involved in major step i under single-pivot row-ordering α. Let c_i and r_i be the pivot column and pivot row, respec-

tively, of major step i and let k_i be the number of rows in Θ_α^i. The structure of B^i in terms of the structure of B^{i-1} is obtained by applying (2.3)

$$\text{(3.1)} \qquad \text{Adj}\,(\{\Theta_\alpha^i\}_\lambda, B^i) = \bigcup_{s=1}^{\lambda} \text{Adj}\,(\{\Theta_\alpha^i\}_s, B^{i-1}) - \{c_i\} \quad \text{for } \lambda = 2, 3, \cdots, k_i,$$

and by removing the pivot row and pivot column and all associated edges. A direct result of (3.1) is the following theorem characterizing the structure of rows involved in a major step using a single pivot row.

THEOREM 3.5. $\text{Adj}\,(\{\Theta_\alpha^i\}_2, B^i) \subseteq \text{Adj}\,(\{\Theta_\alpha^i\}_3, B^i) \subseteq \cdots \subseteq \text{Adj}\,(\{\Theta_\alpha^i\}_{k_i}, B^i)$.

Now consider two rows r_s and r_t involved in major step i that satisfy $\text{Adj}\,(r_s, B^{i-1}) \subset \text{Adj}\,(r_t, B^{i-1})$, where the inclusion is proper. It is natural to process row r_s before r_t to avoid possible unnecessary local fill-in in this major step.

DEFINITION 3.6. A single-pivot row-ordering α is locally acceptable, if whenever $r_s, r_t \in \Theta_\alpha^i$ and $\text{Adj}\,(r_s, B^{i-1}) \subset \text{Adj}\,(r_t, B^{i-1})$, with proper inclusion, r_s is ordered before r_t in Θ_α^i.

The following lemma is useful in proving a theorem about locally acceptable single-pivot row-orderings. The lemma follows from Theorem 3.5 and states that if a previously processed row is involved in a subsequent major step, all rows that followed it in the previous major step are also involved.

LEMMA 3.7. *Let m be the smallest integer $j > i$ such that $\Theta_\alpha^j \cap \Theta_\alpha^i \neq \varnothing$. Then, $\{\Theta_\alpha^i\}_\lambda \notin \Theta_\alpha^m$, $\lambda = 2, \cdots, u-1$, and $\{\Theta_\alpha^i\}_\lambda \in \Theta_\alpha^m$, $\lambda = u, \cdots, k_i$ for some $2 \leqq u \leqq k_i$.*

THEOREM 3.8. *A locally acceptable single-pivot row-ordering is compatible up to the order of rows with identical nonzero structure.*

Proof. Let α be a locally acceptable single-pivot row-ordering and consider Θ_α^i and Θ_α^m of Lemma 3.7. The rows in Θ_α^i satisfy the relationship of Theorem 3.5 after completion of major step i. So, because α is locally acceptable, rows common to major steps i and m must be in the same order in Θ_α^m as in Θ_α^i except possibly rows with identical nonzero structure. This holds for any major step i, so α must be compatible up to the order of rows with identical nonzero structure. □

If two rows have an identical nonzero structure, their interchange will have no effect on the fill-in created during reduction. So a locally acceptable single-pivot row-ordering is essentially compatible. Local acceptability was motivated by reducing fill locally during reduction; however, it turns out that local acceptability reduces fill globally as is stated in Theorem 3.10 below.

Let $\text{Adj}_\alpha\,(r, B^i)$ be the structure of row r in B^i under row-ordering α. Definition 3.9 gives a means of comparing some row-orderings.

DEFINITION 3.9. A row-ordering β is at least as good as row-ordering α if $\text{Adj}_\beta\,(r, B^i) \subseteq \text{Adj}_\alpha\,(r, B^i)$ for all $r \in R^i$ for $i = 1, 2, \cdots, p$.

The use of this definition is not in finding a good row-ordering, because it cannot compare just any two row-orderings, but in demonstrating global properties of locally acceptable row-orderings without assuming a specific criterion, such as operations or Givens rotations required for the complete reduction.

THEOREM 3.10. *For every single-pivot row-ordering α there exists a locally acceptable single-pivot row-ordering β that is at least as good.*

Proof. We first show that an interchange of two rows to satisfy Definition 3.6 produces an ordering that is at least as good. Let δ be a single-pivot row-ordering that is not locally acceptable. Then in some major step i there exist two rows r_s and r_t with structure as in Definition 3.6 such that $\{\Theta_\delta^i\}_a = r_t$ and $\{\Theta_\delta^i\}_b = r_s$, $a < b$. Let row-ordering γ be identical to row-ordering δ up to and including major step i except for interchange of rows r_s and r_t in major step i. The rows affected by this interchange at completion

of major step i are $\{\Theta_\gamma^i\}_h$ for $h = \max(2, a), \cdots, b-1$. Using (3.1), the following holds for these rows

$$\begin{aligned}\mathrm{Adj}_\gamma(\{\Theta_\gamma^i\}_h, B^i) &= \bigcup_{\substack{m \leqq h \\ m \neq a}} \mathrm{Adj}_\gamma(\{\Theta_\gamma^i\}_m, B^{i-1}) \cup \mathrm{Adj}_\gamma(r_s, B^{i-1}) - \{c_i\} \\ &\subseteq \bigcup_{\substack{m \leqq h \\ m \neq a}} \mathrm{Adj}_\gamma(\{\Theta_\gamma^i\}_m, B^{i-1}) \cup \mathrm{Adj}_\gamma(r_t, B^{i-1}) - \{c_i\} \\ &= \mathrm{Adj}_\delta(\{\Theta_\delta^i\}_h, B^i).\end{aligned}$$

Thus $\mathrm{Adj}_\gamma(r, B^j) \subseteq \mathrm{Adj}_\delta(r, B^j)$ for all $r \in R^j$, $j = 1, \cdots, i$. It follows that $\mathrm{Adj}_\gamma(c_{i+1}, B^i) \subseteq \mathrm{Adj}_\delta(c_{i+1}, B^i)$, so if we let γ take the same row order as δ in major step $i+1$ we have $\mathrm{Adj}_\gamma(r, B^{i+1}) \subseteq \mathrm{Adj}_\delta(r, B^{i+1})$ for all $r \in R^{i+1}$. Constructing the remainder of row-ordering γ in this fashion for major steps $i+2, \cdots, p$ gives a row-ordering that is at least as good as δ. Repeating row interchanges such as the above produces in the end a locally acceptable row-ordering that is at least as good as δ. □

Since rows with identical nonzero structure can be reordered without affecting the nonzero structure, Corollary 3.11 immediately follows.

COROLLARY 3.11. *For every single-pivot row-ordering α there exists a locally acceptable compatible row-ordering β that is at least as good.*

The set of locally acceptable compatible row-orderings thus contains row-orderings that are at least as good as any given single-pivot row-ordering. In addition, these row-orderings are attractive because they allow sequential processing by rows, whereas single pivot row-orderings in general do not allow sequential processing by rows.

Three row-ordering strategies were tested in [3], of which two satisfy the local acceptability criterion when ties are handled properly. The performance of the two was uniformly better than the third strategy on the test matrices used. The number of Givens transformations was used as the criterion of comparison.

Only the two restricted classes of row-orderings, as defined at the outset of this section, were discussed here. Theoretically, only the class of row-orderings allowing variable pivots may contain better row-orderings, than the class of locally acceptable compatible row-orderings. Such row-orderings are not applicable to processing sequentially by rows. Variable pivot row-orderings and an algorithm for Givens reduction using such row-orderings are discussed in [14].

4. Implementation of symbolic Givens reduction. The successive bigraphs of the active portion of the matrix after each major step can be generated using (3.1). Since we are taking unions, the number of edges in the bigraph can grow. This intermediate fill seems to require a dynamic data structure. However, by noting the row structure described by Theorem 3.5, a static representation scheme can be developed when a compatible row-ordering is used. Conveniently, as a result of Corollary 3.11, we are only interested in locally acceptable compatible row-orderings, when the numerical factorization is to be done sequentially by rows as in the George–Heath algorithm. The static representation is possible because fill occurring during symbolic reduction need not be represented explicitly. The implicit representation is achieved through a collection of tree structures described below that will be called the reduction forest. All proofs in this section are left to the reader as they are conceptually simple but notationally difficult. The proofs can be also be found in [17].

Suppose the reduction proceeds by columns and the row order is determined at each major step by some compatible row-ordering strategy. Definition 3.3 gives the partial ordering induced by a major step on the rows of a matrix. Note that a rooted

tree describes a partial ordering of its nodes, since only a single path exists between each pair of nodes.

DEFINITION 4.1. If node x is a descendant of node y, then $x < y$.

Each major step builds more ordering information. While each collection of trees (a forest) represents a partial ordering of its nodes, the converse is not true in general but holds for partial orderings induced by compatible row-ordering strategies.

LEMMA 4.2. *With a compatible row-ordering strategy, the accumulated partial ordering of rows after each major step can be represented by a forest.*

This is the *reduction forest* that is used to implicitly represent fill during reduction. First we describe its construction and then its function.

Let $T^i = (X^i; H^i)$ $i = 0, 1, \cdots, p$ be the sequence of reduction forests that corresponds to the sequence of major steps. Before reduction begins, the null ordering of the rows is represented by a forest of isolated root nodes giving the initial reduction forest T^0. The set of nodes ordered and their order in major step i is given by Θ^i_α. Given a set Θ^i_α the following algorithm updates the reduction forest for a major step i, thus producing T^i from T^{i-1}. Recall that k_i is the number of rows in Θ^i_α.

1. For $m = 1$ to $k_i - 1$, do:
2. cut $\{\Theta^i_\alpha\}_m$ from its parent.
3. make $\{\Theta^i_\alpha\}_m$ a child of $\{\Theta^i_\alpha\}_{m+1}$.

The product of a major step is a single tree in the reduction forest that was constructed by combining isolated nodes and/or trees constructed in previous major steps and whose root is the last row processed. The set Θ^i_α forms a parth in this new tree from node $\{\Theta^i_\alpha\}_1$ to the root $\{\Theta^i_\alpha\}_{k_i}$. Figure 4.1 gives an example of the construction of a reduction forest. It shows the major steps of a reduction process on a 9×6 matrix. The six major steps are defined by $\Theta^1_\alpha, \cdots, \Theta^6_\alpha$ that are used to construct the reduction forests $T^1, \cdots, T^6$. The symbols **R**, **I**, and $*$ all denote nonzeros. The completed rows of the **R** factor are those that contain nonzeros denoted by **R**.

The use of the reduction forest is stated in Theorem 4.4, which forms the basis of the representation scheme for symbolic Givens reduction. The theorem states that the sequence of reduction forests together with B^0, the initial bigraph, can be used to generate the sequence B^i of bigraphs. In essence, no fill is represented explicitly and the rows whose union must be taken to obtain the current row structure of an active row are encoded into the reduction forest. This is motivated by the structure in Theorem 3.5 and expression (3.1). The current structure of an active row is given by the union of structures of all its descendant rows in the reduction forest and itself. The formal statements follow.

DEFINITION 4.3. Fam (r, T) is the set of all nodes in the subtree of T rooted at r, where $T = (X; H)$ is a forest. That is, Fam (r, T) is the "family" of r in T.

THEOREM 4.4. Adj $(r, B^i) =$ Adj (Fam (r, T^i), $B^0) \cap C^i$ for all $r \in R^i$.

For example in Fig. 4.1, the structure of row 7 after completing three major steps is given by the union of the original structures of rows 7, 2, 6, and 1 in the remaining active columns (4, 5 and 6).

The reduction forest thus allows a static representation of symbolic Givens reduction. This representation can be further improved by removing nonzeros that become redundant. Note that both row 2 and row 7 have a nonzero in column 6 of the initial matrix. Since row 7 becomes an ancestor of row 2 after the first major step, this nonzero in row 7 becomes redundant. The structure of row 7 is the same in all subsequent major steps regardless of its initial nonzero in column 6. This suggests that some nonzeros can be deleted from the initial structure without affecting the results of

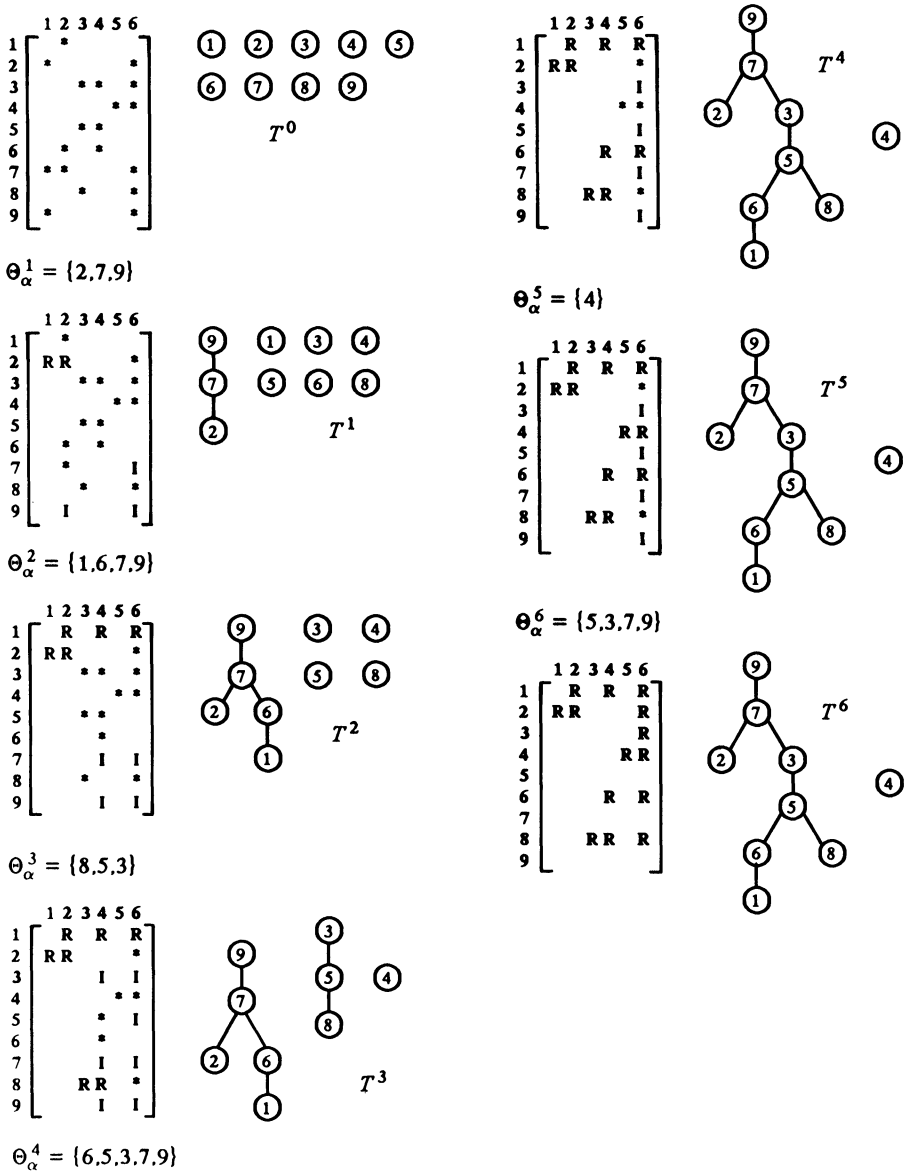


FIG. 4.1. *Symbolic reduction on a* 9×6 *matrix. Sets* Θ_α^i *define the major steps and are used to update the reduction forests shown with each partially reduced matrix. Symbols* *, **R**, *and* **I** *all denote nonzeros, but only the* *'*s are represented in* $\bar{B}^i$. *The* **I**'*s are implied by* T^i *and the* **R**'*s do not contribute to the active portion of each matrix.*

symbolic reduction. In general, whenever row r_1 is a descendant of row r_2 in T^i and both have nonzeros in columns Ω of **X**, the Ω nonzeros in row r_2 are redundant when computing the current structure of r_2 or any of its ancestors. The reduction forest can be used to define a sequence of bigraphs $\bar{B}^i = (R^i, C^i; \bar{E}_i)$ that has all such redundant edges removed. This sequence is generated by correcting each row for nonzeros of its descendants. This can be accomplished by computing

$$\text{Adj}(r, \bar{B}^i) = [\text{Adj}(r, \bar{B}^{i-1}) - \text{Adj}(\text{Fam}(s, T^i), B^0)] \cap C^i, \tag{4.1}$$

where $\bar{B}^0 = B^0$, whenever s becomes a child of r during the construction of T^i from T^{i-1}.

This new representation, containing only nonredundant nonzeros, has some very useful properties. Whenever a row has a nonzero in the $\bar{B}^i$ representation, then all its ancestor rows have that nonzero and all its descendant rows do not have that nonzero; moreover neither its ancestor nor descendants have that nonzero in the $\bar{B}^i$ representation. The reduction forest becomes more informative about the current structure of the matrix as well as its representation. In the example of Fig. 4.1, only nonzeros denoted by * are represented in $\bar{B}^i$; nonzeros denoted by **I** are implied by $\bar{B}^i$ through T^i, and nonzeros denoted by **R** no longer contribute to the active portion of the matrix. The next corollary gives the equivalent of Theorem 4.4 in terms of $\bar{B}^i$.

COROLLARY 4.5. $\text{Adj}(r, B^i) = \text{Adj}(\text{Fam}(r, T^i), \bar{B}^i) \cap C^i$ for all $r \in R^i$.

By Theorem 4.4 and Corollary 4.5, (4.1) can be written entirely in terms of $\bar{B}^i$ as

$$\text{Adj}(r, \bar{B}^i) = [\text{Adj}(r, \bar{B}^{i-1}) - \text{Adj}(\text{Fam}(s, T^i), \bar{B}^{i-1})] \cap C^i. \tag{4.2}$$

Next, a supernode concept is introduced to take care of pivot rows, which are no longer active but may contribute to the structure of an active row. Presently, all rows of the matrix are represented in the reduction forest. Pivot rows from completed major steps are not involved in any subsequent major steps except possibly by their contribution to the structure of their active ancestors through the reduction forest. By combining the pivot row and its active parent into a supernode after each major step, we obtain a reduction forest of supernodes that represent only active rows of the matrix. Let Sup (r) be the set of nodes in the supernode r. The new reduction forest of supernodes is updated for a major step i by the following algorithm.

1. For $m = 1$ to $k_i - 1$, do:
2. cut $\{\Theta_\alpha^i\}_m$ from its parent.
3. if $m = 1$, then merge Sup $(\{\Theta_\alpha^i\}_1)$ into supernode $\{\Theta_\alpha^i\}_2$.
4. else make $\{\Theta_\alpha^i\}_m$ a child of $\{\Theta_\alpha^i\}_{m+1}$.

The addition of the supernode concept into the reduction forest introduces only minor changes in Theorem 4.4, Corollary 4.5, and expressions (4.1) and (4.2). In all of these we now deal with supernodes rather than nodes and the adjacency structure of a supernode is given by the union of the structures of all its component nodes. In addition, (4.2) is applied not only when s becomes a child of r, but also before Sup (s) is merged into r. This last change ensures that each nonzero of a supernode is represented only once in $\bar{B}^i$.

Figure 4.2 gives the equivalent sequence of supernode reduction forests to the sequence given in Fig. 4.1. Supernodes have their representative (active) node listed

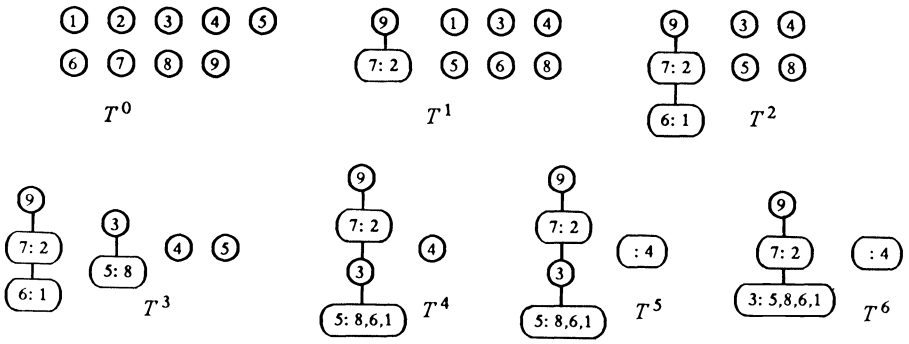


FIG. 4.2. *Reduction forest sequence of Fig.* 4.1 *after introducing the supernode concept.*

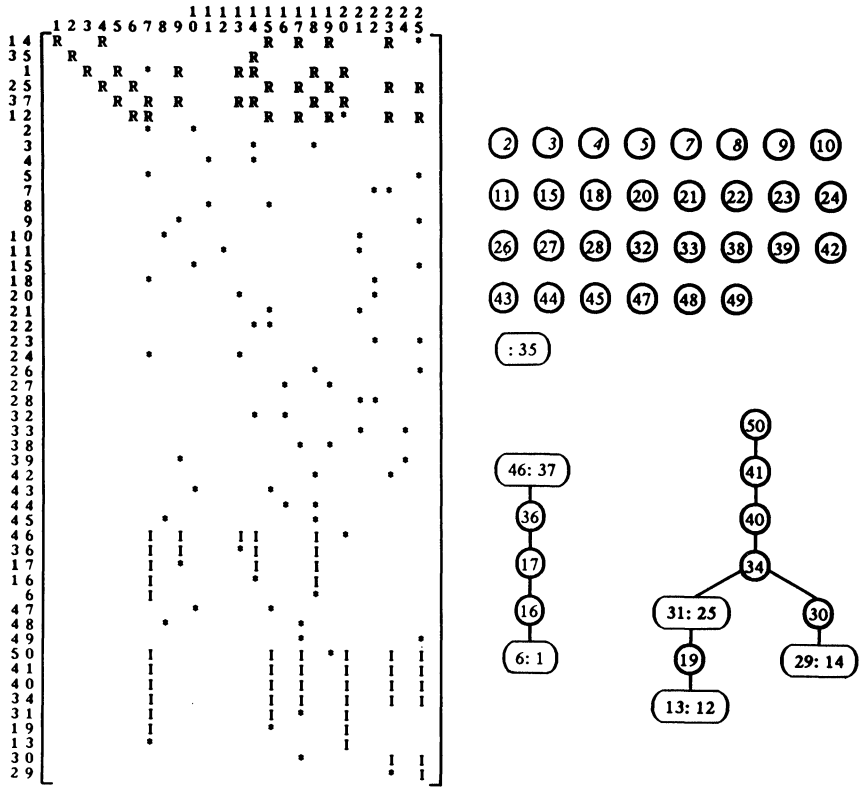


FIG. 4.3a. *A* 50×25 *matrix after* 6 *major steps with the associated reduction forest* T^6.

first followed by the remaining component nodes. A larger example is given in Figs. 4.3a and 4.3b. The figures show one major step of symbolic reduction on a 50×25 matrix with two random entries per row. Figure 4.3a gives the matrix after 6 major steps and the reduction forest T^6. Major step 7 is defined by $\Theta_\alpha^7 = \{2, 5, 6, 13, 16, 17, 18, 19, 24, 31, 34, 36, 40, 41, 46, 50\}$, which is used in constructing T^7 and the matrix in Fig. 4.3b. The rows of the active portion of each matrix are ordered by trees of the corresponding reduction forest and in preorder within trees to illustrate the emerging patterns in the row structure. The symbols *, **R**, and **I** all represent nonzeros and their special meaning is the same as in Fig. 4.1. In what follows, any references to the reduction forest will mean the modified reduction forest of supernodes associated with the active portion of the matrix.

The remaining component of the symbolic Givens reduction algorithm is the computation of Θ_α^i from $\bar{B}^{i-1}$ and the T^{i-1} reduction forest. The set Θ_α^i is the ordered set Adj (c_i, B^{i-1}), where c_i is the pivot column of major step i. If a row-oriented data structure is used, one must essentially check each row to see if it contains a nonzero in the pivot column. The sequence $\bar{B}^{i-1}$ has been constructed so that the information in T^{i-1} can be used for such a search. Whenever $c_i \in$ Adj (Sup (r), $\bar{B}^{i-1}$), then r and all its ancestors in T^{i-1} have a nonzero in the pivot column and all its descendants have a zero in the pivot column. Thus each time we find a supernode with a pivot column nonzero in the $\bar{B}^{i-1}$ representation, we obtain pivot column information on all its ancestors and descendants in T^{i-1}.

Let Θ^i be the complete set of supernodes with a pivot column nonzero in the $\bar{B}^{i-1}$ representation. For example, in the reduction sequence of Fig. 4.1 (using the reduction

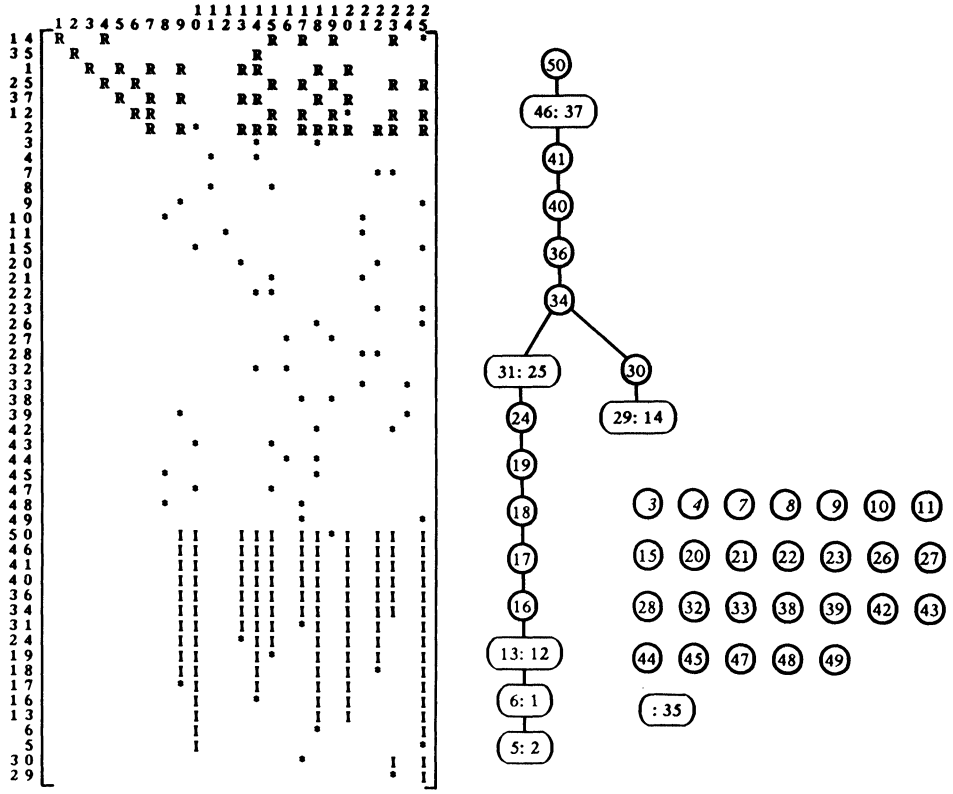


FIG. 4.3b. *The matrix of Fig.* 4.3a *after major step* 7 *with the associated reduction forest* T^7.

forest of Fig. 4.2), $\Theta^1 = \{2, 7, 9\}$, $\Theta^2 = \{1, 6, 7\}$, $\Theta^3 = \{3, 5, 8\}$, $\Theta^4 = \{5, 6\}$, $\Theta^5 = \{4\}$, and $\Theta^6 = \{5, 7\}$. Another example is in Fig. 4.3, where $\Theta^7 = \{2, 5, 6, 13, 18, 24\}$. In each case, the set Θ^i_α is given by ordering the set $\{\Theta^i \cup \text{Ancestors}(\Theta^i, T^{i-1})\}$ according to a given row-ordering α. Algorithm 4.6 computes the set Θ^i given the pivot column c_i of major step i, the reduction forest T^{i-1}, and bigraph $\bar{B}^{i-1}$. The algorithm uses an array I, which is initially $\neq c_i$, to mark rows that are eliminated from the search each time a row is found. The set Θ^i is initially empty.

ALGORITHM 4.6.

1. For each $r \in R^{i-1}$ such that $I_r \neq c_i$ do:
2. If $c_i \in \text{Adj}(\text{Sup}(r), \bar{B}^{i-1})$ then:
3. $\Theta^i \leftarrow \Theta^i \cup \{r\}$.
4. $I_s \leftarrow c_i$ $\forall s$ such that $s \in \text{Descendant}(r, T^{i-1})$.
5. While $I_r \neq c_i$ do:
6. $I_r \leftarrow c_i$.
7. $r \leftarrow \text{Parent}(r, T^{i-1})$.

Because a compatible row-ordering is used, the first row ordered (the pivot row) must come from the set Θ^i, since all other rows in Θ^i_α are ancestors of rows in Θ^i. After picking the pivot row, we can replace it in Θ^i by its parent, unless the parent is also an ancestor of another row already in Θ^i. This can be checked by examining the array I as it appears after the execution of Algorithm 4.6. Note that for each node in Θ^i the path to the root of its tree is marked by c_i in the array I. Thus it is sufficient to check if the parent in question is at a junction of these paths; that is, check if any of

its other children are marked by c_i in I. After this update, Θ^i again contains the set of rows eligible to be chosen next under the compatibility row-ordering restriction. This process of choosing a row and updating Θ^i is accomplished by Algorithm 4.7. This algorithm uses the array I as it appears after execution of Algorithm 4.6.

ALGORITHM 4.7.

1. Choose r from Θ^i.
2. $\Theta^i \leftarrow \Theta^i - \{r\}$.
3. if r has a parent s in T^{i-1}, then:
4. $\quad I_r \leftarrow 0$.
5. $\quad$ If $I_k \neq c_i$ $\forall$ children k of s in T^{i-1}, then $\Theta^i \leftarrow \Theta^i \cup \{s\}$.

By supplying a row-ordering strategy α for step 1, this algorithm produces the set Θ^i_α one element at a time. At this point we can present the complete symbolic Givens reduction algorithm. Initially $\bar{B}^0 = (R^0, C^0; E^0)$ represents the original matrix to be reduced, $T^0 = (R^0; \varnothing)$ is a forest of isolated nodes corresponding to rows of the matrix, and the array I is zero.

ALGORITHM 4.8. Symbolic Givens Reduction.

1. For $i = 1$ to p, do:
2. $\quad$ choose c_i from C^{i-1}.
3. $\quad C^i \leftarrow C^{i-1} - \{c_i\}$.
4. $\quad \Phi \leftarrow \varnothing$.
5. $\quad$ obtain Θ^i with Algorithm 4.6.
6. $\quad$ if $\Theta^i \neq \varnothing$, then:
7. $\quad\quad$ obtain pivot row r_i with Algorithm 4.7.
8. $\quad\quad$ cut r_i from its parent.
9. $\quad\quad \Phi \leftarrow \text{Adj}\,(\text{Fam}\,(r_i, T^{i-1}), \bar{B}^{i-1})$.
10. $\quad\quad \text{Adj}\,(r_i, \bar{B}^i) \leftarrow \text{Adj}\,(r_i, \bar{B}^{i-1}) - \{c_i\}$.
11. $\quad\quad R^i \leftarrow R^{i-1} - \{r_i\}$.
12. $\quad\quad t \leftarrow r_i$.
13. $\quad\quad$ while $\Theta^i \neq \varnothing$, do:
14. $\quad\quad\quad$ obtain next row r with Algorithm 4.7.
15. $\quad\quad\quad$ cut r from its parent.
16. $\quad\quad\quad \text{Adj}\,(s, \bar{B}^i) \leftarrow \text{Adj}\,(s, \bar{B}^{i-1}) - \Phi$ $\forall s \in \text{Sup}\,(r)$.
17. $\quad\quad\quad \Phi \leftarrow \Phi \cup \text{Adj}\,(\text{Fam}\,(r, T^{i-1}), \bar{B}^i)$.
18. $\quad\quad\quad$ if $t = r_i$, then merge $\text{Sup}\,(t)$ into supernode r.
19. $\quad\quad\quad$ else make t a child of r.
20. $\quad\quad\quad t \leftarrow r$.
21. $\quad$ store Φ as the structure of next row of $\mathbf{R}$ factor.

The set Φ represents the structure of the current pivot row; thus at the end of a major step it contains the structure of the next row of $\mathbf{R}$. Row-ordering and column-ordering strategies have been left undefined in this algorithm (step 1 of Algorithm 4.7 and step 2 of Algorithm 4.8, respectively), as there are many possible candidates.

By restricting the choice of the next row only to the set Θ^i, in step 1 of Algorithm 4.7, any row-ordering strategy is forced to be compatible. The resulting row order that is to be used in sequential numerical factorization by rows is determined in two parts. First, the pivot rows are ordered in the order they leave the set R. Then, the remaining rows are ordered after the pivot rows in the order determined by the final reduction forest.

The data structures used in this algorithm are of the standard variety. The bigraph $\bar{B}^0$ is represented in row adjacency form by one array containing the column lists for each row and a pointer array giving the beginning of each list. The removal of column entries for a given row that takes place in steps 10 and 16 of Algorithm 4.8 is done by repacking the list for that row and padding with zeros. The reduction forest T^i is represented in a triply linked tree form (see for example [13]) to facilitate fast searching. The supernodes are represented by a linked list anchored at each representative node. The sets Φ, R^i and C^i are indicator arrays of length p, n and p, respectively, and the set Θ^i is a stack.

The minimum degree column-ordering strategy for symbolic Givens reduction is discussed in § 3. Note that after each major step, only the degree of column nodes in Φ needs to be updated. The degree of a column node c after major step i is $|\text{Adj}(\text{Adj}, (c, B^i), B^i)| - 1$. This is simply the number of nonzeros in the union of all rows that have a nonzero in column c less one. Since the root row of each tree in the reduction forest contains all nonzeros of rows in its tree, it is sufficient to find one row containing c in the tree, identify its root, and add the structure of the root to the union. Also, any $c \in \Phi$ contains Φ in its union and Φ is the root row structure of the tree constructed in the last major step. This eliminates all rows in this tree, whose roots is $\{\Theta^i_\alpha\}_{k_i-1}$, from the search for rows with c. These ideas are used in Algorithm 4.9 to update the degree d for a column node $c \in \Phi$. The array I is initially $\neq c$ and $\Psi = \Theta^i_\alpha$.

ALGORITHM 4.9. Column Degree Computation.

1. For all $r \in R^i - \text{Fam}(\{\Theta^i_\alpha\}_{k_i-1}, T^i)$ with $I_r \neq c$ and $c \in \text{Adj}(\text{Sup}(r), \bar{B}^i)$, do:
2. find root t of tree containing r.
3. $\Psi \leftarrow \Psi \cup \text{Adj}(\text{Fam}(t, T^i), \bar{B}^i)$.
4. $I_s \leftarrow c \ \forall s \in \text{Fam}(t, T^i)$.
5. $d \leftarrow |\Psi|$.

Some column-ordering strategies, as well as the minimum degree tiebreaking strategy, require the number of nonzeros in a given column. This information is not available without some computation, since a row-oriented data structure is used. Again, only columns in Φ need to be updated. Algorithm 4.10 obtains d, the number of nonzeros in column c of B^i. The array I is initially $\neq c$ and $d = 0$.

ALGORITHM 4.10. Column-Count Computation.

1. For each $r \in R^i$ such that $I_r \neq c$ and $c \in \text{Adj}(\text{Sup}(r), \bar{B}^i)$, do:
2. $m \leftarrow r$.
3. while $I_m \neq c$, do:
4. $I_m \leftarrow c$.
5. $d \leftarrow d + 1$.
6. replace m by its parent in T^i if it exists.
7. $I_s \leftarrow c \ \forall s \in \text{Fam}(r, T^i)$.

A more efficient implementation of a column-count update would be directly in steps 16 and 17 of Algorithm 4.8, where the new structure of a row is being computed. Computation of the old structure of a row can be done at the same time with little additional computation. The difference between these two structures would give the necessary adjustments to column counts.

Sometimes a dense or some other less obvious row will cause severe fill-in in **R**. It is possible to leave out such rows from the initial factorization and use them to update only the solution [1], [11]. The question of which rows should be left out can be partially answered by looking at the structure of $\bar{B}^i$. Any row that has many nonzeros

represented in $\bar{B}^i$ is very different from its descendants in T^i and may be the cause of much fill in **R**. The importance of this difference, of course, depends on how many descendants it has. A criterion, such as $|\text{Adj}\,(r, \bar{B}^i)|^\gamma / |\text{Fam}\,(r, T^i) - \{r\}|^\delta$ for some $\gamma > 0$ and $\delta > 0$, could be used to measure a row's "difference." Rows with high values of this criterion should be left out from the factorization and used to update the solution.

5. Algorithm performance and summary. The fill prediction by the symbolic Givens reduction is based on expression (2.2). This type of fill prediction is called the local Givens rule in [2], where it is shown that this rule is exact when **X** has the strong Hall property. If **X** does not have this property and is of full structural rank (structural rank is maximal rank of a given nonzero structure), it can be reordered into a block upper triangular form, where each block has the strong Hall property, and each block can be processed separately. On the other hand, if such **X** is processed directly with the local Givens rule, the fill predicted is no more (less, if **X** happens to be in a block upper triangular form) than what a symbolic Cholesky factorization of **X**'**X** would predict [2]. It seems that when **X** has less than full structural rank, it can be reordered into a similar block upper triangular form, where some blocks are zero and the nonzero blocks have the strong Hall property. Same observation regarding fill prediction as in the full structural rank case applies here.

A special case of structural rank deficiency occurs when $n < p$. Although it is implicitly assumed in (1.1) that $n \geqq p$, the symbolic Givens reduction is applicable to underdetermined systems. The predicted structure of **R** by symbolic Givens reduction is in this case $n \times p$ upper trapezoidal. In highly underdetermined systems with $n \ll p$, the predicted fill is much less than that predicted by symbolic Cholesky factorization of **X**'**X**, which predicts **R** as upper triangular of order p.

The time complexity of the algorithm depends on the row and column ordering strategies used. To avoid this dependence, we assume in the discussion below that the matrix is already ordered, so that any ordering or choice operations take $O(1)$ time. Only expressions for worst case complexity bounds are given.

Algorithm 4.6 operates on $\bar{B}^{i-1}$ and T^{i-1} and finds all r such that $c_i \in \text{Adj}\,(\text{Sup}\,(r), \bar{B}^{i-1})$. Every row in R^{i-1} is visited exactly once, but only some rows are searched for c_i. Its worst case occurs when T^{i-1} is a collection of isolated root nodes and thus contains no information. In this case all rows are searched and the time complexity is $O(|\bar{E}^{i-1}|)$, since every edge in $\bar{B}^{i-1}$ is visited once. As T^{i-1} gains more information, every row with c_i found eliminates its ancestors and descendants from the search and complexity decreases. At the same time $|\bar{E}^{i-1}|$ is decreasing as reduction proceeds.

Algorithm 4.7 chooses a row from Θ^i and then visits all siblings of that row. A given row can at most have as many children as there are nonzeros in it. This is because a row can gain a new child without losing an old child only if it has the current pivot nonzero represented in $\bar{B}^i$. Thus the number of original nonzeros in a row is an upper bound on the number of children it can have as well as the time complexity for adding it into Θ^i.

The steps that dominate the time complexity of the complete symbolic reduction algorithm are 5, 9, 10, 16, and 17. Step 5 is Algorithm 4.6, which has worst case complexity of $O(|\bar{E}^{i-1}|)$. Recall that the result of a single major step is a single tree in the reduction forest that consists of all rows involved in that major step as well as all their descendants. Steps 9, 10, 16, and 17 involve visiting each edge of the rows in this tree that is represented in $\bar{B}^{i-1}$. The number of these edges is at most $|\bar{E}^{i-1}|$, so a bound on the worst case time complexity of the complete algorithm is $O(\sum_{i=1}^{p} |\bar{E}^{i-1}|)$,

where $\bar{E}^0 = E^0$, $|\bar{E}^i| \leqq |\bar{E}^{i-1}|$, and $|\bar{E}^{p-1}| \leqq n - p$. The rate of decrease in $|\bar{E}^{i-1}|$ is very problem dependent.

It is expected that the algorithm will on the average perform much better than the worst case indicated above. When performed with a fixed row and column order, it should be competitive with the Cholesky symbolic factorization algorithm, because the structure of $\mathbf{X}'\mathbf{X}$ need not be formed. However, with an ordering strategy such as minimum degree, the Givens algorithm forms the structure of $\mathbf{X}'\mathbf{X}$ implicitly by computing column degrees and is expected to be slower. For certain classes of problems, the possible increase in symbolic factorization time should be offset by better orderings for the numerical stage. Better column-orderings may be possible, since some heuristic column-ordering strategies, not available with the Cholesky symbolic algorithm, become available. Also, heuristic row-ordering strategies based on local decisions during factorization can now be applied at the symbolic stage. The criterion for selection of rows to be left out from the initial factorization and then used to update only the solution may also prove to be effective for some problems.

The symbolic Givens reduction algorithm was programmed and tested on several sparse problems with a number of row-ordering and column-ordering strategies. It is a young algorithm and it will, no doubt, undergo some changes to improve its efficiency as more experience is gained with it. At this point no timing comparisons have been made with other algorithms. Only nonzero and operation counts for a number of ordering strategies have been compared on a few test problems. These are reported in [17].

To conclude, in this article it was shown that sequential processing by rows (equivalent to compatible row-ordering in processing by columns) can be as effective as any processing order using a single pivot within each column. Also, the notion of local acceptability in row-ordering was introduced and shown to reduce intermediate fill globally while producing essentially compatible row-orderings. The nature of intermediate fill with compatible row-orderings makes an implicit representation of fill possible. The resulting symbolic Givens reduction algorithm is an alternative to the symbolic Cholesky factorization algorithm and represents a more direct approach. It stimulates the actual Givens reduction process and thus it can simulate the effect of a wide range of row and column ordering strategies, some previously not available for the symbolic stage.

Acknowledgment. I wish to thank Professor William J. Kennedy at Iowa State University, who suggested this area of research as a topic for my dissertation and provided invaluable advice and encouragement during the research.

REFERENCES

[1] Å. BJÖRCK, *A general updating algorithm for constrained linear least squares problems*, this Journal, 5 (1984), pp. 394–402.

[2] THOMAS F. COLEMAN, ANDERS EDENBRANDT AND JOHN R. GILBERT, *Predicting fill for sparse orthogonal factorization*, Tech. Report 83-578, Dept. Comput. Sci., Cornell University, Ithaca, NY, 1983.

[3] IAIN S. DUFF, *Pivot selection and row ordering in Givens reduction on sparse matrices*, Computing, 13 (1974), pp. 239–248.

[4] IAIN S. DUFF AND J. K. REID, *A comparison of some methods for the solution of sparse over-determined systems of linear equations*, J. Inst. Math. Appl., 17 (1976), pp. 267–280.

[5] W. MORVEN GENTLEMAN, *Row elimination for solving sparse linear systems and least squares problems*, Proc. Dundee Conference on Numerical Analysis, G. A. Watson, ed., Lecture Notes in Mathematics 506, Springer-Verlag, 1975, pp. 122–133.

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

[7] J. ALAN GEORGE AND JOSEPH W. H. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[8] J. ALAN GEORGE AND ESMOND NG, *On row and column orderings for sparse least squares problems*, SIAM J. Numer. Anal., 20 (1983), pp. 326–344.

[9] J. ALAN GEORGE, MICHAEL T. HEATH AND ESMOND NG, *A comparison of some methods for solving sparse linear least-squares problems*, this Journal, 4 (1983), pp. 177–187.

[10] J. ALAN GEORGE, JOSEPH LIU AND ESMOND NG, *Row-ordering schemes for sparse Givens transformations. I. Bipartite graph model*, Linear Algebra Appl., 61, 1984, pp. 55–81.

[11] MICHAEL T. HEATH, *Some extensions of an algorithm for spare linear least squares problems*, this Journal, 3 (1982), pp. 223–237.

[12] ———, *Numerical methods for large sparse linear least squares problems*, this Journal, 5 (1984), pp. 497–513.

[13] D. E. KNUTH, *The Art of Computer Programming, Vol. 1: Algorithms and Data Structures*, Addison-Wesley, Reading, MA, 1968.

[14] JOSEPH W. H. LIU, *On general row merging schemes for sparse Givens transformations*, this Journal, 7 (1986), pp. 1190–1211.

[15] P. E. MANNEBACK, *A direct approach for column ordering and symbolic factorisation in large sparse least squares problems*, Report No. 83/11, Department of Mathematics, Facultes Universitaires De Namur, Belgium, 1983.

[16] HARRY M. MARKOWITZ, *The elimination form of the inverse and its application to linear programming*, Management Sci., 3 (1957), pp. 255–269.

[17] GEORGE OSTROUCHOV, *Symbolic Givens reduction in large sparse least squares problems*, Tech. Report ORNL-6102, Oak Ridge National Laboratory, Oak Ridge, TN, 1984.

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

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 3, May 1987
© 1987 Society for Industrial and Applied Mathematics
005

ON THE APPROXIMATION OF DERIVATIVES OF SINGULARLY PERTURBED BOUNDARY VALUE PROBLEMS*

V. ERVIN† AND W. LAYTON‡

Abstract. We consider the question of approximating *derivatives* of solutions to singularly perturbed two-point boundary value problems:

$$\text{(E)} \qquad \begin{aligned} -\varepsilon u'' + f(x)u' + g(x)u &= q(x), \qquad 0 < x < 1, \\ u(0) &= \alpha, \qquad u(1) = \beta. \end{aligned}$$

This presents especially difficult problems in the layer. We show rigorously that outside the layers difference quotients of a computed approximation to u give optimal approximations to u', uniformly in ε. By looking at the stretched variables, we obtain a formula for computing u' in the layers accurately.

The success of this formula is due in part to the smoothness of the solution of (E) outside the layer and to the good accuracy of the basic method there. It is shown that the relative error in this formula is $O(h+\varepsilon)$.

For concreteness, we take a numerical method previously proposed by the authors as the basic scheme for approximating u. The results are indicative, however, of using any difference method of positive type on (E).

Key words. convection-diffusion, boundary value problem, finite difference method

AMS(MOS) subject classifications. Primary 65N15; secondary 76R99

1. Introduction.

Singularly perturbed elliptic boundary value problems arise in many important physical phenomena including high Reynolds number flow in fluid dynamics and problems in elasticity where edge effects are important. In many of these problems, computing the actual solution is only a vehicle to obtaining a (usually more difficult) functional of the solution. In the two previously mentioned examples, normal derivatives of the solution are required on the boundaries to compute the skin friction coefficients and stress intensity factors, respectively.

In this paper, we consider the question of approximating derivatives of solutions of singularly perturbed B.V.P.'s. We give two methods of achieving this goal. The first is the standard method of taking differences of a computed solution. It is shown that this gives an optimal approximation, uniformly in ε, outside the layers. An "exponentially fitted" approximation to derivatives is given inside the layer. Numerical experiments are given that illustrate the two approaches.

We analyze these approaches for the two-point B.V.P.

$$(1.1) \qquad \begin{aligned} -\varepsilon u'' + f(x)u' + g(x)u &= q(x), \qquad 0 < x < 1, \\ u(0) &= \alpha, \qquad u(1) = \beta, \end{aligned}$$

although the methods given can be extended to many 2-D problems, as discussed in Ervin and Layton [4].

* Received by the editors June 24, 1985, and in revised form November 15, 1985.

† Department of Mathematics, Spelman College and School of Mathematics, Georgia Institute of Technology, Atlanta, Georgia 30332.

‡ Department of Mathematics, Carnegie-Mellon University and School of Mathematics, Georgia Institute of Technology, Atlanta, Georgia 30332. The work of this author was partially supported by the Air Force Office of Scientific Research under grant 83-0101.

The basic method we begin with is a scheme of positive type with errors $O(h^2+\varepsilon h)$. Let $h=1/N$, $U_j \cong u(x_j)$, $x_j = jh$, $j=0,1,\cdots,N$. U_j is calculated via the equations

$$
\begin{aligned}
L_0^h U_j \equiv & -(\varepsilon+\delta_j f_j^2)\frac{U_{j+1}-2U_j+U_{j-1}}{h^2} \\
& +(f_j-\delta_j f_j f_j'-\delta_j f_j g_j)\frac{U_{j+1}-U_{j-1}}{2h} \\
& +(g_j-\delta_j f_j g_j')U_j = q_j-\delta_j f_j q_j', \\
& U_0=\alpha, \qquad U_N=\beta,
\end{aligned} \tag{1.2}
$$

where $f_n = f(x_n)$, $f_n' = f'(x_n)$, etc. $\delta_n = O(h)$ is chosen to ensure that the scheme is of positive type:

$$
\frac{h|f_n||1-\delta_n(f_n'+g_n)|}{2(\varepsilon+\delta_n f_n^2)} \leqq 1. \tag{1.3}
$$

Henceforth, we shall always assume that g, f, q are smooth functions, ε is very small, $\varepsilon \ll O(h)$, $g(x)>0$ and h is sufficiently small w.r.t. f, g that $g(x)-\delta f(x)g'(x)>0$. Note that (1.2) arises by applying a standard central difference code to a problem with perturbed coefficients:

$$
\begin{aligned}
& L_0 w \equiv -\varepsilon_0(x)w''+f_0(x)w'+g_0(x)w = q_0(x), \\
& w(0)=\alpha, \qquad w(1)=\beta,
\end{aligned} \tag{1.4}
$$

where $\varepsilon_0(x)=\varepsilon+\delta f^2$, $f_0(x)=f-\delta f f'-\delta f g$, $g_0(x)=g-\delta f g'$ and $q_0(x)=q-\delta f q'$.

We calculate an approximation to $u'(x_n)$ outside the layer as follows. For $0 \leqq x_n < 1$

$$
u'(x_n) \cong \frac{U_{n+1}-U_{n-1}}{2h}, \qquad 0 \leqq x_n < 1, \tag{1.5}
$$

where U_{-1} is obtained through the difference equation:

$$
U_{-1} = [-\varepsilon_0(x_0)-hf_0(x_0)]^{-1}\left[h^2 q_0(x_0)-(h^2 g_0(x_0)-2\varepsilon_0(x_0))U_0+\left(\varepsilon_0(x_0)+\frac{h}{2}f_0(x_0)\right)U_1\right].
$$

In § 3 the following optimal error estimate for (1.5) is proven.

THEOREM 1.1. *Suppose $f, g, q \in C^{j+1}$, $u \in C^6$, $\delta = O(h)$ is chosen so that* (1.3) *holds, and*

$$
f_0(x) \geqq a > 0, \qquad g_0(x) > 0.
$$

Then, for $1 \leqq n < N$,

$$
\begin{aligned}
\left|u'(x_n)-\frac{U_{n+1}-U_{n-1}}{2h}\right| \leqq & C\varepsilon\delta\left\{1+\varepsilon^{-3}\exp\left[-a\frac{1-x_n}{\varepsilon}\right]\right\} \\
& + Ch^2\left\{1+\varepsilon_0^{-3}\exp\left[-a\frac{1-x_n}{\varepsilon_0}\right]\right\},
\end{aligned}
$$

where C is independent of ε, δ, h and u.

In the above and throughout this paper we will use the convention that ε_0 (no argument) denotes $\max_{0<x<1}|\varepsilon_0(x)|$.

In Ervin and Layton [4], the method (1.2), (1.3) was proposed. Global and local error estimates were proven using discrete and continuous barrier function arguments. We will need several results from [4]. For completeness, we include them, without proof.

The problem of calculating u' in the layer is more difficult and must ultimately be based upon the theory of singular perturbations. Following, e.g., Eckhaus [3], we study the layer by stretching coordinates: Let $\phi_\varepsilon(\eta) = u(x)$, $\eta = (1-x)/\varepsilon$. We obtain the regular perturbation problem for ϕ_ε:

$$-\phi_\varepsilon''(\eta) + f(x)\phi_\varepsilon'(\eta) + \varepsilon g(x)\phi_\varepsilon(\eta) = \varepsilon q(\eta). \tag{1.6}$$

A good first approximation to $\phi'(0)$ is then given by matching the $\varepsilon = 0$ problem to the boundary condition and the numerical solution:

$$\phi_0(\eta) \cong C_0 + \frac{C_1}{f(1)} \exp(+f(1)\eta),$$

$$\phi(x = x_n) = U_n, \qquad \phi(x = 1) = \beta.$$

We then obtain a method of finding $u'(1)$ approximately:

$$u'(1) \cong \varepsilon^{-1}\left(\frac{\beta - U_n}{1 - \exp(-f(1)((1-x_n)/\varepsilon))}\right) f(1) \equiv \tilde{D}U(1). \tag{1.7}$$

The point x_n is chosen to ensure that $U_n = u(x_n) + O(h^2)$ uniformly in ε. From Theorem 1.2 of Ervin and Layton [4] it is sufficient to choose $n =$ greatest integer $[N - \ln(2/ah)]$. Experimentally we have seen that if δ_n is chosen so equality holds in (1.3) then we may even choose $n = N-1$ or $n = N-2$ with better results.

The estimates of §§ 2 and 3 that are used to prove Theorem 1.1 ensure that u will be smooth outside the layer and that U_n will be a good approximation to $u(x_n)$ there. This accuracy is one of the keys that ensures the accuracy of the approximation (1.7). Since $u'(1) = O(\varepsilon^{-1})$, the following theorem shows that the relative error in (1.7) is of order $O(\varepsilon + h)$.

THEOREM 1.2. *In addition to the assumptions of Theorem* 1.1, *suppose* $0 < \varepsilon \ll h < 1$, x_n *is chosen so that* $1 - x_n = O(h)$, *and* $|U_n - u(x_n)| \leqq C(h^2 + \varepsilon h)$. *Then,*

$$\varepsilon |u'(1) - \tilde{D}U(1)| \leqq C(h + \varepsilon).$$

More accurate formulas than (1.7) can be derived by taking a better approximation to $\int f(x)\,dx$ and higher order terms in ε in solving (1.6) but experiments indicate that (1.7) gives a very good approximation. For different types of boundary layers (e.g., characteristic layers) (1.7) must, of course, be modified appropriately. For other methods of calculating derivatives of u see, e.g., Pruess [9], Berger, Solomon and Ciment [7] and also Berger, Solomon and Ciment [8] for numerical results.

In § 2 we review the results of [4] necessary for the proofs of Theorem 1.1 and the related results of § 3. Section 4 presents numerical experiments which validate the theory of §§ 1-3. In particular, (1.7) is compared to more commonly used methods of find $u'(1)$. For a model problem with $\varepsilon = 10^{-4}$ and $h = 1/20$, (1.7) gives an approximation with 1.1% error. More traditional methods using mesh refinement in the layer combined with extrapolation and/or one-sided difference quotients had errors between 12% and 100%.

2. Preliminaries. In [4] the following two convergence theorems were proven.

THEOREM 2.1 [4, Thm. 1.1]. *There is a* $C > 0$, *independent of* ε, *such that*

$$\max_{0 \leqq j \leqq N} |u(x_j) - U_j| \leqq C\{h^2(\varepsilon + \delta) \max_x |u''''(x)| + \varepsilon h \max_x |u'''(x)|\}$$

$$\leqq Ch[(\varepsilon + \delta) + \varepsilon + h] \max_x |u'''|.$$

Local convergence results were also derived. For example, suppose f, g, δ satisfy:

$$f(x) > a,$$

$$f(x_n) - \delta_n(f(x_n)f'(x_n) + f(x_n)g(x_n)) > a, \tag{2.1}$$

$$g(x_n) - \delta_n f(x_n)g(x_n)' > 0.$$

THEOREM 2.2 [4, Thm. 1.2]. *Let* (2.1) *hold,* $\varepsilon_0(x) = \varepsilon + \delta f^2$ $(= \varepsilon + O(h))$. *Then, there is a* $b > 0$, *independent of* h *and* ε, *such that*

$$|u(x_n) - U_n| \leqq C\varepsilon\delta\left\{1 + \varepsilon^{-2}\exp\left[-a\frac{1-x_n}{\varepsilon}\right]\right\} + Ch^2\left\{1 + \varepsilon_0^{-2}\exp\left[-b\frac{1-x_n}{\varepsilon_0}\right]\right\},$$

where C *is independent of* ε *and* h.

We note also that $\delta = O(h)$ is chosen to be sufficiently large that the derived matrix is a strictly diagonally dominant M-matrix. The following two lemmas are simple consequences of this.

LEMMA 2.1 (Monotonicity). *If, for* $1 \leqq j \leqq N-1$,

$$L_0^h v_j \leqq L_0^h w_j, \quad v_0 \leqq w_0, \quad v_N \leqq w_N,$$

then $v_j \leqq w_j$ *for* $0 \leqq j \leqq N$.

LEMMA 2.2. *For any* v_n, $n = 0, \cdots, N$,

$$\max_{0 \leqq n \leqq N} |v_n| \leqq C[\max_{1 \leqq n \leqq N-1} |L_0^h v_n| + \max\{|v_0|, |v_N|\}],$$

where C *is independent of* ε *and* h.

It will also be necessary to recall some regularity results from [4]. These are proven in detail in [4], based on techniques of Kellogg and Tsan [5]. We state slightly sharper versions of the results of [4]. The proofs will be omitted when they follow, mutatis mutandis, from the proofs of [4].

DEFINITION. q is of class (K, j, n, ε) if

$$|q^{(i)}(x, \varepsilon)| \leqq K[1 + \varepsilon^{-i-1-n}\exp(-a\varepsilon^{-1}(1-x))] \quad \text{for } 0 \leqq i \leqq j.$$

Consider the problem

$$Lu \equiv -\varepsilon u'' + fu' + gu = q(x, \varepsilon),$$
$$u(0) = \alpha, \qquad u(1) = \beta. \tag{2.2}$$

LEMMA 2.3. *Suppose* $f \in C^j$, $g \in C^j$ *and let* q *be of class* (K, j, n, ε) *for some* $n \geqq 0$. *Then, the solution* u *of* (2.2) *satisfies* $u \in (K, j, n-1, \varepsilon)$:

$$|u^{(i)}(x)| \leqq C\{1 + \varepsilon^{-i-n}\exp(-a\varepsilon^{-1}(1-x))\} \qquad 0 \leqq i \leqq j+1, \tag{2.3}$$

where $C > 0$ *does not depend on* ε.

Proof. The proof is analogous to the proof of Lemma 3.3 of [4]. □

The proof of the next lemma is analogous to the proof of Lemma 3.4 of [4].

LEMMA 2.4. *Suppose* $f \in C^j$, $g^j \in C$ *and let* u *satisfy*

$$-\varepsilon u'' + fu' + gu = q, \quad u(0) = \alpha, \quad u(1) = \beta,$$

$$f(x) > a > 0, \qquad g(x) \geqq 0,$$

and q *be a class* $(K, j, n-1, \varepsilon)$. *Then, with* $F(x) = \int_0^x f(t)\,dt$.

$$u(x) = \gamma\varepsilon^{-n}\exp(-F(1)\varepsilon^{-1}(1-x)) + z(x)$$

where $\gamma \leqq C$, *and*

$$|z^{(i)}(x)| \leqq C_2\{1+\varepsilon^{-i+1-n}\exp(-a\varepsilon^{-1}(1-x))\},$$

with $C_1>0$ *and* $C_2>0$ *independent of* ε.

3. Proof of the theorems. We will need careful estimates for the truncation error of the method. Note that by applying $(I-\delta f(d/dx))$ to (1.1) we see that u satisfies

$$\begin{aligned} L_0u &\equiv -\varepsilon_0(x)u''+f_0(x)u'+g_0(x)u \\ &= q_0(x)-\varepsilon\delta f u''', \\ u(0)&=\alpha, \qquad u(1)=\beta. \end{aligned}$$

We thus obtain a formula for the truncation error $\tau(x,h)\equiv(L_0-L_0^h)u+\varepsilon\delta f u'''$:

$$\begin{aligned} \tau(x_n,h) &= -(\varepsilon+\delta_n f_n^2)\left(u_n''-\frac{u_{n+1}-2u_n+u_{n-1}}{h^2}\right) \\ &\quad +f_0(x_n)\left(u_n'-\frac{u_{n+1}-u_{n-1}}{2h}\right)+\varepsilon\delta_n f_n u_n''' \\ &= -\varepsilon_0(x_n)\left(\frac{h^2}{12}u^{(iv)}(x_n)+O(h^4)\right) \\ &\quad +f_0(x_n)\left(\frac{h^2}{6}u'''(x_n)+O(h^4)\right)+\varepsilon\delta_n f_n u'''(x_n). \end{aligned}$$

Summarizing, we have

LEMMA. *Suppose* $u\in C^6$; *then*

$$\begin{aligned} \tau(x_n,h) &= \frac{h^2}{12}\{-\varepsilon_0(x_n)u^{(iv)}(x_n)+2f_0(x_n)u'''(x_n)\}+\varepsilon\delta f_n u'''(x_n) \\ &\quad +\frac{h^4}{6!}\{-2\varepsilon_0(x_n)u^{(vi)}(\xi_n)+6f_0(x_n)u^{(v)}(\xi_n)\}, \end{aligned}$$

where ξ_n *is some intermediate point*: $x_{n-1}<\xi_n<x_{n+1}$.

Pointwise estimates of $\tau(x_n,h)$ can also be obtained. For example,

PROPOSITION 3.1. *If* $\varepsilon_0(x)=\varepsilon+\delta f^2$, *with* δ_n *chosen as above* $\tau(x_n,h)$ *satisfies*:

$$|\tau(x_n,h)| \leqq C\begin{cases} h\left[1+\left(\dfrac{\varepsilon_0}{\varepsilon}\right)\varepsilon^{-2}\exp\left\{-a\dfrac{1-x_n}{\varepsilon}\right\}\right]+\varepsilon\delta_n\left[1+\varepsilon^{-3}\exp\left(-a\dfrac{1-x_n}{\varepsilon}\right)\right], \\ h^2\left[1+\left(\dfrac{\varepsilon_0}{\varepsilon}\right)\varepsilon^{-3}\exp\left\{-a\dfrac{1-x_n}{\varepsilon}\right\}\right]+\varepsilon\delta_n\left[1+\varepsilon^{-3}\exp\left(-a\dfrac{1-x_n}{\varepsilon}\right)\right]. \end{cases}$$

Proof. Lemma 2.3 of Kellogg and Tsan [4] gives

$$\begin{aligned} |u''(t)| &\leqq C\left(1+\varepsilon^{-2}\exp\left(-a\frac{1-t}{\varepsilon}\right)\right), \\ |u'''(t)| &\leqq C\left(1+\varepsilon^{-3}\exp\left(-a\frac{1-t}{\varepsilon}\right)\right), \\ |u^{(iv)}(t)| &\leqq C\left(1+\varepsilon^{-4}\exp\left(-a\frac{1-t}{\varepsilon}\right)\right), \end{aligned}$$

so that, by Lemma 3.5 of Ervin and Layton [4]

$$|\tau(x_n, h)| \leqq C\left\{h + \int_{x_{n-1}}^{x_{n+1}} \varepsilon_0 \varepsilon^{-3} \exp\left(-a\frac{1-t}{\varepsilon}\right) dt\right\} + C\varepsilon\delta_n |u'''(x_n)|$$

$$\leqq C\left\{h + \left(\frac{\varepsilon_0}{\varepsilon}\right)\varepsilon^{-1} \sinh\left(a\frac{h}{\varepsilon}\right) * \exp\left(-a\frac{1-x_n}{\varepsilon}\right)\right\} + C\varepsilon\delta_n |u'''(x_n)|.$$

Since, for t bounded, $\sinh(t) \leqq Ct$,

$$|\tau(x_n, h)| \leqq Ch\left[1 + \left(\frac{\varepsilon_0}{\varepsilon}\right)\varepsilon^{-2} \exp\left(-a\frac{1-x_n}{\varepsilon}\right)\right] + C\varepsilon\delta\left(1 + \varepsilon^{-3} \exp\left(-a\frac{1-x_n}{\varepsilon}\right)\right).$$

Similarly, if the second half of Lemma 3.5 of [4] is used, we obtain the second half of the proposition.

It is convenient to analyze the method (1.2) by introducing the following intermediate problem. Let w be the solution of the two-point B.V.P.

$$L_0 w = q_0, \quad w(0) = \alpha, \quad w(1) = \beta,$$

and let $\omega_n = (L_0 - L_0^h)w(x_n)$. We will study the error via the decomposition $u - U = (u - w) + (w - U)$. Applying Proposition 3.1 of [4] and Lemma 2.3 of Kellogg and Tsan [5] gives

PROPOSITION 3.2. *Under the assumptions of the previous proposition,*

$$|\omega(x_n, h)| \leqq Ch^2\left(1 + \varepsilon_0^{-3} \exp\left(-a\frac{1-x_n}{\varepsilon_0}\right)\right).$$

Following, e.g., Courant, Friedrichs and Levy [2] and Bramble and Hubbard [1] in the context of our problem, we define $\phi(x)$ to be the solution of the B.V.P.

$$L_0\phi(x) = \tfrac{1}{12}\{-\varepsilon_0(x)w^{(iv)}(x) + 2f_0(x)w'''(x)\}, \qquad 0 < x < 1,$$

$$\phi(0) = 0, \qquad \phi(1) = 0.$$

PROPOSITION 3.3. *Under the assumptions of Theorem* 1.1,

$$\tfrac{1}{12}\{-\varepsilon_0(x)w^{(iv)}(x) + 2f_0(x)w'''\} \in (K, j, 2, \varepsilon_0);$$

thus $\phi(x) \in (K, j, 1, \varepsilon_0)$; *for* $0 \leqq i \leqq j$

$$|\phi^{(i)}(x)| \leqq K[1 + \varepsilon_0^{-i-2} \exp(-a\varepsilon_0^{-1}(1-x))].$$

We now define $\eta(x_n)$, $\psi(x_n, h)$ by

$$\eta(x_n) = U_n - w(x_n), \qquad \eta(x_n) = \phi(x_n)h^2 + \psi(x_n, h)h^3,$$

and recall that $\omega_n = (L_0 - L_0^h)w(x_n)$.

THEOREM 3.1. *Let η, ψ, ω be defined as above, and suppose the assumptions of the previous proposition hold. Then,*

$$|\psi(x_n, h)| \leqq Ch\left[1 + \varepsilon_0^{-4} \exp\left(-a\frac{1-x_n}{\varepsilon_0}\right)\right].$$

Proof. From the definitions of ϕ, ψ and η we have

$$L_0^h \eta(x_n) = h^2 L_0^h \phi(x_n) + h^3 L_0^h \psi(x_n, h),$$

so that

$$h^3 L_0^h \psi(x_n, h) = (L_0^h \eta(x_n) - h^2 L_0 \phi(x_n)) + h^2[L_0 - L_0^h]\phi(x_n). \tag{3.1}$$

We estimate the second term, $h^2[L_0-L_0^h]\phi(x_n)$, by using the Peano Kernel theorem. Indeed, following the proof of Lemma 3.5 of [4], we obtain

$$|h^2[L_0-L_0^h]\phi(x_n)| \leqq Ch^3 \int_{x_{n-1}}^{x_{n+1}} \varepsilon_0(t)|\phi^{(iv)}(t)|+|\phi'''(t)|\, dt.$$

Since the previous proposition gives

$$|\phi^{(iv)}(t)| \leqq K[1+\varepsilon_0^{-6}\exp(-a\varepsilon_0^{-1}(1-t))],$$
$$|\phi'''(t)| \leqq K[1+\varepsilon_0^{-5}\exp(-a\varepsilon_0^{-1}(1-t))],$$

we obtain

$$|h^2[L_0-L_0^h]\phi(x_n)| \leqq C\left\{h^4+h^3\int_{x_{n-1}}^{x_{n+1}} \varepsilon_0^{-5}\exp\left(-a\frac{1-t}{\varepsilon_0}\right) dt\right\}$$
$$\leqq Ch^4+Ch^3\varepsilon_0^{-4}\sinh\left(a\frac{h}{\varepsilon_0}\right)\exp\left(-a\frac{1-x_n}{\varepsilon_0}\right).$$

Since, for t bounded, $\sinh(t) \leqq Ct$,

$$|h^2[L_0-L_0^h]\phi(x_n)| \leqq Ch^4\left[1+\varepsilon_0^{-5}\exp\left(-a\frac{1-x_n}{\varepsilon_0}\right)\right].$$

The first term $(L_0^h\eta(x_n)-h^2L_0\phi(x_n))$ is estimated in a likewise manner, only we must apply the Peano Kernel theorem to the sixth derivative terms of w. Indeed, we can write

$$(D_+D_-w-w'')(x)=\frac{h^2}{12}w^{(iv)}(x)+h^{-2}(5!)^{-1}$$
$$\cdot\left[\int_x^{x+h}(x+h-s)^5w^{(vi)}(s)\, ds-\int_{x-h}^{x}(x-h-s)^5w^{(vi)}(s)\, ds\right],$$

$$(D_0w-w')(x)=\frac{h^2}{6}w'''(x)+(2h)^{-1}(4!)^{-1}$$
$$\cdot\left[\int_x^{x+h}(x+h-s)^4w^{(v)}(s)\, ds-\int_{x-h}^{x}(x-h-s)^4w^{(v)}(s)\, ds\right].$$

Using the above two formulas and the definitions of η and ϕ, we can estimate the first term on the R.H.S. of (3.1) as follows.

$$L_0^h\eta(x_n)-h^2L_0\phi(x_n)=\varepsilon_0(x_n)h^{-2}(5!)^{-1}$$
$$\cdot\left[\int_{x_n}^{x_{n+1}}(x_n+h-s)^5w^{(vi)}(s)\, ds-\int_{x_{n-1}}^{x_n}(x_n-h-s)^5w^{(vi)}(s)\, ds\right]$$
(3.2)
$$+f_0(x_n)(2h)^{-1}(4!)^{-1}\left[\int_{x_n}^{x_{n+1}}(x_n+h-s)^4w^{(v)}(s)\, ds\right.$$
$$\left.-\int_{x_{n-1}}^{x_n}(x_n-h-s)^4w^{(v)}(s)\, ds\right].$$

Following Kellogg and Tsan [5], we let $R_{m,n}$ denote a quantity bounded by

$$|R_{m,n}| \leqq Ch^m\int_{x_{i-1}}^{x_{i+1}}|w^{(n)}(s)|\, ds.$$

It then follows from (3.2) that $L_0^h\eta(x_n)-h^2L_0\phi(x_n)=\varepsilon_0(x_n)R_{3,6}+R_{3,5}$. Thus we have

$$|L_0^h\eta(x_n)-h^2L_0\phi(x_n)|\leqq Ch^3\int_{x_{n-1}}^{x_{n+1}}\varepsilon_0|w^{(vi)}(t)|+|w^{(v)}(t)|\,dt. \tag{3.3}$$

We now obtain a pointwise bound by using estimates for $w^{(v)}$ and $w^{(vi)}$ in the above. Lemma 2.3 of Kellogg and Tsan [5] gives

$$|w^{(v)}(t)|\leqq C\left(1+\varepsilon_0^{-5}\exp\left(-a\frac{1-t}{\varepsilon_0}\right)\right),$$

$$|w^{(vi)}(t)|\leqq C\left(1+\varepsilon_0^{-6}\exp\left(-a\frac{1-t}{\varepsilon_0}\right)\right).$$

Using these inequalities on the R.H.S. of (3.3) above, and the usual estimates (as in the first half of this proof) gives

$$|L_0^h\eta(x_n)-h^2L_0\phi(x_n)|\leqq Ch^4\left[1+\varepsilon_0^{-5}\exp\left(-a\frac{1-x_n}{\varepsilon_0}\right)\right].$$

Using this inequality and the corresponding one for $h^2[L_0-L_0^h]\phi(x_n)$ yields that

$$|h^3L_0^h\psi(x_n,h)|\leqq Ch^4\left[1+\varepsilon_0^{-5}\exp\left(-a\frac{1-x_n}{\varepsilon_0}\right)\right],$$

and thus,

$$|\psi(x_n,h)|\leqq Ch\left[1+\varepsilon_0^{-4}\exp\left(-a\frac{1-x_n}{\varepsilon_0}\right)\right].$$

Next we investigate how well derivatives of u can be approximated by the most straightforward method of taking differences of the approximate solution. The next two theorems prove that this approach gives optimal accuracy for $0\leqq x<1$, uniformly in ε. In the boundary layer a different technique must be used.

THEOREM 3.2. *Under the previous assumptions, we obtain, for* $n=1,2,\cdots,N-1$

$$\left|\frac{\eta(x_{n+1})-\eta(x_{n-1})}{2h}\right|\leqq Ch^2\left[1+\varepsilon_0^{-3}\exp\left(-a\frac{1-x_n}{\varepsilon_0}\right)\right].$$

Proof. We compute, from the definition of ϕ, η, ψ, for $n=1,2,\cdots,N-1$,

$$\frac{\eta(x_{n+1})-\eta(x_{n-1})}{2h}=h^2\frac{\phi(x_{n+1})-\phi(x_{n-1})}{2h}+h^3\frac{\psi(x_{n+1},h)-\psi(x_{n-1},h)}{2h}.$$

The first term on the R.H.S. is bounded by the Peano Kernel theorem and the bounds for $\phi'(x)$. Indeed,

$$\phi(x_{n+1})=\phi(x_n)+\int_{x_n}^{x_{n+1}}\phi'(s)\,ds$$

and similarly for $\phi(x_{n-1})$. Thus,

$$\frac{\phi(x_{n+1})-\phi(x_{n-1})}{2h}=(2h)^{-1}\left[\int_{x_n}^{x_{n+1}}\phi'(s)\,ds-\int_{x_{n-1}}^{x_n}\phi'(s)\,ds\right].$$

Since

$$|\phi'(s)|\leqq K\left[1+\varepsilon_0^{-3}\exp\left(-a\frac{1-s}{\varepsilon_0}\right)\right]$$

we get, by the usual argument,

$$\left|\frac{\phi(x_{n+1})-\phi(x_{n-1})}{2h}\right| \leqq C\left[1+\varepsilon_0^{-3}\exp\left(-a\frac{1-x_n}{\varepsilon_0}\right)\right].$$

By the previous theorem, we get a bound on ψ:

$$\left|\frac{\psi(x_{n+1},h)-\psi(x_{n-1},h)}{2h}\right| \leqq C\left[1+\varepsilon_0^{-4}\exp\left(-a\frac{1-x_n}{\varepsilon_0}\right)\right]. \qquad \square$$

Using this, we can show that differences of U_n give an optimal approximation for $w'(x_n)$. This is the first step in estimating the error in approximating $U'(x_n)$.

PROPOSITION 3.4. *Under the previous assumptions, we have for* $n=1,2,\cdots,N-1$:

$$\left|w'(x_n)-\frac{U_{n+1}-U_{n-1}}{2h}\right| \leqq Ch^2\left[1+\varepsilon_0^{-3}\exp\left(-a\frac{1-x_n}{\varepsilon_0}\right)\right].$$

Proof. We write, using the triangle inequality,

$$\left|w'(x_n)-\frac{U_{n+1}-U_{n-1}}{2h}\right| \leqq \left|\frac{\eta(x_{n+1})-\eta(x_{n-1})}{2h}\right| + \left|w'(x_n)-\frac{w(x_{n+1})-w(x_{n-1})}{2h}\right|.$$

We bound the second term by:

$$w'(x_n)-\frac{w(x_{n+1})-w(x_{n-1})}{2h} = -(2!)^{-1}(2h)^{-1}$$

$$\cdot\left[\int_{x_n}^{x_{n+1}}(x_n+h-s)^2 w'''(s)\,ds+\int_{x_{n-1}}^{x_n}(x_n-h-s)^2 w'''(s)\,ds\right]. \tag{3.4}$$

Lemma 3.5 of Kellogg and Tsan [5] gives

$$|w'''(s)| \leqq K\left[1+\varepsilon_0^{-3}\exp\left(-a\frac{1-s}{\varepsilon_0}\right)\right].$$

Using this estimate in (3.4) yields

$$\left|w'(x_n)-\frac{w(x_{n+1})-w(x_{n-1})}{2h}\right| \leqq Ch^2\left[1+\varepsilon_0^{-3}\exp\left(-a\frac{1-x_n}{\varepsilon_0}\right)\right].$$

Using this and the previous theorem yields the result.

Proof of Theorem 1.1. Decompose the error in $U'(x_n)$ as:

$$u'(x_n)-\frac{U_{n+1}-U_{n-1}}{2h} = u'(x_n)-w'(x_n)+w'(x_n)-\frac{U_{n+1}-U_{n-1}}{2h}.$$

The second half has been estimated in the previous proposition. The first half follows by noting that the proof of Theorem 3.1 of Ervin and Layton [4] actually establishes a stronger result. Namely, that

$$|u^{(i)}(x)-w^{(i)}(x)| \leqq \varepsilon\delta C\left\{1+\varepsilon^{-2-i}\exp\left[-a\frac{1-x_n}{\varepsilon}\right]\right\},$$

holds for $0 \leqq i \leqq j$. $\square$

Proof of Theorem 1.2. Let $\Phi(x)$ denote the first term in the standard asymptotic expansion of the solution u:

$$\Phi(x) = \bar{u}_0(x)+\tilde{u}_0\left(\frac{1-x}{\varepsilon}\right).$$

Here, $\bar{u}_0(x)$ is the solution to the initial value problem:

$$f(x)\bar{u}_0' + g(x)\bar{u}_0 = q(x), \qquad \bar{u}_0(0) = \alpha$$

and $\tilde{u}_0$ is the first term of the inner expansion matched with $\bar{u}_0$:

$$\tilde{u}_0(x) = (\beta - \bar{u}_0(1)) \exp\left(-f(1)\frac{1-x}{\varepsilon}\right).$$

Since the coefficients of L_ε are smooth, the construction of $\Phi(x)$ is such that (see Eckhaus [3], for example) the remainder $R(x, \varepsilon) = u - \Phi$ satisfies

(i) $|R(x; \varepsilon)| \leqq C\varepsilon, \qquad 0 \leqq x \leqq 1,$

(ii) $|L_\varepsilon R(x; \varepsilon)| \leqq C\varepsilon, \qquad 0 < x < 1.$

Letting $Q(x, \varepsilon) = L_\varepsilon R(x, \varepsilon)$, R then satisfies the B.V.P.

$$\begin{aligned} &L_\varepsilon R = Q, \qquad 0 < x < 1, \\ &R(0, \varepsilon) = 0, \qquad R(1, \varepsilon) = 0. \end{aligned} \tag{3.5}$$

Let

$$[v]_k \equiv \sup_{0<x<1} \left| \frac{d^k}{dx^k} v(x) \right|$$

denote the kth Hölder seminorm. Then, Theorem 6.2.3.1 of Eckhaus [3, pp. 214-215] applied to (3.5) gives the following estimate on $R = u - \Phi$:

$$\varepsilon^2[u-\Phi]_2 + \varepsilon[u-\Phi]_1 + [u-\Phi]_0 \leqq C[Q]_0.$$

In particular, we have that

$$\sup_{0<x<1} |u'(x) - \Phi'(x)| \leqq C.$$

We complete the proof by comparing the formula (1.7) directly with $\Phi'(1)$. We find that

$$\begin{aligned} \Phi'(1) &= u_0(1) + (\beta - \bar{u}_0(1))\frac{f(1)}{\varepsilon} \\ &= O(1) + (\beta - \bar{u}_0(1))\frac{f(1)}{\varepsilon} \end{aligned}$$

and that

$$\tilde{D}U(1) = \frac{f(1)(\beta - U_n)}{\varepsilon}\left(\frac{1}{1 - \exp\left(-f(1)\frac{1-x_n}{\varepsilon}\right)}\right).$$

Since $1 - x_n = O(h) \gg \varepsilon$ it follows that

$$\exp\left(-f(1)\frac{1-x_n}{\varepsilon}\right) \ll 1$$

and

$$\Phi'(1) - \tilde{D}U(1) = \frac{f(1)}{\varepsilon}(U_n - \bar{u}_0(1)) + O(1) + O\left(\varepsilon^{-1}\exp\left(-f(1)\frac{1-x_n}{\varepsilon}\right)\right).$$

Since $\bar{u}_0(x)$ is smooth uniformly in ε, and $\varepsilon \ll h$

$$\bar{u}_0(1) = \bar{u}_0(x_n) + O(h) = u(x_n) + O(h) + O(\varepsilon)$$
$$= U_n + O(h) + O(\varepsilon) + O(h^2 + \varepsilon h).$$

Thus,

$$\varepsilon|\Phi'(1) - \tilde{D}U(1)| \leqq C(h+\varepsilon),$$

and the result follows. □

4. Numerical experiments. The numerical method was first tested on the constant coefficient problem on $0 < x < 1$:

$$-\varepsilon u'' + au' + bu = 0, \qquad u(0) = 1, \qquad u(1) = 2, \tag{4.1}$$

where $\varepsilon = 10^{-4}$, $a = 2$ and $b = 1$. We give below the maximum error on $0 \leqq x \leqq 1$ and on $0 \leqq x \leqq .8$ for various values of $N = h^{-1}$ and the corresponding decay exponents α. These are computed by assuming that the error is proportional to h^α and solving for α from two different values of h and the errors. (See Table 1.)

Notice that good answers are already obtained with a very coarse h. We also calculate an approximation to $u'(x_n)$ by taking central differences of U_n. The corresponding errors and decay exponents are given in Table 2.

In Table 2 we see that good approximations to $u'(x_n)$ can be obtained for x_n away from 1 but that the central difference approximation diverges at the last two mesh points. We now consider the problem of approximating $u'(1)$ by the special formula that was derived via the stretched coordinate problem.

We suppose several methods of finding $u'(1) = 27869$. Method 1 is the usual method of taking an $O(h^2)$ difference quotient of U_N, U_{N-1}, and U_{N-2}. Method 2

TABLE 1
Errors in $u(x_n)$ for (4.1).

	Number of points, $n =$		
Maximum error	30	20	10
$0 \leqq x \leqq 1$	1.36E−5	3.01E−5	1.15E−4
$0 \leqq x \leqq .8$	1.22E−5	2.73E−5	1.07E−4
Decay exponent on $0 \leqq x \leqq 1$	$N = 20, 30$ $\alpha = 1.99$		$N = 10, 20$ $\alpha = 1.93$

TABLE 2
Errors in approximating $u'(x_n)$ for $N = 10, 20, 30$.

Maximum error	$N = 10$	$N = 20$	$N = 30$
$0 \leqq x \leqq 1$	2.7E+4	2.7E+4	2.7E+4
$0 < x \leqq .8$	5.9E−5	1.4E−5	6.0E−6
Decay exponents for $0 < x \leqq .8$	2.08		2.47

computes $u'(1)$ via the formula (1.7) given in the introduction. In Method 3, $u(x_n)$ is calculated on N points. Then, the problem on $x_{N-1} \leqq x \leqq 1$ is resolved by placing N points there also. (This process can be continued several times and results in a type of "exponentially graded" mesh.) Then, $u'(1)$ is approximated by an $O(h^2)$ difference on the finest mesh. Method 4 extrapolates from U_{N-2} and U_{N-1} to find an approximation $u(1-O(\varepsilon)) \cong u(1-\varepsilon)$. Then, $u'(1)$ is calculated via a backward difference $u'(1) \cong \varepsilon^{-1}(u(1)-u(1-\varepsilon))$. Method 5 interpolates with a parabola whose slope at $x=1$ is the approximation to $u'(1)$. The results of Methods 1-5 are summarized in Table 3.

TABLE 3
Errors in various methods of approximating $u'(1)=27870$ *in* (4.1).

Approx. to $u'(1)$ / % Error	Usual $O(h^2)$ difference	Formula proposed in introduction	Graded mesh plus $O(h^2)$ difference quotient	Extrapolation to $u(1-\varepsilon)$ plus backward difference	Quadratic interpolation at 1, $1-(\varepsilon/2)$ and $1-\varepsilon$
$N=10$	20.5 99.9%	27,245 2.24%	—	—	—
$N=20$	4.15 99.85%	27,562 1.11%	—	—	—
$N=30$	62.4 99.8%	27,665 .735%	22,045 20%	31,100 12%	23,200 16.8%

The superiority of the formula given in the introduction is evident as is the $O(h)$ convergence of the relative error. Naturally, the formula could be combined with the other approaches, e.g., using a graded mesh, or extrapolating U_n and U_{n-1} to obtain a better approximation of $\bar{u}(1)$ to produce an even more accurate answer.

The constant coefficient example (4.1) does not consider the error that occurs when $f(x)$ is replaced by $f(1)$ in calculating u' in the layer. To study this, we use the test problem of Leventhal [6], which has a boundary layer at $x=0$:

$$\begin{aligned} &\varepsilon u''+f(x)u'=q(x,\varepsilon), \qquad 0<x<1, \\ &u(0)=u_E(0), \qquad u(1)=u_E(1), \end{aligned} \tag{4.2}$$

where $f(x)=(x+1)^3$ and $q(x,\varepsilon)$ is chosen so that $u(x)=u_E(x)$ where

$$u_E(x)=e^{-x/2}+\exp\left[-\varepsilon^{-1}\int_0^x (t+1)^3\,dt\right].$$

As pointed out by Leventhal [6], u_E is the first term of the matched asymptotic expansion for $u(x)$ when $q=0$. We compute $u'(0)$ $(=-\frac{1}{2}-\varepsilon^{-1})$ through the analogue of the formula (1.7) for the problem (4.1)

$$u'(0) \cong \frac{U_n-\alpha}{\varepsilon(1-e^{-x_n/\varepsilon})}. \tag{4.3}$$

Note that in Table 4 errors are larger than for the constant coefficient problem, but still the method gives a very good approximation to $u(x_n)$ and to $u'(0)$.

TABLE 4
Errors for the variable coefficient problem (4.2).

	Maximum error $0 \leqq x \leqq 1$	$N = 50$	$N = 100$
$\varepsilon = 10^{-4}$	$0 \leqq x \leqq 1$ $.2 \leqq x \leqq 1$	3.33E−1 1.20E−4	1.60E−6 1.23E−6
$\varepsilon = 10^{-8}$	$0 \leqq x \leqq 1$ $.2 \leqq x \leqq 1$	3.33E−1 1.20E−4	1.62E−6 1.25E−6

Approximation to $u'(0)$ / % Error	$N = 50$ $n = 2$	$N = 100$ $n = 1$
$\varepsilon = 10^{-4}$	−9,090 9.1%	−10,050 .5%
$\varepsilon = 10^{-8}$	−91,000,000 9%	−100,500,000 .5%

REFERENCES

[1] J. H. BRAMBLE AND B. E. HUBBARD, *Approximation of derivatives by finite difference methods in elliptic boundary value problems*, in Contributions to Differential Equations, Vol. III, No. 1, 1964, pp. 399-410.

[2] R. COURANT, K. O. FRIEDRICHS AND H. LEVY, *Über die Partiellen Differenzen gleichungen der Mathematischen Physik*, Math. Ann., V. 100 (1928), pp. 32-74.

[3] W. ECKHAUS, *Asymptotic Analysis of Singular Perturbations*, North-Holland, Amsterdam, 1979.

[4] V. ERVIN AND W. LAYTON, *A second order accurate, positive scheme for singular perturbed boundary value problems*, Technical Report, School of Mathematics, Georgia Tech., Atlanta, GA, 1985.

[5] R. B. KELLOGG AND A. TSAN, *Analysis of some difference approximations for singular perturbation problems without turning points*, Math. Comp., 32 (1978), pp. 1025-1039.

[6] S. H. LEVENTHAL, *An operator compact implicit method of exponential type*, J. Comput. Phys., 46 (1982), pp. 138-165.

[7] A. E. BERGER, J. M. SOLOMON AND M. CIMENT, *An analysis of a uniformly accurate difference method for a singular perturbation problem*, Math. Comp., 37 (1981), pp. 79-94.

[8] ———, *Uniformly accurate difference methods for a singular perturbation problem*, in Proc. Int. Conf. on Boundary and Int. Layers, J. J. H. Miller, ed., Trinity College, Dublin, Ireland, Book Press Dublin, 1980, pp. 14-28.

[9] S. A. PRUESS, *Solving linear boundary value problems by approximating the coefficients*, Math. Comp., 27 (1973), pp. 551-561.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 3, May 1987
© 1987 Society for Industrial and Applied Mathematics
006

THE NUMERICAL SOLUTION OF THE THREE-DIMENSIONAL INVERSE SCATTERING PROBLEM FOR TIME HARMONIC ACOUSTIC WAVES*

DAVID COLTON† AND PETER MONK†

Abstract. The inverse scattering problem under consideration is to determine the shape of an acoustically soft obstacle in $\mathbb{R}^3$ from a knowledge of the time harmonic incident wave and the far field pattern of the scattered wave. It is assumed that the frequency is in the resonance region, i.e. high frequency asymptotic methods are not available. A method for solving this problem is presented which is based on the theory of Herglotz wave functions and avoids the use of integral equations. Numerical examples are given showing the practicality of this approach.

Key words. inverse scattering, numerical solution

AMS(MOS) subject classifications. 35R30, 65P05

1. Introduction. A problem of considerable importance in various areas of technology is the determination of the shape of an obstacle from its effect on a known acoustic or electromagnetic wave. However, inverse scattering problems of this type are particularly difficult to solve due to the fact that they are both improperly posed and nonlinear (cf. [5]). In particular, assertions about the nature of the scattering obstacle are highly dependent upon the limitations and errors in the measured signals. Hence a major task in attempting to solve the inverse scattering problem is to take full recognition of both the nonlinear and improperly posed nature of the problem and to derive suitable methods for its stabilization and solution. For frequencies in the resonance region (i.e. high frequency approximations are not available) two methods have been proposed for solving the inverse scattering problem described above. The first, and oldest, of these approaches is the method of analytic continuation [12], [18]. However, in addition to being restricted to a limited class of boundary conditions, this method is subject to severe instability problems due to the fact that the measured far field data is subject to error and thus one is faced with the improperly posed problem of numerical analytic continuation. Hence, in recent years, a second method has been proposed which is based on the use of nonlinear optimization techniques. Such optimization methods deal directly with the stabilization problem by using regularization techniques to construct a scattering obstacle which gives a best fit to the far field data in some appropriate manner (cf. [2], [3], [6], [7], [15]). The regularization can either be applied to the full nonlinear problem or to the linear problems arising in each step of a Newton type procedure to solve the original nonlinear problem. We note that the main interest in such an approach is in the case where it is not in general possible to make use of various asymptotic approximations, such as physical optics. In those cases where such approximations are valid, the optimization method is still applicable and enjoys the corresponding simplicity of such an approach (cf. [14]).

The optimization methods available at this time can be divided into two separate groups. The first of these assumes a priori knowledge of the geometry of the scattering obstacle in order to restrict the function sought after to be in a compact set and then uses the method of modified integral equations to reformulate the inverse scattering problem as an optimization problem defined for functions in this set (cf. [1], [5], [15]).

* Received by the editors July 24, 1985. This research was supported in part by National Science Foundation grant DMS-8320550 and Office of Naval Research grant N00014-85-K-0532 and was written while the first author was on sabbatical leave at Drexel University, Philadelphia, Pennsylvania 19104.

† Department of Mathematical Sciences, University of Delaware, Newark, Delaware, 19716.

Such a procedure has the advantage of being very flexible, for example it is not necessary to know the phase of the far field pattern in order to implement the method. However, it suffers the disadvantage that it is often difficult to compute the Fréchet derivative of the nonlinear operator mapping the domain onto the far field pattern, as well as the fact that a Fredholm integral equation has to be solved at each step of the iterative process. A second class of optimization methods has recently been developed [6], [7] in which an optimal solution is sought in the orthogonal complement of the closure of a set determined by the set of all possible far field patterns. This method also assumes a priori knowledge of the geometry of the scattering obstacle and has its mathematical basis in the theory of Herglotz wave functions [10], [11]. The practical effect of working in the orthogonal complement of the set of far field patterns is to reduce the problem of solving an *exterior* scattering problem to that of an *interior* boundary value problem. The advantage of this approach is that the minimization problem has an extremely simple Fréchet derivative and in addition avoids the use of integral equations altogether. However, it suffers the disadvantage that in order to implement the procedure it is necessary to know both the phase and amplitude of the far field pattern.

We have already alluded to the fact that the presence of noisy data suggests that the inverse scattering problem is properly formulated as a "best-fit" or optimization problem incorporating certain a priori constraints. Indeed, as indicated in the preface to [16], this theme is central not only to the inverse scattering problem we are considering but to most, if not all, inverse problems in wave propagation. Hence, a major task is to numerically implement the optimization methods described above to realistic three-dimensional inverse scattering problems. Unfortunately, to date, such numerical experiments have only been performed for problems in two dimensions [2], [3], [6], [7], [15]. The problem of numerically implementing the first optimization method for three-dimensional problems is fraught with difficulties at the present state of development since each step of the Newton type procedure for solving the nonlinear optimization problem requires the solution of an exterior boundary value problem. However, the second optimization scheme is relatively simple by comparison and hence there appears to be some hope that this approach can successfully be used to numerically solve the three-dimensional inverse scattering problem. The purpose of this paper is to do this, restricting our attention to the case of acoustic scattering by a sound soft scattering obstacle. In the next section of this paper we shall briefly present the mathematical details of this method for solving the inverse scattering problem, basing our ideas on the results of Colton and Monk [6], [7] for the case of two dimensions. We shall then describe the numerical implementation of our mathematical method and present various numerical examples showing the practicality of our approach. To our knowledge, these examples provide the first successful solution of the three-dimensional inverse scattering problem for obstacles of complicated geometry and for frequencies in the resonance region. Our method is also applicable to inverse scattering problems in acoustic and electromagnetic wave propagation for obstacles satisfying a variety of different boundary conditions, and we plan to examine such problems in the near future.

2. An algorithm for solving the inverse scattering problem. Let D be a bounded, simply connected domain in $\mathbb{R}^3$ containing the origin with C^2 boundary ∂D and let the incident field $u^i(\mathbf{x})$ be given by the "plane wave" $u^i(\mathbf{x}) = \exp[ik\mathbf{x}\cdot\boldsymbol{\alpha}]$ where $\mathbf{x}\in\mathbb{R}^3$, $k>0$ is the wave number, and $\boldsymbol{\alpha}$ is a fixed unit vector. If we denote the scattered field by u^s and define the total field u by $u = u^i + u^s$, then the direct scattering problem

for an acoustically soft obstacle D is to find a solution $u \in C^2(\mathbb{R}^3 \setminus \bar{D}) \cap C(\mathbb{R}^3 \setminus D)$ of the Helmholtz equation

$$\Delta_3 u + k^2 u = 0 \quad \text{in } \mathbb{R}^3 \setminus \bar{D} \tag{2.1}$$

such that

$$u = 0 \quad \text{on } \partial D \tag{2.2}$$

and u^s satisfies the Sommerfield radiation condition

$$\lim_{r \to \infty} r\left(\frac{\partial u^s}{\partial r} - iku^s\right) = 0 \tag{2.3}$$

where $r = |\mathbf{x}|$ and (2.3) is assumed to hold uniformly in all directions. Under these conditions it can be shown [5] that u^s has the asymptotic expansion

$$u^s(\mathbf{x}) = \frac{e^{ikr}}{r} F(\hat{\mathbf{x}}, k; \boldsymbol{\alpha}) + O\left(\frac{1}{r^2}\right) \tag{2.4}$$

where $\hat{\mathbf{x}} = \mathbf{x}/|\mathbf{x}|$ and F is known as the *far field pattern* of the scattered field u^s. The *inverse scattering problem* we are concerned with in this paper is, given F, to find D. In particular let $\partial B = \{\mathbf{x} \in \mathbb{R}^3 : |\mathbf{x}| = 1\}$ and let λ_n be the nth eigenvalue of the interior Dirichlet problem for the Helmholtz equation in D. Then given the far field pattern F for all $\hat{\mathbf{x}} \in \partial B$, some fixed $k^2 \neq \lambda_n$, $n = 1, 2, \cdots$, and $\boldsymbol{\alpha} = \boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \cdots, \boldsymbol{\alpha}_N$, where the $\boldsymbol{\alpha}_i$ are distinct unit vectors, we want to determine ∂D. We note that the restriction on k^2 can be avoided [7]; however, in practice, the possibility of choosing k^2 near enough to an eigenvalue to cause numerical instability is reasonably small and hence for the sake of simplicity we do not concern ourselves in this paper with more general formulations that take this possibility into consideration.

We begin our analysis for solving the inverse scattering problem by noting that from Green's formula we can easily show [5] that

$$F(\hat{\mathbf{x}}, k; \boldsymbol{\alpha}) = -\frac{1}{4\pi} \int_{\partial D} \frac{\partial u}{\partial \nu}(\mathbf{y}) \exp[-ik\hat{\mathbf{x}} \cdot \mathbf{y}] \, ds(\mathbf{y}) \tag{2.5}$$

where ν is the unit outward normal to ∂D. Hence, if $g \in L^2(\partial B)$, then

$$\int_{\partial B} F(\hat{\mathbf{x}}, k; \boldsymbol{\alpha}) \overline{g(\hat{\mathbf{x}})} \, ds(\hat{\mathbf{x}}) = -\frac{1}{4\pi} \int_{\partial D} \left\{ \overline{\int_{\partial B} g(\hat{\mathbf{x}}) \exp[ik\hat{\mathbf{x}} \cdot \mathbf{y}] \, ds(\hat{\mathbf{x}})} \right\} \frac{\partial u}{\partial \nu}(\mathbf{y}) \, ds(\mathbf{y}). \tag{2.6}$$

We now define $v(\mathbf{y})$ by

$$v(\mathbf{y}) = \int_{\partial B} g(\hat{\mathbf{x}}) \exp[ik\hat{\mathbf{x}} \cdot \mathbf{y}] \, ds(\hat{\mathbf{x}}) \tag{2.7}$$

and note that $v(\mathbf{y})$ is a solution of the Helmholtz equation for $\mathbf{y} \in \mathbb{R}^3$. Functions of the form (2.7) for $g \in L^2(\partial B)$ are called *entire Herglotz wave functions* and the function $g(\hat{\mathbf{x}})$ is known as the *Herglotz kernel* of v [4], [6], [7], [10], [11]. Now suppose that the domain D is such that the Herglotz kernel g can be chosen so that v is the solution of

$$\begin{aligned} &\Delta_3 v + k^2 v = 0 \quad \text{in } D, \\ &v(\mathbf{y}) = -e^{-ik\rho}/\rho \quad \text{on } \partial D, \end{aligned} \tag{2.8}$$

where $\rho = |\mathbf{y}|$ (note that a unique solution of (2.8) exists since $k^2 \neq \lambda_n$). Domains for which the solution of (2.8) is an entire Herglotz wave function are called *Herglotz*

domains [8]. Suppose that D is contained in the ball $|\mathbf{y}| < b$. Then from (2.6) and Green's formula we have that if the solution of (2.8) is an entire Herglotz wave function with Herglotz kernel g then for any $\boldsymbol{\alpha}$,

$$
\begin{aligned}
\int_{\partial B} F(\hat{\mathbf{x}}, k; \boldsymbol{\alpha})\overline{g(\hat{\mathbf{x}})}\, ds(\hat{\mathbf{x}}) &= -\frac{1}{4\pi}\int_{\partial D}\frac{\partial u}{\partial \nu}(\mathbf{y})\overline{\nu(\mathbf{y})}\, ds(\mathbf{y}) \\
&= \int_{\partial D}\frac{\partial u}{\partial \nu}(\mathbf{y})\frac{e^{ik\rho}}{4\pi\rho}\, ds(\mathbf{y}) \\
&= \int_{\partial D}\left(\frac{\partial u}{\partial \nu}(\mathbf{y})\frac{e^{ik\rho}}{4\pi\rho} - u(\mathbf{y})\frac{\partial}{\partial \nu}\frac{e^{ik\rho}}{4\pi\rho}\right) ds(\mathbf{y}) \\
&= \int_{|\mathbf{y}|=b}\left(\frac{\partial u^i}{\partial \nu}(\mathbf{y})\frac{e^{ik\rho}}{4\pi\rho} - u^i(\mathbf{y})\frac{\partial}{\partial \nu}\frac{e^{ik\rho}}{4\pi\rho}\right) ds(\mathbf{y}) \\
&= u^i(0) \\
&= 1.
\end{aligned}
\tag{2.9}
$$

Equation (2.9) is the fundamental identity on which our algorithm for solving the inverse scattering problem is based.

We shall now use the above simple analysis to reformulate the inverse scattering problem as a problem in constrained optimization. Assume that, in addition to the assumptions previously made on D, we have that D is starlike with respect to the origin and ∂D is parameterized in the form $\rho = \rho(\hat{\mathbf{x}})$ where $0 < a \leqq \rho(\hat{\mathbf{x}}) \leqq b < \infty$ for some positive, a priori known constants a and b (as in [6] and [7], the assumption of starlikeness is made only to provide a convenient parameterization for ∂D). Define the sets U_1 and U_2 by

$$U_1(M) = \{g \in W_2^2(\partial B): \|g\| \leqq M\},$$

$$U_2 = \{\rho \in C(\partial B): 0 < a \leqq \rho(\hat{\mathbf{x}}) \leqq b < \infty \text{ and } |\rho(\hat{\mathbf{x}}_1) - \rho(\hat{\mathbf{x}}_2)| \leqq C|\hat{\mathbf{x}}_1 - \hat{\mathbf{x}}_2|\}$$

where M and C are positive constants and $W_2^2(\partial B)$ denotes the standard Sobolev space with norm $\|\cdot\|$. The a priori assumptions on ∂D are incorporated in the set U_2 via the a priori known constants a, b and C. It can easily be verified that both U_1 and U_2 are compact in $C(\partial B)$. We now define the optimization problem

$$\mu_1(M) = \min_{g \in U_1(M)} \sum_{n=1}^{N} \left| \int_{\partial B} F(\hat{\mathbf{x}}, k; \boldsymbol{\alpha}_n)\overline{g(\hat{\mathbf{x}})}\, ds(\hat{\mathbf{x}}) - 1 \right|^2. \tag{2.10}$$

Once we have computed a function $g \in U_1(M)$ that minimizes the sum in (2.10), we can compute $\rho(\hat{\mathbf{x}})$ by solving the optimization problem

$$\mu_2(g) = \min_{\rho \in U_2} \int_{\partial B} |v(\rho(\hat{\mathbf{x}})\hat{\mathbf{x}}) + e^{-ik\rho(\hat{\mathbf{x}})}/\rho(\hat{\mathbf{x}})|^2\, ds(\hat{\mathbf{x}}) \tag{2.11}$$

where $v(\mathbf{y})$ is defined by (2.7). From the compactness of U_1 and U_2 and the continuity of the integrals in (2.10) and (2.11) we can conclude that $\mu_1(M)$ and $\mu_2(g)$ exist.

DEFINITION. A function $\rho \in U_2$ will be called *admissible* if there exists a function $g \in U_1$ such that $\mu_1(M)$ is attained at g and $\mu_2(g)$ is attained at ρ.

If D is a Herglotz domain with Herglotz kernel $g \in W_2^2(\partial B)$, and M is large enough, we have already shown that we can choose g and ρ so that both μ_1 and μ_2 are zero. The following theorem tells us that in general we should seek functions g and ρ that make μ_1 and μ_2 arbitrarily small.

THEOREM 2.1. *Suppose* ∂D: $\rho = \rho(\hat{\mathbf{x}})$ *is such that* $\rho \in C^{2,\alpha}(\partial B) \cap U_2$. *Then there exists* $g_M \in U_1(M)$ *such that*

$$\lim_{M\to\infty} \left\{ \sum_{n=1}^{N} \left| \int_{\partial B} F(\hat{\mathbf{x}}, k; \boldsymbol{\alpha}_n) \overline{g_M(\hat{\mathbf{x}})}\, ds(\hat{\mathbf{x}}) - 1 \right|^2 + \int_{\partial B} \left| v_M(\rho(\hat{\mathbf{x}})\hat{\mathbf{x}}) + \frac{e^{-ik\rho(\hat{\mathbf{x}})}}{\rho(\hat{\mathbf{x}})} \right|^2 ds(\hat{\mathbf{x}}) \right\} = 0$$

where v_M *is defined by* (2.7) *with* $g = g_M$.

The proof of this theorem, which is based on the approximation properties of entire Herglotz wave functions, is similar to that of Theorem 2.3 of [6] and we shall omit it here.

Our next theorem shows that the solutions of the optimization problems (2.10) and (2.11) depend continuously on the far field data in a certain weak sense. The proof of this theorem is based on the compactness of the sets U_1 and U_2. We again omit the proof, referring the interested reader to Theorem 2.1 of [6] for the proof of a similar result.

THEOREM 2.2. *Let* $\Phi(F)$ *be the set of admissible solutions corresponding to the far field pattern* F. *Then if* $F_j \to F$ *in* $L_2(\partial B)$, $\rho_j \in \Phi(F_j)$, *there exists a convergent subsequence of* $\{\rho_j\}$ *and every limit point lies in* $\Phi(F)$.

It would be of interest to characterize the set of admissible functions $\Phi(F)$. Unfortunately, we have been unable to do this in general. However, in the case of Herglotz domains with Herglotz kernel $g \in W_2^2(\partial B)$ (so that μ_1 and μ_2 are zero for M large enough) we can show that if k^2 is less than λ_1 then in the limit as N (the number of incoming waves) approaches infinity all admissible functions $\rho^* \in \Phi(F)$ satisfy $\rho^* \geqq \rho$ where ∂D: $\rho = \rho(\hat{\mathbf{x}})$ is the solution of the inverse scattering problem. Furthermore, if k^2 is small enough then $\Phi(F) = \{\rho\}$. The interested reader is referred to [7, Thm. 2.4] for a proof of these results in two dimensions. (This proof can easily be generalized to the present three-dimensional case.)

3. Numerical implementation. In this section we shall describe the numerical implementation of the algorithm developed in the previous section. The data for the inverse problem are approximate far field patterns corresponding to a finite number of incoming plane waves. We further approximate the far field pattern $F(\hat{\mathbf{x}}, k; \boldsymbol{\alpha}_n)$ by a finite Fourier expansion of the form

$$F_a(\hat{x}(\theta, \phi), k; \boldsymbol{\alpha}_n) = \sum_{l=0}^{L_{\max}} \sum_{m=0}^{l} F_{lm1}^n \gamma_l^m P_l^m(\cos\theta) \cos m\phi + \sum_{l=1}^{L_{\max}} \sum_{m=1}^{l} F_{lm2}^n \gamma_l^m P_l^m(\cos\theta) \sin m\phi \tag{3.1}$$

where P_l^m denotes an associated Legendre polynomial,

$$\hat{\mathbf{x}}(\theta, \phi) = (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta), \tag{3.2}$$

and θ, ϕ are spherical polar angles. The weights γ_l^m in (3.1) are chosen so that

$$\int_0^{2\pi} \int_0^{\pi} (\gamma_l^m)^2 (P_l^m(\cos\theta))^2 \begin{Bmatrix} \cos^2 m\phi \\ \sin^2 m\phi \end{Bmatrix} \sin\theta\, d\theta\, d\phi = 1. \tag{3.3}$$

The coefficients $\{F_{lmi}^n\}_{i=1,2}$ are computed using an implementation of the T matrix method due to Kristensson (cf. [13]). Since all the figures we reconstruct are solids of

rotation with axis of rotation along the z axis, we first compute $F_a(\hat{\mathbf{x}}, k; \boldsymbol{\alpha})$ for $\boldsymbol{\alpha}$ lying in the $(\mathbf{x}, \mathbf{z})$ plane with spherical angles $\phi = 0$ and

$$\theta = \cos^{-1}\left(\frac{2i}{P+1} - 1\right), \qquad i = 0, \cdots, P+1.$$

We then obtain F_a for general $\boldsymbol{\alpha}$ by rotation. We thus obtain approximate Fourier coefficients of the far field pattern for incident directions

$$\begin{aligned} &\boldsymbol{\alpha}_{ij} = (\sin\theta_i \cos\phi_j, \sin\theta_i \sin\phi_j, \cos\theta_j), \\ &\theta_i = \cos^{-1}\left(\frac{2i}{P+1} - 1\right), \qquad i = 0, \cdots, P+1, \\ &\phi_j = \frac{2\pi j}{Q}, \qquad j = 1, \cdots Q. \end{aligned} \tag{3.4}$$

This gives a total of $PQ+2$ incoming waves with direction vectors $\boldsymbol{\alpha}_n$, $n = 1, 2, \cdots, PQ+2$.

To solve the inverse problem (without assuming that the scattering obstacle is a solid of rotation), we first solve (2.10) by approximating g by a finite Fourier series

$$\begin{aligned} g_a(\hat{\mathbf{x}}) = &\sum_{l=0}^{L_I} \sum_{m=0}^{l} g_{lm1} \gamma_l^m P_l^m(\cos\theta) \cos m\phi \\ &+ \sum_{l=1}^{L_I} \sum_{m=1}^{l} g_{lm2} \gamma_l^m P_l^m(\cos\theta) \sin m\phi. \end{aligned} \tag{3.5}$$

We always choose $L_I \leqq L_{\max}$ and $\min(P, Q) \geqq L_I$. Our approximation to (2.10) then becomes

$$\mu_1^a(M) = \min_{g_a} \sum_{n=1}^{PQ+2} \left| \left[\sum_{l=0}^{L_I} \sum_{m=0}^{l} F_{lm1}^n g_{lm1} + \sum_{l=1}^{L_I} \sum_{m=1}^{l} F_{lm2}^n g_{lm2} \right] - 1 \right|^2. \tag{3.6}$$

We have not required that $g_a \in U_1(M)$ but instead determine μ_1^a using a truncated singular value decomposition. The value of the truncation level is set by the user after inspecting the singular values of the matrix of coefficients of the far field patterns. The singular value decomposition (cf. [9]) was calculated using the NAG algorithms library routine F02WDF. Since $PQ+2 \geqq (L_I+1)^2$ and we use a truncated singular value decomposition, a single g_a is computed from which we then attempt to compute ρ by solving an approximate version of (2.11).

In order to compute ρ, we first form an approximation to v by using product integration to approximate the integral in (2.7). We use the trapezoidal rule on M_T subintervals for the ϕ (azimuthal) integration and the composite five point Gauss rule on M_G subintervals for the θ integration (cf. [17]). Thus our approximation v_a to v is given by

$$v_a(\mathbf{y}) = \left(\frac{2\pi}{M_T}\right)\left(\frac{1}{M_G}\right) \sum_{j=1}^{M_T} \sum_{l=1}^{M_G} \sum_{m=1}^{5} \left\{ A_m g_a(\hat{\mathbf{x}}(\theta_{lm}, \phi_j)) \exp\left[ik\hat{\mathbf{x}}(\theta_{lm}, \phi_j) \cdot \mathbf{y}\right]\right\} \tag{3.7}$$

where

$$\theta_{lm} = \cos^{-1}\left(\frac{2t_m + 2l - 1}{2M_G} - 1\right), \qquad \phi_j = \frac{2\pi j}{M_T} \tag{3.8}$$

and (t_m, A_m), $m = 1, \cdots 5$, are the five-point Gauss Legendre points and weights. We can now discretize (2.11) by replacing ρ by the finite Fourier series

$$\begin{aligned}\rho_a(\hat{\mathbf{x}}) = &\sum_{l=0}^{L_\rho} \sum_{m=0}^{l} \Gamma_{lm1} \gamma_l^m P_l^m(\cos\theta) \cos m\phi \\ &+ \sum_{l=1}^{L_\rho} \sum_{m=1}^{l} \Gamma_{lm2} \gamma_l^m P_l^m(\cos\theta) \sin m\phi.\end{aligned} \tag{3.9}$$

Then (2.11) becomes (using the same integration scheme as (3.7))

$$\begin{aligned}\mu_2^a(g_a) = \min_{\rho_a} &\left(\frac{2\pi}{M_T}\right)\left(\frac{1}{M_G}\right) \sum_{j=1}^{M_T} \sum_{l=1}^{M_G} \sum_{m=1}^{5} \\ &\left| A_m \left\{ v_a(\rho_a(\hat{\mathbf{x}}_{jlm})\hat{\mathbf{x}}_{jlm}) + \frac{\exp[-ik\rho_a(\hat{\mathbf{x}}_{jlm})]}{\rho_a(\hat{\mathbf{x}}_{jlm})} \right\} \right|^2\end{aligned} \tag{3.10}$$

where $\hat{\mathbf{x}}_{jlm}$ is given by (3.2) with $\theta = \theta_{lm}$ and $\phi = \phi_j$. We have not required that $\rho_a \in U_2$, although for higher dimensional approximations to ρ we recommend the use of a priori constraints. Problem (3.10) was solved by a quasi-Newton algorithm, the NAG algorithms library routine E04GCF.

We note that a major advantage of our algorithm for solving the inverse scattering problem is that the computation of the derivatives of the objective function in (3.10) with respect to the coefficients of ρ_a is trivial (although tedious). One can in principle also compute second derivatives and hence it is possible to use a full Newton's method if desired.

4. Numerical results. In this final section we shall show that the optimization algorithm outlined in §§ 2 and 3 can reconstruct a wide range of bodies from synthetic far field data. For these experiments we have used as our data the approximate far field patterns produced by the T matrix method (see § 3). Although all the bodies to be reconstructed are solids of rotation, no symmetry assumptions are built into our method for solving the inverse problem.

We shall now summarize our results in a series of tables and figures. In all the examples (except where explicitly stated otherwise) we have taken $L_{\max} = 8$, $L_I = 6$, $P = 7$ and $Q = 7$ (see (3.1), (3.4) and (3.5)). Thus g_a has 49 complex coefficients and is found using 51 incoming waves distributed according to (3.4). The choice of L_ρ (see (3.9)) is influenced by the choice of L_I and the level of truncation used in the truncated singular value decomposition solution of (3.6). Presumably, L_ρ should be less that L_I. However, the precise relationship between L_ρ and L_I needed to produce the least error is not known. The initial guess for the quasi-Newton method used to solve (3.10) is $\rho_a(\hat{\mathbf{x}}) = 1$.

Example 1. (a) Prolate Ellipsoid $x^2 + y^2 + (\frac{2}{3}z)^2 = 1$.

(b) Oblate Ellipsoid $(\frac{2}{3}x)^2 + (\frac{2}{3}y)^2 + z^2 = 1$.

For this example $M_T = 10$ and $M_G = 2$ (see (3.7)). Table 1 shows the data for the reconstructions of these bodies, and Fig. 1 shows that a visually excellent reproduction has resulted for the case of an oblate ellipsoid.

Example 2. (a) Long Cylinder $(x^2 + y^2)^5 + (\frac{2}{3}z)^{10} = 1$.

(b) Short Cylinder $((\frac{2}{3}x)^2 + (\frac{2}{3}y)^2)^5 + z^{10} = 1$.

For this example $L_\rho = 6$, $M_T = 15$ and $M_G = 3$. Table 2 shows the data for these reconstructions and Fig. 2 shows a visual reconstruction for the case of a long cylinder. We report results when $k = 1$ principally to record the conditioning of the matrix of

TABLE 1
Results for ellipsoids.

	Oblate ellipsoid				Prolate ellipsoid			
Wave number k	1	2	3	3*	1	2	3	3
L_ρ (see (3.9))	4	4	4	6	4	4	4	6
Condition number[1]	8.8×10^{9}	1.8×10^{6}	2.2×10^{4}	2.2×10^{4}	7.9×10^{11}	1.4×10^{8}	1.3×10^{6}	1.3×10^{6}
Minimum singular value[2]	$.3\times10^{-8}$	$.14\times10^{-4}$	$.12\times10^{-2}$	$.12\times10^{-2}$	$.29\times10^{-10}$	$.17\times10^{-6}$	$.18\times10^{-4}$	$.18\times10^{-4}$
SVD truncation level[3]	$.1\times10^{-5}$	0	0	0	$.1\times10^{-4}$	$.1\times10^{-4}$	0	0
Relative L_2 error (%)[4]	.062	.18	.087	.029	.062	.22	.13	.027

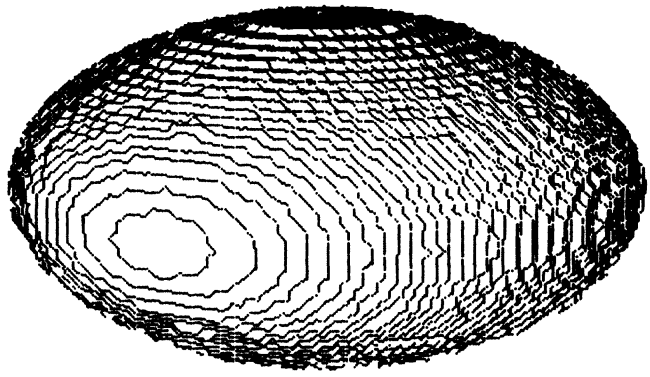
* Indicates that these parameters were used for visual reconstruction.

1. *Condition number*: This is an approximation of the spectral condition number of the matrix of coefficients of the far field pattern used in (3.6).

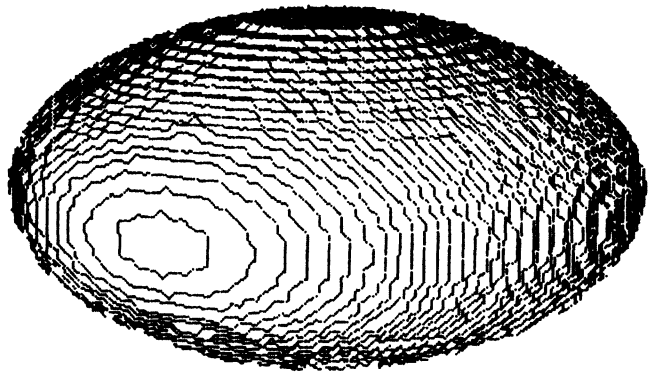
2. *Minimum singular value*: In all cases the largest singular value of the matrix of coefficients of the far field pattern was in the range [20, 30]. We quote the smallest singular value to show how the range of singular values changes with k.

3. SVD *truncation level*: Components in the singular value decomposition with singular values less than this value were not used in the computation of g_a.

4. *Relative L_2 error*: Let ρ^* be the best approximation in the L_2 sense to ρ from functions of the form (3.9). The relative L_2 error is then $\|\rho^* - \rho_a\|/\|\rho^*$ which we express as a percentage.



(a) Original



(b) Reconstruction

FIG. 1. *The oblate ellipsoid and its reconstruction when* $k = 3$.

coefficients of the far field patterns. When $k = 1$ we are forced to truncate the singular value decomposition, and since $L_\rho = L_I$ we would not expect a good reconstruction to result. The fact that a reasonable reconstruction is nevertheless achieved in this case may be due to the rotational symmetry of the body.

For the remaining examples we have not reported the results of computations for a range of values of the wave number k. Instead we show by a single set of results and pictures that a reasonable reconstruction has been achieved in each case.

Example 3. The Peanut $\rho^2(\theta) = \frac{9}{4}(\cos^2\theta + \frac{1}{4}\sin^2\theta)$. For this example $L_\rho = 4$, $M_T = 10$ and $M_G = 2$. Results are reported in Table 2 and Fig. 3. Some smoothing of the waist of the peanut has occurred.

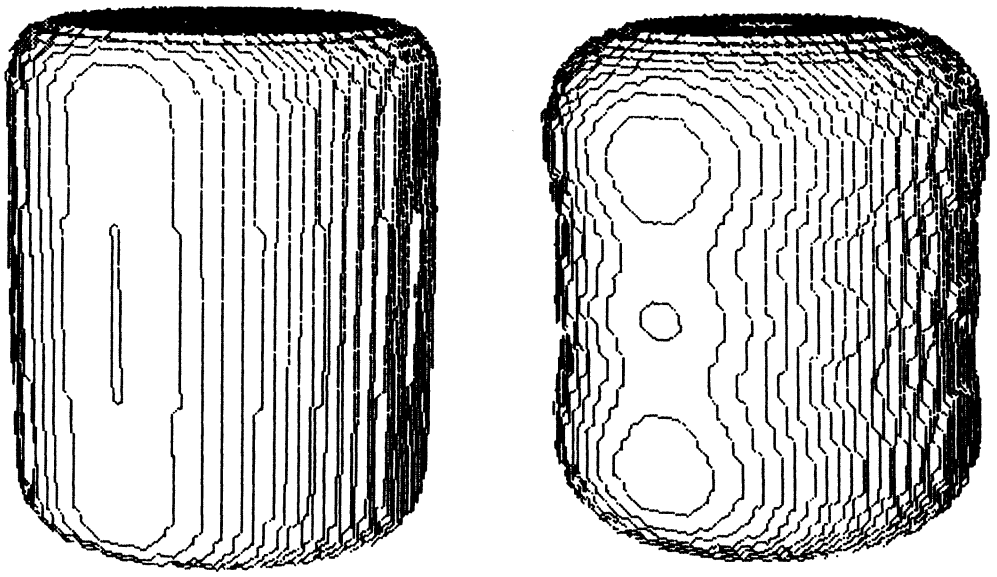
Example 4. The Acorn $\rho^2(\theta) = \frac{9}{25}(\frac{17}{4} + 2\cos 3\theta)$. For this example $L_\rho = 6$, $M_T = 10$ and $M_G = 2$. Results are reported in Table 2 and Fig. 4. Again some smoothing has occurred.

Example 5. The Cushion $\rho(\theta) = 1 - \frac{1}{2}\cos 2\theta$. In this case $L_I = 4$, $P = 5$, $Q = 5$, $L_\rho = 4$, $M_T = 10$ and $M_G = 2$. Note that ρ can be represented by functions of the form (3.9). Results are presented in Table 2 and Fig. 5. The surface is smoothed in the reconstruction so that the indentations at the poles are not present. Our experience in two dimensions [7] suggests that for this example the approximate far field pattern may be rather inaccurate, explaining why our reconstruction is not completely success-

TABLE 2
Results for the reconstruction of Examples 2 to 5.
(For definitions see Table 1.)

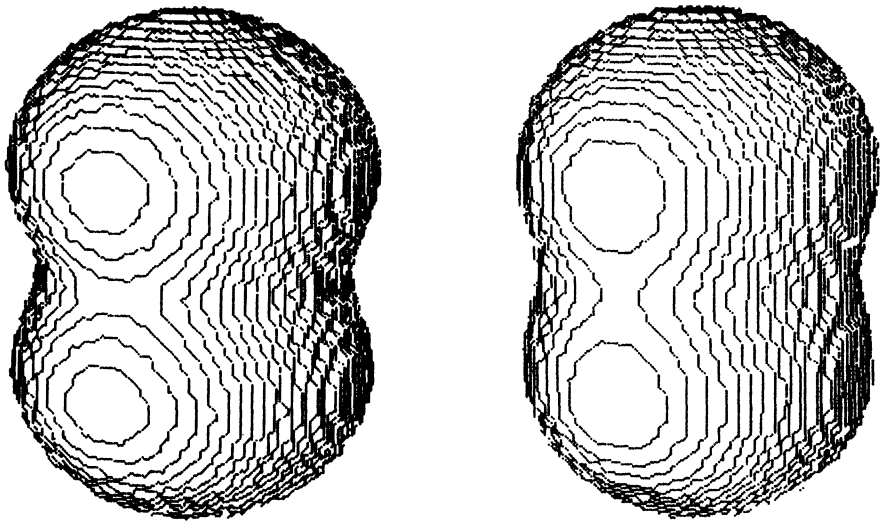
	Long cylinder			Short cylinder			Peanut	Acorn	Cushion	
Wave number k	1	2	3*	1	2	3	3*	3*	2	3*
L_ρ	6	6	6	6	6	6	4	6	4	4
M_T	15	15	15	15	15	15	10	10	10	10
M_G	3	3	3	3	3	3	2	2	2	2
Condition number	1.9×10^{11}	1.2×10^{7}	3.6×10^{6}	2.4×10^{9}	2.3×10^{5}	1.2×10^{4}	1.1×10^{6}	4.9×10^{4}	3.9×10^{3}	1.3×10^{3}
Minimum singular value	$.12\times10^{-9}$	$.21\times10^{-5}$	$.71\times10^{-5}$	$.10\times10^{-5}$	$.10\times10^{-3}$	$.25\times10^{-2}$	$.23\times10^{-4}$	$.54\times10^{-3}$	$.47\times10^{-2}$	$.18\times10^{-1}$
SVD truncation level	$.1\times10^{-5}$	0	0	$.1\times10^{-3}$	0	0	0	0	0	0
Relative L_2 error (%)	3.9	2.5	1.3	3.4	1.3	2.2	1.9	3.3	16	7.3

* Indicates that these parameters were used for visual reconstruction.



(a) Original (b) Reconstruction

FIG. 2. *The long cylinder and its reconstruction when* $k = 3$.

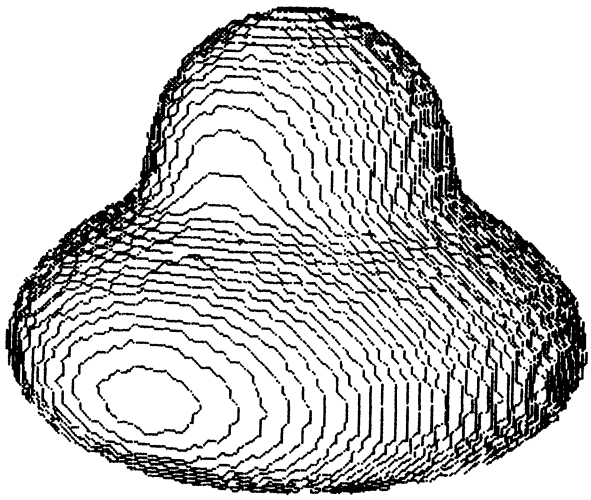


(a) Original (b) Reconstruction

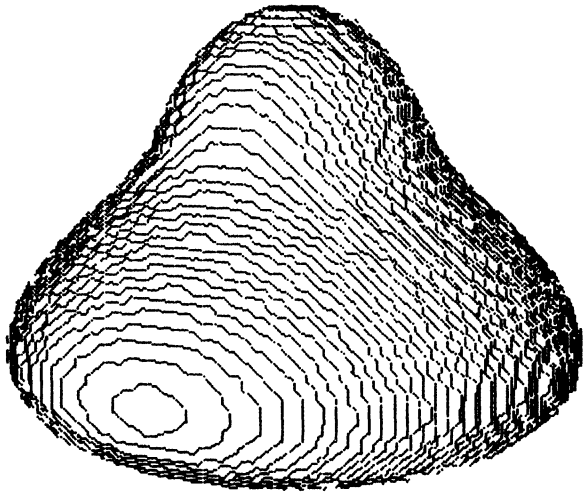
FIG. 3. *The peanut and its reconstruction when* $k = 3$.

ful. This highlights the problem we have experienced in obtaining reliable data for our inverse problem solver.

From these examples we can conclude that the inverse scattering algorithm described in § 2 can be implemented numerically. The detailed results from Examples 1 and 2 suggest that as the wave number k increases the conditioning of the matrix of far field coefficients improves. This improved conditioning should in general result in a more accurate estimate of g_a and hence a more accurate ρ_a. However, the T matrix



(a) Original

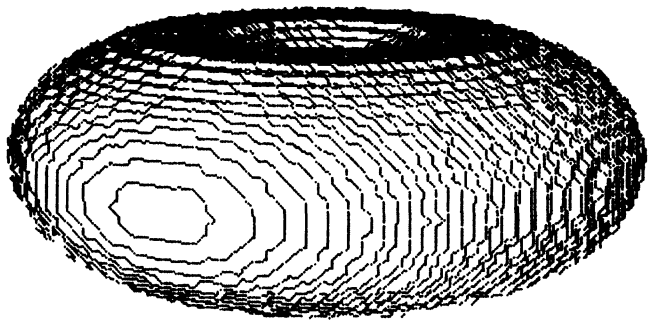


(b) Reconstruction

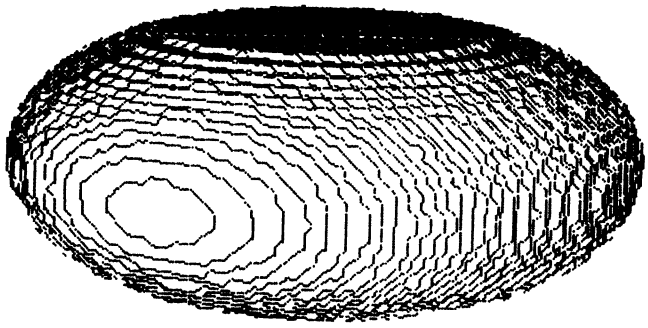
FIG. 4. *The acorn and its reconstruction when* $k = 3$.

method becomes less accurate as k increases and the inverse method performs poorly if k^2 is close to an interior eigenvalue of the domain. Thus the behavior of the error in ρ_a as k increases is difficult to predict. The singular values of the matrix of coefficients of the far field patterns show interesting patterns. In particular, there are many repeated singular values and their distribution seems fairly insensitive to k for a particular body. The connection between these singular values and the scatterer geometry is an interesting open problem since such singular values might be used to classify different scatterers.

We conclude this section by presenting results for the prolate ellipsoid and long cylinder in which random error has been added to the approximate far field pattern. This was accomplished by adding uniformly distributed random numbers in the range $[-\varepsilon, \varepsilon]$ to each coefficient in the matrix of coefficients of the far field patterns (a different random number was added to the real and imaginary parts of each coefficient). In both cases we have set $k = 2$ and the reader can consult Tables 1 and 2 for results



(a) Original



(b) Reconstruction

FIG. 5. *The cushion and its reconstruction when* $k = 3$.

in the absence of such noise (parameters used are as described in Tables 1 and 2 and Examples 1 and 2). The results of adding such error with various different values of ε are shown in Table 3. The method is surprisingly stable to random error and the proper use of a priori information would stabilize it still further.

Acknowledgment. We would like to thank Professor Gerhard Kristensson for making his code for the T matrix method available to us. The figures were drawn using ACM Algorithm 475 [19].

TABLE 3

Results for the reconstruction of the prolate ellipsoid and long cylinder at $k = 2$ *with added noise in the far field pattern. (For definitions see Table* 1 *and the text.)*

	Prolate ellipsoid			Long cylinder		
ε	0.001	0.01	0.1	0.001	0.01	0.1
$\max_{n,l,m,i} \lvert F^n_{lmi} \rvert$	5.6	5.6	5.6	6.56	6.56	6.56
Condition number	8.9×10^4	1.8×10^4	2.2×10^3	8.0×10^4	1.1×10^4	1.0×10^3
SVD truncation level	0.0	0.0	0.0	0.0	0.0	0.0
Relative L_2 error (%)	.47	2.5	8.3	3.8	3.4	7.8

REFERENCES

[1] T. S. ANGELL, D. COLTON AND A. KIRSCH, *The three dimensional inverse scattering problem for acoustic waves*, J. Differential Equations, 46 (1982), pp. 46–58.

[2] Y. M. CHEN AND S. L. WANG, *An efficient numerical method for determination of shapes, sizes and orientations of flaws for nondestructive evaluation*, in Review of Progress in Quantitative Nondestructive Evaluation: 4A, D. O. Thompson and D. E. Chimenti, eds., Plenum Press, New York, 1985, pp. 543–549.

[3] D. COLTON, *The inverse scattering problem for time harmonic acoustic waves*, SIAM Rev., 26 (1984), pp. 323–350.

[4] D. COLTON AND A. KIRSCH, *Dense sets and far field patterns in acoustic wave propagation*, SIAM J. Math. Anal., 15 (1984), pp. 996–1006.

[5] D. COLTON AND R. KRESS, *Integral Equation Methods in Scattering Theory*, John Wiley, New York, 1983.

[6] D. COLTON AND P. MONK, *A novel method for solving the inverse scattering problem for time harmonic acoustic waves in the resonance region*, SIAM J. Appl. Math., 45 (1985), pp. 1039–1053.

[7] ———, *A novel method for solving the inverse scattering problem for time harmonic acoustic waves in the resonance region* II, SIAM J. Appl. Math., to appear.

[8] D. COLTON AND J. WIMP, *Generalized Herglotz domains*, Math. Meth. Appl. Sci., to appear.

[9] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, 1984.

[10] P. HARTMAN, *On solutions of* $\Delta V + V = 0$ *in an exterior region*, Math. Z., 71 (1959), pp. 251–257.

[11] P. HARTMAN AND C. WILCOX, *On solutions of the Helmholtz equation in exterior domains*, Math. Z., 75 (1961), pp. 228–255.

[12] W. A. IMBRIALE AND R. MITTRA, *The two-dimensional inverse scattering problem*, IEEE Trans. Antennas and Propogation, AP-18 (1970), pp. 633–642.

[13] G. KRISTENSSON AND P. WATERMAN, *The T matrix for acoustic and electromagnetic scattering by circular disks*, J. Acoust. Soc. Amer., 72 (1982), pp. 1612–1625.

[14] D. I. OSTRY, *Synthesis of a shaped beam reflector antenna*, in The Application and Numerical Solution of Integral Equations, R. S. Anderssen et al., eds., Sijthoff and Noordhoff, The Netherlands, 1980.

[15] A. ROGER, *Newton-Kantorovich algorithm applied to an electromagnetic inverse problem*, IEEE Trans. Antennas and Propagation, AP-29 (1981), pp. 232–238.

[16] F. SANTOSA, Y. H. PAO, W. SYMES AND C. HOLLAND, *Inverse Problems of Acoustic and Elastic Waves*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1984.

[17] A. H. STROUD, *Approximate Calculation of Multiple Integrals*, Prentice-Hall, Englewood Cliffs, NJ, 1971.

[18] V. H. WESTON AND W. M. BOERNER, *An inverse scattering technique for electromagnetic bistatic scattering*, Canad. J. Physics, 47 (1969), pp. 1177–1184.

[19] T. WRIGHT, *Visible surface plotting program*, Comm. ACM, 17 (1974), pp. 152–155.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 3, May 1987

© 1987 Society for Industrial and Applied Mathematics
007

LOCAL UNIFORM MESH REFINEMENT WITH MOVING GRIDS*

WILLIAM D. GROPP†

Abstract. Local Uniform Mesh Refinement (LUMR) is a powerful technique for solving hyperbolic partial differential equations. However, many problems contain regions where numerical dispersion is very large, such as steep fronts. In these regions, mesh refinement is not very efficient. A better approach in these regions is to locally transform the coordinate system to move with the front. We show how to combine these two approaches in a way that maintains the advantages of LUMR and the effectiveness of moving grids. Experiments with 2-D scalar problems are presented.

Key words. moving grids, mesh refinement

AMS(MOS) subject classification. 65M50

1. Introduction. Hyperbolic partial differential equations typically contain regions of greater activity such as shocks or wave fronts, and these features are dynamic. This behavior, combined with the difficulty of providing a global uniform grid fine enough to resolve the smallest important length scale, has stimulated much work on adaptive grid techniques for hyperbolic problems. There are two major approaches. The first is to use an irregular grid, where the choice of grid points depends on some function of the solution. The grid may be continuous in time. The second is to use locally uniform grids, and to replace these grids frequently. In this case the grids are not continuous in time.

This second method is called Local Uniform Mesh Refinement (LUMR) and is a powerful technique for the solution of partial differential equations. It consists of adaptively refining a uniform coarse grid by overlaying it with uniformly refined subgrids. The uniformity is important in insuring that the overhead of managing the refinement is kept small, and that the refinement does not defeat any vectorizability which may be present in the discretization. This method has been shown to be efficient for a variety of one- and two-dimensional time dependent problems [1], [4], [7]. We discuss how to combine LUMR and moving grids in a way which substantially improves the performance of each of these algorithms at very little additional cost.

In § 2, we motivate this extension. In § 3, we give a detailed description of the algorithm. In § 4, we show computational experiments of this method, and compare it with uniform grid and regular LUMR calculations.

2. Motivation. In a numerical approximation to a hyperbolic PDE, the truncation error typically has leading form

$$T = C(t)k(h^p + k^p) \tag{2.1}$$

where $C(t)$ is independent of h and k. The usual approach in reducing the local error is to reduce h and k. However, since p is usually small ([8] argues that $p = 4$ is optimal for $u_t = u_x$), this is only moderately efficient. In particular, if the problem has n space dimensions, the work goes as (assuming $k/h = \lambda$ is constant) $h^{-(n+1)}$. Thus, to reduce the global error by a factor of 2 by using mesh refinement will require a decrease in h and k by $2^{1/p}$ and an increase in the work of $2^{(n+1)/p}$. Thus, the small p common in

* Received by the editors May 15, 1984; accepted for publication (in revised form) March 11, 1986. The work presented in this paper was supported in part by the Office of Naval Research under contract N00014-82-K-0814 and by the National Science Foundation under grant MCS-8106181.

† Department of Computer Science, Yale University, New Haven, Connecticut 06520.

the solution of hyperbolic problems makes mesh refinement relatively inefficient at reducing the error in the computation.

What alternatives are there? It is also possible to reduce the local error by reducing $C(t)$ in (2.1). We may be able to do this by a change of variables to a moving coordinate system, as $C(t)$ depends on the problem being solved. For example, consider the problem

$$u_t + au_x = 0, \tag{2.2}$$

approximated with the leap-frog scheme. The principle truncation error is

$$T = \tfrac{1}{3}(k^3 u_{ttt} + ah^3 u_{xxx}). \tag{2.3}$$

We can use (2.2) to eliminate u_{ttt} from (2.3) to get

$$T = \tfrac{1}{3}au_{xxx}(-a^2k^3 + h^3), \tag{2.4}$$

so that $C(t) \to 0$ as $a \to 0$. It is easy to pick a coordinate transformation $(x, t) \to (\tilde{x}, \tilde{t})$ which takes (2.2) to

$$u_{\tilde{t}} + 0 \cdot u_{\tilde{x}} = 0.$$

Moreover, leap-frog is exact for this problem.

The important thing to note in (2.4) is that reducing a by 2 reduces the truncation error by (roughly) 2. Thus, a significant savings in the effort required to reach a given error tolerance can be realized by finding an approximate transformation.

This is not a new approach; part of the success of the moving finite element method [9], [10] can be traced to this, as can the method of Davis and Flaherty [5]. What is new here is a way to reduce $C(t)$ in some cases at little additional cost in a mesh refinement code. We will not try to drive $C(t)$ to zero (though that is possible in exceptional cases); rather, we will apply uniform velocity transformations to each refined grid in an attempt to reduce $C(t)$ at low cost.

We also argue below that this is an effective way of reducing numerical dispersion in discontinuous or nearly discontinuous solutions. The best way to see this is to look at graphs of the numerical dispersion of a sample scheme on a model problem. We consider the leap-frog method applied to $u_t + au_x = 0$ for $0 \leqq a \leqq 1$. The model is $u_t + u_x = 0$ where a here is the result of shifting to a moving coordinate system. We let $k/h = \frac{1}{2}$. This gives the dispersion relation

$$\omega h = 2 \sin^{-1}(\tfrac{1}{2}a \sin(h\xi))$$

where the basic mode is $\exp(i(\omega t - \xi x))$. However, this is not quite what we want to compare. Because we are using a moving coordinate system, we actually have (in the original, stationary frame)

$$\omega h = 2 \sin^{-1}(\tfrac{1}{2}a \sin(h\xi)) + (1-a)h\xi.$$

Graphs of this for various values of a are shown in Fig. 1.

The importance of this is that even for $a = \frac{1}{2}$, the numerical dispersion is significantly reduced. Thus, even an approximate moving transformation can be very beneficial. Also note that this reduction in numerical dispersion extends right to the highest frequencies, so this applies to discontinuous solutions as well.

3. Algorithm. In this section we discuss the algorithm for moving LUMR (MLUMR) in two space dimensions. We first discuss the grid structure, then the regridding strategy, and finally the algorithm for advancing the solution one time step. Our algorithm is more in the flavor of [7] than [1] in that we do not allow the grids to be arbitrarily oriented.

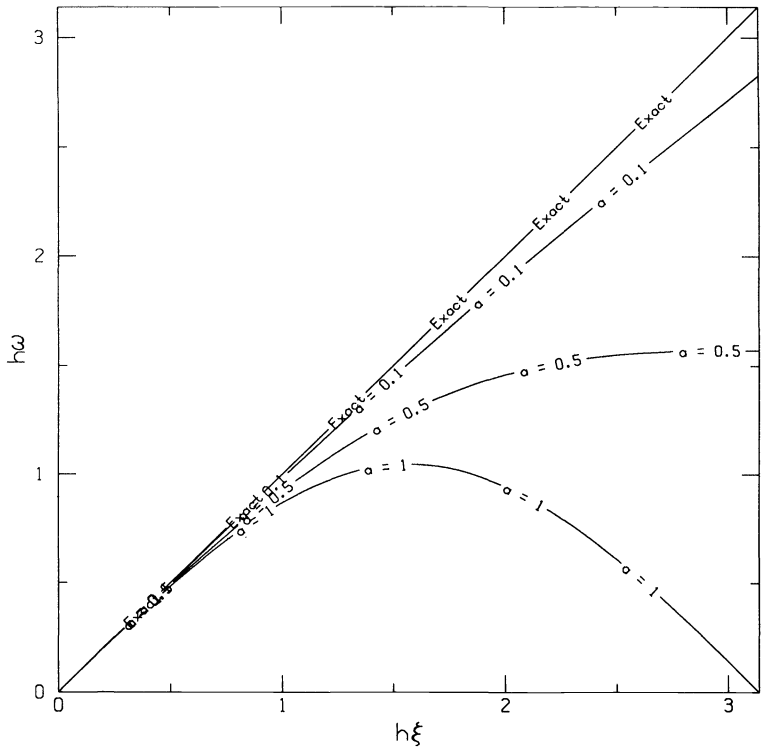


FIG. 1. *Dispersion relation for model problem for various values of a. Parasitic waves have been omitted.*

The algorithm is fully recursive, so multiple levels of refinement are easily accommodated.

3.1. Grid structure. The mesh refinement algorithm produces a series of *levels* of grids, G_l. The initial, user specified, level (corresponding to the spatial domain of the problem) is denoted G_0. All levels are made up of rectangular pieces, called grids, denoted G_{li} for the ith grid in level l. Further, each of these pieces is lined up with the usual cartesian directions. Finally, grids must be strictly nested: if a grid G_{li} lies in $G_{l-1,j}$, then all of G_{li} lies in $G_{l-1,j}$ and none of G_{li} lies in any other grid at level $l-1$. The grid $G_{l-1,j}$ is called the *parent* of G_{li}, and G_{li} a *child* of $G_{l-1,j}$. A sample grid structure is shown in Fig. 2.

Note that the grids G_{11} and G_{12} are two separate grids rather than a single grid lying partly in G_{01} and partly in G_{02}. This restriction significantly simplifies the data structures involved in managing the grids without imposing a restriction on the kinds of refinement.

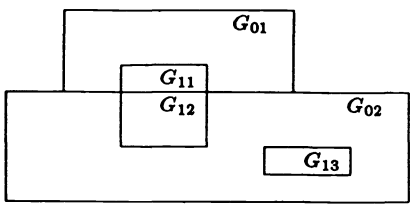


FIG. 2. *Sample grids. One level of refinement is present.*

Each grid G_{lr} contains a uniform mesh of points (x_i, y_j), $i = 1, \cdots, n, j = 1, \cdots, m$, and $x_i = x_0 + (i-1)h_l$, $y_j = y_0 + (j-1)h_l$, where (x_0, y_0) give the coordinates of the lower left corner of G_{lr}. The step sizes in space are h_l and the step sizes in time are k_l. If the boundaries of two grids abut, each contains the mesh points along the common boundary. Thus, points along a common boundary occur in two different places in memory; the algorithm will show how they are maintained with a single value. To simplify the programming, grids may only abut along rows.

It is important to note that the grids of one level overlay the grids of the coarser levels, rather than being patched into the coarser grids. This maintains the uniform nature of the grids, at a small cost in additional grid points. As the same point in the domain may lie in several grids, the algorithm must take steps to insure a single valued solution. Because the grids are regularly oriented, this can be done in an efficient, vectorizable fashion.

Each grid is also assigned a velocity $\vec{v}$. The entire grid will be moved uniformly at this velocity, with $\vec{v}$ changing as often as every time step. This velocity, as a vector, is arbitrary, and thus the grids are not required to move parallel to the coordinate axes. Because the grids move, they can run into each other and escape from their parent.

In the first case, there are two possibilities: one grid is overtaking the other, or two grids run into each other. In the case of one grid overtaking another, we allow the grids to overlap. Nothing else need be done, as this will still provide a good local match to the solution velocity. Alternately, we could force the grids to abut, and reset their velocities to the area-average of their respective velocities. In the case where two grids collide, this means that there is no local velocity transformation, and the error must be reduced by reducing the step sizes h and k. This can be handled naturally by MLUMR by regridding and generating an additional level of refinement as necessary (cf. Fig. 3).

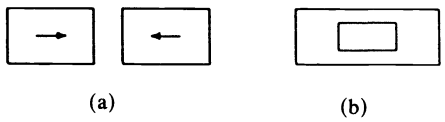


FIG. 3. *Colliding grids as in* (a) *are handled by changing to a stationary grid, with additional levels of refinement as necessary, as in* (b).

In the second case of grids escaping from their parent into another grid as in Fig. 4, we force the grid to stop at the parent boundary, and regrid to create a grid in the adjacent parent. Because the grid cannot move (relative to its parent), it may be necessary to refine it as well; again, this is handled in a natural fashion.

Another approach to this problem is to sacrifice strict nesting by parent and go to the looser *level nesting* discussed in [2]. This complicates the data structures. Because of the grid organization, only grids at the first level of refinement are likely to escape from their parent (a grid in the coarsest grid). The coarsest level will normally have very few components—only one in many cases. Grids at higher levels of refinement are likely to move with the same speed as their parent. Thus grids will rarely escape from their parents, and the additional cost of regridding when that happens is compensated for by the simplicity of handling the other grid-parent interactions.

3.2. Regridding. After some number of time steps on a level l, it is necessary to check to see if the refined grids at the finer levels need to be replaced with new refined

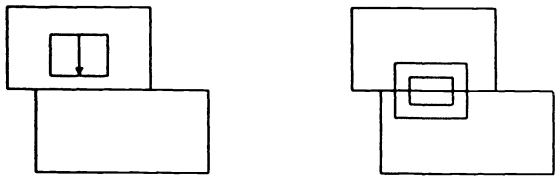


FIG. 4. *Grids leaving their parent as in* (a) *are handled by using stationary grids in the original parent and the new parent. An additional level of refinement, as in* (b), *may be necessary.*

grids. This is done in three steps. First, points in the grids at level l are *flagged* if it is necessary to refine about that point. This may be done by estimating the truncation error at that point or by some ad hoc method based on user knowledge of the behavior of the solution. Second, these flagged points are surrounded by a buffer of marked points. This buffer is needed to isolate the interior of the refined grid from the coarse grid, which is necessary given our strategy of overlaying the refined grids. Further, this buffer allows us to take more steps between regriddings by giving a margin for the region of higher error to move around in. Finally, the marked and flagged points are scanned to determine the new refined grids, and those grids are initialized with values from the previous refined grids (where they overlapped) and the parent grid.

The process of adding the buffers can be done by bit-wise logical operations of left and right shifts and or's. This helps keep the cost of the regridding down. The regridding process is shown in Fig. 5.

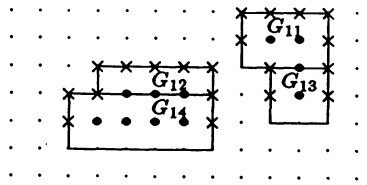
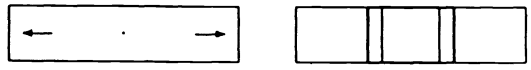


FIG. 5. *Regridding process.* · *denote mesh points,* • *denote flagged points,* × *denote buffer points. The resulting refined grids,* G_{11} *through* G_{14} *are outlined.*

3.3. Finding grid velocities. This is either the hardest or the easiest part, depending on what you plan to do. In the moving finite element (MFE) approach, the nodes move at a velocity automatically calculated to minimize a penalty function made up of the residual and some terms to keep the mesh well-behaved. In our approach, we ask the user to specify the velocity of a grid, given the values on the grid and its location. For many problems this is quite reasonable; detecting near-discontinuities and computing an appropriate approximate velocity is often quite simple. In fluids problems, the appropriate velocity is often the fluid speed, and is immediately available. It should be emphasized that because we are not trying to get an exact value for the velocity, we can be much more cavalier about choosing the velocity. For a completely automatic method, one similar to that in Miller and Miller [9] could be used. The difference between our method and Miller's is that we are trading an automatic velocity calculation from the PDE for a faster, possibly more accurate (particularly at moving

shocks) calculation based on the user's knowledge of the problem behavior. Another advantage is that we avoid the "spring and dashpot" parameters which are necessary in most automatic grid movement methods. Figure 6 shows how rapid velocity changes are handled.



(a) (b)

FIG. 6. *Grids over which the velocity varies greatly as in* (a) *are broken into several grids. In* (b), *the grid has been broken into three overlapping parts, with the center part* (*in thicker lines*) *remaining stationary.*

3.4. Integrating a level. Each level consists of a number of possibly intersecting grids. Here we describe how a single level is integrated; in the next section, we put everything together.

The boundaries between a refined grid and its parent are determined by using linear interpolation from the parent grid values. This has been shown to be stable [3] for stationary grids.

If the grids abut, then the boundary between them is handled in a special way. The points along the common boundary are duplicated in each grid. This is done to maintain the uniform structure of the grids. When integrating the top grid, the common boundary is also integrated, using data from both grids. Then the values computed on the common boundary are copied from the upper grid into the lower grid.

Overlapping grids require special care. In the case where the refinement is 1 (no refinement), overlapping grids can be handled efficiently by just injecting the values on the "refined" grids onto the parent grid, and then using the usual boundary procedure for fine/coarse grid interfaces to communicate the values back to the refined grids.

ALGORITHM 1. Moving Mesh Refinement algorithm for advancing the solution one time step.

procedure step (lev)

1. Integrate level lev.
2. Check for time to regrid the *next* level. If so, regrid the next level.
3. Update grid velocities if enough steps have passed. Must check this *after* the regridding check, since we do not want to reset velocities and immediately regrid.
4. Integrate the finer levels, if any:
   do $i = 1$, refinement (lev)
   call step (lev+1)
5. Having integrated the finer level, we can update this level from the next finer level.

If the refinement is greater than 1, then the boundary values must be computed by interpolating from the overlapping "partner."

As an example, we will describe how one time step is taken on a coarse grid, where there is only one level of refinement and no regridding takes place during this time step. First, the coarse grid is integrated from time t to $t+k_0$. Next, the refined grids are integrated from time t to $t+k_1$, then from $t+k_1$ to $t+2k_1$, and so on until they reach $t+k_0$. The boundary values at the fine/coarse grid interfaces are taken from interpolation of the coarse grid values at times t and $t+k_0$. Finally, the values on the fine grid at time $t+k_0$ are injected into the coarse grid to update the solution there.

3.5. Integration. Now that we have the structure of the grids and the regridding algorithm, we can describe the algorithm for integrating a level one step in time. This algorithm is recursive.

The algorithm described in Algorithm 1 is fairly straightforward. Step 5 is the only one we have not talked about. This step insures that the solution on all the grids is single-valued, and, more importantly, that those areas of the coarse grid which are refined take their solution from the refined grids. If this were not done at every step, there would be a danger that the inaccurate solution on the coarse grid in this area could escape and contaminate the solution everywhere. This contamination is prevented by a combination of finite domain of dependence of the differential equation and the difference approximation, the buffers surrounding the areas of large error, and the injection of the better solution computed on the fine grid into the coarser grids.

A more subtle point is the order of the steps in the algorithm. When the routine to regrid is called, the parent level has been integrated one step beyond the step where the regridding is to take place. This lets us place grids where they will be needed, rather than where they had been needed. Also note that when the regridding routine is called, all grids at *finer* levels no longer have a legitimate parent path (since they now point to nonexistent parents). This is acceptable, since those grids will also be regridded before they will be used to update any parent grid, and they will not be integrated (their only use is to initialize the new grids).

Because handling overlapping grids is complicated and inefficient, particularly if every grid had a different velocity, we may want to force adjacent grids which have nearly the same velocities to have the same (area averaged) velocity. This can significantly improve the performance of the algorithm. We used this approach in our program.

4. Experiments. These experiments are all in two space dimensions and for scalar problems. Each contains discontinuities in either the solution or its derivatives. A single level of refinement was used to show most clearly the effect of the uniformly moving grids. For a very high accuracy solution, additional levels of refinement would be necessary. Each was run on a FPS-164 running SJE and using the E Release of the FORTRAN compiler at optimization level 3. Each problem is computed using a uniform coarse and a uniform fine grid, and both LUMR and MLUMR. In each of the mesh refinement calculations, the level zero grid (the coarsest grid) was the same as the coarse-only calculation. The width of the finest mesh in each calculation is denoted by h_f. No more than 1 level of refinement was used. Errors in the solutions were measured on the coarse grid. Three different measures were used. The l_2-error is defined as $(n^{-1}\sum_{i=1}^{n} e_i^2)^{1/2}$, the l_1-error is defined as $n^{-1}\sum_{i=1}^{n}|e_i|$, and the l_∞-error is defined as $\max|e_i|$. The l_2-error is the appropriate measure for smooth problems. The l_1-error is more appropriate for nonsmooth problems, and should be the primary basis for comparison in these tests. Since in these problems the largest error occurs at discontinuities, then the l_2-error, which preferentially measures the larger errors, will be strongly affected by very small changes in the position of the discontinuities. The l_1-error, on the other hand, weights all errors equally and is a better measure of the numerical dispersion away from the discontinuities. The l_∞-error is easily contaminated by very small errors in positioning discontinuities and is useful only for continuous solutions.

As a first sample problem we take a variant of the revolving cone problem used in [6] and [2]. The problem is:

$$u_t - yu_x + xu_y = 0,$$

with the initial condition

$$u(x, y, 0) = \begin{cases} 1 - 16r & \text{if } r < \frac{1}{16}, \\ 0 & \text{otherwise,} \end{cases}$$

where

$$r = ((x - \tfrac{1}{2})^2 + \tfrac{3}{2}y^2).$$

The boundary conditions at the inflow boundaries are zero. The solution to this variable coefficient problem is given by rigid counter-clockwise rotation of the initial data about the origin with angular frequency $\omega = 1$. This is a good test of both regular and moving mesh refinement because fine difference approximations to this problem generate large, localized errors containing high frequency terms (due to the base and tip of the cone). For moving mesh refinement, there is the additional feature that the appropriate velocity varies substantially over the cone, so that a close match of the velocity of the solution and the fine grids is not possible.

In our test, we use Lax-Wendroff as the difference approximation, with inflow boundaries specified as $u = 0$ and outflow boundaries specified with first order extrapolation. A substantial effort was made to identify hotspots in the code, and those parts of the code were optimized. This was done to insure that these tests were realistic comparisons. For example, the execution time of the Lax-Wendroff scheme was improved by about 40%. The code is written to be efficient in all modes of operation (no refinement, unmoving refinement, or moving refinement), and is fully instrumented.

We can see from Table 1 that the MLUMR solution achieved significantly better results than LUMR at about $\frac{3}{4}$ the cost. Both LUMR and MLUMR were much faster than a uniform grid. The graphs of the solutions in Figs. 7 and 8 show that the error for both the uniform grid and the LUMR calculation is due primarily to numerical dispersion (the high frequency waves lag the solution, as is predicted by Fig. 1), while the error in the MLUMR solution is due to a combination of dispersive error (caused by the variation of velocity over the cone) and dissipation in the scheme. We can see from Fig. 2 that part of the efficiency advantage of MLUMR comes from needing less refinement (a consequence of the smaller dispersive error).

TABLE 1

Results for first test problem. Errors are computed on the coarse grid. Times are for entire program (STEP), *user integrator* (USINTG), *and boundary handling* (BNDY).

Method	h_f	l_2 error	l_1 error	l_∞ error	Time (seconds) STEP	USINTG	BNDY
Coarse	1/20	6.65E−2	2.19E−2	5.98E−1	3.73	3.32	0.33
Fine	1/80	1.38E−2	3.74E−3	1.22E−1	209.	204.	4.0
MR	1/80	1.40E−2	4.20E−3	1.22E−1	46.3	35.0	7.55
Moving MR	1/80	5.65E−3	1.39E−3	7.30E−2	35.4	25.8	6.20

The second test problem is

$$u_t + uu_x + uu_y = 0$$

on the unit square, with $0 \leqq t \leqq 0.6$. Note that under the rotation $x' = (x+y)/\sqrt{2}$, $y' = (-x+y)/\sqrt{2}$, the equation becomes $u_t + \sqrt{2}uu_{x'} = 0$; the solutions of this equation are well known. This is a very difficult problem for any mesh refinement scheme because so much of the domain needs to be refined.

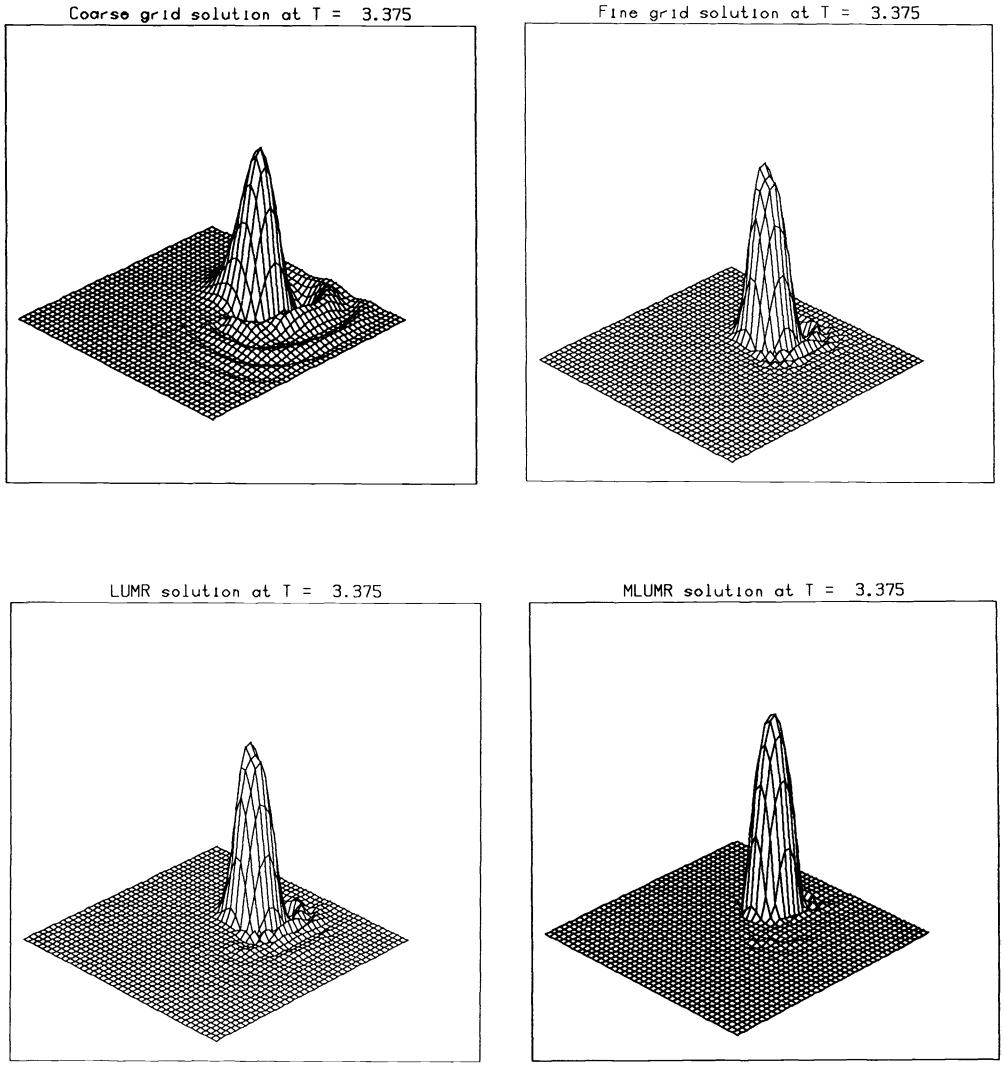


FIG. 7. *Solutions for revolving cone problem.*

The initial data for our second test problem is

$$u(x, y, t=0) = \begin{cases} \frac{1}{2} & \text{if } 0 \leqq x \leqq \frac{1}{2} \quad \text{and } 0 \leqq y \leqq \frac{1}{2}, \\ -\frac{1}{2} & \text{if } \frac{1}{2} < x \leqq 1 \quad \text{and } \frac{1}{2} < y \leqq 1, \\ \frac{1}{4} & \text{otherwise.} \end{cases}$$

The solution of this problem consists of six shocks, two moving with speed $\frac{3}{8}$, two with speed $\frac{1}{8}$, and two stationary. At two points (the problem is symmetric about $x = y$), three shocks with different speeds come together. Further, since this is a nonlinear problem, even a perfect choice for the speed of the local grids will not completely eliminate the error. This is because characteristics converge on the shock, and thus there is no unique local velocity.

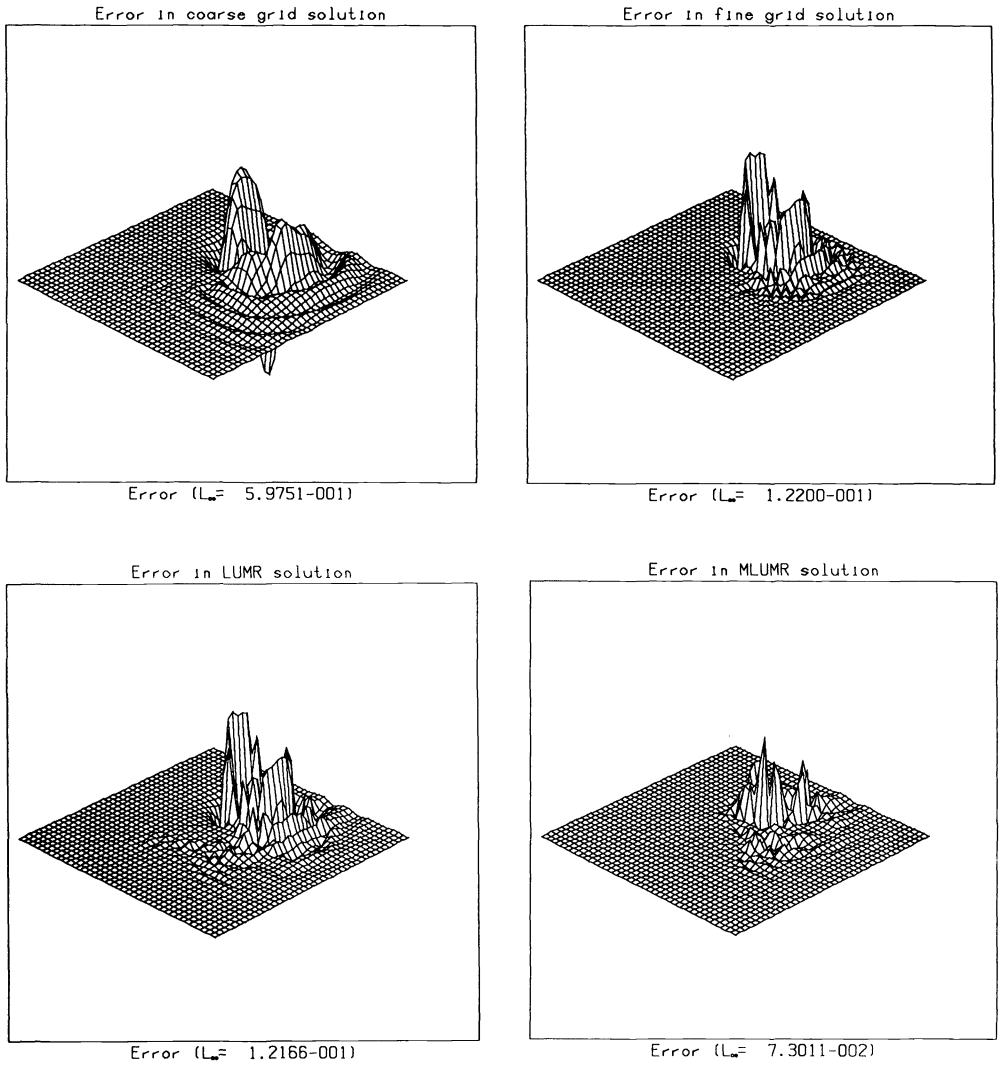


FIG. 8. *Errors in the solutions in Fig. 7.*

For this problem, the grid speeds are chosen nearly exactly; the speeds were set by the location and extent of the grid. If the grid covered several shocks, it was broken up into several overlapping pieces. Because the grid speeds were chosen nearly exactly, the errors in the computation come from only two places: the errors in fitting the moving grids to regions where multiple shocks interact, and the error in integrating a shock even in a co-moving reference frame. Thus, this example is complementary to the first example, where the error is due to range of velocities over the refined grid.

We can see from Table 2 that even without refinement, MLUMR gives a much better solution at a fraction of the cost of either LUMR or uniform grids. In this example, the l_1-norm is the most representative, as an error of one mesh point at a shock can make the l_∞-norm $O(1)$ and have significant impact on the l_2-norm. From Figs. 9 and 10 we see that the majority of the error in all computations is at the shocks, but that the "ringing" trailing the shocks is nearly absent in the MLUMR calculation.

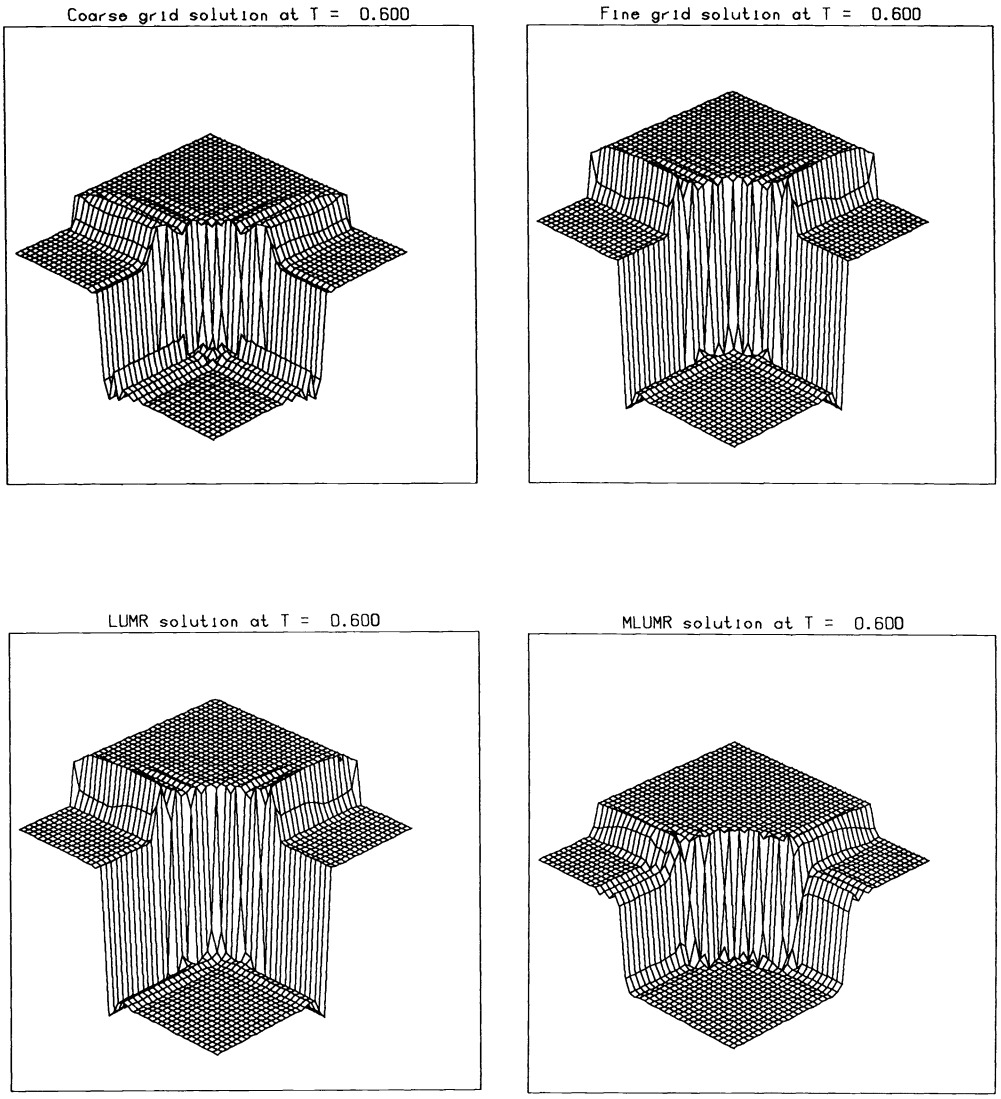


FIG. 9. *Solution for the second test problem.*

TABLE 2

Results for second test problem. Errors are computed on the coarse grid. Times are for entire program (STEP), *user integrator* (USINTG), *and boundary handling* (BNDY).

Method	h_f	l_1 error	l_2 error	Time (seconds) STEP	USINTG	BNDY
Coarse	1/40	1.84E−2	4.78E−2	.94	.88	.04
Fine	1/80	1.18E−2	4.56E−2	7.22	7.05	.14
MR	1/80	1.17E−2	4.47E−2	5.00	3.61	.66
Moving MR	1/40	1.07E−2	3.99E−2	2.45	1.27	.49
Moving MR	1/80	6.88E−3	3.29E−2	4.76	2.97	.97

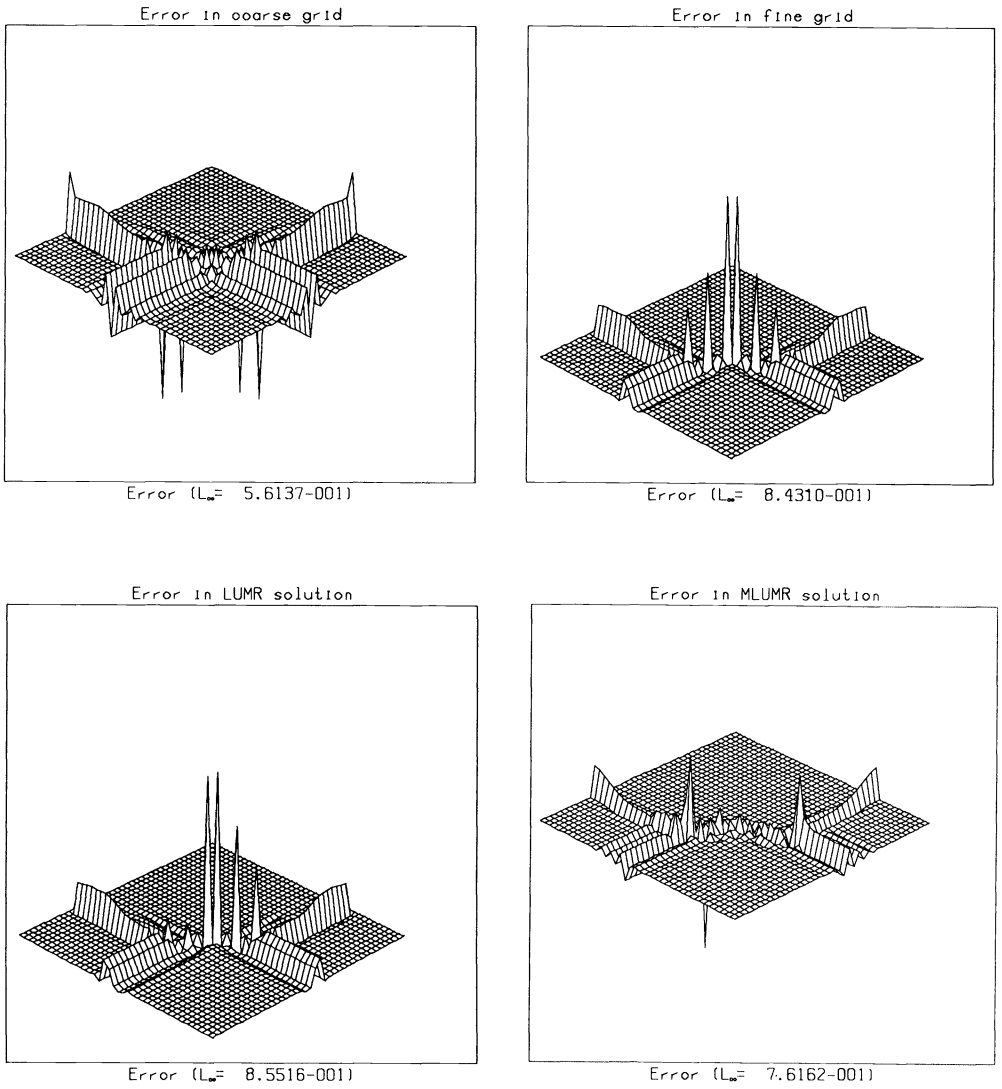


FIG. 10. *Errors in the solutions in Fig.* 9.

The highest error in the MLUMR calculation comes at the intersections of the shocks, where the MLUMR method cannot take advantage of any local frame of reference. Figure 11 shows the grids used in the two test problems.

In conclusion, we have shown that by taking advantage of the fact that numerical dispersion is caused in large part by a mismatch of grid velocity with solution velocity, we can reduce the error in a calculation at very low cost, both in terms of CPU time and program complexity. In addition, instead of forcing the user to pick values for mysterious parameters, the user is instead required to provide a local velocity function. This information is often available in terms of the interpretation of the solution (local wind speed as one of the components) or in terms of auxiliary conditions (such as the Rankine-Hugoniot conditions). Finally, this method easily generalizes to 3 or more dimensions.

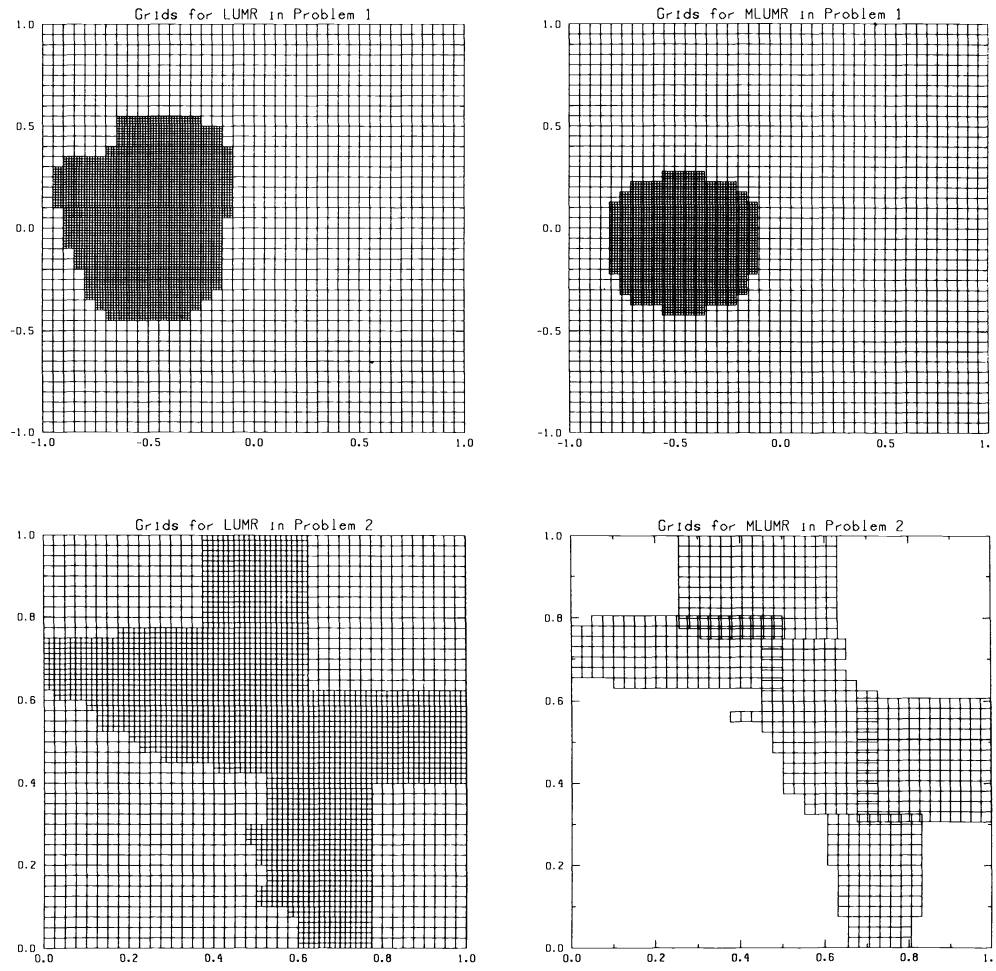


FIG. 11. *Grids for both test problems at the final time* ($T = 3.375$ *for problem* 1, $T = 0.6$ *for problem* 2).

REFERENCES

[1] MARSHA J. BERGER, *Adaptive mesh refinement for hyperbolic partial differential equations*, Ph.D. thesis, Stanford University, 1982.

[2] MARSHA J. BERGER AND JOSEPH OLIGER, *Adaptive mesh refinement for hyperbolic partial differential equations*, Technical Report Manuscript NA-83-02, Stanford University, March 1983.

[3] MARSHA J. BERGER, *Stability of interfaces with mesh refinement*, Technical Report 83-42, ICASE, August 1983.

[4] JOHN H. BOLSTAD, *An adaptive finite difference method for hyperbolic systems in one space dimension*, Ph.D. thesis, Stanford University, 1982.

[5] STEPHEN F. DAVIS AND JOSEPH E. FLAHERTY, *An adaptive finite element method for initial-boundary value problems for partial differential equations*, this Journal, 3 (1982), pp. 6-27.

[6] DAVID GOTTLIEB AND STEVEN A. ORSZAG, *Numerical Analysis of Spectral Methods: Theory and Applications*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1977.

[7] WILLIAM D. GROPP, *A test of mesh refinement for* 2-d *scalar hyperbolic problems*, this Journal, 1 (1980), pp. 191-197.

[8] H.-O. KREISS AND JOSEPH OLIGER, *Comparison of accurate methods for the integration of hyperbolic problems*, Tellus, 24 (1972), pp. 199-215.

[9] KEITH MILLER AND R. MILLER, *Moving finite elements*, I, SIAM J. Numer. Anal., 18 (1981), pp. 1019-1032.

[10] KEITH MILLER, *Moving finite elements* II, SIAM J. Numer. Anal., 18 (1981), pp. 1033-1057.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 3, May 1987
© 1987 Society for Industrial and Applied Mathematics
008

ACCURATE CONSERVATIVE REMAPPING (REZONING) FOR ARBITRARY LAGRANGIAN–EULERIAN COMPUTATIONS*

JOHN K. DUKOWICZ† AND JOHN W. KODIS‡

Abstract. The Arbitrary Lagrangian-Eulerian (ALE) method in computational fluid dynamics requires the periodic remapping of conserved quantities such as mass, momentum, and energy from a Lagrangian mesh to some other arbitrarily defined mesh. This procedure is a type of interpolation which is usually constrained to be conservative and monotone. It is typically carried out by solving a partial differential equation analogous to the continuity equation. Alternatively, the remapping may be carried out using an integral formulation which is (for the remapping of mass)

$$m_k = \iiint_{V_k} \rho(\mathbf{r})\, dV,$$

where m_k is the mass of a cell k of the new mesh whose volume is V_k, and $\rho(\mathbf{r})$ is the known density distribution on the old (Lagrangian) mesh. Remapping using this integral method avoids many drawbacks of the continuous method but the evaluation of such integrals is costly and difficult for arbitrary meshes. Recently, very efficient methods [6], [7] have been demonstrated for the case of constant cell density in two dimensions. We now extend these methods to the much more accurate case in which the density distribution in linear within each cell. This results in second order accuracy. However, to preserve monotonicity the gradients are limited using the option of two simple local limiters. To the extent that the limiting is applied the method locally reverts to first order accuracy.

Key words. partial differential equations, computational fluid dynamics, remapping, rezoning, interpolation, Lagrangian-Eulerian method

AMS(MOS) subject classifications. 35L60, 76N99, 76R99

1. Introduction.

Traditionally, numerical fluid dynamics has taken the form of Lagrangian or Eulerian methods. Lagrangian methods, in which the computational mesh travels with the fluid, are ideal for the many problems which involve interfaces between materials or free surfaces. However, multidimensional Lagrangian calculations can typically be carried out for only a limited time before severe mesh distortion, or even mesh tangling, destroys the calculation. Eulerian methods, in which the mesh is fixed, are ideal for flows with large deformation but the sharp resolution of interfaces or free surfaces is lost.

To overcome these individual limitations, methods have been developed which combine the features of both the Lagrangian and Eulerian methods. A representative method is the Arbitrary Lagrangian-Eulerian (ALE) method of Hirt et al. [1]. These methods permit arbitrary mesh motion, including the Lagrangian and the Eulerian limits. In particular, the mesh along material interfaces can be allowed to move with the fluid (or at least with the normal component of the velocity), thus preserving the sharp resolution of the interface, while away from the interface the mesh may be forced to move in such a way as to preserve mesh regularity (or even adaptively to better resolve some feature of the solution). To achieve this, these methods are typically organized to cycle between two phases: a Lagrangian phase in which the mesh moves in the Lagrangian manner, followed by the remapping (or rezoning) phase in which

* Received by the editors July 29, 1985; accepted for publication (in revised form) April 7, 1986.
† Theoretical Division, Group T-3, Los Alamos National Laboratory, Los Alamos, New Mexico 87545.
‡ Earth and Space Sciences Division, Group ESS-5, Los Alamos National Laboratory, Los Alamos, New Mexico 87545.

all the computational quantities defined on the mesh are transferred from the Lagrangian mesh to a second, arbitrarily defined mesh. It is with this remapping[1] phase that we will be concerned in this paper.

The remapping phase contains no physics (after all, the Lagrangian phase would be sufficient were it not for mesh distortion); it can be considered as simply an interpolation procedure in which the known distributions of quantities on one mesh, such as density, are interpolated to locations on the new mesh. However, for reasons of accuracy and robustness it is generally required that this interpolation procedure possess certain fundamental properties such as monotonicity and conservation. That is, we will require that the interpolation process does not introduce new maxima or minima beyond the range of the existing data (this will ensure the important property of positivity, that is, the method will not generate unphysical negative densities or energies), and that the procedure does not create or destroy integrated values of the remapped quantities. This is particularly important for the physically conserved quantities such as mass, momentum, and total energy.

We will show that this remapping procedure can be expressed by a partial differential equation equivalent to the usual convective or continuity equation. In the context of a remapping procedure, the numerical solution of this equation is known as a "continuous" remapping. There are well-known drawbacks to this procedure, however. In the first place, the numerical procedure has stability restrictions (the CFL condition) which may control the time step of the solution algorithm as a whole. Second, higher order implementations suffer from dispersive errors which cause characteristic oscillations or "wiggles" in the vicinity of steep gradients (a lack of monotonicity or positivity), while first order methods, such as donor cell, which are monotone, are excessively diffusive. Special monotone methods which control diffusion have also been developed [2], [3], [4]. All such methods based on the solution of the continuity equation have the advantage of relative simplicity, and this accounts for their widespread use, but the time step limitation and the inevitable numerical diffusion can be severe drawbacks.

There is an alternative procedure. It turns out that the above-mentioned differential equation, the continuity equation for remapping, is equivalent to a simple integrated form which is (e.g., for the remapping of density)

$$m_k = \iiint_{V_k} \rho(\mathbf{r})\, dV, \tag{1}$$

where m_k is the mass of cell k in the new mesh, V_k is its volume, and $\rho(\mathbf{r})$ is the known density distribution on the old mesh. The density distribution on the new mesh may then be constructed based on the average densities $\bar{\rho}_k = m_k / V_k$. This procedure is clearly conservative. It is also monotone provided the procedure for constructing the density distribution $\rho(\mathbf{r})$ does not create new extrema. Such a procedure may be called an integral remapping. It clearly has no time step limitation in the sense of the continuous remapping since the procedure defines the remapping from one arbitrary mesh to another in a single step, rather than in a sequence of small steps. The main drawback of this method is the difficulty of evaluating the integrals in (1) for arbitrary meshes. The simplest (and practically very important) case is the one in which the density within each cell is constant ($\rho(\mathbf{r}) = \bar{\rho}_k$, $\mathbf{r} \in V_k$), and the two meshes are composed

[1] The terms remapping and rezoning are often used interchangeably in this regard, but remapping is preferable in our sense of an interpolation process, with rezoning referring to the process of defining a new mesh.

of arbitrary quadrilaterals. This is the case for many Lagrangian codes. (Such a simple density distribution is clearly monotone and it can be shown to correspond to the use of donor cell differencing in the continuous remapping case, making it very diffusive, but this is compensated by the fact that relatively few remappings would be carried out.) In this simple case the problem reduces to evaluating the volume of overlap of arbitrarily intersecting cells. However, even this is logically very complicated. One way to overcome this logical complexity is to use Monte Carlo integration [5] but this is expensive if good accuracy is desired. A more accurate density distribution would produce a more accurate, less diffusive method, but in such a case direct integration would be prohibitive.

The problem of evaluating the integral in (1) has recently been greatly simplified in [6] by converting the volume integral to a surface integral through the appropriate use of the divergence theorem. This has made the general integral remapping a practical procedure. The method described in [6] is restricted to quadrilateral meshes, however, and involves the transformation to a square mesh in "logical" coordinates. An undesirable consequence of this transformation is the need to assign storage for several auxiliary constants per cell for each variable to be remapped. More recently, Ramshaw [7] improved this method by showing how to do the integration in physical space, avoiding the transformation, and thus eliminating the need for the additional storage. This also eliminated, at least in principle, the restriction to quadrilateral meshes.

These original papers [6], [7] were concerned only with the case of constant cell density. In this paper we combine the principal ideas of these papers and extend them to the case where the density distribution within each cell is linear. This greatly improves the accuracy of the method. However, the method is no longer intrinsically monotone. To alleviate this, the density gradients which we allow within each cell must be controlled in such a way as to ensure monotonicity. We describe two alternative simple limiters which achieve this.

We have stressed the application of most interest to us: the remapping of Lagrangian calculations. It is useful to point out that there is a significant number of other applications. For example, adaptive mesh calculations may require a large or discontinuous change in the mesh, which would require an integral remapping of the type described here. Another example is in the linking of two codes each of which has different meshes and therefore requires an accurate and conservative transfer of quantities from one to the other.

2. Theoretical description.

2.1. The remapping phase. We will start from the fundamental control volume description of the conservation equations of fluid dynamics [8, p. 41]:

$$\frac{d}{dt}\int_{V(t)} \rho\, dV + \int_{S(t)} \rho(\mathbf{u}-\mathbf{b})\cdot\mathbf{n}\, dS = 0,$$

$$\frac{d}{dt}\int_{V(t)} \rho\mathbf{u}\, dV + \int_{S(t)} \rho\mathbf{u}(\mathbf{u}-\mathbf{b})\cdot\mathbf{n}\, dS - \int_{S(t)} \boldsymbol{\tau}\cdot\mathbf{n}\, dS = 0,$$

$$\frac{d}{dt}\int_{V(t)} \rho E\, dV + \int_{S(t)} \rho E(\mathbf{u}-\mathbf{b})\cdot\mathbf{n}\, dS - \int_{S(t)} \mathbf{u}\cdot\boldsymbol{\tau}\cdot\mathbf{n}\, dS = 0,$$

where $V(t)$ is an arbitrary time-varying control volume, $S(t)$ is its surface, $\mathbf{b}$ is the surface velocity, and $\mathbf{n}$ is the outward unit vector normal to the surface. These equations express the conservation of the mass, momentum, and total energy, respectively, of the fluid in the control volume, where ρ is the density, $\mathbf{u}$ is the velocity, and E is the

specific total energy ($E = e + \frac{1}{2}\mathbf{u} \cdot \mathbf{u}$, where e is the specific internal energy). For the purposes of our discussion we have neglected to show other contributions to these equations, such as the heat flux $\mathbf{q}$ or the forces on the control volume other than the stress tensor $\boldsymbol{\tau}$; nor are we interested in other equations required to close this system, such as the constitutive relations for $\boldsymbol{\tau}$ or the equation of state.

These equations are frequently the starting point for conservative numerical methods (e.g., the finite volume method) in which the control volume is the computational cell and $\mathbf{b}$ is the mesh velocity. The case $\mathbf{b} = 0$ corresponds to Eulerian methods and the case $\mathbf{b} = \mathbf{u}$ to Lagrangian methods. The arbitrary Lagrangian-Eulerian methods decompose the above equations as follows:

(a) Lagrangian phase:

$$\frac{d}{dt}\int \rho \, dV = 0, \tag{2a}$$

$$\frac{d}{dt}\int \rho \mathbf{u} \, dV - \int \boldsymbol{\tau} \cdot \mathbf{n} \, dS = 0, \tag{2b}$$

$$\frac{d}{dt}\int \rho E \, dV - \int \mathbf{u} \cdot \boldsymbol{\tau} \cdot \mathbf{n} \, dS = 0. \tag{2c}$$

(b) Remapping phase:

$$\frac{d}{dt}\int \rho \, dV + \int \rho(\mathbf{u} - \mathbf{b}) \cdot \mathbf{n} \, dS = 0, \tag{3a}$$

$$\frac{d}{dt}\int \rho \mathbf{u} \, dV + \int \rho \mathbf{u}(\mathbf{u} - \mathbf{b}) \cdot \mathbf{n} \, dS = 0, \tag{3b}$$

$$\frac{d}{dt}\int \rho E \, dV + \int \rho E(\mathbf{u} - \mathbf{b}) \cdot \mathbf{n} \, dS = 0, \tag{3c}$$

where it is understood that the volume and surface integrals refer to the moving cell, in the former case to the Lagrangian cell, and in the latter to the cell moving with the relative velocity $(\mathbf{b} - \mathbf{u})$. This decoupling can be viewed as an operator splitting algorithm applied to the original equations [1], [9]. This viewpoint requires that the computation be carried out in a series of short time steps (whose magnitude is determined by stability considerations) such that each phase is immediately followed by the other. This type of remapping is known as continuous remapping, and it is intimately connected to the solution of a partial differential equation analogous to the continuity equation. To see this, consider a typical remapping equation

$$\frac{d}{dt}\iiint q \, dV + \iint q(\mathbf{u} - \mathbf{b}) \cdot \mathbf{n} \, dS = 0, \tag{4}$$

where q is the quantity being remapped, such as ρ, $\rho\mathbf{u}$, or ρE. Using the divergence theorem this equation may be converted to

$$\frac{d}{dt}\iiint q \, dV + \iiint \nabla \cdot q(\mathbf{u} - \mathbf{b}) \, dV = 0.$$

For an infinitesimal control volume V this equation becomes (by the mean value theorem)

$$\frac{d}{dt}(qV) + V\nabla \cdot q(\mathbf{u} - \mathbf{b}) = 0. \tag{5}$$

Since $dV/dt = V\nabla \cdot \mathbf{b}$ [7, p. 16], and $V \neq 0$, we obtain

$$\frac{dq}{dt} + \nabla \cdot q\mathbf{u} - \mathbf{b} \cdot \nabla q = 0. \tag{6}$$

Since $dq/dt \equiv \partial q/\partial t + \mathbf{b} \cdot \nabla q$, this is equivalent to

$$\frac{\partial q}{\partial t} + \nabla \cdot q\mathbf{u} = 0, \tag{7}$$

a continuity equation for q.

We now take a more general viewpoint in which the Lagrangian phase is seen as the solution of the original equations in a moving coordinate system, namely the one in which $\mathbf{b} = \mathbf{u}$. The purpose of the remapping phase is to perform an interpolation of the state variables ρ, $\rho\mathbf{u}$, ρE (assumed to be "frozen" functions of space only, having been determined by the Lagrangian phase) to some new, arbitrarily defined coordinate system (or mesh). We no longer need to define a mesh velocity $\mathbf{b}$, only the new mesh position. Since we are interested in cell-integrated quantities (2), (3), the remapping is trivially given by

$$Q_k = \iiint_{V_k} q(\mathbf{r})\, dV, \tag{8}$$

where V_k is the volume of the kth cell of the new mesh, and $q(\mathbf{r})$ is the known ("frozen") distribution of q on the old (Lagrangian) mesh. Further, as is clear from (8), there need not be a one-to-one correspondence between the cells of the two meshes; in fact the two meshes may be entirely arbitrary. As in any interpolation procedure, however, the new mesh must be at least contained within the old mesh, or else one must be able to extend the old mesh by means of boundary conditions so as to contain the new mesh. Notice that (8) only determines the average of the quantity q within each cell. This ensures the conservation properties of this procedure. To specify a distribution of q within each cell a separate procedure, based on these average values, must be provided (as we will do presently).

This type of remapping, which may be termed "integral" remapping, has several distinct advantages. It permits great flexibility since the Lagrangian equations may be integrated independently a variable number of time steps (limited by the permissible mesh distortion), and it is not limited by the magnitude of the required mesh change. The mesh may change size, shape, and even topology. The ability to make a large mesh change also enhances accuracy. It is well known that stable, monotone, continuous remappings are very diffusive due to the large number of remapping steps required. Since the integral remapping is able to accommodate a large mesh change in a single step, such loss of accuracy is avoided. However, any mesh distortion degrades the accuracy of the solution. Therefore, these two counterweighing effects must be balanced by an appropriate choice of the allowable mesh distortion.

2.2. The remapping integral. The direct implementation of the remapping expressed by (8) is difficult because first, the integration of a general density distribution $q(\mathbf{r})$ over a general spatial domain is very complicated and second, it is rather difficult to keep track of the overlap of cells of one mesh with cells of the other. This is the main reason for the popularity of the "continuous" remapping, based on the solution of (5) (or (7)), as compared to the integral remapping, because these difficulties are effectively minimized by keeping the displacement of one mesh small with respect to the other. This situation was changed with the development of the method of [6] in

which the volume integration of (8) was converted to surface integration permitting a much easier and more efficient method of solution.

This method may be summarized as follows. We wish to find a vector function **F** such that we can write

$$\iiint_{V^*} q(\mathbf{r})\, dV = \iiint_{V^*} \nabla \cdot \mathbf{F}\, dV \tag{9}$$

for an *arbitrary* volume V^* and a known distribution $q(\mathbf{r})$. Having such a function, we can apply the divergence theorem

$$\iiint \nabla \cdot \mathbf{F}\, dV = \iint \mathbf{n} \cdot \mathbf{F}\, dS$$

and therefore evaluate (8) efficiently as

$$Q_k = \iint_{S_k} \mathbf{n} \cdot \mathbf{F}\, dS. \tag{10}$$

The necessary and sufficient conditions for (9) to be valid for arbitrary volumes of integration are

$$\nabla \cdot \mathbf{F} = q(\mathbf{r}) \tag{11}$$

and

$$\mathbf{n} \cdot \mathbf{F}_1 = \mathbf{n} \cdot \mathbf{F}_2, \tag{12}$$

where $\mathbf{F}_1$ and $\mathbf{F}_2$ are the values of **F** on either side of a discontinuity. That is, the normal component of **F** must be continuous across any surface of discontinuity (the discontinuity can only occur in the tangential component). This condition arises because otherwise the divergence of **F** will have delta function type contributions to the integral in (9), which, however, are assumed to be absent in the distribution $q(\mathbf{r})$.

The two conditions on **F** ((11), (12)) do not determine **F** uniquely. A practical choice is obtained by choosing **F** to have only one component, i.e., $\mathbf{F} = (P, 0, 0)$. However, as a result of (12) this choice requires the use of a transformed coordinate system in which cell faces of the old mesh (which determines **F**) are perpendicular to coordinate directions. For simplicity, from now on we confine ourselves to two dimensions. In two dimensions such a transformation will map (x, y) to (i, j), and this mapping is characterized by the Jacobian of the transformation $J(i, j)$. The simplest such transformation is the bilinear transformation which takes an arbitrary quadrilateral into the unit square. Because of the relationship to the conventional index notation for logically rectangular meshes, (i, j) are known as the "logical" coordinates. This also limits the method to logically rectangular meshes, i.e., meshes of arbitrary quadrilaterals. The method is now summarized by the following equations, corresponding to (11) and (10):

$$\frac{\partial P}{\partial i} = q(i, j) J(i, j), \tag{13}$$

$$Q_k = \oint_{C_k} P\, dj, \tag{14}$$

where C_k is the contour around cell k of the new mesh, and the integration takes place in the positive, or counterclockwise, direction. The integration of (13), holding j fixed, is relatively straightforward since in general both q and J are simple polynomials in

i and j. Nevertheless, the continuity of P across cell boundaries, to satisfy condition (12), requires us to store certain cell constants representing the integration constants that arise in the integration of (13). In (14) each line segment of the new mesh makes a contribution to the two cells on either side of it. The integration is most efficiently carried out by tracing out entire mesh lines of the new mesh through the cells of the old mesh, and updating the appropriate cells. The mesh topology of the old mesh determines which cell is entered upon the mesh line crossing any given cell face. In practice the procedure requires two passes, one through each mesh. One pass through the old mesh is required to set up $P(i, j)$ in each cell, and another pass along the mesh lines of the new mesh is required to integrate (14).

The restriction to quadrilateral meshes and the need to store extra cell constants are certainly undesirable. They both result from the need to satisfy condition (12). Recently, Ramshaw [7] has pointed out that it is possible to relax this condition by excluding the delta functions at the discontinuities in **F** from the integration domain. This is illustrated in Fig. 1. Since there is now no need for a coordinate transformation, the method may be carried out in physical coordinates as follows:

$$\frac{\partial P}{\partial x} = q(x, y), \tag{15}$$

$$Q_k = \oint_{C_k^*} P\, dy, \tag{16}$$

where C_k^* is the contour around cell k of the new mesh *excluding* the surfaces of discontinuity of P (cell faces of the old mesh) that pass through cell k. The method is now simpler not only because (15) is simpler than (13) due to the absence of the Jacobian, but primarily because the function $P(x, y)$ can have arbitrary discontinuities at the cell boundaries so that the integration of (15) is local to each cell. This means that integration constants are irrelevant and need not be saved, in contrast to the previous method. It is also obvious that mesh topology is arbitrary and the method is

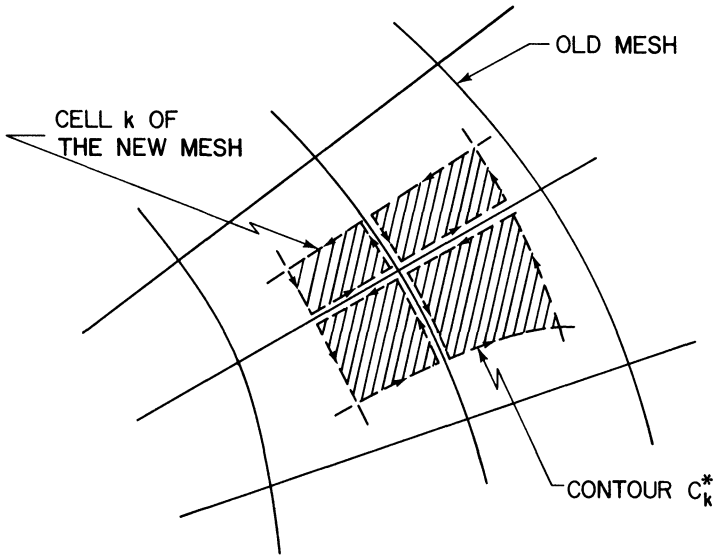


FIG. 1. *The integration contour around a cell of the new mesh, showing how surfaces of discontinuity (cell faces of the old mesh) are excluded.*

not confined to a mesh of quadrilaterals, although that may be its most common application. In practice the method is again applied in two passes: one pass along the mesh lines of the new mesh and another pass along the mesh lines of the old mesh to subtract out the contributions of the delta functions at the discontinuities. Each of these passes is entirely analogous to the sweep along mesh lines of the previous method. We shall discuss aspects of the practical implementation later, but here we mention the one negative aspect as compared to the previous method, and that is the need for very careful handling of the case when cell faces of the new and old meshes coincide.

An alternative to (15), (16) is obtained if we assume the other component of $\mathbf{F}$ to be zero (i.e., $\mathbf{F}=(0, P)$):

$$\frac{\partial P}{\partial y}=q(x, y), \tag{17}$$

$$Q_k=-\oint_{C_k^*} P\,dx. \tag{18}$$

This set of equations may be preferable to (15), (16) depending on the form of $q(x, y)$.

2.3. The density distribution. In remapping and using (15), (16) (or (17), (18)), which are equivalent to (8), we assume that the density distribution $q(\mathbf{r})$ is given within each cell of the old mesh. This density distribution must obviously be obtained from the available values of Q_k. The accuracy of the remapping depends on how this density distribution is determined.

The simplest case occurs when we assume that the density is constant within each cell, equal to the average density, i.e., $q(\mathbf{r})=\bar{q}_k=Q_k/V_k$. This is the most common assumption, and at the same time the least accurate, since it results in a very diffusive method. It has the advantage, however, that the method is automatically monotone, and, if the total number of remappings is low then the amount of diffusion may not be excessive. This is the type of remapping described in [6] and [7].

In this paper we introduce a more accurate density distribution which leads to a method that is substantially less diffusive, but which is still monotone. Any density distribution $q(\mathbf{r})$ that we specify must satisfy a constraint to preserve conservation, namely, that it be consistent with the known average value in each cell:

$$\iiint_{V_k} q(\mathbf{r})\,dV=\bar{q}_k V_k=Q_k. \tag{19}$$

The distribution that we propose is

$$q(\mathbf{r})=\bar{q}_k+\nabla_k q\cdot(\mathbf{r}-\bar{\mathbf{r}}_k), \tag{20}$$

where $\nabla_k q$ is a gradient of q in cell k, to be specified later, and $\bar{\mathbf{r}}_k$ is the centroid of cell k. Since the centroid is defined as

$$\bar{\mathbf{r}}_k\equiv\frac{1}{V_k}\iiint_{V_k}\mathbf{r}\,dV,$$

it is immediately obvious that our proposed distribution is consistent with (19). Equation (20) is equivalent to the first two terms in the Taylor series expansion of $q(\mathbf{r})$ about the point $\bar{\mathbf{r}}_k$ (and is therefore second order accurate) if it is assumed that the cell average $\bar{q}_k$ is located at the centroid and $\nabla_k q$ is at least a first order ($\mathcal{O}(\mathbf{r}-\bar{\mathbf{r}}_k)$, $\mathbf{r}\in V_k$) approximation to the gradient.

Such an approximation to the gradient can be obtained by defining a secondary mesh whose nodes or vertices are the centroids of the primary mesh. The gradient is approximated as the volume average (except in cylindrical coordinates, explained later)

$$\langle \nabla_k q \rangle = \frac{1}{V_j} \iiint_{V_j} \nabla q \, dV, \tag{21}$$

where the averaging volume V_j is the volume in the secondary mesh defined by the centroids of all the neighbors of cell k. This is illustrated in Fig. 2. The numerical evaluation of the above integral is facilitated by the use of the divergence theorem to obtain

$$\langle \nabla_k q \rangle = \frac{1}{V_j} \iint_{S_j} q\mathbf{n} \, dS, \tag{22}$$

where the value of q is required only along the surface S_j of volume V_j. We obtain a first order approximation if we assume a linear variation of q along the surface, since (22) then gives the exact result if q is a linear function of $\mathbf{r}$. The numerical implementation, in particular the boundary conditions used, will be described later. This value of the gradient cannot be used directly in (20), however, since it would result in a nonmonotone method. One might expect, especially in the vicinity of steep gradients, that (20) could produce large overshoots (or undershoots) when compared to neighboring data.

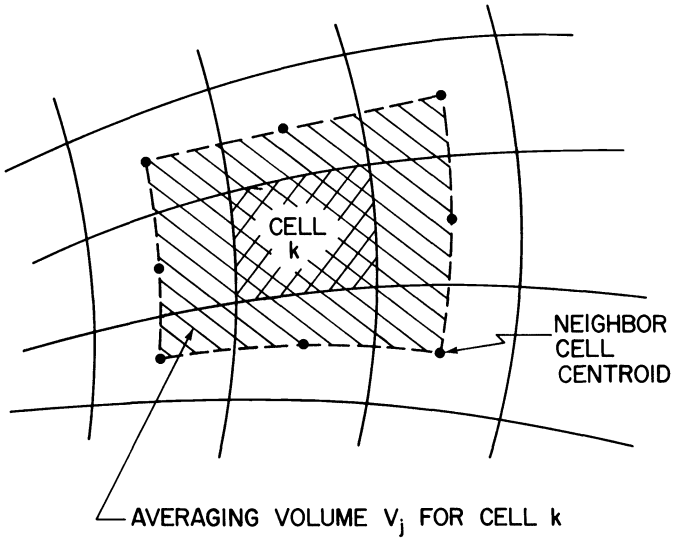


FIG. 2. *The averaging volume used to calculate the gradient for cell* k.

To prevent such overshoots, we limit the value that the gradient is allowed to take. Of course, when the gradient is changed from the value given by (22) the accuracy of the density given by (20) is reduced from second order to first order. Such limiting was first introduced in the one-dimensional case by van Leer [4]. It is also closely related to the flux limiting employed in the FCT [2], [3] method. We now describe two methods of limiting that we have employed.

2.3.1. Van Leer limiting. This is the multidimensional extension of van Leer's one-dimensional method. We introduce a limiting coefficient $\alpha_k (0 \leqq \alpha_k \leqq 1)$ for each cell k such that

$$\nabla_k q = \alpha_k \langle \nabla_k q \rangle$$

and α_k is determined by enforcing local monotonicity such that the density in cell k does not lie outside the range of the average densities in the neighboring cells. This determines α_k to be

$$\alpha_k = \min \{1, \alpha_{\max}, \alpha_{\min}\},$$

where

$$\alpha_{\max} = \max \{0, (\bar{q}_{\max} - \bar{q}_k)/(\bar{q}_{k\max} - \bar{q}_k)\},$$
$$\alpha_{\min} = \max \{0, (\bar{q}_{\min} - \bar{q}_k)/(\bar{q}_{k\min} - \bar{q}_k)\},$$

and $\bar{q}_{\max}, \bar{q}_{\min}$ are the maximum and minimum values of $\bar{q}$ in the neighboring cells, and $q_{k\max}, q_{k\min}$ are the maximum and minimum values of q in cell k. This limiter can still produce a "sawtooth" type of distribution of q at cell boundaries. That is, while the distribution is monotone with respect to neighboring cell averages, it is not necessarily monotone with respect to neighboring distributions of q.

2.3.2. Monotone limiting. This alternative procedure ensures absolute monotonicity, at least in the one-dimensional case. We introduce test gradients $\langle \nabla_{ik} q \rangle$ that are calculated according to (22) and are associated with each vertex i of cell k. Then we find the maximum and minimum among these gradients:

$$(\nabla_k q)_{\max} = \max \{\langle \nabla_{ik} q \rangle, i = 1, 2, \cdots, N_v\},$$
$$(\nabla_k q)_{\min} = \min \{\langle \nabla_{ik} q \rangle, i = 1, 2, \cdots, N_v\},$$

where N_v is the number of vertices of cell k (for a quadrilateral $N_v = 4$). The limiting is given by

$$\nabla_k q = \begin{cases} (\nabla_k q)_{\min}, & (\nabla_k q)_{\min} > 0, \\ (\nabla_k q)_{\max}, & (\nabla_k q)_{\max} < 0, \\ 0 & \text{otherwise.} \end{cases}$$

This limiting is of course done for each component separately.

The van Leer limiting permits the steepest possible gradient without undue oscillations, and therefore the least amount of diffusion or smearing. We have found in practice that the van Leer limiter is satisfactory in almost all cases. The monotone limiter is more conservative and therefore more diffusive. It is an option in those few cases in which the van Leer limiter might produce undesired local oscillations.

3. Practical implementation. The most general case that we consider in two dimensions is the case of cylindrical symmetry. In cylindrical coordinates (r, z) the remapping equation is

$$m_k = 2\pi \int\int_{V_k} \rho(r, z) r \, dr \, dz.$$

Customarily, we drop the factor of 2π and consider quantities to be defined per unit radian. We can then combine the Cartesian and cylindrical symmetry cases by introducing pseudo-Cartesian coordinates (x, y) as follows

$$q(x, y) = \rho(x, y) R(x), \tag{23}$$

where

$$R(x) = \beta + (1-\beta)x,$$

$$\beta = \begin{cases} 1, & \text{Cartesian,} \\ 0, & \text{cylindrical,} \end{cases}$$

and x corresponds to the radial coordinate in the cylindrical case. The density distribution in cell k of the old mesh is given by

$$\rho_k(x, y) = \bar{\rho}_k + \frac{\partial \rho_k}{\partial x}(x - \bar{x}_k) + \frac{\partial \rho_k}{\partial y}(y - \bar{y}_k),$$

where $\bar{\rho}_k = m_k / V_k$, $(\partial \rho_k / \partial x, \partial \rho_k / \partial y)$ are the components of the limited gradient in cell k, and $(\bar{x}_k, \bar{y}_k)$ are components of the centroid (appropriately computed in the cylindrical case). Because of the functional form of $q(x, y)$ given in (23), (17), (18) give a somewhat simpler result, and the flux function P in cell k becomes

$$P_k(x, y) = (a_1^k + a_2^k x + a_3^k x^2) y + \tfrac{1}{2}(a_4^k + a_5^k x) y^2, \tag{24}$$

where

$$a_1^k = \beta \left(\bar{\rho}_k - \frac{\partial \rho_k}{\partial x} \bar{x}_k - \frac{\partial \rho_k}{\partial y} \bar{y}_k \right),$$

$$a_2^k = \beta \frac{\partial \rho_k}{\partial x} + (1-\beta) \left(\bar{\rho}_k - \frac{\partial \rho_k}{\partial x} \bar{x}_k - \frac{\partial \rho_k}{\partial y} \bar{y}_k \right),$$

$$a_3^k = (1-\beta) \frac{\partial \rho_k}{\partial x}, \quad a_4^k = \beta \frac{\partial \rho_k}{\partial y}, \quad a_5^k = (1-\beta) \frac{\partial \rho_k}{\partial y}.$$

These coefficients are calculated as needed within each cell. We do, however, save $\partial \rho_k / \partial x$, $\partial \rho_k / \partial y$, $\bar{x}_k$, $\bar{y}_k$ in temporary working arrays.

The integral in (18) is composed of contributions from straight line segments. We consider an arbitrary straight line segment contained in cell k whose end points are (x_1, y_1) and (x_2, y_2) and whose direction of integration is from point 1 to point 2. It is convenient to introduce a local parameter s $(0 \leqq s \leqq 1)$ along the line segment such that

$$x = x_1 + (x_2 - x_1)s, \qquad y = y_1 + (y_2 - y_1)s, \tag{25}$$

and

$$dx = (x_2 - x_1)\, ds,$$

and so the segment contribution is

$$\Delta_1^2 Q^k = (x_1 - x_2) \int_0^1 P_k(s)\, ds. \tag{26}$$

It is obvious that vertical line segments $(x_2 = x_1)$ make no contribution, and thus it is possible to take advantage of this in certain types of meshes. Substituting (25) into (24), the segment contribution (26) can be explicitly evaluated as

$$\Delta_1^2 Q^k = (x_1 - x_2) \left[\frac{a_1^k}{2}(y_1 + y_2) + \frac{a_2^k}{6}\{y_1(2x_1 + x_2) + y_2(x_1 + 2x_2)\} \right.$$

$$
\begin{aligned}
&+\frac{a_3^k}{12}\{y_1(3x_1^2+2x_1x_2+x_2^2)+y_2(x_1^2+2x_1x_2+3x_2^2)\}\\
&+\frac{a_4^k}{12}\{y_1(2y_1+y_2)+y_2(y_1+2y_2)\}\\
&+\frac{a_5^k}{24}\{x_1(3y_1^2+2y_1y_2+y_2^2)+x_2(y_1^2+2y_1y_2+3y_2^2)\}\Big].
\end{aligned}
\tag{27}
$$

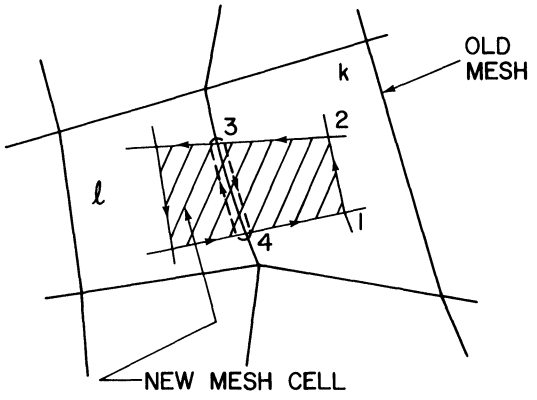
Note that $a_3^k = a_5^k = 0$ for Cartesian coordinates, and $a_1^k = a_4^k = 0$ for cylindrical coordinates.

Referring to Fig. 3a, (27) is precisely the contribution from line 1-2 to the cell of the new mesh shown shaded, while the contribution from line 3-4 of the old mesh, traced in the direction from point 3 to point 4, may be expressed as

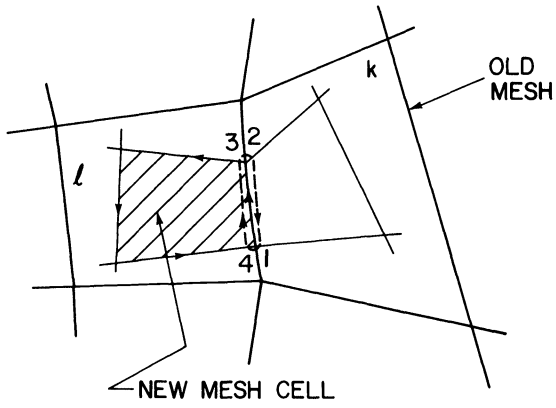
$$\Delta_3^4 Q^{kl} = \Delta_3^4 Q^k - \Delta_3^4 Q^l,$$

where k and l are the neighboring cells of the old mesh on either side of line 3-4. In the exceptional case in which lines 1-2 and 3-4 coincide we adopt the convention, illustrated in Fig. 3b, that for the new cell to the left of the line segment (shaded)

$$\Delta Q_L = \Delta_1^2 Q^k + \Delta_3^4 Q^k - \Delta_3^4 Q^l = -\Delta_3^4 Q^l$$



a.



b.

FIG. 3. *The configuration illustrating the contribution of the old mesh cell face to the line integral in* (a) *the typical case*; (b) *the case of the coincidence of the old and new mesh faces.*

(i.e., $\Delta_1^2 Q^k + \Delta_3^4 Q^k = 0$), while for the new cell to the right of the line segment

$$\Delta Q_R = \Delta_3^4 Q^k - \Delta_3^4 Q^l - \Delta_1^2 Q^l = \Delta_3^4 Q^k$$

(i.e., $\Delta_3^4 Q^l + \Delta_1^2 Q^l = 0$). This prescription for treating coincidences is equivalent to neglecting the contribution of the new mesh lines, and splitting the contribution of the old mesh segment by distributing $\Delta_4^3 Q^l$ to the cell on the left and $\Delta_3^4 Q^k$ to the cell on the right. It is obvious that the same coincidence must be detected when sweeping over the old and the new mesh. Coincidences are determined by requiring that the two line segments be parallel and that the area of the quadrilateral defined by the end points of the two segments be zero. This in general involves the evaluation of three cross products:

$$\mathbf{r}_{21} \times \mathbf{r}_{34} = 0 \rightarrow \text{parallelism},$$

$$\left.\begin{aligned} \mathbf{r}_{31} \times \mathbf{r}_{24} &= 0 \\ \mathbf{r}_{41} \times \mathbf{r}_{32} &= 0 \end{aligned}\right\} \rightarrow \text{zero area}.$$

We have chosen to detect only exact coincidences (i.e., exact zeros). This requires care so that the same arithmetic is performed whether one is sweeping along the old or the new mesh.

The tracing of mesh lines through one or the other mesh, and the calculation of intersections of the mesh line segments, follows the methods described in [6] and [7]. As a practical matter, however, in order to maximize accuracy and preserve symmetry between the sweeps of the old and new mesh, we always calculate intersections using only the end point of the cell sides, rather than using newly calculated intersection points to define line segments.

In general, the intersection of a mesh line segment with a cell side immediately specifies the cell that that segment is entering. In exceptional circumstances, however, a segment will intersect exactly at a cell vertex, and then there is ambiguity regarding which cell it is entering. For such cases we have provided a routine that finds the correct cell for continuing the segment. This routine is also useful for locating the starting cell for initiating the sweep along a given mesh line.

The computation of the density gradient, (21) or (22), requires further comment. Strict interpretation of (21) in cylindrical coordinates would always give a zero radial component of the gradient. We therefore reinterpret (21) to be an area average in our pseudo-Cartesian coordinates

$$\langle \nabla_k q \rangle = \frac{1}{A_j} \int\int_{A_j} \nabla q \, dx \, dy.$$

Again, it is convenient to convert to a surface (line) integral,

$$\left\langle \frac{\partial q_k}{\partial x} \right\rangle = \frac{1}{A_j} \oint_{C_j} q \, dy, \qquad \left\langle \frac{\partial q_k}{\partial y} \right\rangle = -\frac{1}{A_j} \oint_{C_j} q \, dx, \tag{28}$$

where A_j is the area of the averaging cell defined by the centroids, as shown in Fig. 2, and C_j is its contour. The above integrals are evaluated assuming a linear variation of q along the cell sides. However, along mesh boundaries we need boundary conditions to evaluate the integrals in (28). Sometimes these are available, such as when the normal component of velocity or momentum must be zero at reflective boundaries. In other cases (or for other variables) we do not have boundary conditions, but recalling that a zero slope merely lowers the order of the approximation (§ 2.3), we assume a zero normal gradient (reflective or von Neumann boundary condition) in such cases.

4. Computational examples. We will illustrate our technique using two examples. The first example was chosen to illustrate the principal properties of the method in a simple advection problem where there is no confusion due to interaction with other effects. The second example illustrates the use of the method in a realistic fluid dynamics calculation.

The first example consists of the simple advection of a scalar quantity on a Cartesian mesh. The initial density distribution is shown in Fig. 4a. This is advected with a velocity field which corresponds to a circular motion such that each point traces a circle with a radius of 5 units. The problem is trivial hydrodynamically since in a Lagrangian framework neither the density nor the shape of the distribution changes. However, when remapping this is far from the case, since a discontinuous density distribution typically leads to either oscillatory or else highly damped results. The calculation is Lagrangian for a number of time steps or cycles, followed by a remapping back to the original regular square mesh. All the subsequent figures correspond to the completion of a full revolution. Figure 4b shows the results of assuming a constant cell density and remapping on every cycle (no. of cycles = no. of remappings = 160). This calculation is representative of a continuous remapping using the donor cell method, although it was performed using the integral remapping method. The density distribution obtained is monotone and the severe diffusion of this first order calculation is very evident. Figure 4c shows the results of introducing the limited linear gradients, as described in this paper, to obtain a second order method. This calculation, since it

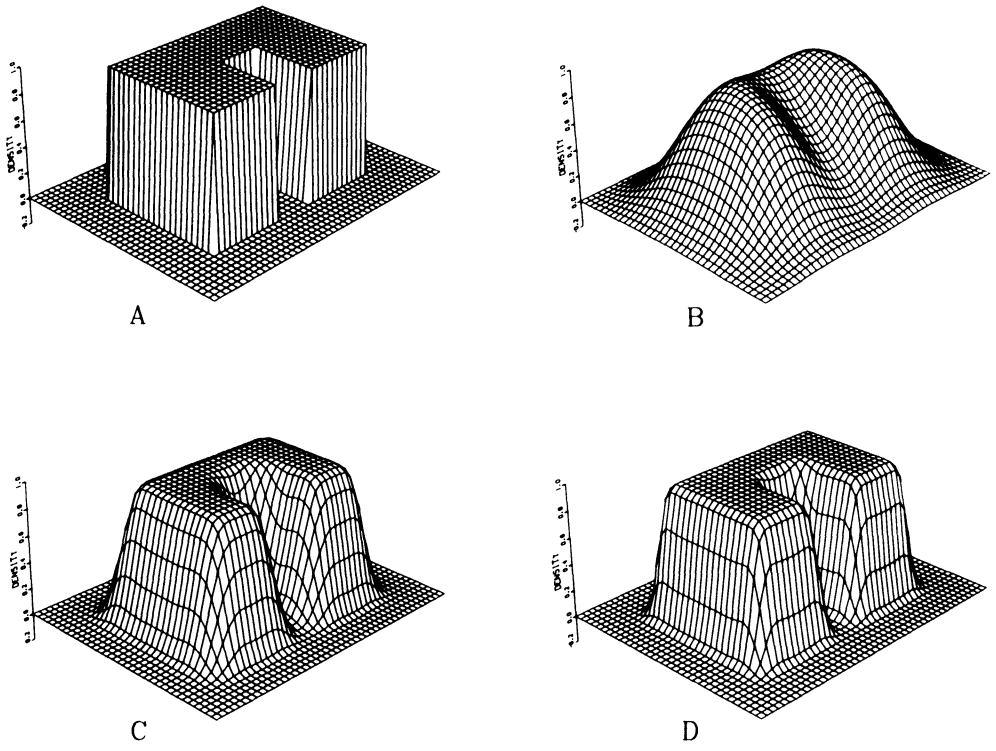


FIG. 4. *The simple remapping of a scalar quantity being advected by a circular velocity field.* (a) *Initial density distribution.* (b) *Constant density (first order) remapping* (160 *cycles*, 160 *remappings*). (c) *Linear density (second order) monotone remapping* (160 *cycles*, 160 *remappings*). (d) *Same as* (c), *but only* 32 *remappings.*

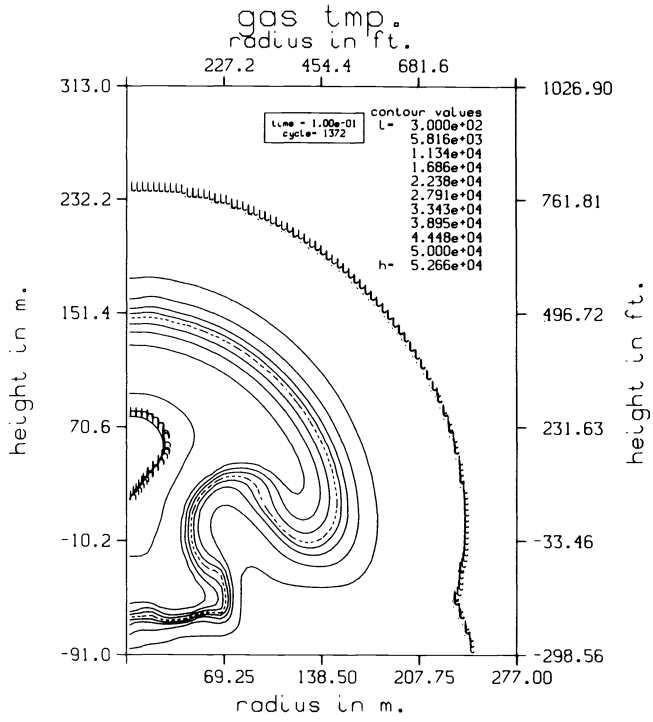


FIG. 5. *Temperature contours for a blast wave interaction problem: First order (constant density) remapping.*

remapped every cycle, corresponds approximately to continuous remapping using an FCT or a van Leer method. We note that a second order method, without limiting, would be expected to show severe oscillations on this problem which contains steep gradients. However, the density distribution that we obtain is monotone, and the diffusion is greatly reduced. Finally, Fig. 4d shows the result of remapping only at every fifth cycle (no. of cycles = 160, no. of remappings = 32). This illustrates the further improvement, possible with the integral remapping of this paper, which avoids the Courant stability restriction of the continuous remapping methods, and which therefore permits fewer remappings.

The second example illustrates a computation of the interaction of a strong spherical blast wave in air with a ground plane. The blast wave reflects from the ground plane, and the subsequent interaction of the reflected shock wave with the flow gradients produces complex flows similar to those due to the Richtmyer-Meshkov instability [10], [11].

The computation is performed in cylindrically symmetric geometry using a mesh of 100×70 cells. The calculation is Lagrangian until a decision is made to change the mesh and perform a remapping. The remapping is triggered under two conditions. In the first case, the mesh is checked for distortion and if a distortion criterion is met the mesh is remapped to a regular mesh within the original boundaries. Second, since there is no disturbance outside the expanding blast wave, it is inefficient to maintain a mesh very far outside the shock. We therefore determine when the shock approaches the top or the right-hand boundary of the mesh, and then expand the mesh appropriately and perform a remapping. The new mesh is regular with the original number of cells (although this is optional). Typically a remapping is required every 30 cycles.

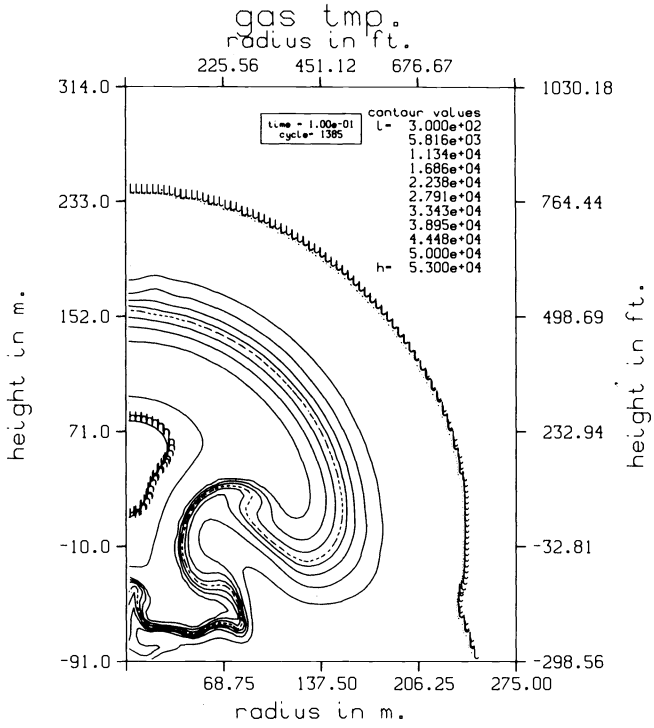


FIG. 6. *Temperature contours for a blast wave interaction problem: Second order (linear density) monotone remapping.*

Figure 5 shows the contours of the temperature at some time following the reflection of the shock, using an ALE code that uses a first order (constant cell densities) remapping [12]. Figure 6 shows the contours of temperature at the corresponding time obtained with the same code but using the new second order (monotone, linear distribution) remapping. The greater detail obtained in the development of the instability with the new remapping is clear from a qualitative comparison of the two results, although the differences are not dramatic since remapping is infrequent in either case.

Acknowledgments. We are grateful to Eric M. Jones for providing us with information regarding our second computational example, and to John D. Ramshaw for helpful comments and suggestions.

REFERENCES

[1] C. W. HIRT, A. A. AMSDEN AND J. L. COOK, *An arbitrary Lagrangian-Eulerian computing method for all flow speeds*, J. Comput. Phys., 14 (1974), pp. 227-253.

[2] J. P. BORIS AND D. L. BOOK, *Flux-corrected transport* I: SHASTA—*A fluid transport algorithm that works*, J. Comput. Phys., 11 (1973), pp. 38-69.

[3] S. T. ZALESAK, *Fully multidimensional flux-corrected transport algorithm for fluids*, J. Comput. Phys., 31 (1979), pp. 336-362.

[4] B. VAN LEER, *Towards the ultimate conservative difference scheme.* V. *A second-order sequel to Godunov's method*, J. Comput. Phys., 32 (1979), pp. 101-136.

[5] H. G. HORAK, E. M. JONES, J. W. KODIS AND M. T. SANDFORD II, *An algorithm for the discrete rezoning of Lagrangian meshes*, J. Comput. Phys., 26 (1978), pp. 277-284.

[6] J. K. DUKOWICZ, *Conservative rezoning (remapping) for general quadrilateral meshes*, J. Comput. Phys., 54 (1984), pp. 411-424.

[7] J. D. RAMSHAW, *Conservative rezoning algorithm for generalized two-dimensional meshes*, J. Comput. Phys., 59 (1985), pp. 193–199.

[8] P. A. THOMPSON, *Compressible-Fluid Dynamics*, McGraw-Hill, New York, 1982.

[9] A. Y. LEROUX AND P. QUESSEVEUR, L.E.A.: *Un code hydrodynamique multifluide bidimensionnel*, Ninth International Conference on Numerical Methods in Fluid Dynamics, Springer-Verlag, Berlin, 1985, pp. 365–368.

[10] R. D. RICHTMYER, *Taylor instability in shock acceleration of compressible fluids*, Comm. Pure Appl. Math., 13 (1960), pp. 297–319.

[11] V. A. ANDROPOV, S. M. BAKHRAKH, E. E. MESHKOV, V. N. MOKHOV, V. V. NIKIFOROV, A. V. PEVNITSKII AND A. I. TOLSHMYAKOV, *Turbulent mixing at contact surface accelerated by shock waves*, Soviet. Phys. JETP, 44 (1976), pp. 424–427.

[12] F. L. ADDESSIO, D. E. CARROLL, J. K. DUKOWICZ, F. H. HARLOW, J. N. JOHNSON, B. A. KASHIWA, M. E. MALTRUD AND H. M. RUPPEL, CAVEAT: *A computer code for fluid dynamics problems with large distortion and internal slip*, Los Alamos National Laboratory Report LA-10613-MS, Feb. 1986.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 3, May 1987
© 1987 Society for Industrial and Applied Mathematics
009

PRACTICAL ASPECTS OF INTERPOLATION IN RUNGE-KUTTA CODES*

I. GLADWELL†, §, L. F. SHAMPINE‡, §, L. S. BACA‡ AND R. W. BRANKIN†

Abstract. Runge-Kutta codes solve numerically the initial value problem for a system of ordinary differential equations in a step-by-step manner. The automatically chosen step sizes may vary widely in size depending on the equations, the nature of the error control, and the accuracy desired. Interpolation schemes approximate the solution of the differential equation within the step. Analytical tools are developed and applied to a selection of interpolation schemes with the aim of understanding how the interpolants behave. A novel matter in this context is the preservation of properties inherent in the values and derivatives computed by the Runge-Kutta code. In particular, we show how to preserve local monotonicity so as to achieve a visually pleasing interpolant, and in a way which is especially appropriate to Runge-Kutta codes.

Key words. ordinary differential equations, Runge-Kutta methods, interpolation, initial value problems

AMS(MOS) subject classifications. 65L05, 65D05

1. Introduction. Consider the numerical approximation of the solution $y(x)$ of an initial value problem for a system of ordinary differential equations,

$$y' = f(x, y), \quad a \leqq x \leqq b, \quad y(a) \text{ given.} \tag{1.1}$$

An explicit Runge-Kutta method starts with $y_0 = y(a)$ and steps from a to b in the following way: Given $y_n \simeq y(x_n)$, it provides an explicit recipe for $y_{n+1} \simeq y(x_{n+1})$ where $x_{n+1} = x_n + h_n$. The local solution $u(x)$ at (x_n, y_n) is that solution of (1.1) with $u(x_n) = y_n$. A code selects the step size h_n automatically so that an estimate of a norm of the local error,

$$\|le\| = \|u(x_n + h_n) - y_{n+1}\|,$$

is smaller than a given tolerance. This can be accomplished by taking h_n small enough, but to be efficient, one must take h_n as large as possible.

To obtain an approximate solution at a specified point, x_{out}, a step can be shortened to make x_{out} a mesh point. This is somewhat inefficient. If solutions are required at a great many specific points, if a solution is to be plotted, or if an x is to be found where $y(x)$ has a certain property, for example where $y(x)$ is a maximum, a Runge-Kutta code can be very inefficient. What is needed is an inexpensive way to approximate the local solution $u(x)$ between integration points, i.e., for $x_n < x < x_{n+1}$. Lately there has been quite a lot of progress in this matter. In this paper we examine the matter from several points of view. Our goals are to understand better the methods proposed and to illustrate how to select the best of several possibilities.

In the investigations of Shampine [1], [2] and Horn [3], [4], [5] the approximation of the local solution on $[x_n, x_n + h]$ is studied as $h \to 0$. The first author employs a local interpolant to intermediate solution values. In § 2 we extend this approach. For some formulas and interpolants, a remarkably detailed picture of the error of the interpolant is obtained: it is shown that as $h \to 0$ the behaviour of the error on $[x_n, x_n + h]$ can be completely specified in terms of the position x and of the local error of the formula.

* Received by the editors July 22, 1985; accepted for publication (in revised form) February 20, 1986.
† Department of Mathematics, University of Manchester, Manchester, England M13 9PL and NAG Ltd., Oxford, England OX2 7DE.
‡ Numerical Mathematics Division, Sandia National Laboratories, Albuquerque, New Mexico 87185.
§ The work of this author was supported by NATO research grant 898/83.

For other formulas and circumstances, a semi-quantitative or semi-qualitative understanding is obtained. In contrast to these results, the truncation error analysis used by Horn identifies the "typical" behaviour on an ensemble of problems. In compensation, it can be applied to any explicit Runge-Kutta method. This kind of analysis has been criticized as being "vague." We present a new way of viewing it which is perfectly concrete. Also we show that when both the interpolation theory and the truncation error methods of analysis are applicable, they lead to consistent results. In [1], [2] several ways to interpolate certain important Runge-Kutta formulas are presented. There they are compared with respect to their continuity, order and cost only. To illustrate how to apply the analytical methods developed in § 2, we compare these interpolants with respect to their accuracy. The conclusions we reach are of independent interest.

In [18] a method of analysis closely related to that of Horn is applied to the development of some new interpolants. This method and the two mentioned above are based on the limit $h \to 0$. This is not always realistic. It is supposed that the f of (1.1) satisfies a Lipschitz condition in y with constant L. In practice hL may not be particularly small. In § 3 the theory of Peano kernels is applied in a way like that of Gladwell [16] to understand better the error of interpolation in this range of h of intermediate size.

When modest accuracy is asked of the Runge-Kutta code, the interpolants studied in §§ 2 and 3 may not share properties of $y(x)$ such as local monotonicity. Motivated by the need for graphical display, many authors have considered how to interpolate sparse, accurate data in a visually pleasing way. In the context of data fitting and graph-plotting, most of the research has concentrated on interpolation to solution values alone. Also, often it is considered merely convenient to have a scheme using local data only. The situation with initial value problems in ordinary differential equations is special: a local procedure is necessary. To be consistent with the data available, the first derivative must be interpolated also. To our knowledge there has been no previous investigation of visually pleasing interpolation which takes into account the special situation of ordinary differential equations. In § 4 we employ some recent work of Gregory and Delbourgo [15] to obtain an elegant solution to the problem of obtaining such a visually pleasing interpolant in this context.

2. Accuracy of interpolants. Runge-Kutta methods proceed from an approximation y_n of the solution $y(x_n)$ of

$$y' = f(x, y), \quad a \leqq x \leqq b, \quad y(a) \text{ given}, \tag{2.1}$$

one step of length h to an approximation y_{n+1} of $y(x_n + h)$. Some methods produce approximations to $y(x)$, or even $y'(x)$, at certain points between x_n and $x_n + h$. It is suggested in [1] that these approximations be interpolated to produce an approximation to $y(x)$ on the whole interval $[x_n, x_n + h]$. Two kinds of analysis were applied in [1] to justify such interpolation procedures. One is based on interpolation theory and the other on truncation errors. In this section we develop further these tools and illustrate their use by considering the DPS formulas derived in [2].

We rely upon [1], [2] for the details, but for the sake of completeness, we now outline the situation. The local solution $u(x)$ at (x_n, y_n) is defined as that solution of (2.1) with $u(x_n) = y_n$. An explicit Runge-Kutta method is used to advance a step of length h to obtain y_{n+1}. The local error of this step is defined to be

$$le = u(x_n + h) - y_{n+1}.$$

For a method of order q, this error is $O(h^{q+1})$. It is supposed that there are distinct points ζ_i in $[x_n, x_n+h]$ such that approximations u_i to $u(\zeta_i)$ are available and, possibly, also approximations

$$u_i' = f(\zeta_i, u_i)$$

to $u'(\zeta_i)$. It is assumed that, at each point ζ_i, the local error

$$u(\zeta_i) - u_i = O(h^{q+1}),$$

which implies

$$u'(\zeta_i) - u_i' = O(h^{q+1})$$

for the Lipschitzian problems which interest us. The local interpolant considered by Shampine [1] is the polynomial $p(x)$ of least degree such that

$$p(\zeta_i) = u_i, \qquad i = 1, \cdots, m,$$
$$p'(\zeta_i) = u_i', \qquad i = 1, \cdots, r.$$

Because the data $u(x_n)$ and $u'(x_n) = f(x_n, u(x_n))$ are always available, it is most natural to take $\zeta_1 = x_n$ and to interpolate value and slope there. It is also natural to take $\zeta_s = x_n + h$ and $u_s = y_{n+1}$ for some s, $1 < s \leqq m$. Then, the local polynomial interpolants connect up to form a globally C^0 piecewise-polynomial approximation to $y(x)$. Even better is to note that interpolation is done only after a successful step, so that one can take $s \leqq r$, form $u_s' = f(x_{n+1}, y_{n+1})$, and later use this evaluation in the integration over the next step. Proceeding in this way the piecewise-polynomial approximation is globally C^1.

In the interpolation theory approach of [1], the polynomial $Q(x)$ is defined by interpolation of the exact values

$$Q(\zeta_i) = u(\zeta_i), \qquad i = 1, \cdots, m,$$
$$Q'(\zeta_i) = u'(\zeta_i), \qquad i = 1, \cdots, r$$

and then the error is written

$$u^{(K)}(x) - p^{(K)}(x) = [u^{(K)}(x) - Q^{(K)}(x)] + [Q^{(K)}(x) - p^{(K)}(x)]. \tag{2.2}$$

The first term is due to the interpolant alone and is $O(h^{m+r-K})$. The second term arises because inexact data is interpolated and it is $O(h^{q+1-K})$. Throughout this section we assume that $m + r > q + 1$.

Taylor series expansion of the local error results in an expression

$$u(x_n + h) - y_{n+1} = h^{q+1} \sum_{i=1}^{\lambda_{q+1}} T_{q+1,i} D_{q+1,i} + O(h^{q+2}). \tag{2.3}$$

Here the elementary differentials $D_{q+1,i}$ depend only on the problem, that is, on f, x_n, and y_n, and the truncation error coefficients $T_{q+1,i}$ depend only on the formula. The interpolants suggested by Shampine can be regarded as a family of Runge-Kutta methods, each method producing an approximation $y_{n+\sigma} = p(x_n + \sigma h)$ to $u(x_n + \sigma h)$. Fixing σ, a Taylor series expansion of the local error of this formula has the form

$$u(x_n + \sigma h) - y_{n+\sigma} = (\sigma h)^{q+1} \sum_{i=1}^{\lambda_{q+1}} T_{q+1,i}(\sigma) D_{q+1,i} + O(h^{q+2}). \tag{2.4}$$

Horn [3], [4], [5] suggests analyzing interpolants which can be viewed in this way by comparing the size of the truncation error coefficients of the error at the end of the

step, the values $T_{q+1,i}$, to those of the error at $x_n+\sigma h$, the values $\sigma^{q+1}T_{q+1,i}(\sigma)$, as $h\to 0$ with σ fixed.

2.1. Interpolation theory approach. The truncation error analysis provides no information about the error of interpolation for a *given* problem. In this section we extend the interpolation theory approach of [1] to address this question. It is possible to obtain some insight in general, and in a few cases, it is possible to obtain a complete understanding. Here we suppose that the data error dominates the error due to the interpolation of exact values of the local solution; in [11] we consider some other cases.

Supposing that $m+r>q+1$, (2.2) becomes

$$u^{(K)}(x)-p^{(K)}(x)\sim Q^{(K)}(x)-p^{(K)}(x).$$

As in [1] we have an expression for the data error of the form

$$Q^{(K)}(x)-p^{(K)}(x)=\sum_{i=1}^{m} A_i^{(K)}(x)[u(\zeta_i)-u_i]+\sum_{i=1}^{r} B_i^{(K)}(x)[u'(\zeta_i)-u_i'] \tag{2.5}$$

The fundamental interpolation polynomials A_i, B_i are written out in [1] where it is argued that $A_i^{(K)}(x)$ is $O(h^{-K})$ and $B_i^{(K)}(x)$ is $O(h^{1-K})$. Because the u_i' are of the same order of accuracy as the u_i, we then have

$$u^{(K)}(x)-p^{(K)}(x)\sim\sum_{i=1}^{m} A_i^{(K)}(x)[u(\zeta_i)-u_i]. \tag{2.6}$$

Because the $A_i(x)$ are known functions, we obtain in this way a description of how the various local errors affect the interpolant. In the following we consider cases for which relations among the local errors permit a much more complete understanding.

Case I. A natural procedure is to interpolate value and slope at both ends of the interval $[x_n, x_n+h]$. This cubic Hermite interpolation has $m+r=4$, so for any formula of order $q\leqq 2$, the data error dominates. There is no local error at $\zeta_1=x_n$, and taking $\zeta_2=x_n+h$ we have

$$u^{(K)}(x)-p^{(K)}(x)\sim A_2^{(K)}(x)[u(\zeta_2)-u_2]=A_2^{(K)}(x)le.$$

This is an ideal result: the error within the interval is related asymptotically in a known way to the error made at the end of the step, *le*. The code is attempting to control *le* so that the error of the approximation at all x in $[x_n, x_n+h]$ is related to the error tolerance specified by the user. In terms of the normalized variable σ defined by $x=x_n+\sigma h$,

$$\frac{\|u(x_n+\sigma h)-p(x_n+\sigma h)\|}{\|le\|}\sim|A_2(x_n+\sigma h)| \quad \text{as } h\to 0. \tag{2.7}$$

Qualitatively it is easy to understand the behaviour of the polynomial $A_2(x_n+\sigma h)$ from the interpolation conditions defining it:

$$A_2(\sigma=0)=0, \qquad A_2'(\sigma=0)=0,$$
$$A_2(\sigma=1)=1, \qquad A_2'(\sigma=1)=0.$$

Figure 2.1 displays $|A_2(x_n+\sigma h)|$, which along with (2.7) gives us a very satisfactory understanding of the behaviour of the local interpolation procedure. We emphasize that (2.7) holds for any problem with sufficiently smooth f.

Case II. Now let us consider quintic Hermite interpolation to value and slope at x_n, $x_n+h/2$, x_n+h. With methods of order $q\leqq 4$, the data error will then dominate. There are two variations of interest. Formulas are implemented as pairs so that in

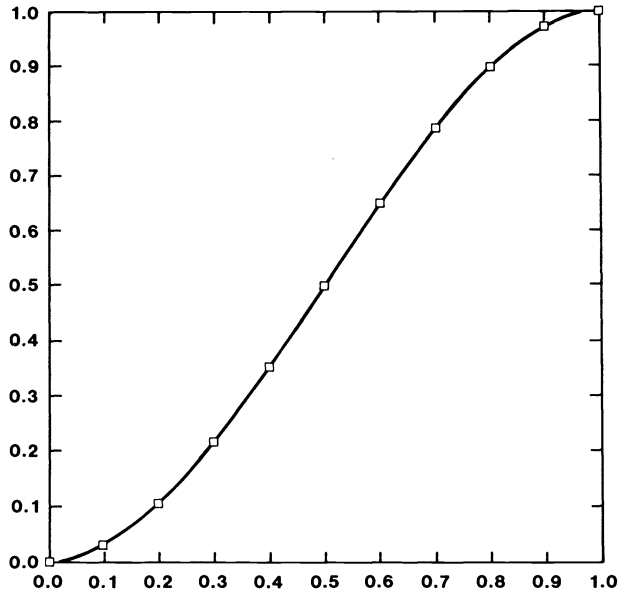


FIG. 2.1. *For Case* I *of* § 2, *this plot shows how the error of interpolation to the local solution compares to the local error.*

addition to the formula of order q, there is a companion of order $q+1$. One variation is to advance the integration with the result of order q. The other is to advance with the result of order $q+1$; this is called local extrapolation.

If local extrapolation is not done,

$$u^{(K)}(x)-p^{(K)}(x)\sim A_2^{(K)}(x)[u(\zeta_2)-u_2]+A_3^{(K)}(x)le.$$

To obtain the kind of detail we saw in Case I, we must relate the local error at $\zeta_2=x_n+h/2$ to le, the local error at x_n+h. There are a number of circumstances for which

$$u(\zeta_2)-u_2\sim le/2.$$

A general possibility is to form u_2 by an independent step of length $h/2$ with the same formula used to form u_3. This possibility is discussed in [1], where it is noted that it is used by a code of Hiebert and Shampine based on a Fehlberg pair. Another general possibility arises when a step of length $h/2$ is taken from x_n to form u_2 and this is followed by another step of length $h/2$ to form u_3. This situation is always present when the local error is estimated by doubling. An elementary proof of the desired relation of the local errors is given in [8], where the current prejudice against this way of estimating local errors is attacked. There are formulas that estimate local error by embedding which have this structure. One such is a (4, 5) pair of Shintani [6]; England [7] gives a family. One of England's formulas showed up very well in the tests of Shampine and Watts [9] and so has been implemented in a number of codes, including the simulation language GASP IV [12]. These examples show that the special relation between the local errors of u_2 and u_3 that we study holds for a number of interesting formulas. Assuming the relation, we have

$$u^{(K)}(x)-p^{(K)}(x)\sim[A_2^{(K)}(x)/2+A_3^{(K)}(x)]le. \tag{2.8}$$

Figure 2.2 presents the right-hand side of

$$\frac{\|u(x_n+\sigma h)-p(x_n+\sigma h)\|}{\|le\|}\sim|A_2(x_n+\sigma h)/2+A_3(x_n+\sigma h)|. \tag{2.9}$$

England's formula is treated as an example of truncation error analysis in [1]. The present analysis applies to it, and provides an incomparably better understanding of the interpolation error. Now we know exactly how the interpolation error within the step is related (asymptotically) to the local error made at the end of the step, and this is true for any given problem.

When local extrapolation is used, the value u_3 is computed from a higher order formula than u_2 so that

$$u^{(K)}(x)-p^{(K)}(x)\sim A_2^{(K)}(x)[u(\zeta_2)-u_2].$$

This is a useful indication of the way the interpolation error is distributed within the step. To obtain a more detailed picture, we suppose we have one of the situations just described for which we know that the local error of u_2 is $le/2$. Then we have the complete asymptotic description

$$u^{(K)}(x)-p^{(K)}(x)\sim A_2^{(K)}(x)le/2.$$

Figure 2.3 presents the right-hand side of

$$\frac{\|u(x_n+\sigma h)-p(x_n+\sigma h)\|}{\|le\|}\sim|A_2(x_n+\sigma h)/2|.$$

Evidently local extrapolation improves the accuracy of the local interpolant as well as the order of accuracy of the values used to advance the integration. In [1] we pointed out that the interpolant for the England scheme can be useful outside the interpolation interval $0\leqq\sigma\leqq 1$ provided one does not go too far. Comparison of Fig. 2.2 with Fig.

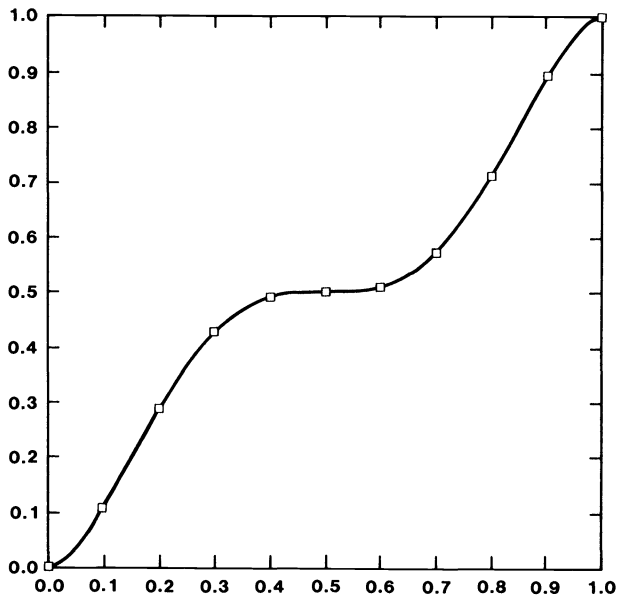


FIG. 2.2. *For Case* II *of* § 2, *this plot shows how the error of interpolation to the local solution compares to the local error. Local extrapolation is not done.*

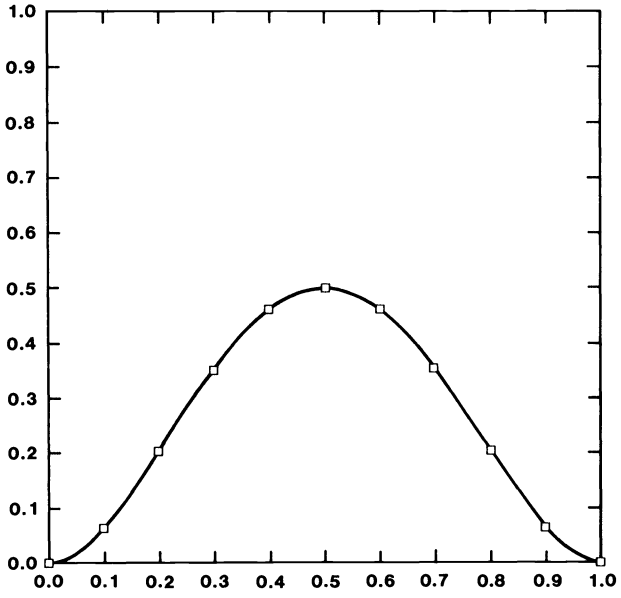


FIG. 2.3. *The situation is that of Fig. 2.2 except that local extrapolation is done.*

2.3 suggests that the interpolant arising from local extrapolation will be the more effective in this context. By way of illustration, we state that

$$\|u(x_n+2h)-p(x_n+2h)\| \sim 148\|le\| \tag{2.10}$$

if local extrapolation is not done, and

$$\|u(x_n+2h)-p(x_n+2h)\| \sim 32\|le\| \tag{2.11}$$

if it is.

2.2. Truncation error analysis. Horn [3], [4], [5] employs an analysis based on truncation errors that is more generally applicable than the approach just investigated. Referring to the outline presented earlier, the question is how the error at $x_n+\sigma h$ relates to the error, *le*, at x_n+h as $h\to 0$. The difficulty is that the elementary differentials $D_{q+1,i}$ depend on the problem and are not known. In contrast to the results of § 2.1, we cannot predict what will happen for a single problem. To obtain a guide for action, a large ensemble of problems is considered and it is asked what behaviour we can "expect." On a suitable set of problems, the "typical" relation of the two local errors (2.3), (2.4) is indicated by the relative sizes of the coefficients of the elementary differentials, that is, the truncation error coefficients. Horn uses

$$\|T_{q+1}\|_2=\left(\sum_{i=1}^{\lambda_{q+1}} T_{q+1}^2\right)^{1/2}$$

as a measure of the size of the coefficients. A plot of

$$\|\sigma^{q+1}T_{q+1}(\sigma)\|_2/\|T_{q+1}\|_2 \tag{2.12}$$

provides an indication as to how the error of the interpolant at $x_n+\sigma h$ compares asymptotically to the error at x_n+h, namely *le*. Although a useful tool, this truncation error analysis has been criticized as vague. We now develop a new interpretation which is concrete. If we let

$$\Gamma_{q+1}=\max_{1\le i\le\lambda_{q+1}}\|D_{q+1,i}\|$$

in the Taylor series expansion (2.3) of the local error, we readily find that

$$\|le\| = \|u(x_n+h) - y_{n+1}\| \leqq h^{q+1}\Gamma_{q+1}\sqrt{\lambda_{q+1}}\,\|T_{q+1}\|_2 + O(h^{q+2}) = B(1, h).$$

Similarly treating (2.4), we have

$$\|u(x_n+\sigma h) - y_{n+\sigma}\| \leqq h^{q+1}\Gamma_{q+1}\sqrt{\lambda_{q+1}}\,\|\sigma^{q+1}T_{q+1}(\sigma)\|_2 + O(h^{q+2}) = B(\sigma, h).$$

For a given problem we would like to compare $\|u(x_n+\sigma h) - y_{n+\sigma}\|$ to $\|le\|$ as $h \to 0$ with σ fixed. This not being possible generally, we shall compare instead the *bounds* just derived. It is pleasing that as $h \to 0$, the ratio $B(\sigma, h)/B(1, h)$ is just (2.12). Of course, comparing bounds on quantities is not nearly so useful as the detailed results of the interpolation theory, but it provides a reasonable way to assess the quality of an interpolation procedure and has the virtue of generality.

England's scheme was treated in [1] as an example of truncation error analysis. The plot of (2.12) given in [1] is apparently the same as our Fig. 2.2. This is no accident. For the cases of § 2.2 where interpolation theory gives a detailed result, we have

$$u(x_n+\sigma h) - p(x_n+\sigma h) \sim g(x_n+\sigma h)\,le$$

for a suitable function $g(x)$. Inserting here the expansion of le, we have

$$u(x_n+\sigma h) - p(x_n+\sigma h) \sim h^{q+1} \sum_{i=1}^{\lambda_{q+1}} g(x_n+\sigma h)\,T_{q+1,i}D_{q+1,i}.$$

On comparing this expression to (2.4) and using the fact that the elementary differentials are independent, it follows that

$$\sigma^{q+1}T_{q+1,i}(\sigma) = g(x_n+\sigma h)\,T_{q+1,i}$$

for these schemes. But then

$$\|\sigma^{q+1}T_{q+1}(\sigma)\|_2 / \|T_{q+1}\|_2 = |g(x_n+\sigma h)|,$$

which is exactly the quantity

$$\frac{\|u(x_n+\sigma h) - p(x_n+\sigma h)\|}{\|le\|} \sim |g(x_n+\sigma h)|$$

that we plotted in Figs. 2.1–2.3.

The DPS formulas derived in [2] allow a number of interpolants. In [2] some possibilities are compared with respect to their cost alone. Now we shall apply our analytical tools to the task of selecting an interpolant for these formulas. Besides illustrating how one might proceed in general, the conclusions for these formulas are of considerable practical interest.

The cheapest possibility costs no extra function evaluations for interpolation. A fourth order result is formed at the midpoint and a quartic is fitted to this result and the value and slope at both ends. With local extrapolation the accuracy of the interpolant is indicated by the plot of (2.12) in Fig. 2.4. This result is a little disappointing since the error at some points of (x_n, x_n+h) can be somewhat larger than the error at x_n+h. Of course, the whole truncation error analysis is a rough measure of relative accuracy and the theory and practice of local error control is somewhat crude, so we should not take these results too seriously. Still, it was pointed out in [2] that an extra function evaluation for the interpolation is of little practical importance, so that an investigation of more expensive procedures is worthwhile.

In [2] a DPS triple is given which produces a fifth order result at the midpoint at a cost of one function evaluation. Figure 2.5 shows the expected behaviour of the

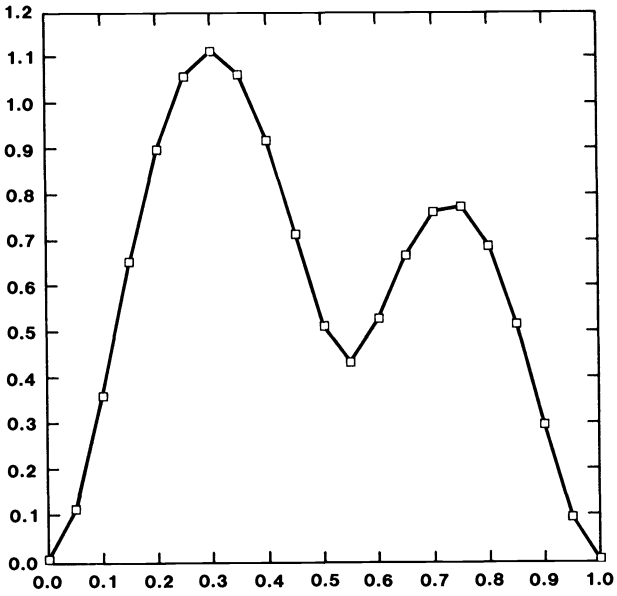


FIG. 2.4. *Truncation error analysis of* DPS *triple with fourth order result at midpoint and local extrapolation.*

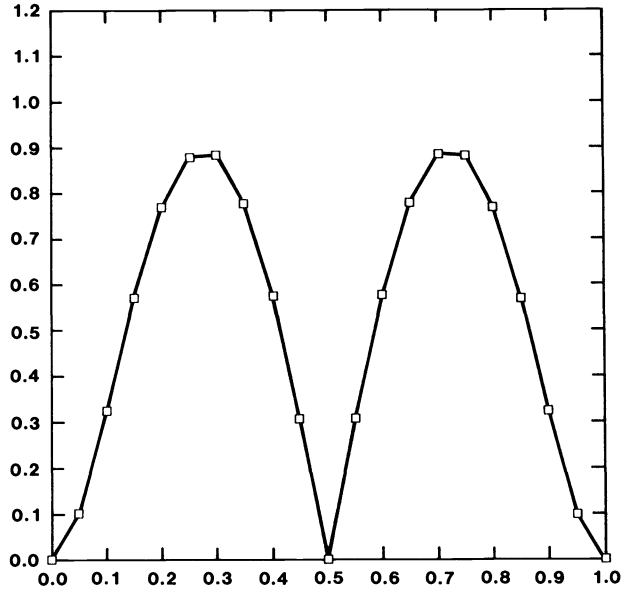


FIG. 2.5. *Truncation error analysis of* DPS *triple with fifth order result at midpoint and local extrapolation.*

error resulting from local quartic interpolation. This appears quite satisfactory, but it was noted in [2] that the fifth order result is less convenient to implement than the fourth order result. For this reason, we also examined a scheme not considered in [2]. This new scheme uses the fourth order result $y_{n+1/2}$ at $x_n + h/2$, but adds $y'_{n+1/2} = f(x_n + h/2, y_{n+1/2})$ and performs quintic Hermite interpolation. The scheme is only a little more trouble than the scheme without the derivative. The truncation error analysis leads to a plot which cannot be distinguished visually from Fig. 2.3. It does differ, however, for the value of the ratio (2.12) which at $\sigma = \frac{1}{2}$ is about 0.51. The interpolation theory approach of § 2.1 applies to this scheme. We find the relative behaviour, but

cannot relate quantitatively the local error at the midpoint to *le*. Here we see that in a somewhat vague, "average" sense, the ratio is about 0.51. Among the possibilities for the DPS triples yielding a fourth order interpolant, we prefer quintic Hermite interpolation to the value and slope at x_n, $x_n + h/2$, $x_n + h$ with the fourth order result at the midpoint and the fifth order result at $x_n + h$ (local extrapolation).

Another interesting possibility with the DPS triples is to form a fifth order interpolant. This is done with quintic Hermite interpolation to the value and slope at x_n, $x_n + h/2$, $x_n + h$ with the fifth order result at both midpoint and the end of the step. As we remarked, it costs a function evaluation to form the fifth order result at the midpoint and the software interface is less convenient than with the fourth order result. It costs still another evaluation to get the slope there. These disadvantages are not so great, as it is explained in [2] how to handle the interface in a reasonable way and why the extra function evaluations are not especially important in practice. The most obvious gain is that the interpolant is now of the same order as the value used to advance the integration. In Fig. 2.6 we relate the error of the interpolant to the local error of the fifth order result at the end of the step. It is important to understand that the situation here is not the same as that of Fig. 2.4. There the error is taken relative to the local error controlled by the code, so that a ratio greater than unity means that the interpolant might have an error greater (asymptotically) than the user's tolerance. Here the error is taken relative to the error of the extrapolated result, which is of higher order than the local error controlled by the code. A ratio greater than unity in Fig. 2.6 means that the error of the interpolant might be greater (asymptotically) than the error of the locally extrapolated result, but this result should have an error much smaller than the user's tolerance, and hence is not a matter for concern.

3. Polynomial interpolation results. As we saw in the previous section, for some practical choices of polynomial interpolant the dominating term in the error of the polynomial fit p to the local solution u is a multiple of the local error itself, at least

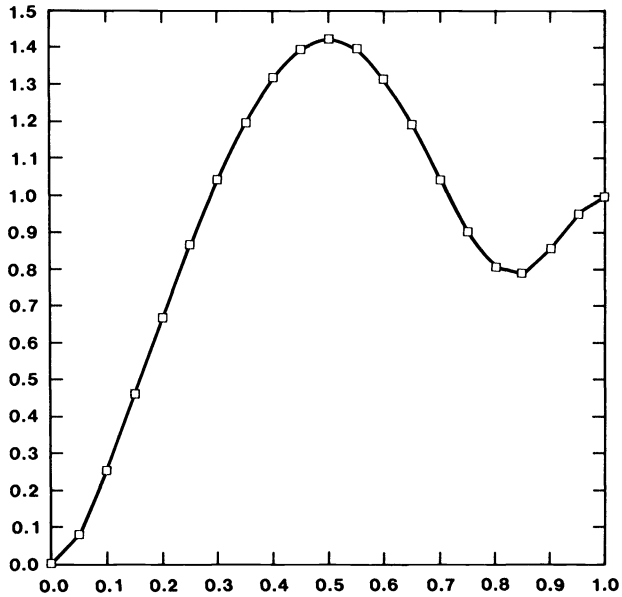


FIG. 2.6. *Truncation error analysis of* DPS *triple with fifth order interpolant taken relative to the local error of the fifth order formula.*

for step sizes h small enough that the asymptotic error expansions are valid. The argument is invalid when the interpolation error dominates and when the local error and interpolation error are of the same order of accuracy. In such cases, the truncation error analysis of the previous section provides insight, yet there is more to be learned from studying the polynomial interpolation error itself.

In practical computation, codes based on explicit Runge-Kutta methods are used often to obtain numerical solutions of low to moderate accuracy. In such circumstances the step sizes chosen by the code may not be small. Although the asymptotic local error theory employed earlier when deriving interpolation results is usually valid, often the leading terms will not dominate significantly; the terms neglected in the analysis there may be of importance almost equal to those analysed. In any case, the user faced with his data (solution and derivative values) is not overly concerned at their provenance; his major interest lies in the properties of the local fit in the context of his particular application.

In applications such as predicting where a function of the solution has a root, we may want to evaluate the fit outside the interval $[x_n, x_n+h]$, that is, to extrapolate the solution instead of interpolate it. The arguments of the previous section give some insight then. In particular, results such as (2.10) and (2.11) are of considerable interest and demonstrate the need for careful analysis. As in this context, any assumption that the data error dominates is likely to be violated except when calculating very high accuracy results, it is important to measure the effect of extrapolation relative to that of interpolation.

A further consideration here is that when locating, say, a discontinuity or a zero, possibly the only control the user has over the accuracy of the computed solution is his ability to manipulate the local error tolerance. A superficial reading of the previous section may lead one to expect that because reducing the local error tolerance reduces the local error, the interpolation error will also be reduced, and hence that the zero or discontinuity will be located more accurately. However, this disregards the fact that changing the local error tolerance modifies the step size sequence. Hence for differing local error tolerances, a fixed event will appear in a different relative position in the integration step. Except in cases when the local error strongly dominates the interpolation error, this latter error will be of significance when determining the position of the event.

Again, we consider the error arising from three interpolation schemes:

(i) Cubic Hermite interpolation on each of the subintervals $[x_n, x_n+h/2]$ and $[x_n+h/2, x_n+h]$, requiring the solution and its derivative at each of the points x_n, $x_n+h/2$, x_n+h, that is, $m=2$, $r=2$ (in contrast to the previous section where one interpolant for the whole interval was considered);

(ii) quartic interpolation to solution and derivative at each of x_n and x_n+h and to solution value only at $x_n+h/2$, that is, $m=3$, $r=2$;

(iii) quintic Hermite interpolation to the solution and its derivative at x_n, $x_n+h/2$ and x_n+h, that is, $m=3$, $r=3$.

As was stated in § 2, for sufficiently smooth functions u the error in the interpolants is $O(h^{m+r})$. In Table 3.1 we give an approximate value for the constant $C^s_{m,r}$ in a more precise result of the form

$$\|u-p\|_{C^\infty[x_n,x_n+h]} \leqq C^s_{m,r} h^s \|u^{(s)}\|_{C^\infty[x_n,x_n+h]} \tag{3.1}$$

for $2 \leqq s \leqq m+r$ where the norm is the maximum norm over the interval stated; this includes the usual interpolation error result for $s=m+r$. These bounds on the interpolation error are derived by applying the standard Peano theory as discussed for example

in [10]. We indicate also the point, $\sigma_{\max}$, >0.5 (using the notation of § 2) at which the bound may be attained. With extrapolation in mind, we give also the location of the points $\sigma > 1$ where the extrapolant has maximum norm error the same as the internal ($\sigma < 1$) maximum, σ_{ex}, twice that value, σ_{2ex}, and ten times that value, σ_{10ex}.

A number of observations follow immediately from Table 3.1. First, in cases when the solution of the differential equation is not as smooth as would be required for the highest order result, the size of the constants indicates that the interpolation error is not affected critically; that is, as s increases, the error constants behave approximately inversely linearly with s, except for very low error orders. Second, there is a rather surprising uniformity in the σ values both as s increases for a particular interpolant and across the interpolants; the value $\sigma_{\max} = 0.75$ for the cubic Hermite interpolant is constant as may be demonstrated analytically.

TABLE 3.1
Error constants and locations for polynomial interpolants.

m	r	s	$C^s_{m,r}$	$\sigma_{\max}$	σ_{ex}	σ_{2ex}	σ_{10ex}
3	3	6	1.94E−4	.79	1.07	1.10	1.18
3	3	5	5.76E−4	.78	1.08	1.11	1.20
3	3	4	2.44E−3	.78	1.09	1.11	1.21
3	3	3	1.37E−2	.78	1.09	1.12	1.23
3	3	2	9.58E−2	.77	1.09	1.13	1.24
3	2	5	2.36E−3	.73	1.11	1.14	1.28
3	2	4	5.72E−3	.72	1.12	1.16	1.30
3	2	3	2.12E−2	.73	1.13	1.17	1.34
3	2	2	9.85E−2	.72	1.13	1.19	1.36
2	2	4	2.60E−3	.75	1.10	1.14	1.27
2	2	3	1.04E−2	.75	1.12	1.17	1.34
2	2	2	6.25E−2	.75	1.13	1.18	1.37

We may use this latter observation to assist us computationally. If we view the integration algorithm in combination with the interpolation technique as a method for computing a globally C^1 approximate solution to the differential system, then we should estimate the global errors not just at the integration points but also at the points where they take their maxima. Of course, we do not know the location of these maxima since integration error and interpolation error may cancel. A pragmatic approach is to monitor a global error estimate close to $\sigma_{\max}$. Table 3.1 leads us to believe that, fortunately, the degree of smoothness of the local solution u will not affect the validity of this approach.

Consider now the possibility of using our interpolants for extrapolation. Unless we are certain that the local error dominates the interpolation error, we are likely to be unhappy with extrapolation to points where the extrapolation error will far exceed the interpolation error. Fortunately the values of σ_{ex}, σ_{2ex}, and σ_{10ex} do not depend critically on the smoothness of the local solution. This observation enables us to predict that extrapolation to points up to, say, 10% of the current interval length outside the interval is likely to be satisfactory. Great care is necessary as may be seen from the closeness of σ_{2ex} to σ_{ex} in all cases. As might be expected, the lower degree piecewise-polynomial fits are the more reliable extrapolants.

Since each of the interpolants under consideration is globally C^1, users will expect to employ first derivatives of these interpolants to approximate first derivatives of the solution. The information in Table 3.2 corresponds to Table 3.1 except for the first derivative. Now

$$\|u'-p'\|_{C^\infty[x_n,x_n+h]} \leqq C^{s\prime}_{m,r} h^{s-1} \|u^{(s)}\|_{C^\infty[x_n,x_n+h]}. \tag{3.2}$$

The value $\sigma'_{\max} = 0.50$ for the quartic Hermite interpolant is a constant value.

From Table 3.2 we observe that, insofar as the error constants may be used as a measure of accuracy, for the same degree of smoothness of u in (3.2) as in (3.1), the cubic Hermite polynomial approximates u' slightly better relative to the way it approximates u than do the quartic or quintic Hermite fits. Also observe that for all degrees of smoothness, s, the error in the derivative of each interpolant takes its maximum at about a constant position $\sigma'_{\max}$. This is similar behaviour as in the case of the error in the interpolant itself. Finally observe that, relatively speaking, the extrapolant to the derivative is generally less accurate than the extrapolant to the function, that is, $\sigma_{\text{ex}} < \sigma'_{\text{ex}}$, $\sigma_{2\text{ex}} < \sigma'_{2\text{ex}}$ and $\sigma_{10\text{ex}} < \sigma'_{10\text{ex}}$.

Tables 3.1 and 3.2 provide a guide as to which of the three interpolants under discussion is best suited to our purposes. Of course, since they are of different orders, the one of highest order will be most accurate for small error tolerances and smooth local solutions. However, if we assume only that the solution is sufficiently smooth that we can consider $s \leqq 4$, Table 3.1 favours the cubic Hermite marginally and Table 3.2 favours it more strongly. The quartic polynomial would seem to be favoured only if it is considered essential to avoid the derivative evaluation at $x_n + h/2$.

4. Visually pleasing interpolants. In this section we examine qualitative properties of the interpolants, in particular, local monotonicity. If $|y'(x)| \geqq \mu > 0$ for $x_n \leqq x \leqq x_n + h$, the interpolants we consider produce a fit monotone on $[x_n, x_n + h]$ for all sufficiently small h. This may not be the case when the step sizes are "large." Working with graphical accuracy ordinarily results in rather large step sizes so that this issue is important in practice.

A stratagem similar to that described in this paper has been used in the NAG library for some time [13]. There, a Runge-Kutta-Merson method is employed to produce fourth order solutions on an uneven mesh. A quintic polynomial is fitted to

TABLE 3.2

Error constants and locations for the derivatives of the polynomial interpolants.

m	r	s	$C^{s\prime}_{m,r}$	$\sigma'_{\max}$	σ'_{ex}	$\sigma'_{2\text{ex}}$	$\sigma'_{10\text{ex}}$
3	3	6	7.67E−4	.92	1.03	1.05	1.13
3	3	5	2.10E−3	.91	1.03	1.05	1.14
3	3	4	8.65E−3	.91	1.03	1.05	1.15
3	3	3	4.83E−2	.90	1.03	1.06	1.18
3	3	2	3.75E−1	.90	1.04	1.07	1.20
3	2	5	8.33E−3	.50	1.05	1.08	1.22
3	2	4	2.08E−2	.50	1.05	1.09	1.27
3	2	3	7.40E−2	.50	1.06	1.11	1.32
3	2	2	4.16E−1	.50	1.08	1.13	1.40
2	2	4	8.02E−3	.60	1.04	1.07	1.22
2	2	3	3.23E−2	.61	1.05	1.09	1.32
2	2	2	2.51E−1	.61	1.07	1.13	1.43

the solution and derivative values at both ends of the current interval and also to the solution and derivative values from the beginning of the *previous* interval. In this case the error in the integrated values dominates the interpolation error; that is, the local error tolerance provides a control of the error of the globally C^1 fitted solution. Since the polynomial is fitted across two intervals of integration, it is necessary to restrict changes of step size so as not to lead to an interpolant whose error is magnified enormously by the choice of successive step sizes. (See [17] for a detailed analysis of how important the ratio of successive step sizes can be; this ratio appears directly in the error term.) The quintic Hermite interpolant has been used successfully in this context for several years for both tabulation and root finding purposes. In Fig. 4.1 we show a plot of the numerical solution of the differential equation

$$y' = -y, \qquad y(0) = 1, \tag{4.1}$$

using the Runge-Kutta-Merson code D02PAF to produce the integration points (marked) and the quintic Hermite polynomial to plot the solution. We show plots for a mixed local error control with local error tolerance choices 10^{-1} and 10^{-2}. In both

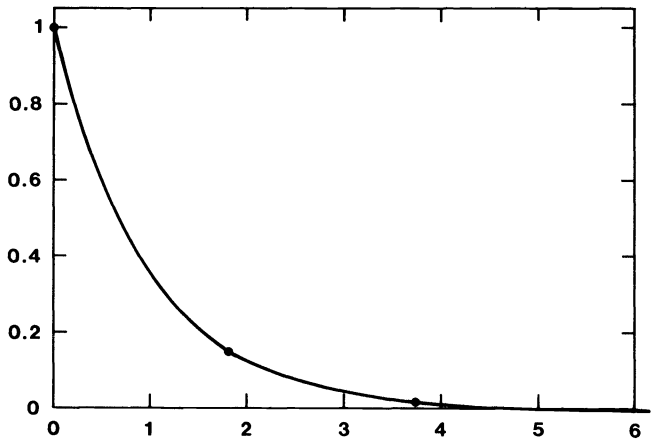


FIG. 4.1a. *Quintic Hermite interpolant to solution values computed by routine* D02PAF *with local error tolerance* $\tau = 10^{-1}$. ● *denotes an integration point.*

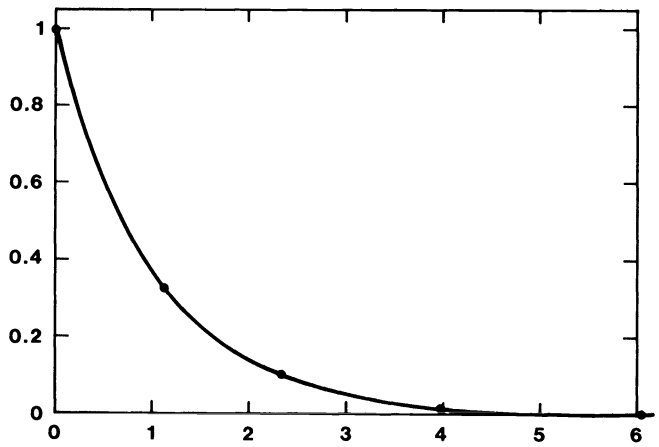


FIG. 4.1b. *Quintic Hermite interpolant to solution values computed by routine* D02PAF *with local error tolerance* $\tau = 10^{-2}$. ● *denotes an integration point.*

cases the initial stepsize is chosen automatically and it turns out to be small enough that, together with the effect of the restriction on step size changes mentioned above, the plotted solution is well behaved, and even monotonic. This is not the case in some plots for the DPS method with the same local error tolerances (see Figs. 4.2 and 4.3). In the DPS method there is no step size restriction forced by interpolation requirements because the interpolant is always fitted on two subintervals of equal length.

As in preceding sections the interpolants for the DPS formulas will be used to illustrate the issues. Naturally we would prefer a single scheme for interpolation insofar as this is possible. However, with a wide range of applications in mind a unique choice is probably not appropriate. For producing results of a more or less uniform accuracy, quintic interpolation using fifth order values everywhere is likely to be suitable. For applications such as root finding, the situation is a little less clear. It is necessary that the interpolant share the properties of the data sufficiently well and that it reproduce

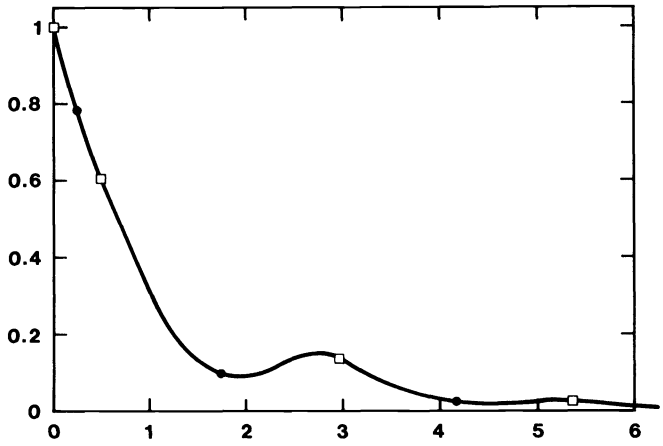


FIG. 4.2. *Quintic Hermite interpolant to fifth order solution values at integration points and fourth order solution values at midpoints computed by a* DPS *code with* $\tau = 10^{-1}$. □ *denotes integration point*; ● *denotes a midpoint value.*

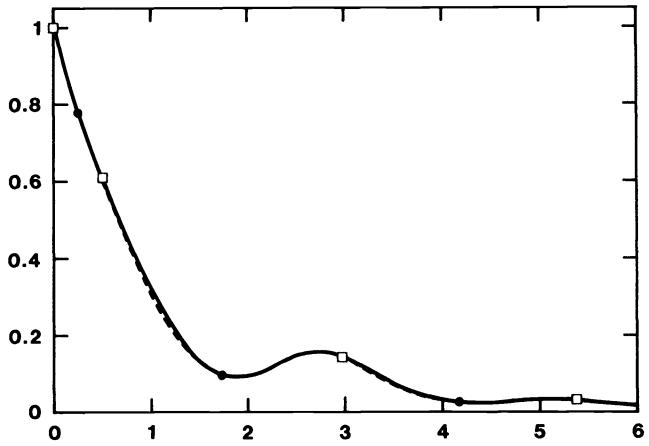


FIG. 4.3. *Cubic Hermite* (——) *and quadratic rational* (- - -) *interpolants to fifth order solution values at integration points and fourth order solution values at midpoints computed by* DPS *code with* $\tau = 10^{-1}$. □ *denotes an integration point*; ● *denotes a midpoint value.*

only those properties which are predictable from the data. Unfortunately, none of the interpolants discussed so far have this property independent of the step size used (since, for example, monotonic data can lead to nonmonotonic interpolants).

It is important to appreciate that convergence of the Runge-Kutta method is sufficient to guarantee that for all sufficiently small step sizes many specific mathematical properties of the solution will be shared by an interpolant of a high enough order of accuracy. Hence, for example, a cubic Hermite interpolant to monotonic data will be monotonic for all step sizes smaller than a certain bound which turns out to be determinable. Unfortunately, the local error tolerance required to ensure that the step size is small enough is unknown. The situation is still less satisfactory for the quintic Hermite fit to monotonic data. Again we know that for all sufficiently small step sizes the quintic fit will be monotone, but this time we do not have a determinable bound. In practice, if it is essential that a property such as monotonicity hold, we must use a type of fit which guarantees the property for *all* stepsizes, or be prepared to expend considerable computational effort on repeated integrations at progressively tighter tolerances until it can be verified that the requisite property holds for our preferred fit.

Turning now to the general question of graph-plotting in the context of initial value solvers, many users consider it particularly important that a visually pleasing plot be produced. This implies that properties of the data, such as monotonicity, convexity, and smoothness be preserved as far as possible. It is also important that the computational cost of producing the plot should be low, that unnecessarily large amounts of data storage should not be needed, and that the plotted function be consistent with the data at least at the integration points. In our case we have available the derivatives, at least at the interval end points, and consistency with the data implies that we should use these values. Given the data available, standard graph-plotting software would always use cubic Hermite polynomials either on $[x_n, x_n + h]$, or on the two subintervals $[x_n, x_n + h/2]$ and $[x_n + h/2, x_n + h]$ if the derivative at $x_n + h/2$ had been calculated.

Consider first the case when the data does not imply a special property such as monotonicity. Such a case is provided by our DPS code operating on problem (4.1) with local error tolerance 10^{-1} and using the fourth order values at the midpoint of the intervals. Of the four intervals shown in Fig. 4.2 the first is rather short due to the conservative initial step size algorithm and the second, third, and fourth intervals have midpoint values which do not lie monotonely between their endpoint values. In this case the quartic fit to the data and the Hermite quintic fit to the data plus the derivative at the midpoint value are hardly distinguishable so only the latter is given in Fig. 4.2. A cubic Hermite fit on each of the two half intervals gives a very similar picture in Fig. 4.3; the rational quadratic fit plotted there will be discussed below. If we repeat the calculations with a local error tolerance of 10^{-2}, the true solution e^{-x} is approximated by locally monotonic data on a fine mesh, and all the interpolants give visually pleasing graphs reproducing the important features of the data. In general, a suitable choice of local error tolerance is not known a priori and the disagreeable behaviour shown in Figs. 4.2 and 4.3 is quite feasible.

If we use the fifth order midpoint result in the DPS code (and its derivative when appropriate), the resulting computed solution values are monotonic even for a local error tolerance of 10^{-1}, this despite the fact that the data are not an accurate representation of the solution of (4.1). In Fig. 4.4 we show the quartic and Hermite quintic fits to this data; clearly neither is monotonic though the quartic fit seems superior. The cubic Hermite fits over the subintervals $[x_n, x_n + h/2]$ and $[x_n + h/2, x_n + h]$ are shown in Fig. 4.5 together with the rational quadratic fits to be discussed below. Observe that

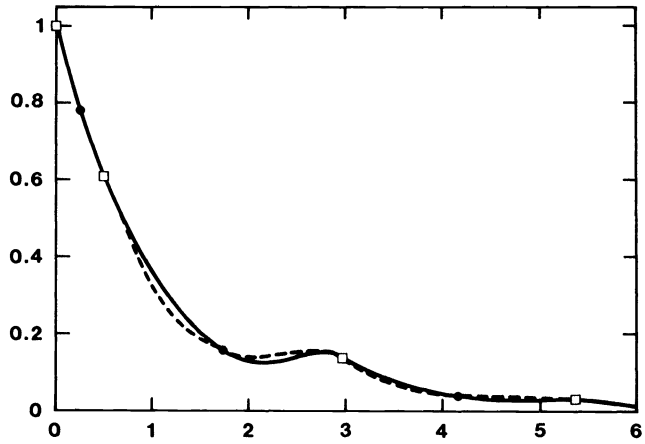


FIG. 4.4. *Quintic Hermite* (——) *and quartic Hermite* (- - -) *interpolants to fifth order solution values at integration points and midpoints computed by* DPS *code with* $\tau = 10^{-1}$. □ *denotes an integration point*; ● *denotes a midpoint value.*

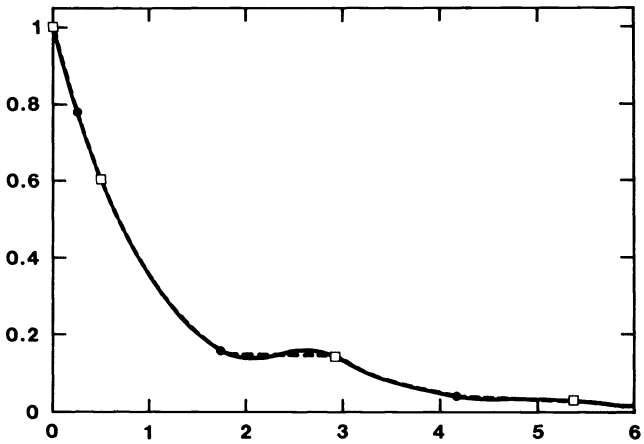


FIG. 4.5. *Cubic Hermite* (——) *and quadratic rational* (- - -) *interpolants to fifth order solution values at integration points and midpoints computed by* DPS *code with* $\tau = 10^{-1}$. □ *denotes an integration point*; ● *denotes a midpoint value.*

the cubic fit is visually very similar to the quintic and that only the rational quadratic interpolant provides a locally monotonic fit to the locally monotonic data.

To illustrate better the behaviour of the local errors, we give in Table 4.1 the errors in the computed solution relative to the local solution for each of the local error tolerances 10^{-1} and 10^{-2}. Observe that the fourth order midpoint value is always less accurate than the fifth order midpoint value and that the error of the fifth order value of the end of each interval is greater than the error in either of the midpoint values. From the point of view of graph-plotting, it is rather unfortunate that the sign of both errors at the midpoint is always the opposite of the sign of the error at the endpoint. Because the example used does not seem special, it appears likely that this behaviour is common.

We consider now in more detail the task of achieving a monotonic fit to monotonic data—a problem studied widely in the literature. Fortunately the case of fitting by

TABLE 4.1
Errors in values computed using a DPS *code relative to the local solution through the endpoint values.*

$TOL = 10^{-1}$					
x_n	$x_n + 0.5h$	$x_n + h$	E_1	E_2	E_3
0.00	0.25	0.50	−2.9E−6	−1.5E−5	5.8E−6
0.50	1.73	2.97	−2.0E−2	−8.3E−2	8.9E−2
2.97	4.17	5.37	−4.0E−3	−1.6E−2	1.7E−2
5.37	6.53	7.68	−7.1E−4	−2.8E−3	2.9E−3
7.68	8.84	10.00	−1.4E−4	−5.5E−4	5.7E−4
$TOL = 10^{-2}$					
x_n	$x_n + 0.5h$	$x_n + h$	E_1	E_2	E_3
0.00	0.25	0.50	−2.9E−6	−1.5E−5	5.8E−6
0.50	1.28	2.06	−1.5E−3	−5.5E−2	4.6E−3
2.06	2.86	3.66	−3.8E−4	−1.4E−3	1.2E−3
3.66	4.74	5.82	−4.5E−4	−1.7E−3	1.8E−3
5.82	6.87	7.91	−6.6E−5	−2.5E−4	2.5E−4
7.91	8.96	10.00	−1.2E−5	−4.4E−5	4.4E−5

E_1 = *Fifth order midpoint value—local solution.*
E_2 = *Fourth order midpoint value—local solution.*
E_3 = *Fifth order end point value—local solution.*

cubic Hermite polynomials has been investigated thoroughly [14]. Let

$$\Delta = (u_{n+1} - u_n)/h, \quad \dot{p}_n = \dot{u}_n/\Delta, \quad \dot{p}_{n+1} = \dot{u}_{n+1}/\Delta. \tag{4.2}$$

Then the cubic Hermite polynomial fit to $(x_n, u_n, \dot{u}_n)$ and $(x_n + h, u_{n+1}, \dot{u}_{n+1})$ is monotonic if and only if $(\dot{p}_n, \dot{p}_{n+1})$ lies in that finite region, R, of the first quadrant of the (x, y)-plane bounded by

$$x^2 + y^2 + xy - 6x - 6y + 9 = 0. \tag{4.3}$$

Observe that the data is monotonic if and only if $(\dot{p}_n, \dot{p}_{n+1})$ lies in the first quadrant. This means that the cubic Hermite fit reproduces monotonic behaviour for only some monotonic data.

In Fig. 4.6 we plot the data corresponding to the tolerance 10^{-1} in Table 4.1, relative to the region R. The data plotted comprise fourth order midpoints with fifth order endpoints, and fifth order midpoints with fifth order endpoints. The fifth order values lie in the first quadrant, but some are outside R, that is, they represent monotonic data which cannot be fitted monotonically by piecewise cubic Hermite polynomials. Observe that some of the fourth order values lie in the third quadrant, demonstrating quite dramatically that they do not represent monotonic data. The local solution values plotted on the same graph would coincide with those fifth order plotted values inside R.

For the case of monotonic data a technique exists which gives a globally C^1 monotonic fit of the same order of accuracy as the cubic Hermite fit, yet which is local to a single interval of the integration at a time. Gregory and Delbourgo [15] achieve this using a rational quadratic fit. With the definitions of (4.2) the rational quadratic interpolant may be written in the centered form

$$\frac{(u_{n+1} + u_n)}{2} + \frac{(u_{n+1} - u_n)[\theta + \frac{1}{2}(\dot{p}_n - \dot{p}_{n+1})(\frac{1}{4} - \theta^2)]}{1 + (\dot{p}_{n+1} + \dot{p}_n - 2)(\frac{1}{4} - \theta^2)} \tag{4.4}$$

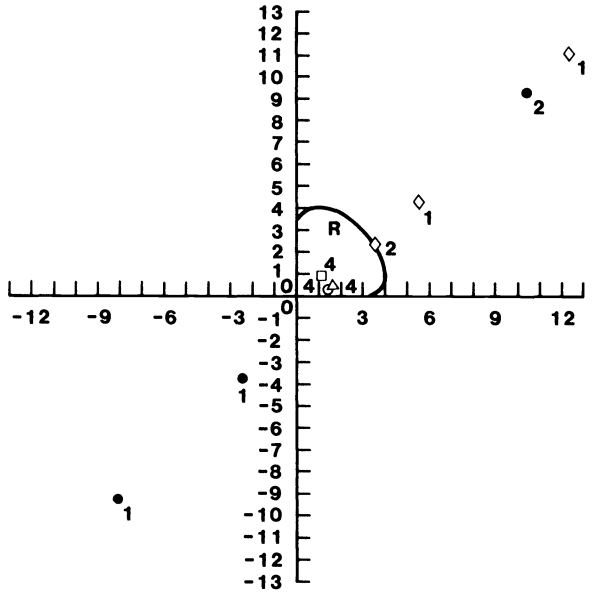


FIG. 4.6. *Solution values from* DPS *code with* $\tau = 10^{-1}$ *plotted in* (p_n, p_{n+1}) *plane. R is the region defined by* (4.3). ● *denotes fourth order data (midpoint-endpoint);* ◇ *denotes fifth order data (midpoint-endpoint);* △ *denotes fifth order data (endpoint-midpoint);* ○ *denotes fourth order data (endpoint-midpoint);* □ *denotes fourth order/fifth order data from the initial integration step. Numerical values printed by the above symbols denote the number of occurrences of the symbol at a given point.*

where $\theta = (x - x_n - h/2)/h$. This form is more stable numerically than that given by Gregory and Delbourgo. They show that (4.4) is monotonic on $[x_n, x_n + h]$ if and only if the data is monotonic on $[x_n, x_n + h]$.

The remaining problem facing us is what to do on those subintervals where the data is not monotonic. The fit (4.4) may then have poles in $[x_n, x_n + h]$. Even if it does not, it is no more accurate than the cubic Hermite interpolant. Thus we have adopted the straightforward stratagem of using (4.4) when the data is locally monotonic and using a cubic Hermite fit otherwise. In most integrations the quadratic rational will be used almost always. We see that the resulting fits in Figs. 4.3 and 4.5 are very satisfactory visually.

Recall that in the DPS method the derivative of the midpoint value is not used to advance the integration and so it is natural to ask here whether, for the purposes of interpolation, we can avoid calculating it directly and instead approximate it from the available data. Gregory and Delbourgo give a simple linear scheme for approximating the midpoint derivative from the local solution values. Using this approximation, they preserve monotonicity but the order of approximation of (4.4) is lowered. It is also possible to use an approximation based on the endpoint derivative values to obtain the same effect. If we insist on preserving the order of the approximation and on using a linear fitting technique (such as polynomial interpolation), then we cannot guarantee that the midpoint derivative will share the sign of its endpoint neighbours. Hence to obtain a visually pleasing fit of the maximum achievable order with either (4.4) or the cubic Hermite fit and to use midpoint values in our interpolation, we recommend evaluating the midpoint derivative directly.

In [11] we consider the monotonicity properties of the quintic Hermite fit relative to those of a pair of cubic Hermite fits. We observe little difference in the two approaches.

This observation and the preceding discussion lead us to conclude that at most two interpolation techniques are necessary, in both cases using fifth order midpoint values and their derivatives. In those cases when it is most important to provide a visually pleasing representation or simply to preserve monotonicity, we recommend the use of the quadratic rational interpolant combined with the cubic Hermite interpolant for those subintervals where the data is not monotone. In all other cases the quintic Hermite fit is recommended, and for sufficiently stringent local error tolerances this fit may be adequate for all purposes.

Acknowledgment. We wish to thank Dr. J. Williams for bringing the work of Gregory and Delbourgo to our attention and for a useful discussion of the question of fitting to monotonic data.

REFERENCES

[1] L. F. SHAMPINE (1985), *Interpolation for Runge-Kutta methods*, SIAM J. Numer. Anal., 22, pp. 1014-1027.

[2] ———, (1985), *Some practical Runge-Kutta formulas*, Math. Comp., to appear.

[3] M. K. HORN (1981), *Scaled Runge-Kutta algorithms for handling dense output*, Rept. DFVLR-FB81-13, DFVLR, Oberpfaffenhofen, Federal Republic of Germany.

[4] ———, (1982), *Scaled Runge-Kutta algorithms for treating the problem of dense output*, Rept. NASA TMX-58239, L. B. Johnson Space Center, Houston, TX.

[5] ———, (1983), *Fourth- and fifth-order, scaled Runge-Kutta algorithms for treating dense output*, SIAM J. Numer. Anal., 20, pp. 558-568.

[6] H. SHINTANI (1966), *Two-step processes by one-step methods of order 3 and of order 4*, J. Sci. Hiroshima Univ., Ser. A-1, 30, pp. 183-195.

[7] R. ENGLAND (1969), *Error estimates for Runge-Kutta type solutions to systems of ordinary differential equations*, Comput. J., 12, pp. 166-170.

[8] L. F. SHAMPINE (1985), *Local error estimation by doubling*, Computing, 34, pp. 179-190.

[9] L. F. SHAMPINE AND H. A. WATTS (1971), *Comparing error estimators for Runge-Kutta methods*, Math. Comp., 25, pp. 445-455.

[10] C. T. H. BAKER (1970), *The error in polynomial interpolation*, Numer. Math., 15, pp. 315-319.

[11] I. GLADWELL, L. F. SHAMPINE, L. S. BACA AND R. W. BRANKIN (1985), *Practical aspects of interpolation in Runge-Kutta codes*, Numer. Anal. Rept. 102, Dept. Math., Univ. Manchester, England.

[12] A. A. B. PRITSKER (1974), *The* GASP IV *Simulation Language*, John Wiley, New York.

[13] I. GLADWELL (1979), *Initial value routines in the NAG Library*, ACM Trans. Math. Software, 5, pp. 386-400.

[14] F. N. FRITSCH AND R. E. CARLSON (1980), *Monotonic piecewise cubic interpolation*, SIAM J. Numer. Anal., 17, pp. 238-246.

[15] J. A. GREGORY AND R. DELBOURGO (1982), *Piecewise rational quadratic interpolation to monotonic data*, IMA J. Numer. Anal., 2, pp. 123-130.

[16] I. GLADWELL (1969), *A note on a paper by Birkhoff and Priver*, SIAM J. Numer. Anal., 6, pp. 294-298.

[17] S. PRUESS (1985), *Stability bounds for local Lagrangian interpolation*, Dept. Math., Stat., Univ. New Mexico, to appear.

[18] W. H. ENRIGHT, K. R. JACKSON, S. P. NORSETT AND P. G. THOMSEN (1985), *Interpolants for Runge-Kutta formulas*, Rept. No. 180/85, Dept. Comput. Sci., Univ. Toronto, Canada.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 3, May 1987
© 1987 Society for Industrial and Applied Mathematics
010

PARALLEL SOLUTION OF ODE's BY MULTI-BLOCK METHODS*

MOODY T. CHU† AND HANS HAMILTON†

Abstract. The notion of linear multi-step methods for solving ordinary differential equations is generalized to a class of multi-block methods. In a multi-block method step values are all obtained together in a single block advance which is accomplished by allocating the parallel tasks on separate processors. The expected benefit of multi-block methods is the speedup in the computation of solutions. The basic formulation is described. Examples are given to demonstrate the existence of such schemes. The predictor-corrector type combination is formed and the resulting stability problem is considered. Test results of one of these multi-block methods on the Denelcor HEP machine are reported.

Key words. multi-block methods, parallel processing, absolute stability, Denelcor HEP

AMS(MOS) subject classification. 65L05

1. Introduction. We are interested in the problem of speeding up the numerical integration of a system of ordinary differential equations by parallel computations. We shall study a class of multi-block methods and report the test results made on the Denelcor HEP machine.

Parallel methods for numerical solutions of systems of ordinary differential equations have been proposed by various researchers. Generally speaking, the parallelism may be achieved by partitioning the tasks either across the algorithm or across the system of equations. In the review paper [4], Franklin exemplified this idea with three classes of parallel algorithms: the equation segmentation methods used quite heavily by engineers in the field of dynamic system simulation, the parallel predictor-corrector methods developed by Miranker and Liniger [8], and the parallel block implicit methods considered by Shampine and Watts [13], [14], Rosser [12], and Worland [15]. Other methods designed or adaptable for parallel processing also exist. Readers can find them in, for example, papers by Birta and Abou-Rabia [2], Donelson and Hansen [3], Mitra [9], Nievergelt [10], Andria et al. [1], and also several other papers cited by Poole and Voigt in [11].

In this paper, we consider parallel block methods. The procedures, described as k-block, r-point methods, generalize those developed by Shampine and Watts in the sense that their schemes correspond to a special case of our 1-block, r-point methods. Also, the cyclic composite multi-step methods originally proposed by Donelson and Hansen [3] for serial computations to circumvent the very real barrier in the condition of zero-stability for multi-step methods, can be easily reformulated in terms of our multi-block settings.

In a k-block, r-point method, time is divided into a series of blocks with each block containing r steps at which solutions to differential systems are to be found; the computation proceeds in blocks, i.e., the computation is based on the computed values at the earlier k blocks. Within a block it is possible to assign the computational tasks at each time step to a single processor and, thus, the computations at all future steps can be performed simultaneously in parallel. As a consequence, speedup in the integration and the function evaluation is expected.

The basic setting of a general k-block r-point method is discussed in § 2 where we also give four examples to show the feasibility of this approach. Section 3 is devoted to the concept of predictor-corrector methods. In § 4 we examine the absolute stability

* Received by the editors May 29, 1985, and in revised form December 23, 1985.
† Department of Mathematics, North Carolina State University, Raleigh, North Carolina 27695-8205.

properties of block methods. Some experimental results are included there. Finally, we discuss the implementation and testing of one particular block method on the Denelcor HEP machine. The test results are reported in § 5.

2. Multi-block methods. We shall be interested in obtaining a numerical solution of

$$\begin{aligned} y'(x) &= f(x, y(x)), \\ y(a) &= y_0, \qquad a \leqq x \leqq b, \end{aligned} \tag{2.1}$$

where we make the usual assumption that f is continuous and satisfies a Lipschitz condition on the region $[a, b]\times(-\infty, \infty)$. The discussions in what follows can be generalized to higher dimensional systems in an obvious way.

For $h \in (0, h_0)$ let $x_k = a + kh$, y_k represent a numerical approximation to the exact solution $y(x_k)$ and $f_k = f(x_k, y_k)$. We also introduce $m = 0, 1, 2, \cdots$ representing the block number, $n = mr$ the first step number in the m-th block, and r the proposed block size (usually the same as the number of processors).

DEFINITION 2.1.1. Let Y_m and F_m be vectors defined by

$$Y_m = [y_n, y_{n+1}, \cdots, y_{n+r-1}]^T, \tag{2.2}$$

$$F_m = [f_n, f_{n+1}, \cdots, f_{n+r-1}]^T. \tag{2.3}$$

Then a general k-block, r-point method is a matrix finite difference equation of the form

$$Y_m = \sum_{i=1}^{k} A_i Y_{m-i} + h \sum_{i=0}^{k} B_i F_{m-i}, \tag{2.4}$$

where all A_i's and B_i's are certain properly chosen $r \times r$ matrix coefficients.

If $r = 1$, then (2.4) is just the classical k-step method. When $B_0 = 0$, we say that (2.4) is an explicit method; otherwise it is implicit. Note that when $r > 1$, there is a combinatorial question of parcelling out time steps among blocks. For instance, when $r = 2$, one can define the block by either $Y_m = [y_{2m}, y_{2m+1}]^T$ as in (2.2) or $Y_m = [y_{2m+1}, y_{2m+2}]^T$ as in [13], [14]. The possible significant effect of these different block definitions is reflected only in the necessary starting procedures where one definition may demand more starting values than the other, but not in the successive block advancing procedures.

DEFINITION 2.1.2. Let $y(x)$ be the exact solution of (2.1) and

$$Z_m = [y(x_n), y(x_{n+1}), \cdots, y(x_{n+r-1})]^T. \tag{2.5}$$

Then the local truncation error E_m of the scheme (2.4) at the mth block is defined to be

$$E_m = Z_m - \sum_{i=1}^{k} A_i Z_{m-i} - h \sum_{i=0}^{k} B_i Z'_{m-i}. \tag{2.6}$$

There are several ways to determine the matrix coefficients involved in (2.4); the Taylor expansion method and the method of undetermined coefficients are probably the most straightforward ones. When applying any of these methods to derive the scheme (2.4), one may often suppress the full usage of the information of certain blocks by setting certain columns of the corresponding matrix coefficients to be identically zero. This action is especially suitable when the desired order of accuracy is much lower than what would have been achieved if all steps inside the blocks were active. We now give some examples below to demonstrate the existence of schemes of the form (2.4). Also listed are the corresponding principal local truncation errors.

Example 1. $k=2$, $r=2$, explicit, partial usage of block values.

$$
\begin{aligned}
Y_m = &\begin{bmatrix} 0, & 1+a_1+a_2 \\ 0, & 1+a_3+a_4 \end{bmatrix} Y_{m-2} + \begin{bmatrix} -a_1, & -a_2 \\ -a_3, & -a_4 \end{bmatrix} Y_{m-1} \\
&+ \frac{h}{12}\begin{bmatrix} 0, & 9+5a_1+4a_2 \\ 0, & 32+5a_3+4a_4 \end{bmatrix} F_{m-2} \\
&+ \frac{h}{12}\begin{bmatrix} 8a_1+16a_2, & 27-a_1+4a_2 \\ -64+8a_3+16a_4, & 80-a_3+4a_4 \end{bmatrix} F_{m-1},
\end{aligned} \tag{2.7}
$$

$$
\text{Local error} = \frac{1}{72}\begin{bmatrix} (27+3a_1)h^4y^{(4)} \\ (192+3a_3)h^4y^{(4)} \end{bmatrix}. \tag{2.8}
$$

Example 2. $k=1$, $r=2$, implicit, lower order of accuracy.

$$
\begin{aligned}
Y_m = &\begin{bmatrix} 1+a_1, & -a_1 \\ 1+a_2, & -a_2 \end{bmatrix} Y_{m-1} \\
&+ \frac{h}{72}\begin{bmatrix} 56+46a_1-24b_1, & 72b_1 \\ 27+27a_2, & 81+57a_2 \end{bmatrix} F_{m-1} \\
&+ \frac{h}{72}\begin{bmatrix} 120+42a_1-72b_1, & -32-16a_1+24b_1 \\ 81-15a_2, & 27+3a_2 \end{bmatrix} F_m,
\end{aligned} \tag{2.9}
$$

$$
\text{Local error} = \frac{1}{720}\begin{bmatrix} (320+190a_1-240b_1)h^4y^{(4)} \\ (-27-19a_2)h^5y^{(5)} \end{bmatrix}. \tag{2.10}
$$

Example 3. $k=2$, $r=2$, explicit, full usage of block values.

$$
\begin{aligned}
Y_m = &\begin{bmatrix} 1+a_1+a_2+a_3, & -a_1 \\ 1+a_4+a_5+a_6, & -a_4 \end{bmatrix} Y_{m-2} + \begin{bmatrix} -a_2, & -a_3 \\ -a_5, & -a_6 \end{bmatrix} Y_{m-1} \\
&+ \frac{h}{24}\begin{bmatrix} 9a_1+8a_2+9a_3, & 64+19a_1+32a_2+27a_3 \\ -55+9a_4+8a_5+9a_6, & 275+19a_4+32a_5+27a_6 \end{bmatrix} F_{m-2} \\
&+ \frac{h}{24}\begin{bmatrix} -32-5a_1+8a_2+27a_3, & 64+a_1+9a_3 \\ -325-5a_4+8a_5+27a_6, & 225+a_4+9a_6 \end{bmatrix} F_{m-1},
\end{aligned} \tag{2.11}
$$

$$
\text{Local error} = \frac{1}{720}\begin{bmatrix} (224-19a_1-8a_2-27a_3)h^5y^{(5)} \\ (2125-19a_4-8a_5-27a_6)h^5y^{(5)} \end{bmatrix}. \tag{2.12}
$$

Example 4. $k=1$, $r=2$, implicit, higher order of accuracy.

$$
\begin{aligned}
Y_m = &\begin{bmatrix} 1+a_1, & -a_1 \\ 1+a_2, & -a_2 \end{bmatrix} Y_{m-1} + \frac{h}{24}\begin{bmatrix} 8+9a_1, & 32+19a_1 \\ 9+9a_2, & 27+19a_2 \end{bmatrix} F_{m-1} \\
&+ \frac{h}{24}\begin{bmatrix} 8-5a_1, & a_1 \\ 27-5a_2, & 9+a_2 \end{bmatrix} F_m,
\end{aligned} \tag{2.13}
$$

$$
\text{Local error} = \frac{1}{720}\begin{bmatrix} (-8-19a_1)h^5y^{(5)} \\ (-27-19a_2)h^5y^{(5)} \end{bmatrix}. \tag{2.14}
$$

It is interesting to note that the predictor used in [13] is a special case of (2.7) with parameter values $a_1=a_2=a_3=a_4=-\frac{1}{3}$, and the well-known Clippinger and Dimsdale formula used as the corrector in [13] corresponds to (2.9) with parameter values $a_1=a_2=-1$ and $b_1=\frac{5}{12}$.

It was pointed out in [13] that the stability problem would appear to be the most serious limitation of block methods when used as a predictor-corrector type combination. We shall see in § 4 that by taking advantage of the free parameters of the preceding formulas, the stability properties can be greatly improved.

The procedure (2.4) may be regarded as a package of r multi-step methods (possibly with different step numbers) working independently. The convergence of this procedure, therefore, is assured by the standard arguments for multi-step methods. We shall omit the formal proof of convergence in this presentation.

3. Predictor-corrector methods. Based on the usual setting for multi-step methods, one can easily set up the analogies of PC modes for multi-block methods. Henceforth, let the quantities related to the predictor be denoted by the mark "*". Then the P(EC)^{t}E mode, for example, is defined as follows:

$$
\begin{aligned}
&\text{P: } Y_m^{(0)} = \sum_{i=1}^{k^*} A_i^* Y_{m-i}^{(t)} + h \sum_{i=1}^{k^*} B_i^* F_{m-i}^{(t)};\\
&\text{E: } F_m^{(s)} = [f(x_n, y_n^{(s)}, \cdots, f(x_{n+r-1}, y_{n+r-1}^{(s)})]^T,\\
&\text{C: } Y_m^{(s+1)} = \sum_{i=1}^{k} A_i Y_{m-i}^{(t)} + h \sum_{i=1}^{k} B_i F_{m-i}^{(t)} + hB_0 F_m^{(s)} \quad \text{for } s = 0, \cdots, t-1,\\
&\text{E: } F_m^{(t)} = [f(x_n, y_n^{(t)}), \cdots, f(x_{n+r-1}, y_{n+r-1}^{(t)})]^T.
\end{aligned}
\tag{3.1}
$$

Let us consider now the effect of such a combination on the local truncation error. Under the localizing assumption (i.e., assuming that $Z_{m-i} = Y_{m-i}^{(t)}$ for all $i = 1, 2, \cdots, k$), it can be shown that

$$
\begin{aligned}
Z_m - Y_m^{(s+1)} &= hB_0(Z'_m - F_m^{(s)}) + E_m\\
&\approx hLB_0(Z_m - Y_m^{(s)}) + E_m,
\end{aligned}
\tag{3.2}
$$

where L is bounded by the Lipschitz constant of f with respect to y. It is seen that formally the local truncation error of this multi-block algorithm behaves in exactly the same way as that of the multi-step algorithm. For the discussion of the latter case readers are referred to [6]. The expression (3.2) gives us a practical idea of choosing the predictor and the corrector when forming the combination (3.1). Let us consider the simplest case when $t = 1$ as an example. Then (3.2) suggests that one should use a predictor in which all components have the same order of accuracy, because the multiplication by the matrix B_0 will usually mix up all components and, hence, contaminate the superior accuracy if there is any. It also suggests that for economical purposes one should use a corrector in which each component has equal or one order higher accuracy than the predictor. In this case, the local truncation error of the PECE mode is of the same order as the corrector. Finally, it suggests that restricting oneself to the case $k^* = k+1$ is generally sufficiently for the practical purpose.

4. Stability. From now on let it be assumed that $k^* = k+1 = p$. We find that the following definitions of vectors facilitate discussions of the stability problem.

Let $U_m^{(s)}$ and $V_m^{(s)}$ for $s = 0, 1, \cdots, t$ be $pr \times 1$ vectors defined by

$$
U_m^{(s)} = \begin{bmatrix} Y_m^{(s)} \\ Y_{m-1}^{(t)} \\ \vdots \\ Y_{m-p+1}^{(t)} \end{bmatrix}, \qquad V_m^{(s)} = \begin{bmatrix} F_m^{(s)} \\ F_{m-1}^{(t)} \\ \vdots \\ F_{m-p+1}^{(t)} \end{bmatrix}.
\tag{4.1}
$$

It is easy to see that (3.1) is equivalent to the following scheme:

$$U_m^{(0)} = \begin{bmatrix} A_1^* & A_2^* & \cdots & A_{p-1}^* & A_p^* \\ I & 0 & & 0 & 0 \\ \vdots & & & & \vdots \\ & & & 0 & 0 \\ 0 & \cdots & & I & 0 \end{bmatrix} U_{m-1}^{(t)}$$

$$+h \begin{bmatrix} B_1^* & B_2^* & \cdots & B_{p-1}^* & B_p^* \\ 0 & 0 & & 0 & 0 \\ \vdots & & & & \vdots \\ 0 & & \cdots & & 0 \end{bmatrix} V_{m-1}^{(t)} \tag{4.2}$$

$$\equiv \mathbf{A}^* U_{m-1}^{(t)} + h\mathbf{B}^* V_{m-1}^{(t)};$$

$$U_m^{(s+1)} = \begin{bmatrix} A_1 & A_2 & \cdots & A_{p-1} & 0 \\ I & 0 & & 0 & 0 \\ 0 & & & & \vdots \\ \vdots & & & 0 & 0 \\ 0 & \cdots & & I & 0 \end{bmatrix} U_{m-1}^{(t)}$$

$$+h \begin{bmatrix} B_0 & B_1 & \cdots & B_{p-2} & B_{p-1} \\ 0 & 0 & & 0 & 0 \\ \vdots & & & & \vdots \\ 0 & & \cdots & & 0 \end{bmatrix} V_m^{(s)} \tag{4.3}$$

$$\equiv \mathbf{A} U_{m-1}^{(t)} + h\mathbf{B} V_m^{(s)},$$

where $\mathbf{A}^*$, $\mathbf{B}^*$, $\mathbf{A}$ and $\mathbf{B}$ are $pr \times pr$ square matrices defined accordingly, and I represents an $r \times r$ identity matrix. When applying this scheme to the model problem

$$y' = \lambda y \quad \text{with} \quad \lambda < 0 \tag{4.4}$$

and letting $\mathbf{h} = \lambda h$, we find that

$$U_m^{(0)} = (\mathbf{A}^* + \mathbf{hB}^*) U_{m-1}^{(t)}, \tag{4.5}$$

$$U_m^{(s+1)} = \mathbf{A} U_{m-1}^{(t)} + \mathbf{hB} U_m^{(s)}, \tag{4.6}$$

and hence

$$\begin{aligned} U_m^{(t)} &= \{[I + (\mathbf{hB}) + \cdots + (\mathbf{hB})^{t-1}]\mathbf{A} + (\mathbf{hB})^t(\mathbf{A}^* + \mathbf{hB}^*)\} U_{m-1}^{(t)} \\ &\equiv T(\mathbf{h}; \mathbf{A}, \mathbf{B}, \mathbf{A}^*, \mathbf{B}^*) U_{m-1}^{(t)}. \end{aligned} \tag{4.7}$$

From (4.7) it is clear that the PC block algorithm (3.1) is absolutely stable for a value $\mathbf{h}$ if and only if

$$s(T(\mathbf{h}; \mathbf{A}, \mathbf{B}, \mathbf{A}^*, \mathbf{B}^*)) < 1 \tag{4.8}$$

where $s(T)$ represents the spectral radius of the matrix T.

As we have seen earlier, in accordance with certain prescribed orders of accuracy, generally one can determine the components in the matrix coefficients $\mathbf{A}$, $\mathbf{B}$, $\mathbf{A}^*$, $\mathbf{B}^*$

up to some free parameters. These parameters are not totally free in practice because some of them must satisfy the constraint of the so called zero-stability.

DEFINITION 4.1. A multi-block method (2.4) is said to be zero-stable if solutions of the equation

$$Y_m = \sum_{i=1}^{k} A_i Y_{m-i} \tag{4.9}$$

remain bounded for all m.

In terms of (4.2) and (4.3), the zero-stability is equivalent to, respectively, that there are no eigenvalues of $\mathbf{A}^*$ and $\mathbf{A}$ lying outside the unit circle and that the eigenvalues on the unit circle have simple elementary divisors.

Now an interesting optimization problem has arisen: Given $\mathbf{A}^*$, $\mathbf{B}^*$, $\mathbf{A}$ and $\mathbf{B}$ being determined up to some free parameters (so that a certain prescribed order of accuracy is fulfilled), we want to find the infimum of $\mathbf{h}$ from all possible parameter values which are subject to the constraint that the corresponding matrices $\mathbf{A}^*$, $\mathbf{B}^*$, $\mathbf{A}$ and $\mathbf{B}$ satisfy $s(T(\mathbf{h}; \mathbf{A}^*, \mathbf{B}^*, \mathbf{A}, \mathbf{B})) < 1$, and that $\mathbf{A}$ is zero-stable. (It turns out that the zero-stability of $\mathbf{A}^*$ is immaterial.) This problem is by no means simple because both the objective and the constraint functions are implicitly described, and can be shown numerically to possess many discontinuous or nondifferentiable points. Some experimental results concerning the left endpoint H_B of the interval of absolute stability for the PECE mode using (2.7) as the predictor and (2.9) as the corrector are collected in Table 1. As is seen, the PC pair studied by Shampine and Watts in [13] is only a special case of a 7-parameter family of methods and certainly does not give the best absolute stability. Table 2 contains similar results for the same mode using (2.11) as the predictor and (2.13) as the corrector.

Note that the PC pair of (2.11) and (2.13) constitutes a fourth order block method. At the time this paper was written, the best stability boundary we had obtained was

TABLE 1

P				C			PECE
a_1	a_2	a_3	a_4	a_1	a_2	b_1	H_B
−1/3	−1/3	−1/3	−1/3	−1.0	−1.0	5/12	−0.439
−1/3	−1/3	−1/3	−1/3	−1.0	0.0	5/12	−0.848
2.0	−7/3	−1/3	−1/3	−1.0	−1/5	7/12	−0.955
7/3	−8/3	−1/3	−1/3	−1.0	−3/5	11/12	−1.130
7/3	−8/3	−1/3	−1/3	−1.0	−3/5	1.0	−1.249

TABLE 2

P						C		PECE
a_1	a_2	a_3	a_4	a_5	a_6	a_1	a_2	H_B
−1.000	−0.667	−0.333	−1.000	−0.667	−0.667	−0.500	−1.000	−0.311
−1.000	−0.667	−0.333	−1.000	−0.667	−0.667	−1.000	−0.500	−0.603
−1.233	−0.567	−0.333	−1.133	−0.567	−0.567	−0.999	−0.200	−0.680
−1.333	−0.567	−0.133	−1.333	−0.467	−0.567	−1.000	−1.000	−0.692
−1.233	−0.467	−0.033	−1.333	−0.767	−0.367	−1.000	−1.000	−0.740

$H_B = -0.810$ by setting

$$(4.10) \qquad \text{P:} \begin{cases} a_1 = -1.23000, \\ a_2 = -0.46300, \\ a_3 = -0.02770, \\ a_4 = -1.33000, \\ a_5 = -0.77212, \\ a_6 = -0.37000, \end{cases} \qquad \text{C:} \begin{cases} a_1 = -1.01000, \\ a_2 = -0.04520. \end{cases}$$

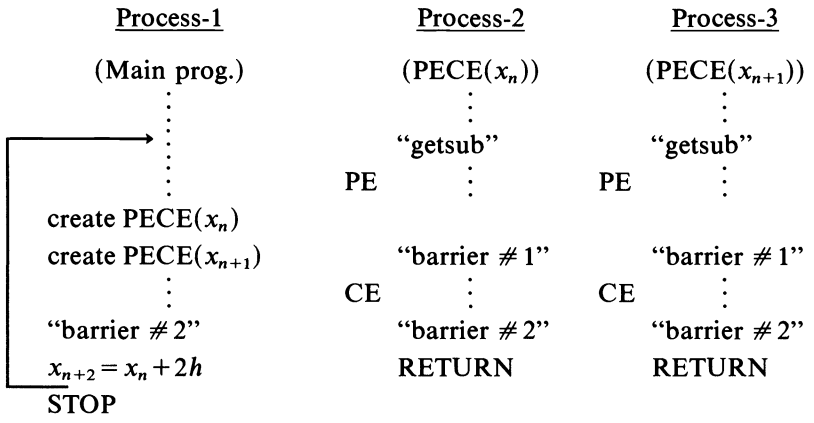
We recall that the stability boundary H_{ABM} of the fourth order Adams-Bashforth-Moulton (ABM) pair in PECE mode is $H_{\text{ABM}} = -1.285$ [5]. Thus, it seems that our multi-block method is severely restricted by its stability property and is not competitive with the fourth order ABM method. One must be aware, however, that each call of an r-point, multi-block method with step size h advances the time by the total length rh. Consider the case $r = 2$. Suppose the length of integration for a particular problem is T and both methods take the maximum allowable stepsize for this problem. Then the ABM method will take T/H_{ABM} steps and $2T/H_{\text{ABM}}$ function evaluations. The multi-block method will take $T/(2H_B)$ blocks and $2T/H_B$ function evaluations. Although the multi-block method uses more function evaluations than the ABM method, the parallel nature of the function evaluation effectively halves the time needed for these evaluations, i.e. it will need only T/H_B real function evaluation time. This argument is applicable to general comparisons and of compatible (same order, same mode, etc.) multi-step and multi-block methods.

The data we have furnished in (4.10) are certainly not conclusive. We propose to continue experimenting with that optimization problem for other multi-block schemes. Before ending, we note that a similar problem of optimizing the stability region for a class of parallel predictor-corrector formulas developed in [8] has been studied by Katz et al. [5]. They showed that if the corrector is fixed to be the 4th order Adams-Moulton corrector then the optimally stable parallel predictor is the Adams-Bashforth predictor shifted to the right by one integration step. In this case, the left endpoint of the interval of absolute stability is -0.846. It is also worth noting that in [14] Watts and Shampine derived a class of implicit 1-block methods from Newton-Cotes type interpolatory formulas and conclusively established that these block methods for size $r = 1, 2, \cdots, 8$ are A-stable whereas those for $r = 9, 10$ are not.

5. Implementation on HEP. A prototype code of the PECE scheme from formulas (2.11) and (2.13) has been implemented with fixed stepsize to study the real-time savings in using a multi-processor machine. The machine used was the Denelcor HEP at Argonne National Laboratory. Before examining the results of our tests, let us briefly consider some details of the HEP and how our code was implemented to take advantage of the structure of the HEP. Obviously, most of the ideas discussed in the following about the interprocess communication and synchronization should serve as general patterns on other multi-processing systems as well.

The HEP (Heterogeneous Element Processor) is a large-scale scientific multi-processor system which can execute a number of sequential (SISD) or parallel (MIMD) programs simultaneously. The HEP supports an enhanced version of Fortran-77 which allows for parallel execution of multiple tasks within a single program. These extensions are implemented in HEP-Fortran, but are most commonly used through higher level monitors (macros) supplied by the installation at Argonne. These macros allow for the scheduling of multiple tasks, and other associated concepts useful in multi-processing. Readers are referred to [7] for more detailed discussions.

Various tasks to be done on the HEP are referred to as "processes". The objective of the macros is to schedule the execution of the processes and provide the necessary synchronization between these processes. The macros "getsub" and "barrier" used in our code accomplish these two jobs. Recall that (2.11) predicts at x_n and x_{n+1} and (2.13) corrects at x_n and x_{n+1}. The PECE at x_n is handled by one process while PECE at x_{n+1} is handled by a second process. The main program represents a third process. The allocation of PECE at x_n or x_{n+1} to a process is handled by the "getsub" macro. Since the corrector step requires the function evaluation from the predictor step, the "barrier" macro is used to synchronize the process after PE has been completed by both processes. The "barrier" is again used to synchronize processes after CE has been completed. This may be visualized as follows:



In the above, Process-2 and -3 may be viewed as subroutines running concurrently with Process-1, the main program. The "create" statements are equivalent to a "CALL" in Fortran except that execution continues after the statement is encountered. This provides the concurrent execution of the processes. The "getsub" is contained in subroutine PECE and assigns the process the x_n or x_{n+1} step. The "barrier" statements cause the processes to "wait" until all other processes in that category have completed their work before continuing. The "RETURN" statements cause both Process-2 and -3 to "die" off and the main program to prepare for another block of two steps.

The problem we tested on the HEP was

$$y' = -2y, \qquad y(0) = 1. \tag{5.1}$$

This equation was duplicated to simulate higher-dimensional problems as well as three types of overhead demand of the derivative evaluation routine. This is a "make-work" problem, but suited our purpose better than any specific problem we could find.

The timing on the HEP is handled by the "clock" macro where time is measured in increments of 100 nanoseconds. The performance figures we have reported include only the call to the integrator. I/O and other "housekeeping" chores are not included. The serial code is simply the code that runs on our IBM-3081 system, i.e. it uses only 1 process on the HEP.

Each type of problem has been run with different integration interval lengths with the dimension being varied between 1 and 128 in powers of 2 for each interval. It turns out that the average time per unit length of integration is fairly constant for each dimension. Therefore, we shall report the average behavior only. Theoretically, with three processes we would expect a speedup of 66%, but actually only two processes

are heavily loaded while the main program is idle most of the time. Hence any speedup should be closer to 50%.

In the Type-1 problem we had the number of flops (one flop means either an addition or multiplication) per component in the derivative evaluation match the number of components. In Fig. 1 we have graphed time vs. dimension for the serial and parallel methods. As expected, time behaves like $O(\dim^2)$. The relation of speedup vs. dimension is shown in Fig. 2. As the dimension grows the speedup nears the theoretical 50%. To determine the cause of the less than optimal speedup for the lower-dimensional problems we then run the other two types of problems.

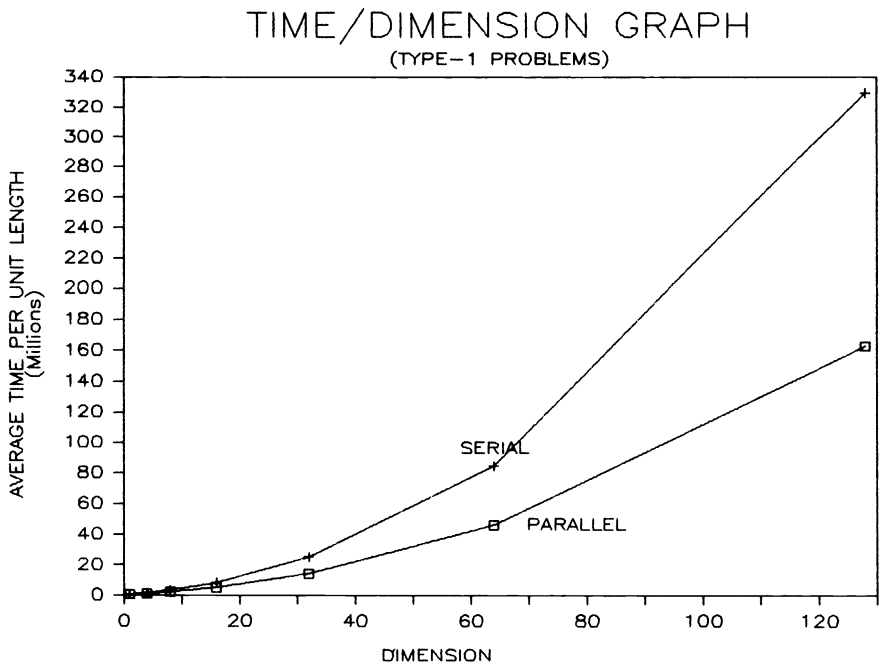


FIG. 1

In the Type-2 problem the number of flops was fixed at a very high level so that the cost of the derivative evaluation would be approximately the same for each dimension. The relation of time vs. dimension for this problem is shown in Fig. 3. In Fig. 4 we have speedup vs. dimension. The speedup is approximately 47% for all the dimensions.

In the Type-3 problem we fixed the number of flops per component at a relatively high level so that time would behave as $O(\dim)$ as the dimension varied. This result is shown in Fig. 5. Speedup vs. dimension is about the same as the Type-2 problem, so Fig. 3 is applicable to this version also.

The tests on the last two types of problems indicate that the theoretical speedup becomes possible when the derivative evaluation cost is high. In the Type-1 problem the derivative evaluation cost was low for the low-dimensional problems, so overhead costs in creating and synchronizing the processes decreased the speedup.

6. Conclusion. In the above we have discussed a class of multi-block methods for solving systems of ordinary differential equations. These block methods appears to be well suited to computers with parallel architectures and, hence, are expected to enhance

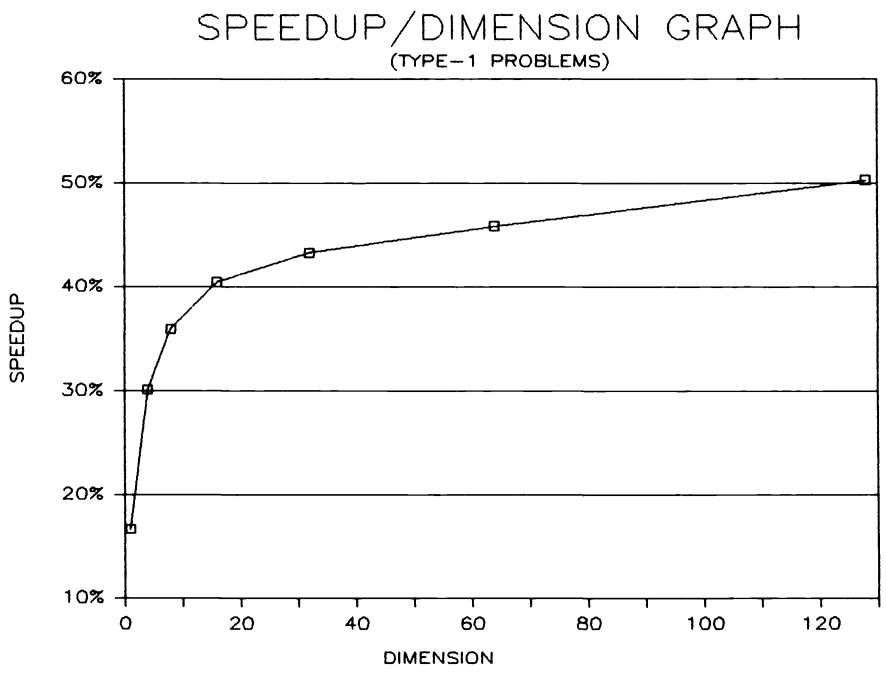

SPEEDUP/DIMENSION GRAPH
(TYPE-1 PROBLEMS)
SPEEDUP
60%
50%
40%
30%
20%
10%
0
20
40
60
80
100
120
DIMENSION

FIG. 2

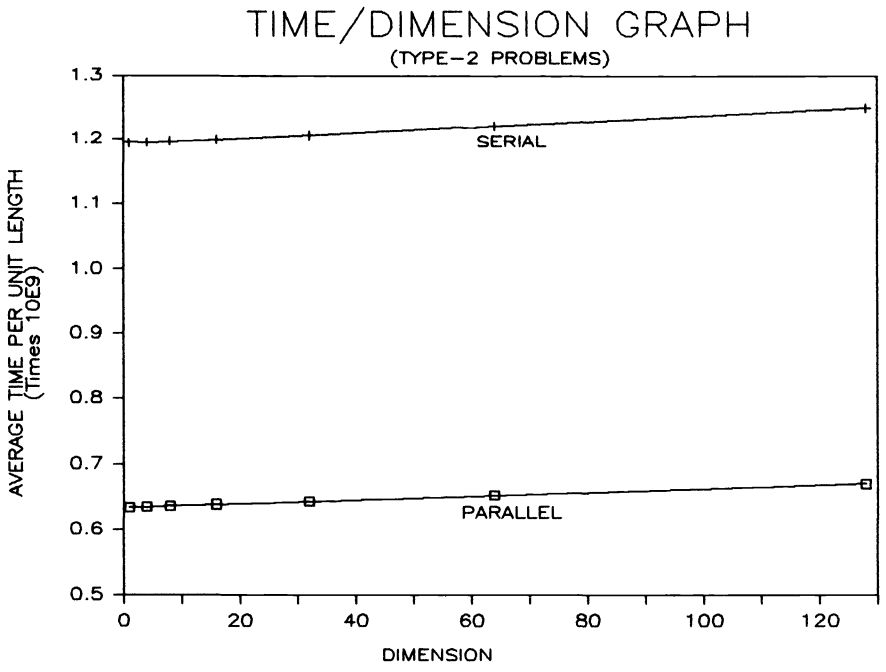

TIME/DIMENSION GRAPH
(TYPE-2 PROBLEMS)
AVERAGE TIME PER UNIT LENGTH
(Times 10E9)
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
SERIAL
PARALLEL
0
20
40
60
80
100
120
DIMENSION

FIG. 3

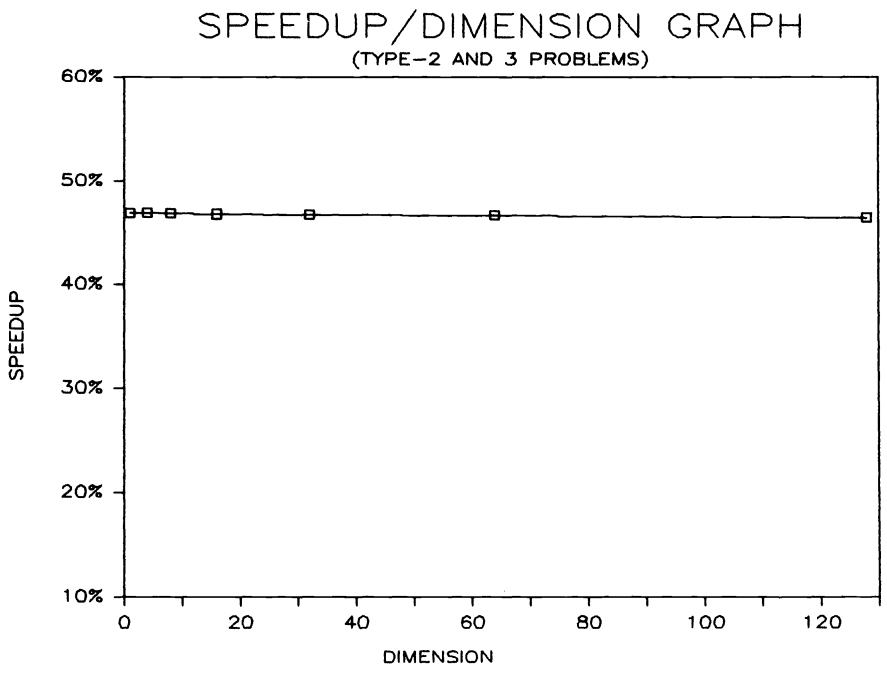

SPEEDUP/DIMENSION GRAPH
(TYPE-2 AND 3 PROBLEMS)
SPEEDUP
60%
50%
40%
30%
20%
10%
0
20
40
60
80
100
120
DIMENSION

FIG. 4

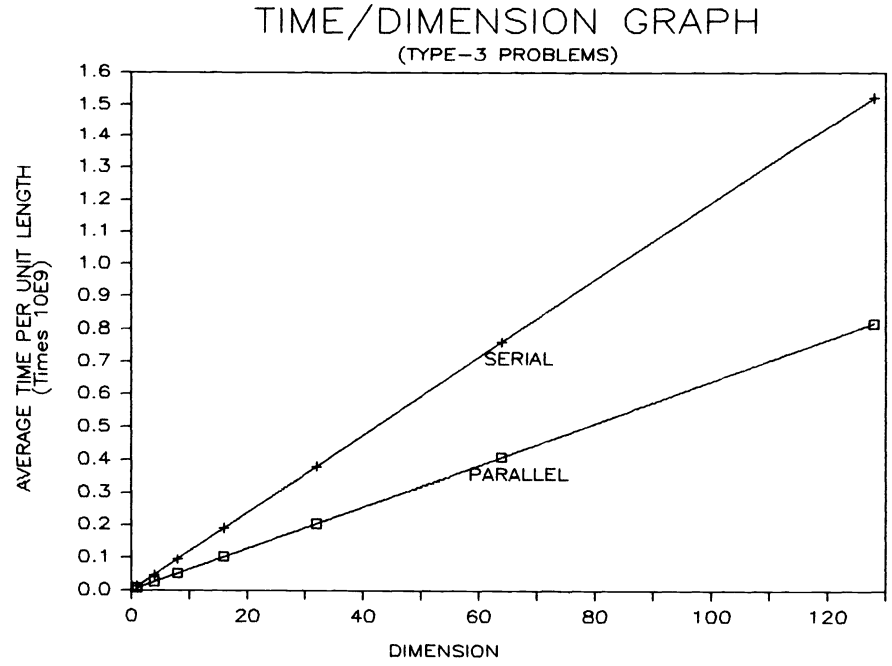

TIME/DIMENSION GRAPH
(TYPE-3 PROBLEMS)
AVERAGE TIME PER UNIT LENGTH
(Times 10E9)
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
SERIAL
PARALLEL
0
20
40
60
80
100
120
DIMENSION

FIG. 5

the speedup of the computation of numerical solutions. The following conclusions are drawn from experiments that we have conducted thus far:

(1) The stability region, though smaller than that of a compatible Adams-Bashforth-Moulton method, is not a restriction as severe as has been reported in the literature. In fact, we have found sets of data which make the corresponding multi-block methods competitive as far as the absolute stability and the accuracy are concerned.

(2) The theoretical speedup 50% can almost be achieved on an MIMD machine, such as the Denelcor HEP system, when the cost of derivative evaluation is high. This is true even for lower-dimensional problems. When the cost of derivative evaluation is low, the overhead costs with creating and synchronizing the processes may have the strongest effect in degrading the performance of speeding-up. The total speedup, however, is still significant.

Finally, since the algorithm we have written is highly vectorizable, it is desirable to experiment with a multi-processor computer which has vector instructions. One such a machine is the CRAY X-MP. When this becomes available we shall report the results elsewhere.

REFERENCES

[1] G. D. ANDRIA, G. D. BYRNE AND D. R. HILL, *Natural spline block implicit methods*, BIT, 13 (1973), pp. 131-144.

[2] L. G. BIRTA AND O. ABOU-RABIA, *A multi-microprocessor system for continuous system simulation*, Technical Report TR8308, Dept. of Computer Science, Univ. Ottawa, Ontario, Canada, 1983.

[3] J. DONELSON AND E. HANSEN, *Cyclic composite multistep predictor-corrector methods*, SIAM J. Numer. Anal., 8 (1971), pp. 137-157.

[4] M. A. FRANKLIN, *Parallel solution of ordinary differential equations*, IEEE Trans. Comput., C-27 (1978), pp. 413-420.

[5] I. N. KATZ, M. A. FRANKLIN AND A. SEN, *Optimally stable parallel predictors for Adams-Moulton correctors*, Comput. Math. Appl., 3 (1977), pp. 217-233.

[6] J. D. LAMBERT, *Computational Methods In Ordinary Differential Equations*, John Wiley, London, 1973.

[7] E. L. LUSK AND R. A. OVERBEEK, *Implementation of monitors with macros: A programming aid for the* HEP *and other parallel processors*, Technical Report ANL-83-97, Argonne National Laboratory, Argonne, IL.

[8] W. L. MIRANKER AND W. M. LINIGER, *Parallel methods for the numerical integration of ordinary differential equations*, Math. Comput., 21 (1967), pp. 303-320.

[9] D. MITRA, *Chaotic, asynchronous relaxations for the numerical solution of differential equations by parallel processors*, preprint, AT&T Bell Laboratories, Murray Hill, NJ, 1985.

[10] J. NIEVERGELT, *Parallel methods for integrating ordinary differential equations*, Comm. ACM, 7 (1964), pp. 731-733.

[11] W. G. POOLE AND R. G. VOIGT, *Numerical algorithms for parallel and vector computers; An annotated bibliography*, Computing Rev., 15 (1974), pp. 379-388.

[12] J. ROSSER, *A Runge-Kutta for all seasons*, SIAM Rev., 9 (1967), pp. 417-452.

[13] L. F. SHAMPINE AND H. A. WATTS, *Block implicit one-step methods*, Math. Comp., 23 (1969), pp. 731-740.

[14] H. A. WATTS AND L. F. SHAMPINE, *A-stable block implicit one-step methods*, BIT, 12 (1972), pp. 252-266.

[15] P. B. WORLAND, *Parallel methods for the numerical solution of ordinary differential equations*, IEEE Trans. Comput., C-25 (1976), pp. 1045-1048.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 3, May 1987
© 1987 Society for Industrial and Applied Mathematics

SOLVING TRIDIAGONAL SYSTEMS ON ENSEMBLE ARCHITECTURES*

S. LENNART JOHNSSON†

Abstract. The concurrent solution of tridiagonal systems on linear and 2-dimensional arrays, complete binary trees, shuffle-exchange and perfect shuffle networks, and boolean cubes by elimination methods are devised and analyzed. The methods can be obtained by symmetric permutations of some rows and columns, and amounts to cyclic reduction or a combination of Gaussian elimination and cyclic reduction (GECR). The ensembles have only local storage and no global control. Synchronization is accomplished via message passing to neighboring processors.

The parallel arithmetic complexity of GECR for N equations on a K processor ensemble is $O(N/K + \log_2 K)$, and the communication complexity is $O(K)$ for the linear array, $O(\sqrt{K})$ for the 2-dimensional mesh, and $O(\log_2 K)$ for the networks of diameter $O(\log_2 K)$. The maximum speed-up for the linear array is attained at $K \approx (N/\alpha)^{1/2}$ and for the 2-d mesh at $K \approx (N/2\alpha)^{2/3}$, where $\alpha =$ (the time to communicate one floating-point number)/(the time for a floating-point arithmetic operation). For the binary tree the maximum speed-up is attained at $K = N$, and for the perfect shuffle and boolean k-cube networks, $K = N/(1+\alpha)$ yields the maximum speed-up. The minimum time complexity is of order $O(N^{1/2})$ for the linear array, of order $O(N^{1/3})$ for the mesh, and of order $O(\log_2 N)$ for the binary tree, the shuffle-exchange, the perfect shuffle and the boolean k-cube.

The relative decrease in computational complexity due to a truncation of the reduction process in a highly concurrent system is much greater than on a uniprocessor. The reduction in the arithmetic complexity is proportional to the number of steps avoided, if the number of processing elements equals the number of equations. So also is the reduction in the communication complexity for ensembles configured as binary trees, shuffle-exchange and perfect shuffle networks, and boolean cubes.

Partitioning the ensemble into subsets of processors is shown to be more efficient for the solution of multiple independent problems than pipelining the solutions over the entire ensemble. A *balanced* cyclic reduction algorithm is presented for the case where each system is spread uniformly over the processing elements, and its complexity is compared with Gaussian elimination.

Key words. odd-even cyclic reduction, Gaussian elimination, tridiagonal systems, parallel computers, distributed computing, ensemble architectures, MIMD

AMS(MOS) subject classifications. 15A06, 65F05, 65N05, 65Q25, 68Q35

1. Introduction.

The rapidly developing integrated circuit technology already allows hundreds of thousands of devices on a single chip or, equivalently, a few microprocessors with local storage. With a small amount of storage per processor, on the order of 10 16-bit processors can fit on a single chip in state-of-the-art technology [51]. It is expected that this number will increase by one to two orders of magnitude within a decade. The low cost of reproduction of an integrated circuit makes architectures consisting of a large number of identical processing elements sparsely and regularly interconnected a viable alternative to large mainframe computers. We refer to such architectures as *ensemble architectures.* A high nominal performance is attained by using a large number of processing elements built in standard technology. An excellent overview of the principles guiding VLSI architectures can be found in [52]. An assessment of the impact of VLSI on computer architecture is also given in [17].

We assume for convenience that the architectures are of the MIMD type (Multiple Instruction streams Multiple Data streams) [7], but a SIMD (Single Instruction Multiple Data streams) architecture is equally appropriate in most cases. Synchronization is obtained via message passing. There is a high degree of uniformity in the algorithms described in this paper. There are only a few different types of codes. The

* Received by the editors December 12, 1984; accepted for publication (in revised form) February 11, 1986. This work was supported by the Office of Naval Research under contract N00014-84-K-0043.

† Departments of Computer Science and Electrical Engineering, Yale University, New Haven, Connecticut 06520.

uniformity conceptually simplifies concurrent algorithms, enhances program clarity, simplifies verification of correctness, and makes program loading more efficient. Identical codes for different processors can be reproduced within the ensemble. If the encoding of what processors should receive which code can be made compactly, then the program loading may be particularly efficient [38]. With a unique program for each processor the loading time might be of a complexity that is of the same order as the arithmetic complexity of sequential algorithms. The time for program loading may be a significant fraction of the total time, if the ensemble serves as an attached processor, and only a few problems are solved for each loading of the program.

In the ensemble architectures we consider the processing elements with their storage are interconnected as linear and 2-dimensional arrays, complete binary trees, shuffle-exchange and perfect shuffle networks, and boolean cubes. These interconnection schemes have different characteristics with respect to wiring complexity, extensibility, and communication capabilities between arbitrary pairs of nodes. For a feasibility evaluation of these interconnection schemes, it is necessary to investigate the mapping of typical computations on to the ensembles. Several efficient algorithms for matrix multiplication, FFT, sorting, and many other problems are known. In this report we develop efficient distributed algorithms for the solution of tridiagonal systems.

The solution of tridiagonal systems of equations is part of several methods for the solution of partial differential equations. For instance, fast Poisson solvers employing the method of Fourier Analysis-Cyclic Reduction (FACR) [19], [20] and the Alternate Direction Implicit (ADI) method both include the solution of tridiagonal systems. If the number of processing elements is large enough to allow for one grid point per processing element, then a mapping may be made such that a processing element stores only one equation and a tridiagonal system is solved in a distinct subensemble. However, in general, multiple systems have to be solved in a set of processing elements. In some instances it may be possible to allocate the systems freely, but in other cases each system may be distributed evenly over a subensemble [24].

We first give a brief characterization of a few model ensemble architectures, then review Gaussian elimination and odd-even cyclic reduction [5] in the context of concurrent computation, and define the optimization of communication in an ensemble architecture as a graph embedding problem. In § 4 we investigate the arithmetic and communication complexity of Gaussian elimination and odd-even cyclic reduction on a parallel computer with ideal communication capabilities, but finite computational resources. We also investigate the concurrent solution of a tridiagonal system with multiple right-hand sides, and the solution of multiple independent systems. In § 5 we present algorithms for the mapping of odd-even cyclic reduction onto linear and 2-dimensional arrays, followed by a complete binary tree algorithm of arithmetic and communication complexity $O(\log_2 N)$ for a system of N irreducible equations and N processing elements. We use this tree algorithm to obtain an algorithm of comparable complexity for the shuffle-exchange network. Odd-even cyclic reduction on a perfect shuffle is discussed next. The boolean cube algorithms we present have a complexity that is of minimum order, and use properties of Gray codes in the local control of communication operations. All algorithms have entirely distributed control. Section 6 contains a summary and conclusions.

Throughout the analysis we consider the performance gain possible through truncated cyclic reduction, and the solution of multiple tridiagonal systems as well as the solution of a tridiagonal system with multiple right-hand sides.

Independently and concurrently, Gannon and Van Rosendale [9] have undertaken a similar study. In the case of finite resources, i.e., multiple equations per processor,

they perform local eliminations according to an algorithm by Sameh et al. [49], [35], which yields a reduced pentadiagonal system with two equations per processor. In the case of tridiagonal systems this difference is insignificant with respect to computational complexity if $N \gg K$. However, in the generalization of the algorithms described here to arbitrary banded systems the difference becomes significant. With the partial elimination order determined by a symmetric permutation of rows and columns, as in nested dissection and the partitioning method of Wang, not only is the arithmetic complexity lower, but the scheme also preserves symmetry and positive definiteness, should the original matrix have these properties [29]. We also treat the case where multiple independent tridiagonal systems shall be solved and describe a *balanced* cyclic reduction algorithm. We also compare the complexity of this algorithm with Gaussian elimination and derive the conditions under which one or the other is of minimum complexity.

In the following, $N = 2^n - 1$ denotes the number of equations in the tridiagonal system, K the number of processors in the ensemble, P the number of tridiagonal systems to be solved, NR the number of right-hand sides, t_a the time for local data fetching and arithmetic, t_c the time for interprocessor communication, $\alpha = t_c/t_a$, and τ the time for a communication start-up. For our complexity estimates we assume that a processor can perform one communication on two of its ports concurrently with arithmetic operations.

A few sample programs in pseudo code are contained in the Appendix. The programs serve to illustrate a programming style for multiprocessor systems in which synchronization is obtained via message passing. All sample programs are written for the case $N = K$ and $NR = 1$. The programs also illustrate the degree of program uniformity for cyclic reduction on the various ensembles.

2. Ensemble architectures. One important property of an ensemble architecture with respect to algorithm performance is the diameter of its corresponding graph. The diameter gives a lower bound for the time to perform global communication [10]. Global communication is required for the solution of irreducible systems of equations, since any component of the solution vector depends on all elements of the right-hand side, and all matrix elements. However, note that the global dependence may be sufficiently weak to allow for good local approximations, and thereby eliminate the requirement for global communication. In the case of cyclic reduction the reduction process may be truncated after a few steps if the system is sufficiently diagonally dominant. Communication is everywhere local.

The diameter of a K node linear array is $K/2$ and that of a 2-dimensional mesh with end-around connections $\sqrt{K}$. The lower bound of the communication complexity of those two ensemble configurations is of a higher order than the arithmetic complexity for the solution of tridiagonal systems on a PRAM [8] model of computation.

The diameter of a complete binary tree of $2^k - 1$ nodes is $2(k-1)$, the diameter of a 2^k node shuffle-exchange graph is $2k-1$ and that of a boolean cube of 2^k nodes is k. The lower bound for the communication complexity for these ensembles is the same as the order of the arithmetic complexity for the solution of tridiagonal systems on a PRAM model. The ensembles with a diameter of order $O(k)$ are potentially capable of solving tridiagonal systems in a time proportional to the lower bound for limited *fan-in* circuitry. In this paper we devise mappings of cyclic reduction onto these ensembles such that the system is solved in a time proportional to the lower bound.

In light of the evolving VLSI technology it is conceivable that dedicated circuitry can be used for tridiagonal solvers. For the design of hardware for tridiagonal solvers

in VLSI the area requirement of the various ensemble configurations is also important. Another important characteristic is the number of communication channels needed between chips when the ensemble is implemented on several chips, which is the case in today's technology. A complete binary tree of K nodes requires an area of $O(K)$ based on the Thompson grid model [56], [57], [4], if there is no restriction on the placement of nodes. The area is increased to $O(K \log_2 K)$ if all leaf nodes are placed on the boundary [3]. A shuffle-exchange network requires a layout area of order $O(K^2/\log_2^2 K)$ and a boolean cube an area of order $O(K^2)$ [36]. The complete binary tree not only requires the smallest area, but also allows for a partitioning such that only 4 off-chip channels are required independent of the size of the subtree that fits on a single chip and the size of the tree being assembled [37], [2]. It also has the smallest lower bound, $O(\sqrt{K}/\log_2 K)$ for the maximum length of any wire interconnecting nodes [41], [48]. For a boolean cube the maximum wire length is of order $O(K)$ [36] and the number of channels required for interchip connections is equal to the number of nodes implemented on-chip $\times$ the number of off-chip dimensions.

The binary tree has definite advantages over the shuffle-exchange and boolean cube configurations from a designer's point of view. Moreover, the shorter wires may allow the clock rate for the binary tree to be higher than for the other two configurations, giving it a possible edge also with respect to running time.

3. Elimination methods on parallel architectures.

3.1. Sparse elimination. In this paper we will make use of Gaussian elimination as well as odd-even cyclic reduction. Odd-even cyclic reduction is equivalent to Gaussian elimination performed on a permuted system of equations. The relationship between the two methods is easily seen from the graph representation of elimination [40]. In this representation a nonzero matrix element is represented by a directed edge between two vertices labeled with the indices of the matrix elements. For a symmetric matrix the edges may be considered as undirected. In the graph model the elimination of variable i corresponds to the removal of edges incident on vertex i. New edges are inserted, or the values associated with existing edges are changed, so that after the elimination of variable i there exist, for each edge ji incident upon i, edges from node j to all nodes upon which edges emanating from i terminate.

The graph corresponding to a symmetric tridiagonal matrix of order N is a path of N nodes. The graph corresponding to a tridiagonal matrix is a *perfect elimination graph* [46], and the elimination order defined by sequentially eliminating the variables from one end of the path to the other (or from both ends towards the middle) is a *perfect elimination order*, i.e., no *fill-in* is generated.

The cyclic reduction algorithm defines a partial order of elimination. This partial order is the same as the partial order produced by nested dissection [13], [12], which minimizes the order of the solution time on a PRAM model of computation [25]. Even though nested dissection is asymptotically optimal with respect to the order of the arithmetic operations [39], its arithmetic complexity is approximately twice that of Gaussian elimination for tridiagonal systems ($17N - 18\log_2 N + 2$ compared to $8N - 7$), because the partial elimination order defined by cyclic reduction (nested dissection) yields *fill-in*. The partial order of elimination is often represented as *elimination trees* [6], or *quotient trees* [11]. The elimination trees for Gaussian elimination, 2-way Gaussian elimination, and cyclic reduction are shown in Fig. 1.

In cyclic reduction 2^{n-j} variables are eliminated in step $j = \{1, 2, \cdots, n-1\}$. On a uniprocessor the elimination in each step is performed sequentially. But, in spite of this fact and the higher arithmetic complexity of odd-even cyclic reduction, it is

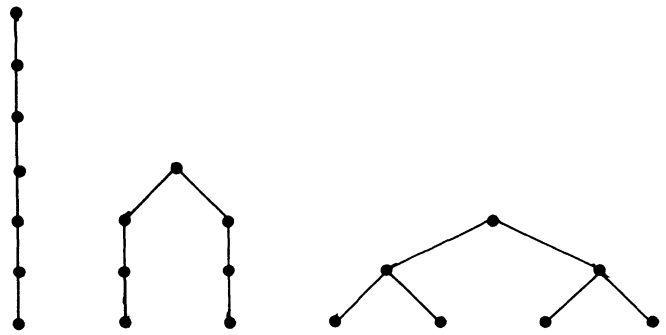


FIG 1. *Elimination trees for Gaussian elimination, 2-way Gaussian elimination, and cyclic reduction.*

preferable on most architectures with pipelined arithmetic units, such as vector architectures. The reason is that in Gaussian elimination only one (or two) variable(s) can be eliminated concurrently, and the number of operations required for each such elimination is very limited in a tridiagonal system (5 in the forward phase, 3 in the backsubstitution). Note however that for a banded system of half bandwidth $\gg 1$ the concurrency in the elimination of a single variable increases, and the number of variables that can be eliminated concurrently decreases [31].

To derive estimates of the solution time on an ensemble architecture it is necessary to incorporate the communication required for the elimination operations. We consider several cases:

- The ensemble is sufficiently large to hold one equation per processing element;
- Multiple equations per processing element;
- Multiple right-hand sides;
- Multiple independent tridiagonal systems.

3.2. Odd–even cyclic reduction. A tridiagonal system of irreducible linear equations $Ax = y$, where A is of dimension $N = 2^n - 1$, can be presented in matrix vector form as:

$$\begin{pmatrix} b_1 & c_1 & & & & x_1 & y_1 \\ a_2 & b_2 & c_2 & & & x_2 & y_2 \\ & a_3 & b_3 & c_3 & & x_3 & y_3 \\ & & \ddots & \ddots & \ddots & = & \\ & & & a_N & b_N & x_N & y_N \end{pmatrix}.$$

Odd-even cyclic reduction proceeds in a reduction phase succeeded by a backsubstitution phase. Using subscripts for equation numbers and superscripts to denote reduction and backsubstitution steps, cyclic reduction is defined by the following set of equations:

Reduction:

$$\begin{aligned} a_i^j &= e_i a_{i-2^{j-1}}^{j-1}, \\ c_i^j &= f_i c_{i+2^{j-2}}^{j-1}, \\ b_i^j &= b_i^{j-1} + e_i c_{i-2^{j-1}}^{j-1} + f_i a_{i+2^{j-1}}^{j-1}, \\ y_i^j &= y_i^{j-1} + e_i y_{i-2^{j-1}}^{j-1} + f_i y_{i+2^{j-1}}^{j-1}, \\ e_i &= -a_i^{j-1} / b_{i-2^{j-1}}^{j-1}, \\ f_i &= -c_i^{j-1} / b_{i+2^{j-1}}^{j-1} \end{aligned}$$

where $i = 2^j, 2\times 2^j, 3\times 2^j, \cdots, 2^n - 2^j$, for reduction steps $j = 1, 2, \cdots, n-1$.

The initial conditions are $a_i^0 = a_i$, $b_i^0 = b_i$, $c_i^0 = c_i$, and $y_i^0 = y_i$.

After $n-1$ reduction steps only one equation of the following form remains:

$$a_{2^{n-1}}^{n-1} x_0 + b_{2^{n-1}}^{n-1} x_{2^{n-1}} + c_{2^{n-1}}^{n-1} x_{2^n} = y_{2^{n-1}}^{n-1}.$$

A correct solution for $x_{2^{n-1}}$ is obtained with $x_0 = x_{N+1} = 0$. Remaining variables are obtained through backsubstitution.

Backsubstitution:

$$x_{2^{n-1}} = y_{2^{n-1}}^{n-1} / b_{2^{n-1}}^{n-1},$$

$$x_i = (y_i^{j-1} - a_i^{j-1} x_{i-2^{j-1}} - b_i^{j-1} x_{i+2^{j-1}}) / b_i^{j-1}$$

where $i = \{2j-1, 3\times 2^{j-1}, 5\times 2^{j-1}, \cdots, 2^n - 2^{j-1}\}$, and $j = \{n-1, n-2, \cdots, 1\}$.

In the above algorithm, 12 arithmetic operations are needed per equation in the reduction computation, and 5 per unknown in the backsubstitution. A careful count gives a total of $17N - 18n + 2$ arithmetic operations, disregarding index computations.

If the matrix A is strongly diagonally dominant, then a_i^j and c_i^j tend to zero with j. The stronger the diagonal dominance the more rapid is the convergence. If for $j = m$, $|a_i^j| \ll \varepsilon$ and $|c_i^j| \ll \varepsilon$, where ε is an acceptable error bound, then the reduced tridiagonal system at step m can be treated numerically as a diagonal system. No further reduction computations are necessary. Instead, a diagonal system of $2^{n-m} - 1$ equations is solved, and the backsubstitution process started. This truncated cyclic reduction method requires $2m$ steps. The reduction in the total number of operations is $16(2^{n-m} - 1) - 18(n-m) + 2$. Solving partial differential equations by difference approximation yields matrices for which acceptable precision renders values of m in the order of $10-20$, independent of the size of the matrix [18].

The reduction phase terminates when one equation with one unknown remains. If several reduction steps terminating in different equations are carried out concurrently, then the backsubstitution phase becomes unnecessary. Hockney describes such a cyclic reduction method and refers to it as Parallel Cyclic Reduction [18]. The tridiagonal system is extended with the equations for $i < 1$ and $i > N$ such that $a_i = c_i = 0$ and $b_i = 1$ for $i > 1$ and $i > N$. The parallel cyclic reduction algorithm is of arithmetic complexity $O(N \log_2 N)$, but needs only half the number of sequential steps ($\log_2 N$ instead of $2 \log_2 N$).

3.3. A computation graph for cyclic reduction. In studying the mapping of computations on to a network of processing elements we represent the variable dependencies and the operations performed upon the variables by a directed graph. Nodes correspond to operations consuming and producing data, and directed edges to communication of data. Edges are directed from the source towards the sink. Similarly, we describe the processing ensemble as a graph with nodes corresponding to processing elements with local storage, and edges to interconnections. The mapping problem becomes a problem of embedding one graph in another. In general, the embedding problem should be viewed as a dynamic problem due to the dynamic character of the computation graph.

We represent the coefficients of one equation, the corresponding right-hand side(s), and the unknown(s) corresponding to the column of the diagonal element with one node in the computation graph. Hence, for one right-hand side 5 storage cells are required in each node. The nodes in the computation graph in Fig. 2 are labeled with the equation number, starting from 0. The superscript denotes the reduction step during which the equation is modified.

In the interest of conserving storage, nodes in the same column of the computation graph in Fig. 2 can be identified. The storage requirement is approximately half of that required with the nodes stored in distinct storage cells. The storage conserving scheme is one source of poor performance of many implementations of cyclic reduction on architectures with primary storage partitioned into banks [34]. The degradation is easily seen from Fig. 3, in which nodes in the same column are identified.

With 2^k storage banks and cyclic storage, as performed by most FORTRAN compilers, all equations j such that $i = j \bmod 2^k$ are assigned to bank i. After k reduction steps all equations are in the same bank. The performance may be degraded further towards the end of the reduction phase if the storage is pipelined. In such a case, and with only few equations participating in each reduction step, Gaussian elimination may be preferable to cyclic reduction with respect to storage bandwidth. (It may also be preferable with respect to the time for arithmetic on a pipelined architecture for few equations.) With an odd number of storage banks no bank conflicts occur in cyclic reduction.

4. Finiteness. If the number of equations is greater than the number of processing elements, then several equations have to be identified with the same element (assuming it has sufficient storage). It is desirable to perform the identification of equations such that the amount of arithmetic is distributed as evenly as possible throughout the computations, without excessive communications. There are two commonly used schemes for identification: *consecutive* and *cyclic*. In the *consecutive* scheme all equations $i = \{0, 1, \cdots, N-1\}$ that satisfy the relation $p = \lfloor iK/N \rfloor$ are identified with partition p, $p = \{0, 1, \cdots, K-1\}$. The number of partitions is K. In *cyclic* storage all equations i that satisfy the relations $p = i \bmod K$ are identified with partition p. The consecutive scheme for identification is also often referred to as *domain decomposition*

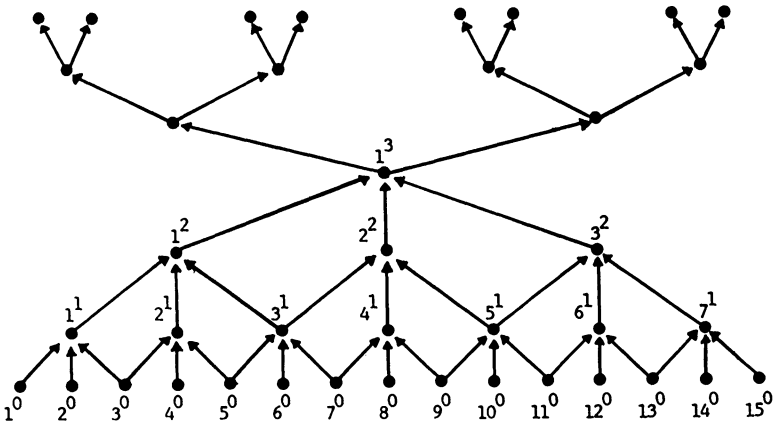


FIG. 2. *A computation graph for odd-even cyclic reduction.*

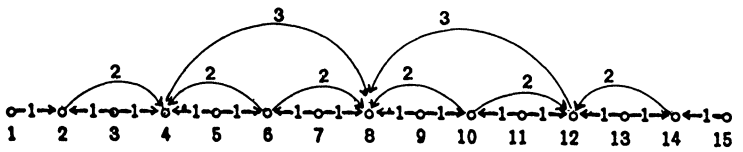


FIG. 3. *A storage conserving computation graph for cyclic reduction.*

or *substructuring*. The cyclic storage scheme yields a higher processor utilization for LU-decomposition on dense matrices [21], [30], [28].

Both the consecutive and the cyclic schemes for identification result in quotient graphs with either $\lfloor N/K \rfloor$ or $\lceil N/K \rceil$ equations per node. Though this is the best possible balance in the distribution of equations, it does not necessarily translate to an even distribution of arithmetic (as already mentioned in the case of banked storage architectures).

We now show that the consecutive scheme for identification is superior for the solution of tridiagonal systems of equations. The analysis will be carried out first for cyclic reduction, then for a combination of Gaussian elimination and cyclic reduction, where the Gaussian elimination is performed within partitions, as in the substructured Gaussian elimination in [58]. Then, we show the relationship between the consecutive scheme, and the algorithms by Sameh et al., and Wang, and (incomplete) nested dissection.

The set of partitions is denoted $P = \{P_0, P_1, \cdots, P_{K-1}\}$. Let the subset of P that is assigned $\lfloor N/K \rfloor$ equations be $Q = \{Q_i\}$, and the subset assigned $\lceil N/K \rceil$ equations be $R = \{R_i\}$, where $|R| = N \bmod K$. For an ensemble of linearly interconnected processors there is no topologically natural choice of K. For a 2-dimensional mesh, K is naturally chosen as the product of two integers, often as a square for reasons of symmetry. For complete binary trees we take $K = 2^k - 1$, implying that we assign one partition to each node of the tree, not only to the leaf nodes. For shuffle networks and boolean cubes, $K = 2^k$.

4.1. Domain decomposition.

4.1.1. Communication aspects of cyclic reduction. For the mapping of partitions onto nodes in a complete binary tree, the following observation is useful. For $K = 2^k - 1$ and $N = 2^n - 1$, the number of partitions with $\lceil N/K \rceil$ equations is equal to the number of nodes in a tree with $n \bmod k$ levels, i.e., $|R| = 2^{n \bmod k} - 1$. If $\lceil N/K \rceil$ equations are assigned to the top $n \bmod k$ levels of a k-level complete binary tree labeled in order [1], then the first $n - k$ reduction steps result in a reduced system that has one equation per tree node. Moreover, the difference in the number of equations subject to elimination operations in any pair of partitions is at most one, throughout the entire reduction phase [26].

For $K = 2^k$, $|R| = K - 1$, i.e., one partition performs elimination operations on one equation less than the other partitions during the first $n - k$ reduction steps.

With respect to interpartition communication needs we first establish the following.

LEMMA 4.1. *In the first $n - k$ reduction steps, partition P_i only communicates with partition P_{i+1} for $i = \{0, 1, 2, \cdots, K-2\}$ and partition P_{i-1} for $i = \{1, 2, \cdots, K-1\}$.*

Proof. The number of indices assigned to a partition is at least $\lfloor N/K \rfloor$. The index difference between equations used in the elimination operations is increasing monotonically during the reduction phase, and is 2^{n-k-1} for step $n - k$. But, the equation index set in each partition is consecutively ordered and $2^{n-k-1} \leqq \lfloor N/K \rfloor$. □

By Lemma 4.1 the interpartition communication during the first $n - k$ reduction steps is entirely between adjacent partitions. However, it is also necessary to establish the direction of communication, particularly if the direction of communication changes. If the direction of communication is changing from one reduction step to the next, pipelining of the communications for successive reduction steps cannot be used to "hide" the latency associated with long routing distances, if adjacent partitions are mapped to processing elements far apart. A proximity preserving path embedding is not necessarily optimal with respect to communication time. For the complete binary

tree a proximity preserving path embedding yields a communication complexity of order $O(\log_2^2 K)$ [26].

THEOREM 4.1. *In the set of partitions* $P_s = 2^{k-s-1}\{1, 3, 5, \cdots, (2^{s+1}-1)\}$, $s = \{0, 1, \cdots, k-1\}$, $|P| = 2^k - 1$ *there exists at least one partition which changes direction of communication with one of its adjacent partitions* $s+1$ *times for every* k *reduction steps.*

We omit the proof of Theorem 4.1 and refer to [26]. The theorem states that among the odd partitions there exists at least one partition that changes the direction of communication with one of its neighboring partitions k times for every k reduction steps. Figure 4 illustrates the interpartition communication.

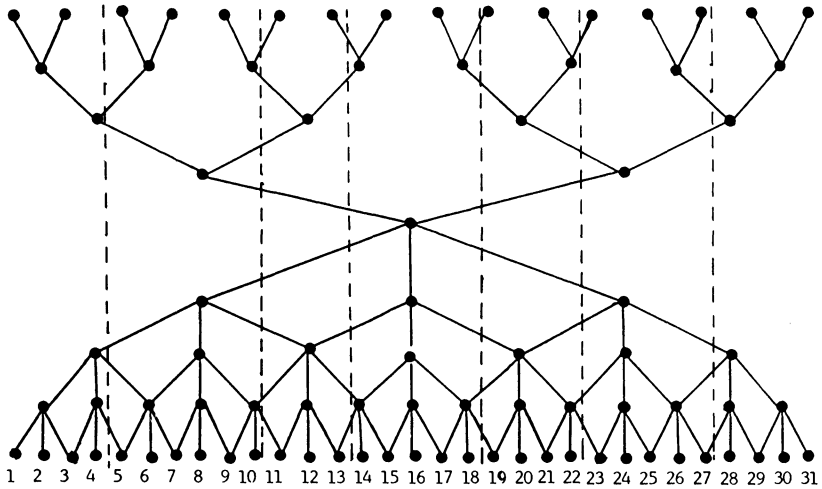


FIG. 4. *Partitioning of the computation graph of cyclic reduction.*

COROLLARY 4.1. *The communication for successive reduction steps can only be partially pipelined.*

COROLLARY 4.2. *If each partition is assigned* 2^{n-k+1} *or* 2^{n-k} *consecutive indices,* $|P| = 2^k - 1$, *then the direction of communication is constant for the first* $n-k+1$ *or* $n-k$ *reduction steps.*

The benefit of modifying the number of equations N to the form $2^m(2^k-1)$ is that the number of indices per partition is even for the first several reduction steps and it is possible to pipeline the communications. This advantage is obtained at the expense of a possible increase in the arithmetic complexity of at most a factor of 2 for the first $n-k+1$ steps. The arithmetic complexity is the same for subsequent steps.

THEOREM 4.2. *For* $K = 2^k$ *the direction of exchange is constant during the first* $n-k$ *steps* [26].

The number of arithmetic operations performed in sequence is approximately $(8+9NR)N/K + (5+6NR)\log_2 K$, where NR is the number of right-hand sides. For this estimate it is assumed that the computations required for the elimination operations on the last K rows are shared between two processing elements: the one holding the row used for the elimination, and the one holding the row subject to elimination. It is also assumed that the reduction and backsubstitution are performed identically on all right-hand sides. We will later present a scheme for improved load balance during the final k reduction steps. The backsubstitution phase can be implemented with communications as in a complete binary tree, in which case most communications include two floating-point numbers, or in a way corresponding to the communication

pattern in the reduction phase. In the latter case each communication only includes one floating-point number per right-hand side. For *NR*-1 an arithmetic complexity of $9 \log_2 K$ for the reduced tridiagonal system can be achieved at the expense of $2(\log_2 K - 1)$ additional communications.

4.1.2. Gaussian elimination locally, cyclic reduction globally. In cyclic reduction the partition assigned equation $2^{n-1} - 1$ requires a total of $n-1$ communications. For the first $n-k$ reduction steps the communication is with adjacent partitions, but during the last k steps with partitions differing in index by $2^j, j = \{0, 1, \cdots, 2^{k-2}\}$. The lower bound for the communication time is $O(\log_2 K)$ for ensembles with nodes of degree three.

If instead of the partial elimination order defined by cyclic reduction, the elimination order within partitions is taken to be the order of the equation indices (increasing or decreasing order), then the communication requirements are reduced to one communication for the first $n-k$ reduction steps. The reduced system after $n-k$ local eliminations can still be made a tridiagonal system. The fact that local forward and backward elimination and one communication with adjacent partitions yield a reduced tridiagonal system was observed by Wang [58]. The method of Sameh et al. [49], [35] uses no communication in the first $n-k$ reduction steps, but results in a pentadiagonal system. Similar schemes have also been described in [45] and [9]. The advantage of the local elimination order with one communication, as described below, is that it preserves not only diagonal dominance but also symmetry and positive definiteness, also when applied to arbitrary banded matrices [29]. We refer to the algorithm with reduced communication complexity as GECR for Gaussian elimination–cyclic reduction. It requires $\log_2 K$ communications for the reduction phase. Algorithm GECR proceeds in three phases:

- Local forward and backward elimination. The backward elimination requires one communication with an adjacent partition.
- Solve a reduced system of equations with one equation per partition by cyclic reduction.
- Local backsubstitution.

In phase 1 *fill-in* occurs in columns $i \times N/K - 1$ of partition i, $i = \{1, 2, \cdots, K-1\}$, and columns $(i+1)N/K - 1$ of partition i, $i = \{0, 1, \cdots, K-1\}$. The last equation of each partition in this system of equations forms a tridiagonal system in the variables corresponding to the columns $(i+1)N/K - 1$, $i = \{0, 1, 2, \cdots, K-1\}$. The appearance of the matrix after phase 1 is shown in Fig. 5.

The arithmetic complexity of GECR is almost identical to that of cyclic reduction, because of the fill-in in the forward and backward eliminations. Phase 1 of GECR requires 4 arithmetic operations for each elimination operation on the matrix, except in the first and last partitions, which yield a total of 8 arithmetic operations for the forward and back eliminations. For the first and last partitions the forward and backward eliminations require a total of 5 arithmetic operations per equation. There are 2 operations on each right-hand side in each of the forward and backward eliminations, i.e., a total of 4 arithmetic operations per right-hand side in phase 1. There are 5 arithmetic operations per right-hand side in phase 3. The communication complexity of phase 1 is 1. The number of communication actions for phase 2 is $2(\log_2 K - 1)$. With a symmetric allocation of matrix elements as described in § 4.3, the number of arithmetic and communication operations may change slightly.

Note that if a processing element has pipelined arithmetic units and communication is fast, then the fact that cyclic reduction "vectorizes" may make it preferable to GECR

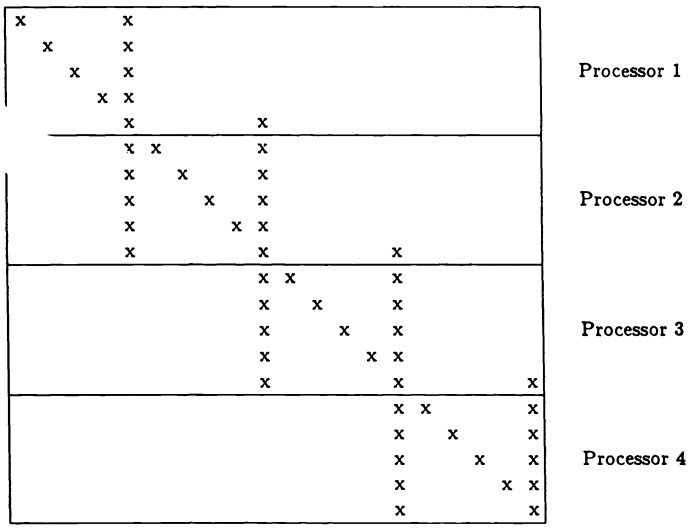


FIG. 5. *The matrix after phase* 1.

with respect to computational complexity. However, if there are many right-hand sides, or if multiple independent problems shall be solved, then "vectorization" across right-hand sides, or across different problems, can be used in conjunction with GECR.

The fact that algorithm GECR preserves diagonal dominance in phase 1 can be exploited for truncation of the reduction process if the diagonal dominance is sufficiently strong [15], [16], [18], [29], [45].

4.2. Cyclic storage. Under the consecutive scheme for identifying equations with partitions, the arithmetic complexity is of order $O(N/K + \log_2 K)$ and the communication complexity for GECR is of order $O(\log_2 K)$. The order of either of these two complexity measures cannot be reduced.

With the cyclic scheme for identifying equations, K must be odd in order to keep the balance of equations subject to elimination operations as even as possible throughout the computations. However, cyclic partitioning is inferior to the consecutive partitioning scheme with respect to communications. This is true for cyclic reduction, and GECR offers no advantage since successive equations are in adjacent partitions. There is no locality in data allocation that can be used advantageously with respect to communication. For a dense system of equations there is no data locality of elimination operations, and the cyclic storage scheme can be used to balance the load [21], [30]. In the solution of tridiagonal systems all elimination operations are local. The cyclic scheme is inherently inferior to the consecutive scheme with respect to computational complexity for tridiagonal (and banded) systems.

4.3. Divide-and-conquer. In this section we relate to each other the methods of incomplete nested dissection [13], Wang [58], and Sameh et al. [49], [35]. Nested dissection on the graph of a tridiagonal matrix is a recursive bisection. The separators are single nodes. The bisection is terminated when there are $K = 2^k$ partitions for K processes. Labeling the bisectors last and in reverse order, and the remaining nodes in the same order as in the initial labeling yields an elimination tree as in Fig. 6. This relabeling corresponds to row and column permutations to yield a system of equations as in Fig. 7. The method by Wang can also be described in terms of a set of separators, and permutation of rows and columns. The elimination tree is labeled K-section in Fig. 6, and the matrix is shown in Fig. 8.

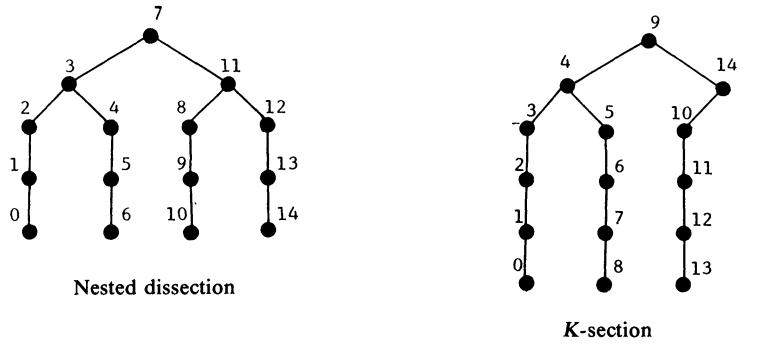


FIG. 6. *Partitioning of tridiagonal systems of equations.*

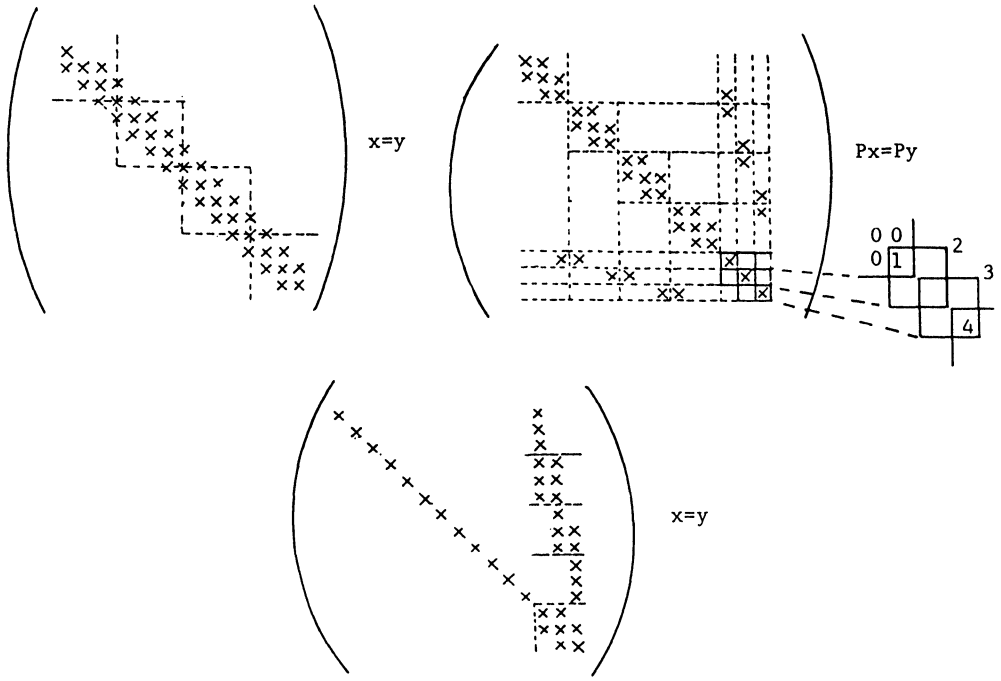


FIG. 7. *Matrix partitioning and permutation for nested dissection.*

Figures 7 and 8 also indicate the association of matrix elements with partitions. If the association of matrix elements with partitions is made the same in the method by Wang as in incomplete nested dissection, then the elimination operations in the first phase are completely local, but an "assembly" of the reduced tridiagonal system requires communication. The matrices to be assembled are 2×2 matrices, except for the first which is a 1×1 matrix. In case of nested dissection the last matrix is also of size 1×1. The matrices to be assembled are shown as squares in Figs. 7 and 8.

If an even number of partitions is desired, as for boolean cube configured ensembles, then incomplete nested dissection is a convenient partitioning strategy. If the number of partitions is of the form 2^k-1, as in complete binary trees, then the K-section strategy is preferable. With the symmetric association of matrix elements

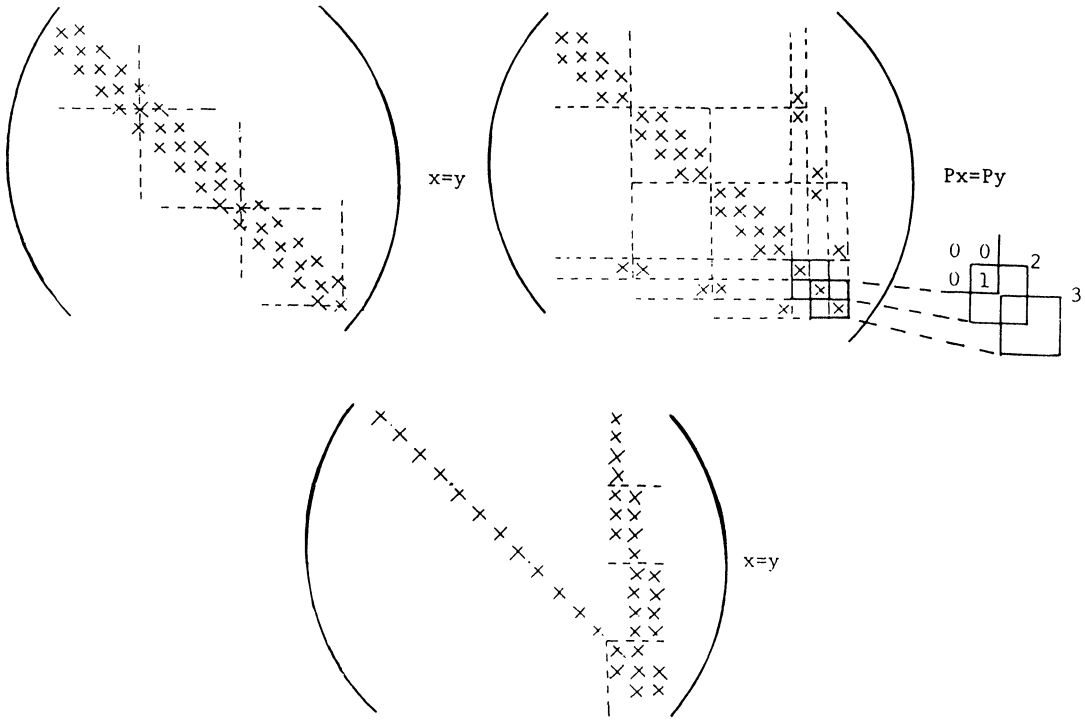


FIG. 8. *Matrix partitioning and permutation for K-section.*

and partitions, the elimination operations on the reduced system can take place as the block matrices are assembled, as is common practice in sparse matrix codes, see e.g. [22], [6]. In the case of nested dissection, elimination is performed on pairs of matrices in the form of a complete binary tree. In the case of K-section one leaf node is missing. The indicated elimination order corresponds to a particular permutation of the separators. After the elimination phase follows a backsubstitution phase that proceeds from the root towards the leaf nodes. Embeddings of the elimination trees in various ensemble architectures will be discussed in § 5.

The algorithm by Sameh et al. cannot be viewed as a symmetric permutation of rows and columns ($PAPP^T x = Py$), a local solve, global solution of a reduced system, and a local backsubstitution. The reduced system is pentadiagonal and of order $2(K-1)$. It can be arranged into a tridiagonal system by pairwise column permutations, but the potential symmetry and diagonal dominance is lost.

4.4. Multiple tridiagonal systems. If there is complete freedom in allocating equations and right-hand sides, then the computations could be localized by solving the system for each right-hand side (or each independent system) in a subensemble of $\lceil K/NR \rceil$ processing elements.

We assume that the system of equations with the multiple right-hand sides is distributed with one equation per partition for all right-hand sides. For multiple independent tridiagonal systems we also assume one equation per partition. Evenly distributed tridiagonal systems occur naturally in the solution of Poisson's problem on ensemble architectures [30] by so-called fast solvers, and in the Alternating Direction Implicit (ADI) method [24]. If the number of equations is greater than the number of partitions, then we assume the consecutive scheme for identifying equations with

partitions, and local forward and backward elimination. We first compare the complexities of solving a system with NR right-hand sides and one equation per partition by either Gaussian elimination or cyclic reduction, then consider the concurrent solution of P independent tridiagonal systems.

The analysis in this section uses an idealized model of computation in which all interpartition communication requires one start-up. The results for Gaussian elimination apply to all ensembles considered in the next section. The arithmetic complexity derived for cyclic reduction also applies for the ensembles studied next, but the communication complexity is higher, in general.

4.4.1. Multiple right-hand sides. For Gaussian elimination the operations on different right-hand sides can be pipelined to keep the processor utilization as high as possible. The parallel arithmetic complexity of 2-way Gaussian elimination for $N = K$ is $(3+5NR)(K-3)/2+5+6NR$ without pipelining of the operations for different right-hand sides. The number of elements communicated in sequence is $(1+NR)\times(K-2)$ and the number of communication actions (start-ups) is $K-2$. In the case of maximum pipelining the arithmetic complexity is $3K+5NR-6$, the number of element transfers in sequence is $K+2NR-1$, and the number of communication actions is also $K+2NR-1$. Optimizing packet sizes, i.e., the amount of data per communication, such that the total time is minimized, yields

$$2\sqrt{(K/2-2)\tau}\left(\sqrt{(NR+2)(t_c+2t_a)}+\sqrt{(NR(t_c+3t_a)}\right) + (5NR-(K/2-1))t_a+2NRt_c+(K-4)\tau. \tag{4.1}$$

The optimum packet size in the forward elimination is

$$\sqrt{\frac{(NR+2)\tau}{(K/2-2)(t_c+2t_a)}}$$

and in the backward elimination

$$\sqrt{\frac{NR\tau}{(K/2-2)(t_c+3t_a)}}.$$

A naive implementation of cyclic reduction in which the computations for all right-hand sides are treated identically yields an arithmetic complexity of $(5+6NR)\times(\log_2 K-2)+5+6NR$ for $N=K$, if the operations required for the elimination of the two off-diagonal elements in a row are distributed over two processing elements. Two communication actions per reduction step are required. The communication complexity is $(3+3NR)(\log_2 K-2)+2(1+NR)$ and the number of start-ups is $3(\log_2 K-1)$, assuming one start-up per interpartition communication. If all operations for the elimination of the off-diagonal elements in a row are performed in one processing element, then the arithmetic complexity is $(8+9NR)(\log_2 K-2)+5+6NR$ and the number of start-ups is reduced to $2(\log_2 K-2)$.

In the naive implementation of cyclic reduction for multiple right-hand sides the processor holding equation $2^{k-1}-1$ is a bottle-neck. It participates in $\log_2 K$ reduction steps for each right-hand side. Balancing the computations such that for $NR<K$ right-hand sides the reduction phase in cyclic reduction converges to distinct processors for different right-hand sides eliminates this problem. The balancing is accomplished by adding "virtual" equations at either end of the system of equations. These virtual equations do not change the solution to the real system, and do not incur any arithmetic or communication operations. The principle in this *balanced* cyclic reduction algorithm is the same as in Hockney's parallel cyclic reduction [18], but we perform concurrent reduction processes on *different* right-hand sides.

To estimate the arithmetic and communication complexities for the balanced cyclic reduction algorithm, one can proceed from the computation graph in Fig. 2. $NR - \lceil NR/K \rceil$ computation graphs of height $\log_2 K$ and $\lceil NR/K \rceil$ of height $\log_2 K - 1$ are superimposed, shifted one step with respect to each other in the horizontal direction. It is easily seen that each processor performs a number of arithmetic operations identical to that of one complete cyclic reduction computation. Hence, the arithmetic complexity is

$$17\left(\log_2 K - 1 + \sum_{i=1}^{\log_2 K - 2} \left\lfloor \frac{NR \bmod K - 1}{2^i} \right\rfloor\right)$$

approximately if $NR \bmod K \leqq (K+1)/2$, else

$$17\left(\frac{K-1}{2} + \sum_{i=1}^{\log_2 K - 2} \left\lfloor \frac{NR \bmod K - (K+1)/2}{2^i} \right\rfloor\right).$$

The speed-up is linear. It is also clear from superimposing the computation graphs that the number of communication actions (start-ups) occurring sequentially is $2 \log_2 K$. However, the amount of data communicated increases, and is approximately $(5K - \log_2 K)\lceil NR/K \rceil$. The complexity of the balanced cyclic reduction algorithm is approximately

$$(4.2) \qquad ((17K - 18 \log_2 K + 2)t_a + (5K - \log_2 K)t_c)\lceil NR/K \rceil + 2\tau \log_2 K.$$

Note that in the balanced cyclic reduction algorithm the operations on the system matrix are repeated for each right-hand side. The parallel arithmetic complexity of cyclic reduction is always lower than that of Gaussian elimination for $NR = 1$, $K > 3$, but for $NR > 1$ pipelined Gaussian elimination may be of a lower complexity. The value of NR at which the cross-over occurs depends on the communication and arithmetic bandwidths, and start-up times. The cross-over point in the idealized architecture is determined by (4.1) and (4.2).

In general, interpartition communication requires more than one communication, as will be shown in subsequent sections. From the complexity expressions for the idealized architecture it is clear that if the start-up time for communication is high, then cyclic reduction is always preferable. On the other hand if the start-up time can be ignored compared to the times for arithmetic or the data transfer time, then Gaussian elimination has an advantage for NR sufficiently large. Both Gaussian elimination and the balanced cyclic reduction algorithm compute the solution *in-place*. Another alternative is to perform a transpose of the data, solve the system locally and transpose back. We will analyze the total complexity of this scheme for the boolean k-cube and compare it with the *in-place* algorithms.

4.4.2. Multiple independent tridiagonal systems. We first establish the following:

THEOREM 4.3. *If P tridiagonal systems shall be solved on a K processor ensemble, $P \leqq K$, and there is no restriction on data allocation, then partitioning of the ensemble into P subensembles yields a lower complexity than sharing the entire ensemble for all P problems.*

Proof. Assume that the solution processes for the P problems on the entire K processor ensemble are pipelined. Let the time be $\alpha(K) + \beta P$. The first term corresponds to the propagation time, and the second to the time for arithmetic in one processor. Then, if the ensemble is partitioned into P subensembles, the time for arithmetic remains constant, since each subensemble must perform P times the work, but for only one problem instead of P problems. The propagation term decreases to $\alpha(K/P)$. □

If each tridiagonal system is distributed evenly over the ensemble, then the same techniques as were discussed for multiple right-hand sides can be applied. For ensembles sufficiently small compared to the number of problems, Gaussian elimination yields a lower complexity than cyclic reduction, if start-ups can be ignored. For sufficiently large P the cross-over point ignoring start-ups is at approximately

$$P = \frac{K(4ta + tc)}{(9ta + 3tc)}.$$

For a 2-dimensional $N \times N$ grid problem P is of the form N/K. Cyclic reduction yields a lower complexity than Gaussian elimination for $N < 85$–128 if $K = 16$, and for $N < 1365$–2048 for $K = 64$ according to the approximate formula, again ignoring start-ups. If start-ups are the determining factor, then cyclic reduction is preferable. An alternative to solving each problem distributed across the ensemble is to perform local forward and backward elimination, then rearrange the data for the global phase such that for $P \geqq K$ problems each tridiagonal system is solved locally and then the solution is distributed to the original data order. We analyze this scheme for the boolean cube.

5. Tridiagonal systems solvers on ensemble architectures. From the analysis above it follows that we only need to consider consecutive partitioning (domain decomposition, substructuring). The computations proceed in three phases of which phase 1 and 3 are local, and phase 2 requires global communication. In the following we focus on phase 2, and assume that there is one equation per processing element. Depending upon the number of systems to be solved, ensemble topology, the arithmetic and communication bandwidths, and start-up times, cyclic reduction or Gaussian elimination may be preferable with respect to the time complexity for phase 2. The mapping of cyclic reduction onto linear arrays, 2-dimensional meshes, complete binary trees, shuffle-exchange and perfect shuffle networks, and boolean cubes is the subject of this section. The arithmetic complexity has been given above, and it suffices to consider the communication complexity for the various mappings.

5.1. Linear arrays. We first assume that partitions are mapped statically to processing elements by identifying partition indices with processor indices. We refer to the algorithm based on the static mapping as an *in-place* algorithm. The communication in reduction step j is with processors at distance $2^j, j = \{0, 1, \cdots, k-2\}$. Instead of a static map an *unshuffle* operation can be performed between each reduction step. The unshuffle operation moves equations subject to further elimination operations into a subarray, preserving the relative order. We refer to the second algorithm as the *shuffle* algorithm.

It is easily verified that the *in-place* algorithm has $K-2$ start-ups, the same as 2-way Gaussian elimination. The *shuffle* algorithm has $2(K-3)$ start-ups. Unshuffle operations on an array of 8 processing elements are shown in Fig. 9. The number of elements communicated are the same in the *in-place* and *shuffle* algorithms, and it follows that the *in-place* algorithm is superior. The *shuffle* algorithm is useful for 2-dimensional arrays, however.

THEOREM 5.1. *An in-place algorithm for cyclic reduction on a linear array, with equation indices identified with indices of consecutively labeled processors, has a lower communication complexity than algorithms using shuffle operations.*

For truncated cyclic reduction the *in-place* algorithm is even more advantageous than the *shuffle* based algorithm. For an *in-place* algorithm the communication cost

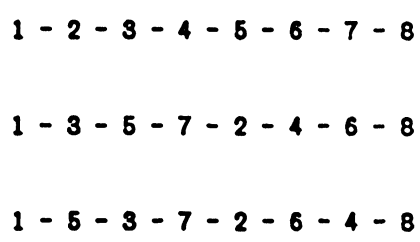


FIG. 9. *Implementing shuffles of decreasing sizes on a linear array.*

doubles for each reduction step, whereas the highest communication cost (half of the total) is incurred in the first step of a *shuffle* based algorithm.

The total complexity for GECR on a linear array is of the form $\alpha N/K+\beta \log_2 K+\gamma K$. It follows that the complexity attains a minimum for K of order $O(\sqrt{N})$. If instead of cyclic reduction Gaussian elimination is used for phase 2, then the total complexity is of the form $\alpha N/K+\gamma K$, and the order of the optimum size array is the same as for GECR.

In comparing the complexity of Gaussian elimination and cyclic reduction for one right-hand side we derive a complexity estimate for 2-way Gaussian elimination of $(4K+2)t_a+(2K-4)t_c+(K-2)\tau$, and an estimate of $(17(\log_2 K-2)+14)t_a+(9K/4-5)t_c+(K-2)\tau$ for cyclic reduction.

THEOREM 5.2. *Cyclic reduction on a linear array always yields a lower complexity than Gaussian elimination for the solution of one tridiagonal system if* $\alpha=t_c/t_a\leqq 1$. *Conversely, Gaussian elimination is of a lower communication complexity if* $\alpha=t_c/t_a\geqq 16$.

The theorem follows directly from the complexity estimates. For $\alpha=4$ cyclic reduction is of a lower complexity for $k\geqq 4$, and for $\alpha\geqq 8$ it is of lower complexity for $k\geqq 5$, assuming $K=2^k-1$.

For multiple right-hand sides or multiple independent problems a similar analysis can be carried out. The complexity of Gaussian elimination for multiple independent problems is approximately $(K+2P-4)(4t_a+2t_c+\tau)$, if the communications are pipelined such that one equation is sent per communication. Without pipelining the complexity is $((4t_a+2t_c)P+\tau)(K-2)$, and with optimized packet sizes for the forward and backward substitution, the complexity becomes approximately

$$\sqrt{2(K-4)P\tau}(\sqrt{3t_c+5t_a}+\sqrt{t_c+3t_a})+8Pt_a+4Pt_c+(K-4)\tau. \tag{5.1}$$

The optimum packet size for the forward phase is

$$\sqrt{\frac{3P\tau}{(K/2-2)(t_c+5/3t_a)}}$$

and for the backward phase

$$\sqrt{\frac{P\tau}{(K/2-2)(t_c+3t_a)}}.$$

The time for the balanced cyclic reduction algorithm is approximately

$$((17K-18\log_2 K+2)t_a+(5K-\log_2 K)t_c)\lceil P/K\rceil+(K-2)\tau. \tag{5.2}$$

The last terms of the complexity estimates (5.1) and (5.2) are approximately equal. Basing the comparison between the two methods on the remaining highest order terms of the two expressions yields a cross-over point of

$$P\approx 0.2\frac{1+\alpha/2+\sqrt{15(1+3\alpha/5)(1+\alpha/3)/16}}{(1+\alpha/9)^2}(K-4)\tau/t_a \quad (\alpha=t_c/t_a).$$

5.2. Two-dimensional meshes. We assume for convenience that the mesh has $2^{k/2}$ processing elements in each dimension. We estimate the communication complexity of *in-place* and *shuffle* algorithms for two "serpentine" embeddings of the partitions.

With a "serpentine" embedding proceeding from the upper left-hand corner to the lower right-hand corner of the mesh, an *in-place* algorithm requires communication between processors at distance 2^{j-1} in reduction step $j, 1 \leqq j \leqq k/2$. After $k/2$ steps there is one equation per row that is subject to further elimination operations. The equations are at opposite ends of successive rows, and reduction step $k/2+1$ requires communication between processors at a distance of $2^{k/2}$. After yet another reduction step, processors with equations to be part of subsequent reduction operations are located in every other row of the first column. The linear array algorithm is applied at this point. The communication for reduction steps $j, k/2+2 \leqq j \leqq k-1$ are between processors at distance $2^{j-k/2-1}$. The total number of start-ups for the reduction and backsubstitution is $6(\sqrt{K}-1)$. With the "serpentine" starting and ending close to the center, as in Fig. 10, the number of start-ups is reduced to $2(2\sqrt{K}-3)$, a reduction in start-ups by a factor of $\frac{1}{3}$.

The number of start-ups can be reduced further by using a *shuffle* algorithm for the first $k/2$ reduction steps (and last $k/2$ backsubstitution steps) and an *in-place* algorithm for the last $k/2-1$ steps. The resulting number of start-ups for the "serpentine" embedding proceeding from the upper to the lower left hand corners is $3\sqrt{K}+\log_2 K-4$. After the first reduction computation an exchange operation is performed between distinct pairs of adjacent processors in every other row. This operation brings equations that are involved in the second reduction step into the same set of columns. An unshuffle operation on columns moves the columns with even equations into one half of the array. After $k/2$ communication, reduction, and exchange operations, and $k/2-1$ unshuffle operations of successively decreasing sizes, the equations that are taking part in the last $k/2-1$ reduction steps are within one column. Figure 11 shows some reduction steps.

52	51	50	49	48	47	46	45
53	54	55	56	41	42	43	44
60	59	58	57	40	39	38	37
61	62	63	64	33	34	35	36
4	3	2	1	32	31	30	29
5	6	7	8	25	26	27	28
12	11	10	9	24	23	22	21
13	14	15	16	17	18	19	20

FIG. 10. *An embedding with* $2(2\sqrt{K}-3)$ *start-ups.*

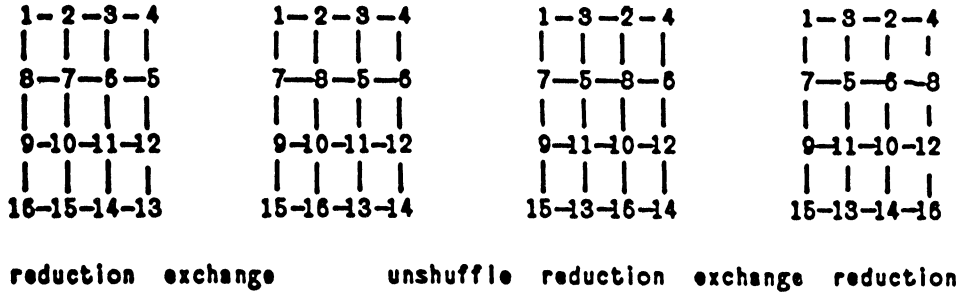


FIG. 11. *Odd-even cyclic reduction on a mesh.*

If the mesh has end-around connections from the end of one row to the beginning of the next row, i.e., the mesh is effectively a twisted cyclinder, then the *in-place* algorithm yields $3\sqrt{K}-4$ start-ups, if the equations are embedded from top to bottom in row major order.

Truncated cyclic reduction reduces the communication time, but not in proportion to the number of reduction steps avoided, because of the nonuniform communication cost.

Parallel cyclic reduction can be implemented on the mesh such that the number of start-ups is approximately $3\sqrt{K}-2$. All processors are used throughout the $O(\log_2 K)$ reduction steps.

The number of start-ups for multiple independent problems, or multiple right-hand sides, is approximately $2(3\sqrt{K}-2)$. The total number of elements transferred between any pair of processors is approximately $5NR$ for NR right-hand sides (or NR problems). Note that two problems can be solved concurrently by mapping one problem to processors 1 through $K-1$, and the other to processors 2 through K. After the first shuffle operation the two problems are confined to half of the array (left and right in Fig. 11).

5.3. Complete binary trees.

5.3.1. Inorder embedding. Presnell and Pargas [43] propose a cyclic reduction algorithm of communication complexity $O(\log_2^2 K)$ for complete binary tree configured ensembles. They map the equations to leaf nodes and use rotation operations for communication. In [27] we present an algorithm for complete binary trees that has a communication complexity of $O(\log_2 K)$.

The partial elimination order of cyclic reduction as defined by the elimination tree suggests an inorder mapping of partitions to processing elements. Such a mapping also allows for conservation of storage, Fig. 3.

With an inorder mapping of partitions to tree nodes, the first step of the cyclic reduction algorithm requires $k-1$ communication start-ups. Partition 2^k-1 is at distance $k-1$ from partitions 2^k-2 and 2^k. Successive reduction steps can be pipelined. Each reduction step following the first only incurs one additional communication action. The number of communications for the reduction phase is $2k-3$. Backsubstitution requires $k-1$ communications, and the total number of start-ups is $3\log_2 K-4$. Figure 12 illustrates the inorder mapping of partitions to tree nodes. The Appendix contains pseudo code for cyclic reduction on a complete binary tree.

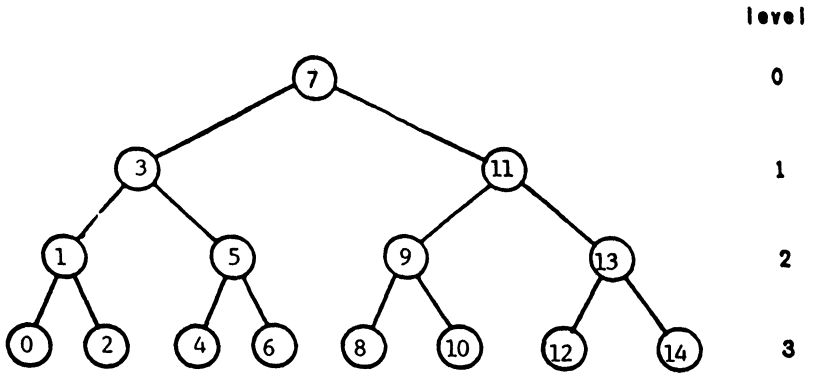


FIG. 12. *Inorder labeling of a complete binary tree.*

In the binary tree algorithm based on an inorder map of equations to nodes of the tree, the order of the time complexity cannot be reduced for truncated cyclic reduction. It takes $k-1$ communications to complete the first reduction step. However, fewer communications are required subsequent to this propagation phase. The backsubstitution phase still needs $k-1$ communication steps, since $x_{K/2-1}$ has to propagate to the leaf level. For m reduction steps the number of start-ups is $2(k-1)+m-1$, and the arithmetic complexity is proportional to $2m+1$.

The reduction in the estimated time complexity is proportional to the reduction in the number of steps in the reduction process. The constant of proportionality varies from 1 to $\frac{1}{3}$. The relative reduction in time for concurrent cyclic reduction is much greater than on a uniprocessor. This property is a consequence of the fact that approximately half of the number of arithmetic operations is performed in the first reduction step and the last backsubstitution step, one quarter in the second step, etc.

5.3.2. A proximity preserving path embedding. The inorder embedding fails to take full advantage of truncating the reduction phase. A proximity preserving path embedding has the potential to take full advantage of an early truncation of the reduction phase, but requires $O(\log_2^2 K)$ communications for the complete cyclic reduction algorithm. We describe a proximity preserving path embedding derived from the loop embedding due to Sekanina [54], [47].

The path embedding in Fig. 13 is generated by an algorithm that is a simple variation of the algorithm described by Rosenberg and Snyder [47]. The labeling proceeds in a depth-first manner. Every other node is labeled on descent. Nodes are labeled in postorder on ascent, except nodes on the path from the root to the leftmost and rightmost child which are labeled in inorder.

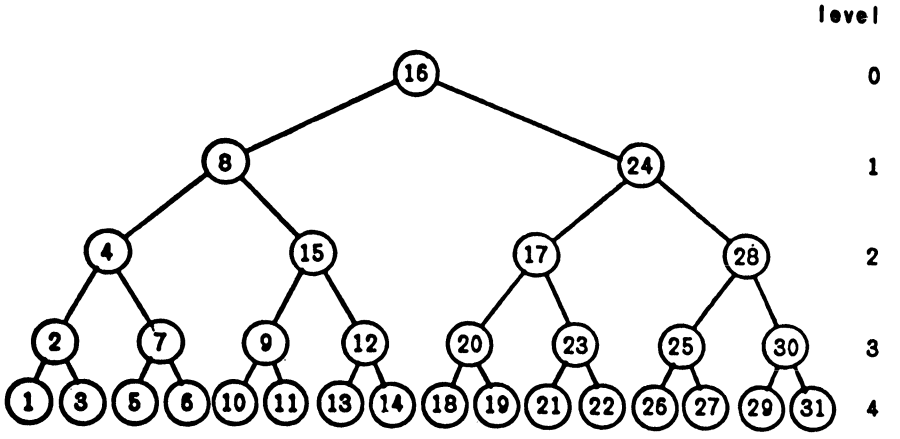


FIG. 13. *Mapping of a path to a binary tree preserving proximity.*

1. The nodes in the left subtree of the root are labeled in descending order, and the nodes in the right subtree in ascending order.
2. After having descended one level the right child of a node is always visited before the left child in the left subtree of the root. The converse is true for the right subtree of the root.
3. The nodes on the paths from the leftmost and rightmost leaf nodes to the root are labeled in inorder.

The first modification only affects the label assigned to a node. The second modification is made only to increase the formal similarity with a tree labeled in

inorder. The two labelings generated by the Rosenberg and Snyder algorithm and an algorithm with only the second modification are isomorphic. The third modification can be made without the maximum distance between adjacent nodes being increased, since we are only embedding a path, not a loop. The distance between the last labeled node in the left subtree and the first labeled node in the right subtree is of no concern in the embedding of a path.

THEOREM 5.3. *The maximum distance between adjacent nodes in the path is* 3 *when embedded by the path embedding algorithm.*

The proof follows that of Rosenberg and Snyder [47]. The modification of the arguments are simple, and can be found in [26].

Though not used for truncated cyclic reduction, we notice the following property.

THEOREM 5.4. *The number of communication steps necessary to change the embedding generated by the path embedding algorithm to an inorder embedding is* $k-3$.

Proof. The proof is in three parts.

(i) The nodes on the paths from the leftmost and rightmost leaf nodes to the root, including the leaf nodes, are labeled in inorder. The root is initially assigned its final inorder label. Its left child is labeled after the right subtree of that node is labeled. Hence, its label is $2k^{-1}-(2k^{-2}-1)-1=2^{k-2}$. By induction it is also true for the remaining nodes along the path. The same property is clearly true for the path to the rightmost leaf node.

(ii) The number of communication steps is at least $k-3$. In the left subtree of the root one node at level 2 is labeled $2^{k-1}-1$, i.e., it is odd and shall be at the leaf level in an inorder labeling.

(iii) It remains to be shown that there is no conflict in data movement. Since the arguments for the right subtree are symmetric, it suffices to prove that there are no conflicts in data movement for the left subtree of the root. The left subtree of the node at level 2 that needs to be relabeled is labeled after its right subtree. The labels start at $2^{k-2}+1$ and end at $3\times 2^{k-3}-1$. Hence, no exchange between the left and right subtrees is necessary. The right child of the node at level 2 being considered has the label $3\times 2^{k-3}$, since it is labeled last in the right subtree. A local exchange gives the node at level 2 the correct label. The theorem follows by induction. □

5.3.3. Parallel cyclic reduction, and multiple independent systems. Inorder mapping of equations to processors is not feasible for Hockney's parallel cyclic reduction. With all odd equations mapped to the leaves, each step requires $O(\log_2 N)$ time. Pipelining cannot be performed for all of the parallel reductions.

The inorder map allows the reduction phases of independent problems to be pipelined. The computation of one problem is initiated before the reduction phase of the preceding problem is complete. Hence, except for the first problem, the propagation time from the leaves to the root is masked by computations.

5.4. Shuffle-exchange networks. A shuffle-exchange network can be defined in terms of the binary encoding of the integers $\{0, 1, 2, \cdots, 2^k-1\}$. Let $(p_{k-1}p_{k-2}\cdots p_1p_0)$, $p_i=\{0,1\}$, $i=\{0,1,\cdots,k-1\}$ be node addresses. Then, node $(p_{k-1}p_{k-2}\cdots p_1p_0)$ is connected to node $(p_{k-2}p_{k-3}\cdots p_0p_{k-1})$ (obtained by a cyclic shift). Furthermore, node $(p_{k-1}p_{k-2}\cdots p_10)$ is connected to $(p_{k-1}p_{k-2}\cdots p_11)$. The edges obtained by cyclic shifts are called *shuffle edges*, and the even-to-odd edges *exchange edges*, [36]. Most nodes in a shuffle-exchange network are connected to three nodes. Nodes $(0\cdots 0)$ and $(1\cdots 1)$ are only connected to one other node, since a cyclic shift yields the same number. Nodes with addresses obtained by cyclic shifts of the same bit string are said to belong to the same necklace. The number of degenerate

necklaces, i.e., necklaces with fewer than k nodes, is of $O(\sqrt{K})$, whereas the number of full necklaces is of $O(K/\log_2 K)$. An 8-node shuffle-exchange network is shown in Fig. 14, and a 16-node network in Fig. 15.

The 8- and 16-node shuffle-exchange networks are planar, but arbitrary shuffle-exchange networks are in general not planar. The diameter of a shuffle-exchange graph is $2k-1$, i.e., almost identical to that of a complete binary tree. The maximum distance between consecutively labeled nodes is $2(k-2)$ and occurs between nodes $(100\cdots0)$ and $(011\cdots1)$. In general, the distance between a pair of nodes is at most twice the number of bit reversals in the addresses.

THEOREM 5.5. *A binary tree of 2^k-1 nodes can be mapped to a shuffle-exchange graph of 2^k nodes such that the distance between nodes adjacent in the tree is at most 2 in the shuffle-exchange graph.*

Proof. Label the nodes in the tree in breadth-first order, starting at the root and assign it the label 1. The label assigned to the left child of node i is $2i$. The label of the right child is $2i+1$. The label of the left child can be obtained by a left cyclic shift of the parent's address. All nodes above the leaf level have the highest order bit 0, and the left cyclic shift always generates an even address. Clearly, if a node of the binary tree is identified with a node in the shuffle-exchange network having the same address, then a parent node and its left child are adjacent. The right child of a tree node is mapped to the odd processor adjacent to the even processor of the left child. Hence, for every node in the tree, its left child is at distance 1, and its right child is at distance 2 (see Fig. 16).

For cyclic reduction on a shuffle-exchange network the partitions are mapped to the processors as described by the tree embedding algorithm, and the inorder tree algorithm used with minor modifications.

Equation $2^{k-1}-1$ is stored at a distance of $2k-3$ from equation 2^{k-1}. The number of communication start-ups is $5k-7$. The above shuffle-exchange algorithm inherits the properties of the tree algorithm. Hence, for truncated cyclic reduction there is still a propagation time of order $O(k)$.

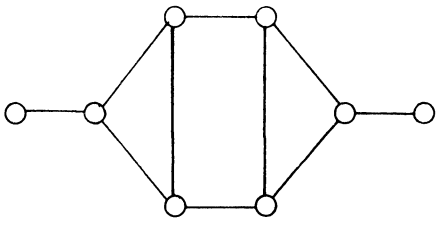


FIG. 14. *An 8-node shuffle-exchange network.*

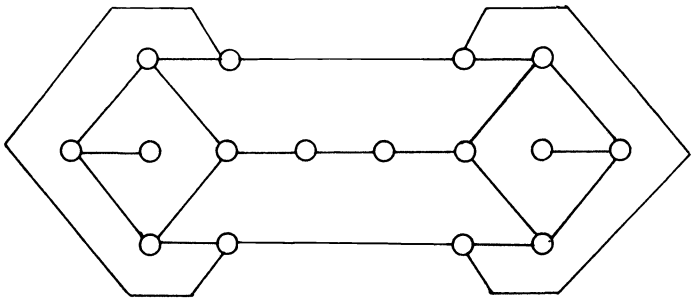


FIG. 15. *A 16-node shuffle-exchange network.*

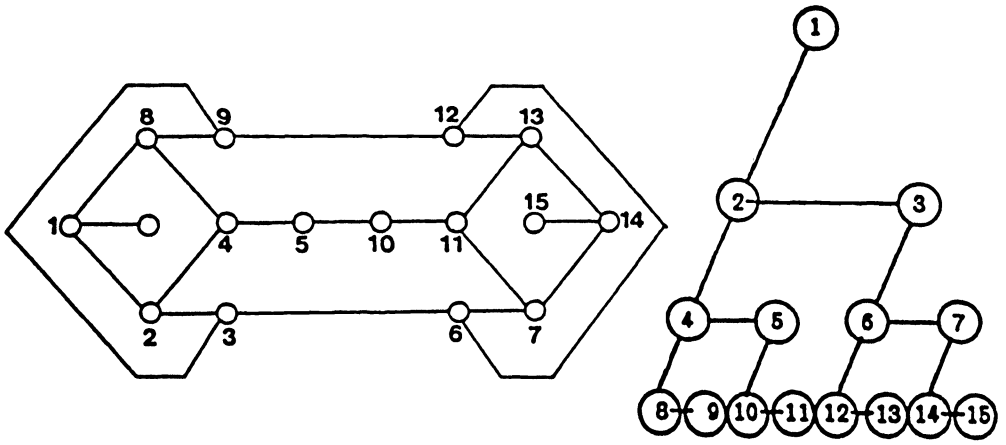


FIG. 16. *A binary tree embedding in a 16-node shuffle-exchange network.*

It is possible to embed two trees in the shuffle-exchange network in such a way that for cyclic reduction the computations for the two trees take place in distinct sets of processors. A processor may be shared for communication purposes. If one tree is mapped to the shuffle-exchange network as described above, then the other tree can be embedded by identifying a tree node with a processor whose address is equal to the bit-wise complementation of the tree node number.

Multiple independent problems can be pipelined in the same way as in the binary tree, in order to reduce the effects of the propagation time on the performance.

5.5. Perfect shuffle networks. A perfect shuffle network differs from a shuffle-exchange network in that an even node is also connected to the preceding odd node [55]. Hence, in the perfect shuffle network processors having successive addresses form a path.

Identify a partition with the processor having the same number. Then, the first reduction step only requires local communication. After the first reduction step is completed, an unshuffle (right cyclic shift) operation is carried out, bringing every even, one step reduced, equation into the processors with addresses in the lower half of the address space. The second reduction step is now carried out in these processors, requiring only nearest neighbor communication. The reduction phase is completed after $k-1$ reduction steps and $k-2$ unshuffle operations. The backsubstitution requires k equation solutions with nearest neighbor communication (in addresses) and $k-2$ shuffle operations (left cyclic shift).

This simple algorithm is not applicable to the shuffle-exchange network, since processors with successive addresses are in general not nearest neighbors. The number of communication start-ups for the perfect shuffle algorithm is $2(2k-3)$.

Truncating the reduction after m steps reduces the number of unshuffle operations to $m-1$. The total number of communication steps in each phase of the cyclic reduction computation is reduced to $2m-1$. The number of start-ups is $2(2m-1)$.

Parallel cyclic reduction can be implemented on the perfect shuffle with no principal difficulty.

Multiple independent problems can be solved concurrently using the balanced cyclic reduction algorithm. The number of start-ups is $2(2k-3)$. After the first shuffle operation half of the problems are contained in the processors with addresses in the lower half of the address space, and the other half of the problems in the processors

with addresses in the upper half of the address space. The total number of elements communicated over any interprocessor communication link is approximately $5NR$.

5.6. Boolean k-cubes. In a boolean k-cube, processors can be assigned addresses so that adjacent processors differ by only 1 bit. Each processor in a k-cube of $K = 2^k$ processors has k neighbors. There is a total of $kK/2$ connections. The diameter of the k-cube is k. A 3-cube is the common 3-dimensional cube. A boolean 4-cube is shown in Fig. 17.

Boolean k-cube algorithms for cyclic reduction can be obtained by embedding a binary tree in the cube, and adapting the tree algorithm to the cube. One tree embedding is obtained by assigning the root of the breadth-first numbered binary tree to processor 0 in the k-cube. Then, for a tree node assigned to processor $(00\cdots 0p_rp_{r-1}\cdots p_0)$, its left child is assigned to processor $(00\cdots 1p_rp_{r-1}\cdots p_0)$, and its right child to processor $(00\cdots 1\bar{p}_rp_{r+1}\cdots p_0)$, where $\bar{p}_r$ is the complement of p_r. The right child of a node is at distance 2 from its parent. Another embedding in which some adjacent tree nodes are at distance 2 when embedded in the cube is obtained by labeling the tree in inorder. Yet another one is used below. It is also possible to embed a 2^k-1 node binary tree in a $2k$-cube such that adjacent tree nodes are at distance 1 in the cube [32].

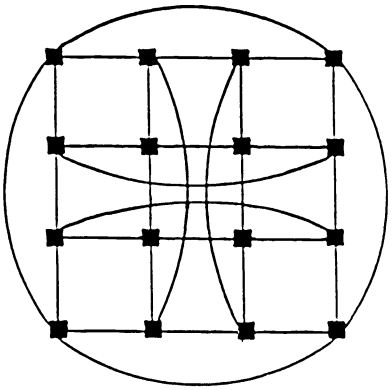


FIG. 17. *A boolean 4-cube.*

We will now briefly describe two k-cube algorithms: an *in-place* algorithm, and a *folding* algorithm. In the first algorithm equations remain in their original location throughout the computations. In the second, computations are performed in cubes of successively lower dimensions by properly moving equations to lower dimensional cubes as the computations progress. The *folding* algorithm is described in detail in terms of pseudocode in the Appendix. Note that in the k-cube, processors with successive addresses are in general not neighbors. For instance, processors $K/2-1$ and $K/2$ are a distance k apart. The objective is to embed the graph of Fig. 2. For convenience we number the equations starting from 0. The equations are initially mapped to the processors in the cube using a *binary-reflected* Gray code [44]. Gray codes for successive integers differ by only 1 bit. This property guarantees that the first reduction step only involves nearest neighbor communication.

Let G_i be the binary-reflected Gray code of i, and $G(k) = (G_0, G_1, \cdots, G_{2^k-1})$ be a k-bit code. Then, a binary-reflected Gray code is defined by

$$G(k+1) = (0G_0, 0G_1, \cdots, 0G_{2^k-1}, 1G_{2^k-1}, \cdots, 1G_1, 1G_0)$$

or

$$G(k+1) = (G_0 0, G_0 1, G_1 1, G_1 0, G_2 0, G_2 1, \cdots, G_{2^k-1} 1, G_{2^k-1} 0).$$

The binary-reflected 4-bit Gray code is

$$G(4) = (0000, 0001, 0011, 0010, 0110, 0111, 0101,$$
$$0100, 1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000).$$

Moreover, let $i = (r_k r_{k-1} \cdots r_0)$ and $G_i = (g_k g_{k-1} \cdots g_1)$. Then the encoding and decoding is given by $g_j = (r_j + r_{j-1}) \bmod 2$ (encoding) and $r_j = (\sum_{i=j+1}^{k} g_i) \bmod 2$ (decoding) (note $r_k = 0$) [44].

LEMMA 5.1. *For the binary-reflected Gray code G_i and G_{i+2^j} differs in precisely* 2 *bits for $j > 0$. Those bits are g_j and g_{s+1}, where s is the bit position in which the carry stops propagating when 2^j is added to i.*

Proof. Let the binary encoding of i be $(i_n i_{n-1} \cdots i_0)$ and that of $r = i + 2^j$ be $(r_n r_{n-1}, \cdots, r_0)$. Furthermore let the Gray code of i be $G(n) = (g_n g_{n-1} \cdots g_1)$ and that of r be $H(n) = (h_n h_{n-1} \cdots h_1)$. Then $i_m = r_m$, $m = \{0, 1, \cdots, j-1\}$ and $r_m = \bar{i}_m$, $m = \{j, j+1, \cdots, s\}$, where $s \geqq j$ is the bit where the carry stops propagating. It follows from the encoding formula that $h_m = g_m$, $m = \{1, 2, \cdots, j-1, j+1, \cdots, s\}$ and $h_j = \bar{g}_j$, $h_{s+1} = \bar{g}_{s+1}$. □

THEOREM 5.6. *The number of communication steps per reduction step is* 2 *for the in-place algorithm, except for the first step for which it is* 1.

Proof. The communication distance for the first step is 1 from the definition of the code, and 2 for the remaining reduction steps by Lemma 5.1. It remains to be shown that the communication paths can be made disjoint. The following routing guarantees disjoint paths:

- Processor G_i sends data to processor G_{i+2^j} by first sending it to the processor with address obtained by complementing the lowest order bit that differs in the codes G_i and G_{i+2^j}.
- Processor G_{i-2^j} sends data to processor G_i by first sending it to the processor with address obtained by complementing the highest order bit that differs in G_{i-2^j} and G_i. □

In the *in-place* algorithm some processors only serve as routing nodes for all but the first reduction step. It is necessary to compute the addresses of these processors, and to insert the necessary communication code, also for the reduction and backsubstitution steps in which no computations take place. In the *folding* algorithm there is no need for communication code when the processor no longer takes part in the reduction computations.

For the *folding* algorithm we observe that the equations that participate in reduction step j have indices in the set $M^j = \{(m_j + 1)2^j - 1\}$, $m_j = \{0, 1, 2, \cdots, 2^{k-j} - 2\}$. Elimination is performed on equations with indices in the set M^{j+1}. If the equations in M^j are embedded in the cube with one equation per node, and such that consecutive equations are in adjacent nodes, then from Lemma 5.1 consecutively ordered nodes in M^{j+1} are at distance 2.

LEMMA 5.2. *By performing one exchange operation on selected pairs of adjacent nodes between successive reduction steps the equations subject to elimination operations form a path with successively higher ordered equations in adjacent processors.*

Proof. It is true for $j = 0$ by construction. From the definition of the binary-reflected Gray code, it follows that $G_{2q+1}(k-p) = (G_q(k-p-1)X)$, where $X = \{0, 1\}$, $p = 0, 1, \cdots, k-2$ and $q = 0, 1, \cdots, 2^{k-p-1} - 1$. The equations in M^j on which reduction

is performed are the ones with $m_j = \{1, 3, \cdots, 2^{k-j}-3\}$, i.e., the indices are of the form $2i+1$, $i = 0, 1, \cdots, 2^{k-j}-2$. Let $p = j-1$. Then $G_{2q+1}(k-(j-1)) = (G_q(k-j)X)$, $q = 0, 1, \cdots, 2^{k-j}-1$. Hence, an equation with index in the set $\{(2i+2)2^j-1\}$, $i = \{0, 1, \cdots, 2^{k-j}-2\}$ is either in a node with $X = 1$ or can be moved to that node by an exchange operation between nodes $(G_q(k-j)0)$ and $(G_q(k-j)1)$. The theorem follows by induction. □

From the encoding and decoding formulas and the above lemma it follows that each processor has sufficient information locally to determine if an exchange is necessary, and with which processor to exchange data. The necessary information items are: processor address, equation index (which can be computed from the address), and reduction step index (j). The second reduction step is carried out in the subcube $(G_i 1)$, $i = K/2-1$. The exchange-reduction steps are repeated $k-3$ additional times on successively smaller subcubes. In the jth step, exchange is performed on the jth lowest order bit, if the leading $k-j$ bits encode an even integer and $g_j = 1$, or the integer is odd and $g_j = 0$.

Processor $(01 \cdots 11)$ computes $x_{2^{k-1}-1}$. Backsubstitution is then carried out in reverse order compared to the reduction phase. The number of processors computing x-values doubles for every step. In the final step half of the processors compute half of the unknowns.

The communication complexity is the same for both the *in-place* and *folding* algorithm. The number of start-ups is $2(2k-3)$.

In the k-cube algorithms based on binary-reflected Gray codes, full advantage can be taken of truncated cyclic reduction. Each reduction step is carried out on all relevant equations during the same time step. Hence, truncating the reduction after m steps reduces the total time proportionally.

Parallel cyclic reduction can be carried out on the k-cube in $O(k)$ time.

Multiple independent problems can be solved by pipelined Gaussian elimination with a complexity of

$$(5.1) \qquad \sqrt{2(K-4)P\tau}(\sqrt{3t_c+5t_a}+\sqrt{t_c+3t_a})+8Pt_a+4Pt_c+(K-4)\tau$$

and by the balanced cyclic reduction algorithm with a complexity of

$$(5.3) \qquad ((17K-18\log_2 K+2)t_a+(5K-\log_2 K)t_c)\lceil P/K \rceil+2(2\log_2 K-1)\tau.$$

An alternative to these two *in-place* algorithms is to transpose the data such that each tridiagonal system is solved locally. If the transpose operation is carried out recursively, then $\log_2 K$ steps are required on the k-cube [32]. The communication complexity for the two transpose operations are $(5P/2t_c+2\tau)\log_2 K$, $P \geqq K$. Pipelining the communication and optimizing the packet size in order to minimize the communication time yields a total complexity of

$$(5.4) \qquad (8-7/K)Pt_a+2\sqrt{5P(\log_2 K-1)\tau t_c}+5P/2t_c+2(\log_2 K-1)\tau.$$

In (5.4) the same packet size $(\sqrt{5P\tau/(4(\log_2 K-1)t_c)})$ is used in both transpose operations. The number of elements communicated is approximately half of that for the balanced cyclic reduction algorithm, and so is the arithmetic complexity, since Gaussian elimination is used locally instead of cyclic reduction. The complexity of the optimally pipelined transpose—local solve—transpose algorithm TGET and that of the balanced cyclic reduction algorithm compares approximately as $a+b+2\sqrt{ab}+c$ and $2(a+b+c)$ (with $a = 2(\log_2 K-1)$, $b = 5Pt_c/2$, and $c = 8Pt_a$).

THEOREM 5.7. *An optimally pipelined transpose—local solve—transpose tridiagonal solver is of a lower complexity than a balanced cyclic reduction solver by a small constant factor (between* 0.5 *and* 1).

A total optimum may be achieved by combining cyclic reduction with Gaussian elimination for the last several reduction steps [23].

5.7. Shared memory architectures. We have in our computational model assumed that each processor has its own local storage. In architectures such as the Ultracomputer [50], [14] and the TRAC [53], processors and storage units are on opposite sides of a switching network. The communication requirements for parallel cyclic reduction [18] have been analyzed for the TRAC architecture by Kapur and Browne [33], and for the Ultracomputer by Peskin [42]. The communication complexity for the shared memory architectures increases by a factor proportional to the depth of the switching network, compared to the previously described ensemble architectures. The sequential dependencies in the cyclic reduction algorithm are such that communication through the network is required for each reduction step. Pipelining the switch network does not reduce the effect on the switch latency. The communications complexity is proportional to $\log_2^2 K$. This complexity is higher than for the binary tree, the shuffle-exchange, the perfect shuffle, and the k-cube by a factor of $\log_2 K$.

5.8. Programming issues. All algorithms described above have distributed control, and data is distributed throughout the storage of all processors. The binary tree algorithm has three kinds of programs: one for the root, one for intermediate level processors, and one for the leaf processors. The shuffle-exchange network algorithm also uses three different codes. The boolean k-cube algorithm uses the same code in all processors. Hence, even though all algorithms are of the MIMD type, the degree of program uniformity across the ensemble is indeed high. The time for program loading can be made short for the tree by using recursive program loading, and an encoding of the binary tree as described in [38]. For a few equations per node the ratio of total program store to data store is significant.

6. Summary and conclusions. The communication complexity of the tridiagonal system solvers described here is of the same order as the diameter of the ensemble configuration. Hence, for networks of N processing elements and a diameter of order $O(\log_2 N)$, the total solution time is of the same order as the lower bound for the solution of an irreducible system of equations on any circuitry with bounded fan-in.

In the case of one equation per processing element, $N = K$, and assuming that all elimination and backsubstitution operations are performed in the processing element storing the row subject to elimination, the arithmetic complexity of cyclic reduction is the same for all ensembles, namely $17(\log_2 K - 2) + 14$ operations. This complexity can be reduced to $11(\log_2 K - 2) + 11$ operations if the operations are shared between the processing elements storing the pivot rows and the ones storing the rows subject to elimination. However, this requires additional communication per reduction phase. The arithmetic complexity can be further reduced to $9(\log_2 K - 2) + 11$ with two additional communications per step.

In the first case, the number of elements per communication is 4 in the reduction phase and 1 in the backsubstitution phase, except for the binary tree (and the shuffle-exchange) algorithm, for which 2 elements are communicated between a pair of processors in each step. In all but the binary tree (and shuffle-exchange) algorithm the two variables needed in the backsubstitution on an equation can be communicated over different links. If the operations required for the elimination of a single element

is shared between two processing elements, then the maximum number of elements per communication in the reduction phase is reduced to 3.

With all operations for the elimination of the two off-diagonal elements in a row carried out in one processor the number of communication start-ups is $K-2$ for the linear array and an *in-place* algorithm. For the 2-dimensional mesh the number of start-ups is $3\sqrt{K}+\log_2 K-4$ (or $3\sqrt{K}-4$ if the mesh has end-around connections). The number of start-ups for the binary tree is $3\log_2 K-4$, $5\log_2 K-6$ for the shuffle-exchange network, and $2(2\log_2 K-3)$ for the perfect shuffle and boolean cube networks. The number of start-ups increases by 50% for the linear array, if the operations for the elimination of a single element is shared between a pair of processors, and doubles if load sharing is performed as indicated for the lowest arithmetic complexity. For the perfect shuffle and boolean cube algorithms the corresponding numbers are 25% and 50%, respectively. For the binary tree and shuffle-exchange algorithms the load sharing prevents effective pipelining. The communication complexity increases to order $O(\log_2^2 K)$.

We conclude that if $\tau \ll t_a$ then the load sharing can be used effectively on a perfect shuffle and boolean cube. However, if $\tau \geqq t_a$, then load sharing does not pay off (but it may for a banded matrix [29]). The complexity term distinguishing the ensemble configurations is the communication complexity which is summarized in Table 1.

For a linear array cyclic reduction is more efficient than 2-way Gaussian elimination if the communication time is equal to the time for an arithmetic operation. However, if the communication is at least an order of magnitude slower than the arithmetic (a factor of 16 in our estimates), then Gaussian elimination is of a lower time complexity.

TABLE 1
Communication complexities (approximate).

Configuration	Total
Linear array	$(K-2)(5/2t_c+\tau)$
2-dim mesh w/o end-around	$(3\sqrt{K}+\log_2 K-4)(5/2t_c+\tau)$
2-dim mesh w/end-around	$(2k-1)t_a+(3\sqrt{K}-4)(5/2t_c+\tau)$
Binary tree	$10(k-2)t_c+(3k-4)\tau$
Shuffle-exchange	$2(6k-11)t_c+(5k-4)\tau$
Perfect shuffle	$(2k-3)(5t_c+2\tau)$
Boolean k-cube	$(2k-3)(5t_c+2\tau)$

With a large number of equations per node we use the consecutive scheme for identifying nodes with processing elements, i.e., substructuring. Moreover, we perform local forward and backward elimination such that a tridiagonal system with one equation per processing element results, and the previous analysis applies. The number of local arithmetic operations is approximately $17N/K$. The total complexity for GECR is of order $O(N/K+K)$ for the linear array, $O(N/K+\sqrt{K})$ for the 2-dimensional mesh, and $O(N/K+\log_2 K)$ for the complete binary tree, the shuffle-exchange and perfect shuffle networks, and the boolean k-cube. The difference in complexity between

different ensemble configurations is insignificant for $N \gg K$. The speed-up is at first linear in K. The range for linear speed-up depends on the type of interconnection, and the ratio $\alpha = t_c/t_a$.

The minimal computational time is $O(\log_2 N)$ for the complete binary tree (and the shuffle-exchange network), the perfect shuffle and the boolean k-cube. The speed-up has a maximum for $K \approx N/(1+\alpha)$. The maximum speed-up for a 2-dimensional mesh is obtained for $K \approx (N/2\alpha)^{2/3}$, and for the linear array for $K \approx (N/\alpha)^{1/2}$. The minimal computational times are $O(N^{1/3})$ and $O(N^{1/2})$, respectively. Table 2 summarizes the optimum ensemble sizes and the corresponding time complexities.

TABLE 2
Optimum ensemble sizes and time complexities (approximate).

Configuration	Optimum Size	Minimum Time
Linear array	$O(\sqrt{N/\alpha})$	$O(\sqrt{N})$
2-dim Mesh	$O(N/2\alpha)^{2/3}$	$O(N^{1/3})$
Binary tree	$N/(1+\alpha)$	$O(\log_2 N)$
Shuffle-exchange	$N/(1+\alpha)$	$O(\log_2 N)$
Perfect shuffle	$N/(1+\alpha)$	$O(\log_2 N)$
Boolean k-cube	$N/(1+\alpha)$	$O(\log_2 N)$

The decrease in speed-up is quite dramatic for a linear array with slow communication as can be seen in Fig. 18. The figure displays graphically the complexity estimates in Table 4. The tree algorithm is superior to the perfect shuffle and k-cube algorithms if there is only one equation per processor, but inferior if there are multiple equations per processor. The difference is small for fast communication, but significant for slow communication. The shuffle-exchange algorithm is inferior by a factor of 2, at most. The efficiencies, i.e., the (speed-up)/(number of processors), decrease fairly rapidly if the number of processors is increased beyond a certain number that depends on problem size and the relative cost of communication. The dependence on the problem size for the binary tree, the perfect shuffle, and the k-cube, is illustrated in Fig. 19, and the dependence on the relative communication cost in Fig. 20. Figure 21 shows how the ratio of processors to problem size for a given efficiency decreases as a function of increased relative communication cost. The ratio increases with the problem size.

Truncated cyclic reduction reduces the total time in proportion to the number of steps avoided for the binary tree, the shuffle-exchange, perfect shuffle, and boolean k-cube algorithms. The relative reduction for the binary tree ranges from $(k-m)/k$ for $t_c \ll t_a$ to $(k-m)/3k$ for $t_c > t_a$. The relative reduction for the perfect shuffle and the k-cube is $(k-m)/k$. For the mesh the reduction is not linear due to the shuffle operations. Since the tree algorithm benefits from truncated reduction to a lesser extent than the perfect shuffle and k-cube algorithms, it may lose its edge over those algorithms for truncated reduction. Table 3 gives the complexity estimates for truncated cyclic reduction on a binary tree, perfect shuffle, and k-cube.

The proportional reduction in computational time offered by truncated cyclic reduction on the binary tree, the shuffle-exchange, the perfect shuffle and the k-cube

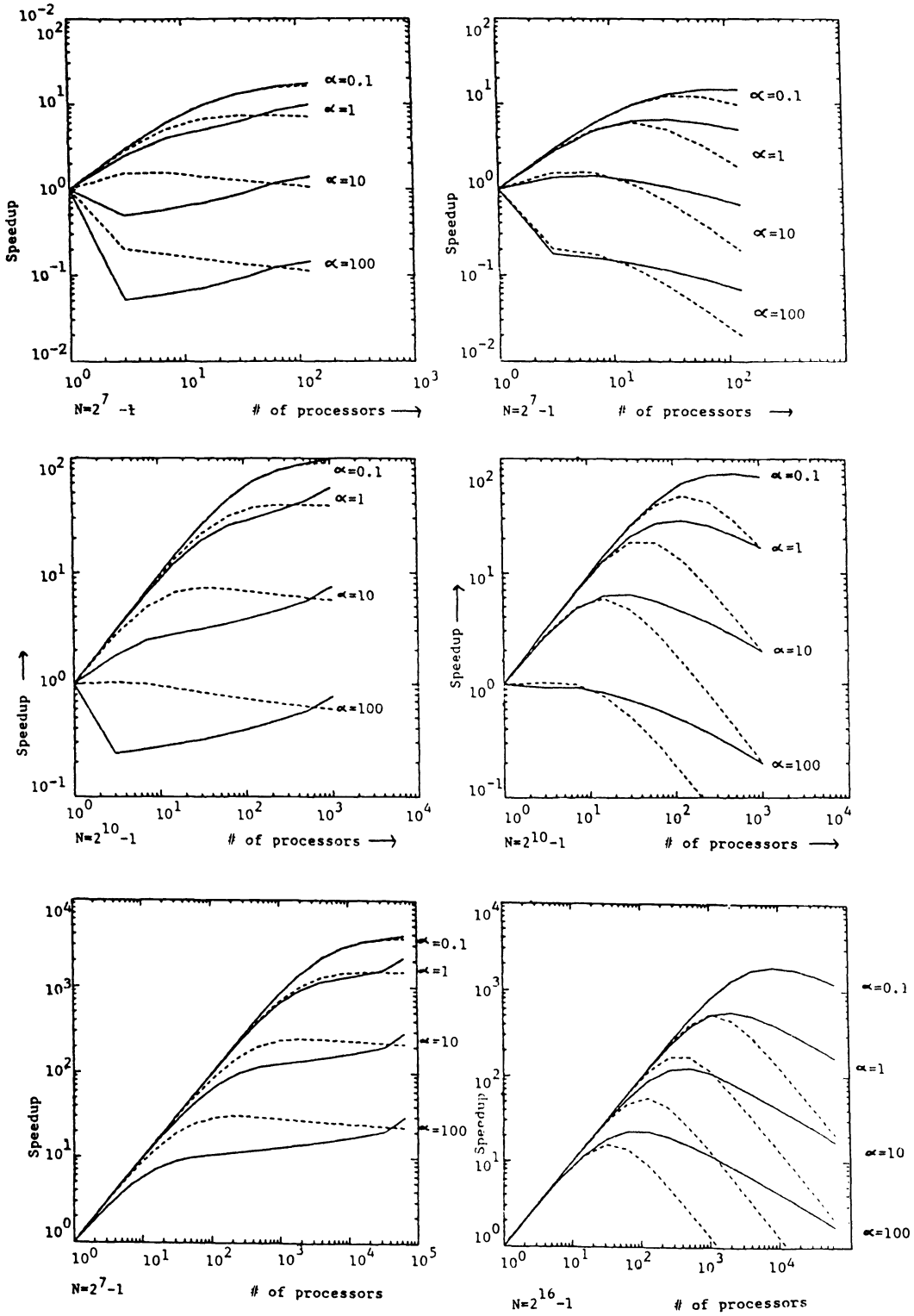


FIG. 18. *Speed-up for* $N = 2^7 - 1$, $N = 2^{10} - 1$, *and* $N = 2^{16} - 1$. $\alpha = 0.1$, 1, 10, *and* 100.

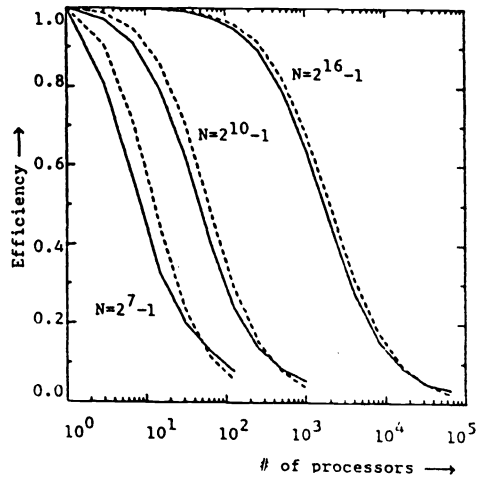


FIG. 19. *The efficiency for a binary tree, perfect shuffle and k-cube,* $\alpha = 1 N = 2^7 - 1, 2^{10} - 1,$ *and* $2^{16} - 1$.

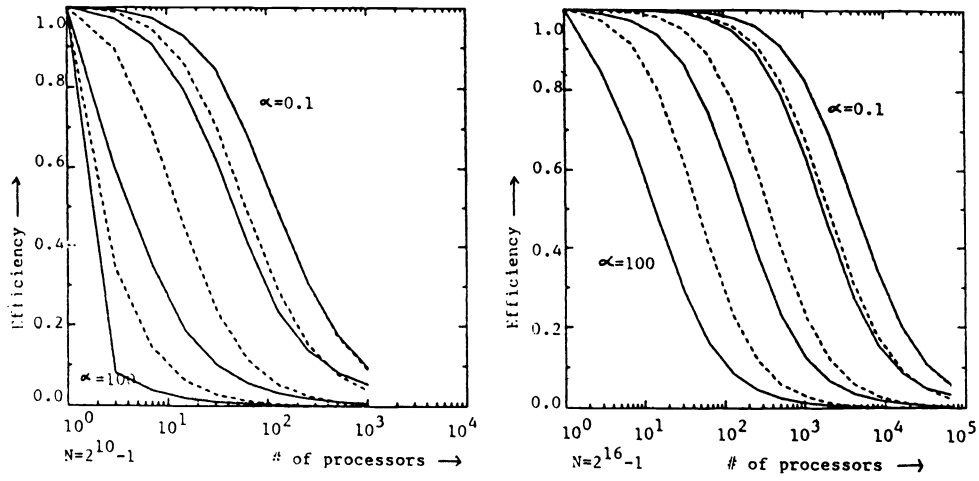


FIG. 20. *The efficiency for a binary tree, perfect shuffle and k-cube,* $N = 2^{10} - 1, 2^{16} - 1$. $\alpha = 0.1, 1, 10, 100$.

is significantly better than on a uniprocessor. Most operations are performed in the first few reduction steps, and the last few backsubstitution steps.

For multiple independent problems partitioning makes the most efficient use of the processors for all investigated topologies. If the data for each problem is uniformly distributed across the ensemble by domain decomposition (substructuring), then the balanced cyclic reduction algorithm yields high processor utilization without an increase in the number of start-ups. The parallel arithmetic complexity eventually becomes approximately twice that of pipelined Gaussian elimination. The number of elements communicated is only slightly larger for cyclic reduction. We summarize the complexity estimates for multiple independent problems on a linear array and a boolean k-cube in Table 4. The complexity of Gaussian elimination is the same for both configurations.

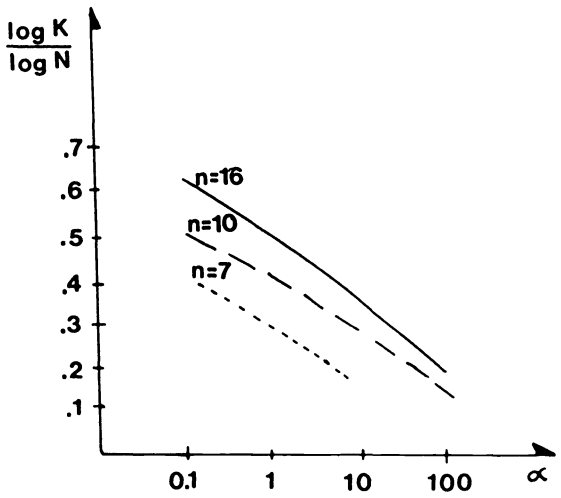


FIG. 21. *Relative number of processors for* 80% *efficiency (tree, perfect shuffle, cube).*

TABLE 3
Estimated complexities for truncated, m-step, cyclic reduction.

Configuration	Complexity
Binary tree	$(m+2)t_a+(m-1)\max(t_a, t_c)+2(k-1)t_c$
Perfect shuffle and k-cube	$(2m+1)t_a+2m(2m-1)t_c$

TABLE 4
Estimated complexities for some ensembles, multiple equations per processor, $k>1$.

Configuration	Complexity
(Linear array)—GE	$\sqrt{2(K-4)P\tau}(\sqrt{3t_c+5t_a}+\sqrt{t_c+3t_a})+8Pt_a+4Pt_c+(K-4)\tau$
Linear array—CR	$((17K-18\log_2 K+2)t_a+(5K-\log_2 K)t_c)\lceil P/K\rceil+(K-2)\tau$
Boolean k-cube—CR	$((17K-18\log_2 K+2)t_a+(5K-\log_2 K)t_c)\lceil P/K\rceil$ $+2(2\log_2 K-1)\tau$
Boolean k-cube—TGET	$(8-7/K)Pt_a+2\sqrt{5P(\log_2 K-1)\tau t_c}+5P/2t_c+2(\log_2 K-1)\tau$

On a linear array Gaussian elimination is of a lower complexity than the balanced cyclic reduction algorithm for

$$P\approx 0.2\frac{1+\alpha/2+\sqrt{15(1+3\alpha/5)(1+\alpha)/16}}{(1+\alpha/9)^2}(K-4)\tau/t_a,$$

approximately. Hence, if the overhead in communication is high relative to the time for a floating-point operation, then the pipelined Gaussian elimination algorithm is preferable only if $P \gg K$. On a boolean cube an optimally pipelined transpose algorithm, local Gaussian elimination, and another transposition is of slightly lower complexity than the balanced cyclic reduction algorithm, which in turn always has fewer start-ups

($O(\log_2 K)$ instead of $O(K)$) than the pipelined Gaussian elimination algorithm. If start-ups are ignored, then the pipelined Gaussian elimination algorithm is of a lower complexity than the balanced cyclic reduction if $P > K(4ta + tc)/(9ta + 3tc)$. Algorithm TGET is always of a lower complexity than pipelined Gaussian elimination.

Finally, we note that all algorithms have distributed control. Moreover, even though the algorithms make use of the MIMD feature of the architecture, only a few different codes are used. In the binary tree there are three kinds: one for the root, one for the intermediate level processors, and one for the leaves. For the k-cube all processors execute the same code. The code for these architectures can be loaded recursively.

7. Appendix.

7.1. A mesh algorithm. Let rows and columns be numbered from 0 to $\sqrt{K}-1$. Then, the algorithm is:

```
begin
  for i := 1 to k/2 − 1 do
    begin
      Communicate with adjacent processors
      Perform a reduction computation
      Odd rows execute exchange operations between even columns and the
          succeeding odd column
      Perform a shuffle operation of size 2^(k/2−(i−1))
    end
  end
  Communicate with adjacent processor
  Perform a reduction computation
  Odd rows execute exchange operations between the even column and the
      succeeding odd column
  invoke a linear array algorithm
end
```

7.2. A complete binary tree algorithm.

Root processor(i):

$x_0 := 0$; $x_{N+1} := 0$

$k := 1$

$m := i/(2k)$

Reduction computations

while m is even do

receive $(al, bl, cl, yl, a_{i-k}, b_{i-k}, c_{i-k}, y_{i-k})$ from the left child

receive $(a_{i+k}, b_{i+k}, c_{i+k}, y_{i+k}, ar, br, cr, yr)$ from the right child

$e_i := -a_i/b_{i-k}$

$f_i := -c_i/b_{i+k}$

$a_i := e_i a_{i-k}$

$c_i := f_i c_{i+k}$

$b_i := b_i + e_i c_{i-k} + f_i a_{i+k}$

$y_i := y_i + e_i y_{i-k} + f_i y_{i+k}$

$k := 2k$

$m := m/2$

enddo

The last reduction step

for m odd do
  receive $(a_{i-k}, b_{i-k}, c_{i-k}, y_{i-k})$ from the left child
  receive $(a_{i+k}, b_{i+k}, c_{i+k}, y_{i+k})$ from the right child
  $e_i := -a_i / b_{i-k}$
  $f_i := -c_i / b_{i+k}$
  $a_i := e_i a_{i-k}$
  $c_i := f_i c_{i+k}$
  $b_i := b_i + e_i c_{i-k} + f_i a_{i+k}$
  $y_i := y_i + e_i y_{i-k} + f_i y_{i+k}$
enddo

Backsubstitution

  $x_i := y_i / b_i$
  send (x_{i-2k}, x_i) to the left child
  send (x_i, x_{i+2k}) to the right child

Intermediate level processor(i):

  $k := 1$
  $m := i/(2k)$

Reduction computations

while m is even do
  receive $(al, bl, cl, yl, a_{i-k}, b_{i-k}, c_{i-k}, y_{i-k})$ from the left child
  receive $(a_{i+k}, b_{i+k}, c_{i+k}, y_{i+k}, ar, br, cr, yr)$ from the right child
  send $(al, bl, cl, yl, ar, br, cr, yr)$ to the parent
  $e_i := -a_i / b_{i-k}$
  $f_i := -c_i / b_{i+k}$
  $a_i := e_i a_{i-k}$
  $c_i := f_i c_{i+k}$
  $b_i := b_i + e_i c_{i-k} + f_i a_{i+k}$
  $y_i := y_i + e_i y_{i-k} + f_i y_{i+k}$
  $k := 2k$
  $m := m/2$
enddo

The last reduction step for node i

for m odd do
  receive $(a_{i-k}, b_{i-k}, c_{i-k}, y_{i-k})$ from the left child
  receive $(a_{i+k}, b_{i+k}, c_{i+k}, y_{i+k})$ from the right child
  send $(a_{i-k}, b_{i-k}, c_{i-k}, y_{i-k}, a_{i+k}, b_{i+k}, c_{i+k}, y_{i+k})$ to the parent
  $e_i := -a_i / b_{i-k}$
  $f_i := -c_i / b_{i+k}$
  $a_i := e_i a_{i-k}$
  $c_i := f_i c_{i+k}$
  $b_i := b_i + e_i c_{i-k} + f_i a_{i+k}$
  $y_i := y_i + e_i y_{i-k} + f_i y_{i+k}$
  send (a_i, b_i, c_i, y_i) to the parent
enddo

Backsubstitution

receive (x_{i-2k}, x_{i+2k}) from the parent
$x_i := (y_i - a_i x_{i-2k} - c_i x_{i+2k})/b_i$
send (x_{i-2k}, x_i) to the left child
send (x_i, x_{i+2k}) to the right child

Leaf processor(i):

Reduction computations

send (a_i, b_i, c_i, y_i) to the parent

Backsubstitution

receive (x_{i-1}, x_{i+1}) from the parent
$x_i := (y_i - a_i x_{i-1} - c_i x_{i+1})/b_i$

7.3. A boolean k-cube algorithm. {Let $(g_k g_{k-1} \cdots g_1)$ be the address of the processor in a k-cube, let $G_m(k-j)$ be the $(k-j)$-bit binary-reflected Gray code of m, and let $j+1$ denote reduction step.}

$j := 0$
$g_0 := 1$
$x_{-1} := 0$
$x_K := 0$
while $g_j = 1$ and $j < k-1$ do
  $m := dec(g_k g_{k-1} \cdots g_{j+1})$ (m is the integer encoded by $k-j$ highest order bits $(G_m(k-j)111)$)
  $G_{m+1}(k-j) := enc(m+1)$ (address of the processor holding the succeeding integer in the $(k\text{-}j)$-cube)
  $G_{m-1}(k-j) := enc(m-1)$ (address of the processor holding the preceding integer in the $(k\text{-}j)$-cube)

{even processors in the $(k-j)$-subcube send their equations to the processors holding preceding and succeeding odd integers, except that the processor holding row $m=0$ only communicates with the processor holding the row $m=1$, and the processor holding the row $m = 2+k-j-2$ only communicates with the processor holding row $m-1$}
{processors holding odd integers communicate and perform reduction computations}

if m even and $m > 0$ then
  send (a_i, b_i, c_i, y_i) to the processor $(G_{m-1}(k-j)11 \cdots 1)$
elseif m even and $m < 2^{k-j} - 2$ then
  send (a_i, b_i, c_i, y_i) to the processor $(G_{m+1}(k-j)11 \cdots 1)$
elseif m odd and $mne 2^{k-j} - 1$ then
  receive $(a_{i-2^j}, b_{i-2^j}, c_{i-2^j}, y_{i-2^j})$ from processor $(G_{m-1}(k-j)11 \cdots 1)$
  receive $(a_{i+2^j}, b_{i+2^j}, c_{i+2^j}, y_{i+2^j})$ from processor $(G_{m+1}(k-j)11 \cdots 1)$
  $e_i := -a_i/b_{i-k}$
  $f_i := -c_i/b_{i+k}$
  $a_i := e_i a_{i-k}$
  $c_i := f_i c_{i+k}$
  $b_i := b_i + e_i c_{i-k} + f_i a_{i+k}$
  $y_i := y_i + e_i y_{i-k} + f_i y_{i+k}$
endif

{exchange data so that equations needed for the next reduction step are in the next lower dimensional subcube (with yet another address bit 1). No exchange is needed

when the cube is reduced to a 2-cube}
if $j < k-2$ then
if m is odd and $g_{j+1} = 0$ then
send (a_i, b_i, c_i, y_i) to the processor $(g_k g_{k-1} \cdots g_{j+2} 111 \cdots 1)$
receive (a_i, b_i, c_i, y_i) from the processor $(g_k g_{k-1} \cdots g_{j+2} 111 \cdots 1)$
endif
if m is even and $g_{j+1} = 1$ then
send (a_i, b_i, c_i, y_i) to the processor $(g_k g_{k-1} \cdots g_{j+2} 011 \cdots 1)$
receive (a_i, b_i, c_i, y_i) from the processor $(g_k g_{k-1} \cdots g_{j+2} 011 \cdots 1)$
endif
endif
$j := j+1$
enddo
{the reduction is now completed for equation i. j is the index of the last reduction step in which the processor participated. The reduction phase is complete when $j = n-1$ for a subcube of dimension 2.}

{The backsubstitution starts with processor $(0111 \cdots 1)$ computing $x_{2^{k-1}}$. Processors $(0011 \cdots 1)$ and $(1111 \cdots 1)$ then compute $x_{2^{k-2}-1}$ and $x_{3*2^{k-2}-1}$, respectively. For computation of additional unknowns it is necessary successively to increase the dimensions of the cube, and make the proper exchanges reversing the exchanges in the reduction phase.}
while $j \geqq 0$ do
{Solve for $x_{2^{k-1}-1}$, $x_{2^{k-2}-1}$ and $x_{3*2^{k-2}-1}$ in the 2-cube of the final step of the reduction phase}
if $j = k-1$ then
{solve for $x_{2^{k-1}-1}$ in processor $(0111 \cdots 1)$}
if $m = 1$ then
$x_i := y_i / b_i$
send (x_{-1}, x_i) to processor $(G_{m-1} 11 \cdots 1)$
send (x_i, x_K) to processor $(G_{m+1} 11 \cdots 1)$
endif
{Solve for $x_{2^{k-2}-1}$}
if $m = 0$ then
receive $(x_{i-2^{k-2}}, x_{i+2^{k-2}})$ from processor $(G_{m+1} 11 \cdots 1)$
$x_i := (y_i - a_i x_{i-2^{k-2}} - c_i x_{i+2^{k-2}}) / b_i$
endif
{Solve for $x_{3*2^{k-2}-1}$}
if $m = 2$ then
receive $(x_{i-2^{k-2}}, x_{i+2^{k-2}})$ from processor $(G_{m-1} 11 \cdots 1)$
$x_i := (y_i - a_i x_{i-2^{k-2}} - c_i x_{i+2^{k-2}}) / b_i$
endif
$j := j-1$
endif
{Reverse the exchange of equations that occurred during the reduction phase}
$j := j-1$
$m := dec(g_k g_{k-1} \cdots g_{k-j})$
$G_{m-1} := enc(m-1)$
$G_{m+1} := enc(m+1)$

if m is odd and $g_{j+1}=0$ then
send (a_i, b_i, c_i, y_i) to the processor $(g_k g_{k-1} \cdots g_{j+2} 111 \cdots 1)$
received $(a_i, b_i, c_i, y_i, x_i)$ from the processor $(g_k g_{k-1} \cdots g_{j^2} 111 \cdots 1)$
endif
if m is even and $g_{j+1}=1$ then
send $(a_i, b_i, c_i, y_i, x_i)$ to the processor $(g_k g_{k-1} \cdots g_{j+2} 011 \cdots 1)$
receive (a_i, b_i, c_i, y_i) from the processor $(g_k g_{k-1} \cdots g_{j+2} 011 \cdots 1)$
endif
{Processors with m even receives x_{i-2^j} from processor $(G_{m-1}(k-j)11 \cdots 1)$, except if $m=0$. Processors with m even receives $x_{i^{2^j+1}}$ from processor $(G_{m+1}(k-j)11 \cdots 1)$, except if $m=2^{k-j}-2$}
if m is odd and $m<2^{k-j}-1$ then
send (x_i) to processor $(G_{m-1}11 \cdots 1)$
send (x_i) to processor $(G_{m+1}11 \cdots 1)$
endif
if m is even then
if $m>0$ then receive $(x_{i-2^{j+1}})$ from processor $(G_{m-1}(k-j)11 \cdots 1)$
if $m<2^{k-j}-2$ then receive $(x_{i+2^{j+1}})$ from processor $(G_{m+1}(k-j)11 \cdots 1)$
$x_i := (y_i - a_i x_{i-2^{j+1}} - c_i x_{i+2^{j+1}})/b_i$
endif
enddo

Acknowledgments. This report has benefited from many stimulating discussions with and suggestions by Stanley C. Eisenstat. Michael J. Fischer suggested the binary tree embedding in the shuffle-exchange networks. Andrea Pappas and Chris Hatchell helped with the manuscript.

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
[2] S. N. BHATT AND F. T. LEIGHTON, *A framework for solving* VLSI *graph layout problems*, J. Comput. System Sci., 28 (1984), pp. 300-343.
[3] R. P. BRENT AND H. T. KUNG, *On the area of binary tree layouts*, Inform. Process. Lett., 11 (1980), pp. 44-46.
[4] S. A. BROWNING, *The Tree Machine: A Highly Concurrent Computing Ennvironment*, Technical Report 1980:TR:3760, Computer Science, California Institute of Technology, Pasadena, CA, January 1980.
[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] S. C. EISENSTAT, M. H. SCHULTZ AND A. H. SHERMAN, *Applications of an element model for Gaussian elimination*, in Sparse Matrix Computations, Academic Press, New York, 1976, pp. 85-96.
[7] M. J. FLYNN, *Very high-speed computing systems*, Proc. of the IEEE, 12, 1966, pp. 1901-1909.
[8] S. FORTUNE AND J. WYLLIE, *Parallelism in random access machines*, Proc. 10th ACM STOC, 1978, pp. 114-118.
[9] D. GANNON AND J. VAN ROSENDALE, *On the impact of communication complexity in the design of parallel numerical algorithms*, IEEE Trans. Computers, C-33/12 December 1984, pp. 1180-1194.
[10] W. M. GENTLEMAN, *Some complexity results for matrix computations on parallel processors*, J. ACM, 25 (1978), pp. 112-115.
[11] A. GEORGE AND W. H. LIU, *Algorithms for matrix partitioning and the numerical solution of finite element systems*, SIAM J. Numer. Anal., 15 (1978), pp. 297-327.
[12] A. GEORGE AND J. W. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.
[13] A. GEORGE, *Nested dissection of a regular finite element grid*, SIAM J. Numer. Anal., 10 (1973), pp. 345-363.

[14] A. GOTTLIEB, R. GRISHMAN, C. P. KRUSKAL, K. P. MCAULIFFE, L. RUDOLPH AND M. SNIR, *The NYU ultracomputer—designing an MIMD shared memory parallel computer*, IEEE Trans. Computers, C-32/2 (1983), pp. 175-189.

[15] D. HELLER, D. K. STEVENSON AND J. F. TRAUB, *Accelerated iterative methods for the solution of tridiagonal linear systems on parallel computers*, J. ACM, 23 (1976), pp. 636-654.

[16] D. HELLER, *Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems*, SIAM J. Numer. Anal., 13 (1976), pp. 484-496.

[17] W. D. HILLIS, *The Connection Machine*, MIT Press, Cambridge, MA, 1985.

[18] R. W. HOCKNEY AND C. R. JESSHOPE, *Parallel Computers*, Adam Hilger, 1981.

[19] R. W. HOCKNEY, *A fast direct solution of Poisson's equation using Fourier analysis*, J. ACM, 12 (1965), pp. 95-113.

[20] ———, *The potential calculation and some applications*, Meth. Comput. Phys., 9 (1970), pp. 135-211.

[21] I. C. P. IPSEN, Y. SAD AND M. H. SCHULTZ, *Complexity of Dense Linear System Solution on a Multiprocessor Ring*, Technical Report RR YALEU/DCS/RR-349, Dept. of Computer Science, Yale University, New Haven, CT, January 1985.

[22] B. M. IRONS, *A frontal solution program for finite element analysis*, Internat. J. Numer. Meth. Engrg., 2 (1970), pp. 5-32.

[23] S. L. JOHNSSON, C.-T. HO AND F. SAIED, *Fast linear algebra routines on hypercubes*, in Parallel Processing and Medium Scale Multiprocessors, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1986.

[24] S. L. JOHNSSON, Y. SAAD AND M. H. SCHULTZ, *Alternating Direction Methods on Multiprocessors*, Technical Report YALEU/CSD/RR-382, Dept. of Computer Science, Yale University, New Haven, CT, August 1985. Commun. Appl. Numer. Meth., to appear.

[25] S. L. JOHSSON, *Gaussian Elimination on Sparse Matrices and Concurrency*, Technical Report 4087:TR:80, Caltech Computer Science Department, December 1980.

[26] ———, *Odd-Even Cyclic Reduction on Ensemble Architectures and the Solution Tridiagonal Systems of Equations*, Technical Report YALE/CSD/RR-339, Dept. of Computer Science, Yale University, New Haven, CT, October 1984.

[27] ———, *Cyclic reduction on a binary tree*, Comput. Phys. Comm., 37 (1985), pp. 195-203.

[28] ———, *Fast Banded Systems Solvers for Ensemble Architectures*, Technical Report YALEU/CSD/RR-379, Dept. of Computer Science, Yale University, New Haven, CT, March 1985.

[29] ———, *Solving Narrow Banded Systems on Ensemble Architectures*, ACM TOMS, 11/3 November (1985), pp. 271-288. Also available as Report YALEU/CSD/RR-418, November 1984.

[30] ———, *Data Permutations and Basic Linear Algebra Computations on Ensemble Architectures*, Technical Report YALEU/CSD/RR-367, Dept. of Computer Science, Yale University, New Haven, CT, February 1985.

[31] ———, *Band matrix systems solvers on ensemble architectures*, in Algorithms, Architecture, and the Future of Scientific Computation, University of Texas Press, 1985. Technical report YALEU/CSD/RR-388, Dept. of Computer Science, Yale University, New Haven, CT.

[32] ———, *Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures*, Technical Report YALEU/CSD/RR-361, Dept. of Computer Science, Yale University, New Haven, CT, January 1985.

[33] R. KAPUR AND J. C. BROWNE, *Block tridiagonal system solution on reconfigurable array computers*, in International Conference on Parallel Processing, IEEE Computer Society, 1981, pp. 92-99.

[34] D. Kershaw, *Solution of single tridiagonal linear systems and the vectorization of the* ICCG *Algorithm on the* CRAY-1, in Parallel Computations, Academic Press, New York, 1982, pp. 85-92.

[35] D. H. LAWRIE AND A. H. SAMEH, *The computational and communication complexity of a parallel banded system solver*, ACM Trans. Math. Software, 10 (1984), pp. 185-195.

[36] F. T. LEIGHTON, *Complexity Issues in* VLSI: *Optimal Layouts for the Shuffle-Exchange Graph and Other Networks*, MIT Press, Cambridge, MA, 1983.

[37] C. E. LEISERSON, *Area-Efficient* VLSI *Computation*, MIT Press, Cambridge, MA, 1982.

[38] P. LI AND L. JOHNSSON, *The Tree Machine: An evaluation of program loading strategies*, in 1983 International Conference on Parallel Processing, IEEE Computer Society, August 1983, pp. 202-205.

[39] RICHARD J. LIPTON, DONALD J. ROSE AND ROBERT ENDRE TARJAN, *Generalized nested dissection*, SIAM J. Numer. Anal., 16 (1979), pp. 346-358.

[40] S. PARTER, *The use of linear graphs in Gaussian elimination*, SIAM Rev., 3 (1961), pp. 119-130.

[41] M. S. PATERSON, W. L. RUZZO AND L. SNYDER, *Bounds on minimax edge length for complete binary trees*, Proc. of the 19th Annual Symposium on the Theory of Computing, ACM, 1981, pp. 293-299.

[42] C. S. PESKIN, *Ultracomputer Implementation of Odd-Even Cyclic Reduction*, Technical Report Ultracomputer note #19, Dept. of Computer Science, New York University, New York, January 1981.

[43] H. A. PRESNELL AND R. P. PARGAS, *Communication along shortest paths in a tree machine*, Proc. of the 1981 Conference on Functional Programming Languages and Computer Architecture, ACM, 1981, pp. 107-114.

[44] E. M. REINGOLD, J. NIEVERGELT AND N. DEO, *Combinatorial Algorithms*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

[45] E. REITER AND G. RODRIGUE, *An incomplete Cholesky factorization by a matrix partitioning algorithm*, in Elliptic Problem Solvers II, Academic Press, New York, 1983, pp. 161-174.

[46] D. J. ROSE, *A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations*, in Graph Theory and Computations, Academic Press, New York, 1973, pp. 183-217.

[47] A. L. ROSENBERG AND L. SNYDER, *Bounds on the costs of data encodings*, Math. Syst. Theory, 12 (1978), pp. 9-39.

[48] W. L. RUZZO AND L. SNYDER, *Minimum edge length planar embeddings of trees*, in VLSI Systems and Computations, Computer Sciences Press, 1981, pp. 119-123.

[49] A. H. SAMEH AND D. J. KUCK, *On stable parallel linear system solvers*, J. ACM, 25 (1978), pp. 81-91.

[50] J. T. SCHWARTZ, *Ultracomputers*, ACM Trans. on Programming Languages and Systems, 2 (1980), pp. 484-521.

[51] C. LUTZ, S. RABIN, C. L. SEITZ AND D. SPECK, *Design of the mosaic element*, Proc., Conference on Advanced Research in VLSI, Artech House, 1984, pp. 1-10.

[52] C. L. SEITZ, *Concurrent* VLSI *Architectures*, IEEE Trans. Comp., C-33/12 (1984), pp. 1247-1265.

[53] M. C. SEJNOWSKI, E. T. UPCHURCH, R. N. KAPUR, D. P. S. CHARLU AND G. J. LIPOVSKI, *An overview of the Texas reconfigurable array computer*, Proc. National Computer Conference, IEEE, 1980, pp. 631-641.

[54] M. SEKANINA, *On an ordering of the set of vertices of a connected graph*, Publication of the Faculty of Science of the University of Brno, 412 (1960), pp. 137-142.

[55] H. S. STONE, *Parallel processing with the perfect shuffle*, IEEE Trans. Comput., C-20 (1971), pp. 153-161.

[56] C. D. THOMPSON, *Area-time complexity for* VLSI, Proc. of the 11th ACM Symposium on the Theory of Computing, ACM, 1979, pp. 81-88.

[57] ———, *A Complexity Theory for* VSLI, Technical Report, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1980.

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

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 3, May 1987
© 1987 Society for Industrial and Applied Mathematics
012

INTERPOLATORY TENSION SPLINES WITH AUTOMATIC SELECTION OF TENSION FACTORS*

R. J. RENKA†

Abstract. A powerful and versatile method of shape-preserving interpolation is developed in terms of piecewise exponential functions with a tension factor associated with each interval. Knots coincide with data points, and the interpolant is formulated in terms of its values and first derivatives at these points. For a given set of derivatives, this enables the efficient computation of the minimum tension factor for which the interpolant satisfies locally defined properties such as monotonicity and convexity, as well as more general bounds on function values and derivatives, in each interval. A local derivative-estimation procedure results in a C^1 interpolant satisfying the constraints with minimum tension, and an iterative procedure can be used to obtain a C^2 spline fit which satisfies the constraints. Test results are presented which show both methods to produce visually pleasing interpolants to various data sets.

Key words. convexity-preserving, cubic splines, exponential splines, interpolation, local data, monotonicity-preserving, shape-preserving, splines under tension, tension factors

AMS(MOS) subject classifications. 65D05, 65D07

1. Introduction.

Given a strictly increasing sequence of abscissae $x_1 < x_2 < \cdots < x_n$ with associated data values $\{y_1, y_2, \cdots, y_n\}$, the interpolation problem is to find a function $f:[x_1, x_n] \to R$ such that

$$f(x_i) = y_i \quad \text{for } i = 1, \cdots, n.$$

The data are assumed to be sufficiently accurate that interpolation is not inappropriate. We shall use a Hermite formulation of the interpolant involving first derivatives at the abscissae $y_i' = f'(x_i)$ for $i = 1, \cdots, n$, and consider both local derivative-estimation procedures resulting in $f \in C^1[x_1, x_n]$, and a global procedure for $f \in C^2[x_1, x_n]$.

The purpose of shape-preserving interpolation methods is twofold: to construct a visually pleasing fit, free of extraneous inflection points, such as a draftsman might produce, and to satisfy constraints inherent in the data such as positivity and monotonicity as, for example, in the case of a probability density function.

In recent years several shape-preserving methods have been developed which make use of piecewise cubic interpolants. Local methods, such as those of Akima [1] and Fritsch and Carlson [6], are based on appropriate choice of derivatives at the abscissae, while the cubic spline methods of de Boor [3] and Pruess [14] employ additional knots between the abscissae. Although it often results in a satisfactory fit, Akima's method does not ensure that any data properties are preserved. The Fritsh–Carlson method preserves monotonicity; de Boor's method preserves convexity; and Pruess's method preserves both of these properties. McAllister and Roulier [11] and Schumaker [16] have developed piecewise quadratic interpolation methods which preserve both monotonicity and convexity by combining the techniques of choosing derivatives appropriately and using additional knots.

A third method of preserving shape properties is provided by tension splines, which can be made arbitrarily close to the piecewise linear interpolant while retaining two continuous derivatives. This method offers more versatility than polynomial interpolants, at the cost of more expensive evaluation in intervals for which the tension factor is nonzero.

* Received by the editors January 15, 1984; accepted for publication (in revised form) January 13, 1986.
† Department of Computer Sciences, North Texas State University, Denton, Texas 76203.

Tension splines with uniform tension were introduced by Schweikert [17] and generalized by Späth [18] to allow tension factors to vary with intervals. Cline [2] has developed software for both interpolation and smoothing using splines with uniform tension, and convergence rates and asymptotic behavior of interpolatory tension splines with variable tension have been analyzed by Pruess [12]. Späth [18] describes an iterative procedure for selecting tension factors to avoid extraneous inflection points, and Lynch [9] provides methods and software for selecting a tension factor to satisfy bounds on function values and derivatives, as well as preserving local convexity. The latter method applies only to the case of uniform tension, however, and thus generally results in far more tension than is necessary in some intervals.

Since continuity of second derivatives is not always necessary and C^1 interpolants often result in a more pleasing appearance, the method presented here generalizes tension splines to piecewise exponential or cubic functions with C^2 splines as an option corresponding to appropriate choice of derivatives at the abscissae. This approach also provides a basis for automatic selection of tension factors.

The interpolant is formulated in § 2. Methods for selecting tension factors are described in § 3. Section 4 is addressed to numerically stable evaluation procedures. Selection of nodal derivatives is discussed in § 5. Test results are presented in § 6.

2. Interpolant. The following is a generalized definition of a tension spline f. Given tension factors $\sigma_k \geqq 0$ associated with intervals $[x_k, x_{k+1}]$ of length $h_k = x_{k+1} - x_k$, define

$$f \in C^1[x_1, x_n] \quad \text{and} \quad f'''' - (\sigma_k/h_k)^2 f'' = 0 \text{ in } [x_k, x_{k+1}]$$

for $k = 1, \cdots, n-1$. This definition differs from those in the literature in two respects: the continuity restriction on f has been relaxed, and σ_k has been scaled by $1/h_k$. This normalization makes the interpolant independent of a scaling of the abscissae (as well as the data values). Note that, on $[x_k, x_{k+1}]$, $\sigma_k = 0$ implies f is cubic, and $\sigma_k > 0$ implies $(h_k/\sigma_k)^2 f'' - f$ is linear and f therefore approaches linear as σ_k increases.

In order to simplify the notation, we will restrict attention to the interval $[x_1, x_2]$ and omit the subscripts on σ and h. The following notation will be used throughout the remainder of this paper. Let y_1, y_2 and y_1', y_2' denote the data values and derivatives, respectively, associated with x_1, x_2, and define

$$h = x_2 - x_1, \quad b = (x_2 - x)/h, \quad s = (y_2 - y_1)/h, \quad d_1 = s - y_1', \quad d_2 = y_2' - s.$$

A convenient set of basis functions for the interpolant may be obtained from the following modified hyperbolic functions:

$$\text{sinhm}(z) = \sinh(z) - z \quad \text{and} \quad \text{coshm}(z) = \cosh(z) - 1.$$

Further, we define

$$E = \sigma * \sinh(\sigma) - 2\,\text{coshm}(\sigma) = \text{coshm}^2(\sigma) - \text{sinhm}(\sigma)\sinh(\sigma),$$

$$\alpha_1 = \sigma * \text{coshm}(\sigma) d_2 - \text{sinhm}(\sigma)(d_1 + d_2),$$

$$\alpha_2 = \sigma * \sinh(\sigma) d_2 - \text{coshm}(\sigma)(d_1 + d_2).$$

Note that $E > 0$ for $\sigma > 0$. This is verified by differentiating $E(\sigma)$ with respect to σ: $E'(\sigma) = \sigma * \cosh(\sigma) - \sinh(\sigma)$ and $E''(\sigma) = \sigma * \sinh(\sigma)$. Thus, $E(0) = E'(0) = 0$ and $E''(\sigma) > 0$ for $\sigma > 0$. For $x \in [x_1, x_2]$ and $\sigma > 0$, the interpolant and its derivatives are

given by the expressions

$$f(x) = y_2 - y_2' hb + h/(\sigma E)[\alpha_1 \operatorname{coshm}(\sigma b) - \alpha_2 \operatorname{sinhm}(\sigma b)],$$
$$f'(x) = y_2' - (1/E)[\alpha_1 \sinh(\sigma b) - \alpha_2 \operatorname{coshm}(\sigma b)],$$
$$f''(x) = \sigma/(hE)[\alpha_1 \cosh(\sigma b) - \alpha_2 \sinh(\sigma b)].$$

It is a straightforward exercise to verify that f interpolates the specified values and derivatives and satisfies the differential equation in the definition above. Continuity of f and f' across interval boundaries follows from the interpolatory conditions.

For the case $\sigma = 0$, expressions may be obtained from the standard Hermite cubic interpolation formulas. Alternatively, we may compute limits of the above expressions from the Taylor series expansions $\sinh(z) = z + z^3/6 + O(z^5)$ and $\cosh(z) = 1 + z^2/2 + z^4/24 + O(z^6)$. Thus, for $x \in [x_1, x_2]$, letting f_0 denote the interpolant with $\sigma = 0$,

$$f_0(x) = \lim_{\sigma \to 0} \{y_2 - y_2' hb + h[(\alpha_1/\sigma^3) \operatorname{coshm}(\sigma b)/\sigma^2 - (\alpha_2/\sigma^2) \operatorname{sinhm}(\sigma b)/\sigma^3]/(E/\sigma^4)\}$$
$$= y_2 - y_2' hb + h[((2d_2 - d_1)/6)(b^2/2) - ((d_2 - d_1)/2)(b^3/6)]/(1/12)$$
$$= y_2 - h[y_2' b + (d_1 - 2d_2)b^2 + (d_2 - d_1)b^3]$$

and

$$f_0'(x) = y_2' + 2(d_1 - 2d_2)b + 3(d_2 - d_1)b^2.$$

While a more standard approach to formulating the interpolant involves two local coordinates, b and $1 - b$, the expressions chosen here are more convenient for the analysis to follow.

3. Selection of tension factors. This section is addressed to methods for computing the minimum tension necessary to satisfy a constraint in an interval for which data values and derivatives are given at the endpoints. In the case of C^2 interpolation, tension factors can be chosen by alternating between computation of nodal derivatives and the procedures discussed below. This is further discussed in § 5. The three types of constraints treated are (i) bounds on function values, (ii) bounds on first derivative values, with monotonicity treated as a special case, and (iii) convexity (or concavity). All bounds are taken to be strict.

For each type of constraint there are three possibilities:

(a) the constraint cannot be satisfied,

(b) the constraint is satisfied by the cubic ($\sigma = 0$),

(c) $\sigma > 0$ is necessary and sufficient to satisfy the constraint.

The methods consist of testing for the three situations in the order specified, and computing the minimum necessary tension factor in case (c).

We first consider conditions under which the constraint can be satisfied. The following is a restatement of Theorem 3 from Pruess [12]:

$$\lim_{\sigma \to \infty} f(x) = by_1 + (1 - b)y_2$$

uniformly in $[x_1, x_2]$, and

$$\lim_{\sigma \to \infty} f'(x) = s \quad \text{and} \quad \lim_{\sigma \to \infty} f''(x) = 0$$

uniformly in any proper closed subinterval of $[x_1, x_2]$. Since $f'(x_1) = y_1'$ and $f'(x_2) = y_2'$, f' is not continuous in the limit. From the expressions

$$f''(x_1) = \sigma/(hE)[\sigma * \operatorname{coshm}(\sigma) d_1 - \operatorname{sinhm}(\sigma)(d_1 + d_2)],$$
$$f''(x_2) = \sigma/(hE)[\sigma * \operatorname{coshm}(\sigma) d_2 - \operatorname{sinhm}(\sigma)(d_1 + d_2)],$$

it can be shown that, excluding the case of f_0 linear ($d_1 = d_2 = 0$), the magnitudes of $f''(x_1)$ and $f''(x_2)$ increase without bound and have the signs of d_1 and d_2, respectively, for σ sufficiently large ($\sigma^2 \operatorname{coshm}(\sigma)/(hE) \to \infty$ and $\sigma * \operatorname{sinhm}(\sigma)/(hE) \to (1/h)$ as $\sigma \to \infty$). Thus, letting β denote the bound defining a constraint, the constraint can be satisfied if and only if β is a lower (upper) bound on the set Q where $Q = \{y_1, y_2\}$ for a bound on f, $Q = \{y_1', y_2', s\}$ for a bound on f', and $Q = \{d_1, d_2\}$ for a bound on $f''(\beta = 0)$.

We next present theorems concerning the location of extrema of the Hermite cubic interpolant f_0 and its first derivative. The condition required for convexity of f_0 is treated in § 3.3. The following notation will be used:

$$T_0 = 3s - y_1' - y_2',$$

$$D_0 = T_0^2 - y_1'y_2' = B_0^2 - y_2'A_0 = C_0^2 - y_1'A_0,$$

$$C_0 = T_0 - y_1' = 3s - 2y_1' - y_2',$$

$$B_0 = T_0 - y_2' = 3s - y_1' - 2y_2',$$

$$A_0 = -B_0 - C_0 = 3(y_1' + y_2' - 2s).$$

Then, for $x \in [x_1, x_2]$,

$$f_0(x) = y_2 - h[y_2'b + B_0b^2 + (A_0/3)b^3],$$

$$f_0'(x) = y_2' + 2B_0b + A_0b^2,$$

$$f_0''(x) = -(2/h)(B_0 + A_0b),$$

$$f_0'''(x) = 2A_0/h^2.$$

In the remainder of this section, f and its derivatives will be treated as functions of the local variable $b \in [0, 1]$; i.e., $f_0'(b)$ denotes the derivative of f_0 with respect to x evaluated at $b = (x_2 - x)/h$.

The following theorem provides an algorithm for determining whether a bound on f is satisfied by f_0.

THEOREM 1. (a) *If $y_1' \geqq 0$ and $y_2' \leqq 0$, then the minimum of f_0 occurs at an endpoint of the interval. If $y_1' \leqq 0$ and $y_2' \geqq 0$, then the maximum of f_0 occurs at an endpoint of the interval.*

(b) *Either f_0' is identically zero or f_0' has no zero in the open interval $(0, 1)$ (and thus both extrema of f_0 occur at the endpoints of the interval) if and only if $y_1'y_2' \geqq 0$ and either $D_0 < 0$ or $(y_1' + y_2')T_0 \geqq 0$.*

(c) *If $A_0 = 0$ and $y_1'y_2' < 0$, the local extremum of f_0 is $f_0(b^*) = y_2 + hy_2'^2/(4B_0)$ at $b^* = -y_2'/(2B_0)$. If $A_0 \neq 0$, the local extrema, if they exist, are given by $f_0(b^*) = y_2 + h(y_2'B_0 + 2D_0b^*)/(3A_0)$, where the local minimum corresponds to $b^* = (-B_0 - \sqrt{D_0})/A_0 = y_2'/(-B_0 + \sqrt{D_0})$, and the maximum occurs at $b^* = (-B_0 + \sqrt{D_0})/A_0 = y_2'/(-B_0 - \sqrt{D_0})$. (In the computational procedure, the expression for b^* is chosen to avoid cancellation error.)*

Proof. Part (a) follows from the fact that f_0' is quadratic and therefore has at most two zeros in the interval, and part (c) is readily verified from expressions for the zeros of f_0' and the sign of f_0''. To prove part (b), we first consider the case $A_0 = 0$. Since $y_1' + y_2' = 2s$, $(y_1' + y_2')T_0 = 2s^2 \geqq 0$, and it suffices to show that f_0' has no zero in $(0, 1)$, or f_0 is constant, if and only if $y_1'y_2' \geqq 0$. If $B_0 = 0$, then f_0 is linear and $y_1'y_2' = s^2 \geqq 0$. For $B_0 \neq 0$, f_0' has a zero at $b^* = -y_2'/(2B_0)$ and $b^* \in (0, 1)$ if and only if $b^*(1 - b^*) = [-y_2'/(2B_0)][y_1'/(2B_0)] = -y_1'y_2'/(4B_0^2) > 0$. The result follows.

For $A_0 \neq 0$, $b^* \in (0,1)$ if and only if $D_0 \geqq 0$ and $b^*(1-b^*) = (-B_0 \pm \sqrt{D_0}) * (A_0 + B_0 \mp \sqrt{D_0})/A_0^2 = (-B_0 \pm \sqrt{D_0})(-C_0 \mp \sqrt{D_0})/A_0^2 = (B_0C_0 - D_0 \pm (B_0 - C_0)\sqrt{D_0})/A_0^2 > 0$. Thus, f_0 has a local extremum or inflection point in the open interval if and only if $D_0 \geqq 0$ and $B_0C_0 - D_0 > -|B_0 - C_0|\sqrt{D_0}$. Since $D_0 < 0$ implies $y_1'y_2' = T_0^2 - D_0 > 0$, the theorem is true in that case, and we may assume $D_0 \geqq 0$. Note that $(B_0C_0 - D_0)^2 - (B_0 - C_0)^2 D_0 = B_0^2C_0^2 + D_0^2 - (B_0^2 + C_0^2)D_0 = (D_0 + y_2'A_0)(D_0 + y_1'A_0) + D_0^2 - (2D_0 + y_2'A_0 + y_1'A_0)D_0 = y_1'y_2'A_0^2$, and thus $y_1'y_2' < 0$ and only if $(B_0C_0 - D_0)^2 < (B_0 - C_0)^2 D_0$; i.e., $-|B_0 - C_0|\sqrt{D_0} < B_0C_0 - D_0 < |B_0 - C_0|\sqrt{D_0}$. The theorem is therefore true for $y_1'y_2' < 0$, and it suffices to show that $B_0C_0 - D_0 > -|B_0 - C_0|\sqrt{D_0}$ if and only if $(y_1' + y_2')T_0 < 0$ where $y_1'y_2' \geqq 0$ and $D_0 \geqq 0$. If $(y_1' + y_2')T_0 < 0$, then $B_0C_0 - D_0 = (T_0 - y_2')(T_0 - y_1') - T_0^2 + y_1'y_2' = 2y_1'y_2' - (y_1' + y_2')T_0 > 0$ and hence $B_0C_0 - D_0 > -|B_0 - C_0|\sqrt{D_0}$.

Conversely, suppose $(y_1' + y_2')T_0 \geqq 0$. Note that

$$
\begin{aligned}
&(y_1' + y_2')^2 T_0^2 - (2y_1'y_2' - |y_1' - y_2'|\sqrt{D_0})^2 \\
&\quad = (y_1' + y_2')^2 T_0^2 - 4y_1'^2 y_2'^2 + 4y_1'y_2'|y_1' - y_2'|\sqrt{D_0} - (y_1' - y_2')^2 (T_0^2 - y_1'y_2') \\
&\quad = y_1'y_2'[2T_0^2 - 4y_1'y_2' + 4|y_1' - y_2'|\sqrt{D_0} + 2T_0^2 + (y_1' - y_2')^2] \\
&\quad = y_1'y_2'[4D_0 + 4|y_1' - y_2'|\sqrt{D_0} + (y_1' - y_2')^2] \geqq 0.
\end{aligned}
$$

Hence, $2y_1'y_2' - |y_1' - y_2'|\sqrt{D_0} \leqq (y_1' + y_2')T_0$, and thus $B_0C_0 - D_0 = 2y_1'y_2' - (y_1' + y_2')T_0 \leqq |y_1' - y_2'|\sqrt{D_0} = |B_0 - C_0|\sqrt{D_0}$. Also, if $y_1'y_2' = 0$, then $B_0C_0 - D_0 = -(y_1' + y_2')T_0 = |B_0 - C_0|\sqrt{D_0} = 0$ since $-(y_1' + y_2')T_0 \leqq 0$ and $|B_0 - C_0|\sqrt{D_0} \geqq 0$. If $y_1'y_2' > 0$ then, from the previous paragraph, $(B_0C_0 - D_0)^2 > (B_0 - C_0)^2 D_0$, and thus $B_0C_0 - D_0 < -|B_0 - C_0|\sqrt{D_0}$ since $B_0C_0 - D_0 \leqq |B_0 - C_0|\sqrt{D_0}$. In either case, we have $B_0C_0 - D_0 \leqq -|B_0 - C_0|\sqrt{D_0}$, and that concludes the proof. □

The following theorem is similar to the lemmas proved by Fritsch and Carlson [6], but provides for a more convenient computational procedure to ascertain the monotonicity of f_0.

THEOREM 2. (a) *f_0' has a local minimum ($A_0 > 0$) or maximum ($A_0 < 0$) given by $f_0'(b^*) = -D_0/A_0$ at $b^* = -B_0/A_0 \in (0,1)$ if and only if $B_0C_0 > 0$.*

(b) *$B_0C_0 > 0$ if and only if $A_0B_0 < 0$ and $A_0C_0 < 0$ (and thus $B_0C_0 > 0$ implies $A_0 \neq 0$).*

(c) *If the data is nondecreasing (y_1', y_2', and s nonnegative and $s = 0 \Rightarrow y_1' = y_2' = 0$), then $b^* \in (0,1)$ and $f_0'(b^*) < 0$ (the cubic interpolant is not monotone) if and only if $D_0 > 0$ and $T_0 < 0$.*

(d) *If the data is nonincreasing, then $b^* \in (0,1)$ and $f_0'(b^*) > 0$ if and only if $D_0 > 0$ and $T_0 > 0$.*

Proof. (a) For $A_0 \neq 0$, $f_0''(b^*) = 0$, f_0''' has the sign of A_0, and, since $b^*(1-b^*) = B_0C_0/A_0^2$, $b^* \in (0,1)$ if and only if $B_0C_0 > 0$. (Alternatively, $f_0''(0)f_0''(1) < 0$ if and only if $B_0C_0 > 0$.)

(b) $B_0C_0 = (-A_0 - C_0)C_0 = -A_0C_0 - C_0^2 = B_0(-A_0 - B_0) = -A_0B_0 - B_0^2 > 0 \Rightarrow A_0B_0 < -B_0^2$ and $A_0C_0 < -C_0^2 \Rightarrow A_0B_0 < 0$ and $A_0C_0 < 0$. Conversely, A_0B_0 and A_0C_0 both negative imply $A_0 \neq 0$ and B_0 and C_0 both have the sign opposite that of A_0.

(c) Suppose $f_0'(b^*) = -D_0/A_0 < 0$ for $b^* \in (0,1)$. Then, since $y_1' \geqq 0$ and $y_2' \geqq 0$, $f_0'(b^*)$ is a local minimum and hence $A_0 > 0$, $D_0 > 0$, $B_0 < 0$, and $C_0 < 0$ from results (a) and (b). Also, $B_0 < 0 \Rightarrow T_0 < y_2'$, $C_0 < 0 \Rightarrow T_0 < y_1'$, and $D_0 > 0 \Rightarrow T_0^2 > y_1'y_2'$. It follows that $T_0 < 0$. Conversely, suppose $D_0 > 0$ and $T_0 < 0$. Then, since $s \geqq 0$, $y_1' + y_2' > 3s \geqq 2s$ and thus $A_0 > 0$. Also, $B_0 = T_0 - y_2' < 0$ and $C_0 = T_0 - y_1' < 0$. Hence $b^* \in (0,1)$ and $f_0'(b^*) = -D_0/A_0 < 0$.

(d) The proof of the last result is analogous to the proof of part (c). □

We now consider the case $\sigma > 0$. In addition to the quantities α_1, α_2, and E introduced in § 2, we define the following functions of σ:

$$A = \alpha_2 - \alpha_1, \qquad C = \alpha_2 + \alpha_1,$$
$$B = Ey_2' - \alpha_2, \qquad D = B^2 - AC.$$

It should be noted that the computation of these expressions must be carefully arranged in order to avoid cancellation error and overflow. Thus, as in the evaluation expressions described in § 4, α_1, α_2, and E (and hence the variables above) are scaled by $2e^{-\sigma}$ when σ is large. Note that the expressions derived below involve ratios of these quantities and are thus independent of the scaling. Also, cancellation error is eliminated in the evaluation of A so that A is generally nonzero even when α_1 and α_2 are computationally identical.

3.1. Bound on f. We first treat the case of a bound β on f and assume that $\sigma > 0$ is necessary and sufficient to satisfy the constraint. It follows that the relevant extremum of f_0 occurs in the open interval. Since

$$\begin{aligned} 2E\,e^{\sigma b}f'(x) &= 2E\,e^{\sigma b}y_2' - 2\,e^{\sigma b}[\alpha_1 \sinh(\sigma b) - \alpha_2 \cosh(\sigma b) + \alpha_2] \\ &= 2\,e^{\sigma b}B + A\,e^{2\sigma b} + C, \end{aligned}$$

the zeros of f', provided they exist, are defined by

$$\exp(\sigma b^*) = \begin{cases} (-B \pm \sqrt{D})/A = C/(-B \mp \sqrt{D}) & \text{if } A \neq 0 \text{ and } D \geqq 0, \\ C/(-2B) & \text{if } A = 0 \text{ and } B \neq 0, \end{cases}$$

and, for $b^* \in (0, 1)$, the local extrema of f are given by

$$\begin{aligned} f(b^*) &= y_2 - y_2' h b^* + h/(\sigma E)[-(A/2)\exp(\sigma b^*) + (C/2)\exp(-\sigma b^*) - \alpha_1 + \sigma b^* \alpha_2] \\ &= y_2 - h/(\sigma E)[\sigma b^*(Ey_2' - \alpha_2) + (-B \pm \sqrt{D})/2 - (-B \mp \sqrt{D})/2 + \alpha_1] \\ &= y_2 - h/(\sigma E)[\sigma b^* B \pm \sqrt{D} + \alpha_1]. \end{aligned}$$

Also,

$$\begin{aligned} f''(b^*) &= \sigma/(hE)[\alpha_1 \cosh(\sigma b^*) - \alpha_2 \sinh(\sigma b^*)] \\ &= \sigma/(hE)[-(A/2)\exp(\sigma b^*) + (C/2)\exp(-\sigma b^*)] \\ &= \begin{cases} -\sigma/(hE)(\pm\sqrt{D}) & \text{if } A \neq 0 \text{ and } D \geqq 0, \\ -\sigma/(hE)B & \text{if } A = 0 \text{ and } B \neq 0. \end{cases} \end{aligned}$$

Define $SGN = -1$ if β is a lower bound, and $SGN = 1$ if β is an upper bound. Then, the relevant extremum corresponds to $SGN * f''(b^*) < 0$, and for $D > 0$, the appropriate value of $\exp(\sigma b^*)$ may be computed without cancellation error as

$$R = \begin{cases} T/A & \text{if } SGN * B < 0 \text{ and } A \neq 0, \\ C/T & \text{if } SGN * B \geqq 0 \text{ or } A = 0, \end{cases}$$

for

$$T = -B - \operatorname{sign}(B)\sqrt{D} \text{ where } \operatorname{sign}(B) \text{ is arbitrary if } B = 0.$$

Note that, since $D > 0$, A and B are not both zero and $T \neq 0$. Also, if $A = 0$, then $SGN = \operatorname{sign}(B)$ and $\operatorname{sign}(B)\sqrt{D} = B$. For $R \in (1, e^{\sigma})$, the corresponding extremum is

$$f(b^*) = y_2 - h/(\sigma E)[B * \ln(R) + SGN\sqrt{D} + \alpha_1],$$

and, if $D \leqq 0$ or $R \notin (1, e^{\sigma})$, the extremum occurs at an endpoint of the interval.

Let σ^c denote the largest value of σ for which $D>0$ and $R\in(1, e^\sigma)$, and define

$$g(\sigma)=\begin{cases} SGN\,[\beta-f(b^*)] & \text{if } 0\leqq\sigma\leqq\sigma^c, \\ g_{\max} & \text{if } \sigma>\sigma^c, \end{cases}$$

where

$$g_{\max}=\begin{cases} \min(y_1, y_2)-\beta & \text{if } SGN=-1, \\ \beta-\max(y_1, y_2) & \text{if } SGN=1. \end{cases}$$

Then $g(\sigma)$ is a continuous function which satisfies $g(0)<0$, by the assumption that nonzero tension is necessary to satisfy the constraint, and $g(\sigma^c)>0$ by the assumption that the constraint can be satisfied. The first occurrence of a zero $\sigma^*\in(0, \sigma^c)$ is the minimum tension required to satisfy the constraint. Intuition backed by numerical evidence indicates that g is strictly increasing on $[0, \sigma^c]$, and thus has only one zero. A straightforward approach to proving this statement is extremely tedious and, since no alternative approach has been found, it remains an open question. Unless g can be shown to be monotonic, the method described below must be modified to ensure that the smallest zero is located.

A procedure found to be effective for computing the zero of g combines the secant and false position methods. Initial estimates are taken to be $\sigma=0$ and σ sufficiently large that $g(\sigma)>0$. These define an interval I containing σ^*. At each iteration, the new estimate is obtained by linear interpolation applied to either the two previous estimates or the endpoints of I, and the bracketing interval I is updated. The two previous estimates are used if and only if the new estimate is contained in I. Convergence is defined by a bound on either $|g(\sigma)|$ or the relative change in σ. In a large number of tests conducted on a DEC PDP-10 with a tolerance of 100ε, for machine precision ε (approximately 10^{-8}), convergence was found to require three to eight evaluations of g.

3.2. Bound on f'. We next consider a bound β on f' and again assume that nonzero tension is necessary and sufficient to satisfy the bound. For $\sigma>0$,

$$f'(b)=y_2'-(1/E)[\alpha_1\sinh(\sigma b)-\alpha_2\,\mathrm{coshm}(\sigma b)],$$
$$f''(b)=\sigma/(hE)[\alpha_1\cosh(\sigma b)-\alpha_2\sinh(\sigma b)],$$
$$f'''(b)=-\sigma^2/(h^2E)[\alpha_1\sinh(\sigma b)-\alpha_2\cosh(\sigma b)].$$

In order to locate the extrema of f', define

$$\begin{aligned}\gamma&=[(hE/\sigma)f''(0)][(hE/\sigma)f''(1)]\\&=\alpha_1[\alpha_1\cosh(\sigma)-\alpha_2\sinh(\sigma)]\\&=\alpha_1[\sigma*\mathrm{coshm}(\sigma)d_1-\mathrm{sinhm}(\sigma)(d_1+d_2)].\end{aligned}$$

Then, since f'' is continuous and has at most one zero in the closed interval, f' has an extremum at $b^*\in(0, 1)$ if and only if f'' changes sign; i.e., if and only if $\gamma<0$. In this case, it is easily verified that $AC>0$ and the extremum is defined by

$$\exp(\sigma b^*)=\sqrt{C/A}.$$

Thus, for $\gamma<0$,

$$\begin{aligned}f'(b^*)&=-(1/E)[\alpha_2-y_2'E-(A/2)\exp(\sigma b^*)-(C/2)\exp(-\sigma b^*)]\\&=(1/E)[B+\mathrm{sign}(A)\sqrt{AC}/2+\mathrm{sign}(C)\sqrt{AC}/2]\\&=(1/E)[B+\mathrm{sign}(A)\sqrt{AC}],\end{aligned}$$

and

$$f'''(b^*) = \sigma^2/(h^2E)[\text{sign}\,(A)\sqrt{AC}].$$

As in the previous subsection, we define

$$g(\sigma) = \begin{cases} SGN\,[\beta - (B - SGN\,\sqrt{AC})/E] & \text{if } \gamma < 0, \\ SGN\,(\beta - s) & \text{if } \gamma \geqq 0, \end{cases}$$

where $SGN = -\text{sign}\,(A)$. Note that $SGN = -1$ if β is a lower bound. By the assumption that $\sigma > 0$ is necessary and sufficient to satisfy the bound, the minimum value of f (for sufficiently small tension) must occur in the open interval. Since f' can have only one local extremum, $f'''(b^*) > 0$ and $A > 0$. Similarly, $SGN = 1$ if β is an upper bound. Note also that $g(0) < 0$ and $SGN\,(\beta - s) > 0$ by assumption. An initial estimate is obtained by analyzing $f'(b^*)$ for large σ. If 2σ is insignificant relative to e^σ, then the (modified) hyperbolic functions are computationally identical to $e^\sigma/2$, α_1 and α_2 are approximately equal to $(e^\sigma/2)(\sigma d_2 - d_1 - d_2)$, and E is approximately $(\sigma - 2)\,e^\sigma/2$. Thus, as σ increases, $f'(b^*)$ approaches $y_2' - (\sigma d_2 - d_1 - d_2)/(\sigma - 2) = (\sigma s - y_1' - y_2')/(\sigma - 2)$. (Note that this expression has limit s, as expected.) Let $\bar{g}(\sigma) = SGN\,[\beta - (\sigma s - y_1' - y_2')/(\sigma - 2)]$. Then it can be shown that, for $\sigma > 2$, $\bar{g}(\sigma) < g(\sigma)$ and thus $g(\sigma) > 0$ for $\sigma = (2\beta - y_1' - y_2')/(\beta - s) = 2 - A_0/(3(\beta - s))$—the zero of $\bar{g}$. This value provides the right end of the bracketing interval for the zero-finding method, and is in fact the zero of g if it is sufficiently large. Convergence required at most seven function evaluations in our tests.

3.3. Convexity.

THEOREM 3. *If $d_1 d_2 > 0$, then f is convex (concave for $d_1 < 0$ and $d_2 < 0$) if and only if $g(\sigma) > 0$ where*

$$g(\sigma) = \begin{cases} \sigma * \text{coshm}\,(\sigma)/\text{sinhm}\,(\sigma) - 1 - \max\{d_1/d_2, d_2/d_1\} & \text{if } \sigma > 0, \\ 2 - \max\{d_1/d_2, d_2/d_1\} & \text{if } \sigma = 0. \end{cases}$$

(*Note that $\sigma * \text{coshm}\,(\sigma)/\text{sinhm}\,(\sigma)$ has limit 3 as σ approaches zero.*) *Furthermore, g is strictly increasing and therefore positive for σ sufficiently large.*

Proof. Suppose f''' has a zero at $b^* \in (0, 1)$. Then $AC < 0$, $\exp(\sigma b^*) = \sqrt{-C/A}$, $f''(b^*) = -\sigma/(hE)[\text{sign}\,(A)\sqrt{-AC}]$, and $f''''(b^*)$ has the same sign as $f''(b^*)$. Thus, the minimum of f'' over $b \in [0, 1]$ is positive for $A < 0$ and the maximum of f'' is negative for $A > 0$; i.e., f is convex. It is therefore sufficient to show that $g(\sigma) > 0$ if and only if the endpoint values of f'' have the same sign.

Suppose d_1 and d_2 are positive. For $\sigma = 0$, the values of f'' at the endpoints are positive if and only if $B_0 < 0$ and $C_0 > 0$, or equivalently, $d_1 < 2d_2$ and $d_2 < 2d_1$. For $\sigma > 0$, $f''(0) = \sigma\alpha_1/(hE) > 0$ if and only if $\sigma * \text{coshm}\,(\sigma)/\text{sinhm}\,(\sigma) > (d_1 + d_2)/d_2 = 1 + d_1/d_2$, and $f''(1) = \sigma/(hE)[\sigma * \text{coshm}\,(\sigma) d_1 - \text{sinhm}\,(\sigma)(d_1 + d_2)] > 0$ if and only if $\sigma * \text{coshm}\,(\sigma)/\text{sinhm}\,(\sigma) > 1 + d_2/d_1$. Similarly, if d_1 and d_2 are negative, then the endpoint values of f'' are negative if and only if $g(\sigma) > 0$.

The derivative of g is given by

$$\begin{aligned} g'(\sigma) &= \{\text{sinhm}\,(\sigma)[\sigma * \sinh(\sigma) + \text{coshm}\,(\sigma)] - \sigma * \text{coshm}^2(\sigma)\}/\text{sinhm}^2(\sigma) \\ &= [\text{coshm}\,(\sigma)\,\text{sinhm}\,(\sigma) - \sigma E]/\text{sinhm}^2(\sigma). \end{aligned}$$

Thus

$$\begin{aligned} \text{sinhm}^2(\sigma) g'(\sigma) &= \cosh(\sigma)\sinh(\sigma) + \sigma * \text{coshm}\,(\sigma) - \sinh(\sigma) - \sigma^2 * \sinh(\sigma) \\ &= .5 * \sinh(2\sigma) + \sigma * \text{coshm}\,(\sigma) - \sinh(\sigma) - \sigma^2 * \sinh(\sigma) \end{aligned}$$

$$= \sum_{\substack{k=3 \\ k \text{ odd}}}^{\infty} (\sigma^k/k!)[2^{k-1} - k(k-2) - 1] > 0,$$

and g is therefore strictly increasing. □

In order to find the minimum tension required for convexity when $g(0) < 0$, Newton's method is used to compute the zero of g. An expression for $g'(\sigma)$ is contained in the proof above and, omitting the details,

$$g''(\sigma) = [\sigma * \text{coshm}(\sigma)/\text{sinhm}(\sigma) - 1]$$
$$\times [\sigma * \text{sinhm}(\sigma) - 4\,\text{coshm}(\sigma) + 2\sigma^2]/\text{sinhm}^2(\sigma).$$

Taking limits as σ approaches zero, we have $g'(0) = 0$ and $g''(0) = .2$. Thus, an initial estimate is obtained from the quadratic approximation

$$g(0) + \sigma g'(0) + \sigma^2 g''(0)/2 = 0 \Rightarrow \sigma = \sqrt{-2g(0)/g''(0)} = \sqrt{-10g(0)}.$$

Convergence to six or more decimal places required only one to three iterations in all cases tested.

4. Evaluation expressions. This section is addressed to numerically stable expressions for evaluating the interpolant, its derivatives, and its integral over an arbitrary domain. As noted in the previous section, similar expressions must be used in the computation of tension factors. A straightforward computation of the modified hyperbolic functions $\text{sinhm}(z)$ and $\text{coshm}(z)$ is subject to failure both in the case of small z, due to cancellation error, and with large values of z, due to overflow. Cancellation error occurs because $\sinh(z)$ approaches z and $\cosh(z)$ approaches 1 as z approaches 0, and overflow results from computing e^z when z is large. With the exception of the recent work of Rentrop [15], these problems have not been adequately treated in the literature. Rentrop's algorithm is not directly applicable to our formulation of the interpolant, but we make use of its essential features—polynomial approximations for a small tension factor σ, and scaling by $e^{-\sigma}$ for large σ. It is assumed that underflow results in assignment of the value zero.

Following Rentrop, $\text{sinhm}(z)$ and $\text{coshm}(z)$, $0 \leqq z \leqq .5$, are computed from the approximation

$$\sinh(z) = zP(z^2)$$

where

$$P(z) = 1 + c_1 z + c_2 z^2 + c_3 z^3 + c_4 z^4$$

for coefficients to be defined below. Then, for $0 \leqq z \leqq .5$ and

$$Q(z) = (P(z) - 1)/z = c_1 + c_2 z + c_3 z^2 + c_4 z^3,$$
$$\text{sinhm}(z) = \sinh(z) - z \cong zP(z^2) - z = z^3 Q(z^2)$$

and

$$\text{coshm}(z) = 2\sinh^2(z/2) \cong (z^2/2)P(z^2/4)^2.$$

In order to avoid cancellation error in the evaluation of integrals, we require an accurate approximation to the additional function

$$\text{coshmm}(z) = \text{coshm}(z) - z^2/2.$$

Letting $w = z^2/4$, $0 \leqq z \leqq .5$, we have

$$\begin{aligned}\text{coshmm}\,(z) &\cong 2w[P(w)^2 - 1] \\ &= 2w[(wQ(w)+1)^2 - 1] \\ &= 2w^2 Q(w)[wQ(w)+2].\end{aligned}$$

Rentrop defines $P(z)$ in terms of truncated coefficients from the formula *SINH* 1984 from Hart et al. [7]. This is the best fourth degree polynomial approximation to $\sinh(z)$ on $[0, .5]$ but is not optimal for the modified hyperbolic functions. We determined optimal coefficients by fitting $z^3 Q(z^2)$ to $\text{sinhm}\,(z)$ on a set of 10,000 equally spaced points in $[0, .5]$ in the uniform norm. Accurate values of $\text{sinhm}\,(z)$ were obtained from the sixth degree polynomial approximation *SINH* 1986 from Hart et al. The results are as follows:

$$c_1 = .16666\ 66666\ 659\ \text{E0}, \qquad c_3 = .19841\ 07350\ 948\ \text{E}-3,$$

$$c_2 = .83333\ 33431\ 546\ \text{E}-2, \qquad c_4 = .27682\ 86868\ 175\ \text{E}-5.$$

Using these coefficients, the maximum relative errors in $\text{coshm}\,(z)$, $\text{sinhm}\,(z)$, and $\text{coshmm}\,(z)$ were found to be approximately 6.47E−14, 4.60E−12, and 4.62E−12, respectively, on a machine with sufficient word length. Using *SINH* 1984 with constant term $c_0 = 1$, the maximum errors were found to be 1.88E−13, 5.68E−11, and 5.68E−11, respectively. Note that $c_0 = 1$ is necessary in order that the relative errors in $\text{sinhm}\,(z)$ and $\text{coshmm}\,(z)$ are bounded for $z = 0$.

In the remainder of this section, we list evaluation expressions chosen for computational efficiency within the constraint of numerical stability. For a large number of evaluation points, efficiency can be increased by a pre-evaluation phase in which various quantities which are independent of the evaluation point are computed and stored in arrays. It should be noted that in many applications the evaluation time is dominated by the time required to locate the interval containing the evaluation point. For $x \in [x_j, x_k]$, $k > j$, an estimate of i such that $x \in [x_i, x_{i+1}]$ can be obtained by fitting a linear function to the points (x_j, j) and (x_k, k). This provides the basis for an efficient search technique which is particularly effective when the mesh spacing is close to uniform.

We restrict attention to the interval $[x_1, x_2]$ and define

$$y_1 = f(x_1), \quad y_2 = f(x_2), \quad y_1' = f'(x_1), \quad y_2' = f'(x_2), \quad h = x_2 - x_1,$$

$$s = (y_2 - y_1)/h, \quad d_1 = s - y_1', \quad d_2 = y_2' - s, \quad u_1 = x_2 - x, \quad b_1 = u_1/h,$$

$$u_2 = x - x_1 \quad \text{and} \quad b_2 = u_2/h.$$

1. For $\sigma = 0$,

$$f(x) = y_1 + u_2\{y_1' + b_2[d_1 + b_1(d_1 - d_2)]\},$$

$$f'(x) = y_1' + b_2[d_1 + d_2 - 3b_1(d_2 - d_1)],$$

$$f''(x) = [d_1 + d_2 + 3(b_2 - b_1)(d_2 - d_1)]/h,$$

$$\int_{x_1}^{x} f(t)\,dt = u_2(y_1 + (u_2/12)\{6y_1' + b_2[4d_1 + (4 - 3b_2)(d_1 - d_2)]\}),$$

$$\int_{x}^{x_2} f(t)\,dt = u_1(y_2 - (u_1/12)\{6y_2' - b_1[4d_2 + (3b_1 - 4)(d_1 - d_2)]\}),$$

$$\int_{x_1}^{x_2} f(t)\,dt = (h/12)[6(y_1 + y_2) - h(d_1 + d_2)].$$

2. For $0<\sigma\leqq .5$, define

$SINH=\sinh(\sigma)$, $CM=\cosh(\sigma)$, $SM=\operatorname{sinhm}(\sigma)$, $CMM=\operatorname{coshmm}(\sigma)$,
$COSH_2=\cosh(\sigma b_2)$, $SINH_2=\sinh(\sigma b_2)$, $CM_2=\cosh(\sigma b_2)$,
$SM_2=\operatorname{sinhm}(\sigma b_2)$, $CMM_2=\operatorname{coshmm}(\sigma b_2)$, $SM_1=\operatorname{sinhm}(\sigma b_1)$,
$CMM_1=\operatorname{coshmm}(\sigma b_1)$ and $E=CM^2-SM*SINH$.

Then

$$f(x)=y_1+y_1'u_2+h[(CM*SM_2-SM*CM_2)(d_1+d_2)+\sigma(CM*CM_2-SINH*SM_2)d_1]/(\sigma E),$$

$$f'(x)=y_1'+[(CM*CM_2-SM*SINH_2)(d_1+d_2)+\sigma(CM*SINH_2-SINH*CM_2)d_1]/E,$$

$$f''(x)=\sigma[(CM*SINH_2-SM*COSH_2)(d_1+d_2)+\sigma(CM*COSH_2-SINH*SINH_2)d_1]/(hE),$$

$$\int_{x_1}^{x} f(t)\,dt=u_2(y_1+y_1'u_2/2)+h^2[(CM*CMM_2-SM*SM_2)(d_1+d_2)+\sigma(CM*SM_2-SINH*CMM_2)d_1]/(\sigma^2E),$$

$$\int_{x}^{x_2} f(t)\,dt=u_1(y_2-y_2'u_1/2)+h^2[(CM*CMM_1-SM*SM_1)(d_1+d_2)+\sigma(CM*SM_1-SINH*CMM_1)d_2]/(\sigma^2E),$$

$$\int_{x_1}^{x_2} f(t)\,dt=(h/2)[y_1+y_2-hE(d_1+d_2)/(\sigma^2CM)].$$

3. For $\sigma>.5$, define

$$e_1=\exp(-\sigma b_1),\quad e_2=\exp(-\sigma b_2),\quad e=e_1e_2=e^{-\sigma},$$
$$a=1-e \text{ and } E'=a[\sigma(1+e)-2a]=2e^{-\sigma}E.$$

Then

$$f(x)=y_1+su_2+h\{a(1+e-e_1-e_2)(d_1+d_2)+\sigma([e_2+e(e_1-2)-b_1a^2]d_1+[e_1+e(e_2-2)-b_2a^2]d_2)\}/(\sigma E'),$$

$$f'(x)=s+\{a[(e_2-e_1)(d_1+d_2)+a(d_1-d_2)]+\sigma[(e_1e-e_2)d_1+(e_1-e_2e)d_2]\}/E',$$

$$f''(x)=\sigma\{\sigma[(e_1e+e_2)d_1+(e_1+e_2e)d_2]-a(e_1+e_2)(d_1+d_2)\}/(hE'),$$

$$\int_{x_1}^{x} f(t)\,dt=u_2(y_1+y_1'u_2/2)+h^2\{\sigma a[(1+e)(\sigma^2b_2^2/2+1)-e_1-e_2-\sigma b_2a]d_1-(a[a(\sigma^2b_2^2/2+1)-e_2+e_1-(1+e)\sigma b_2]+\sigma[e_2e-e_1+2\sigma b_2e])(d_1+d_2)\}/(\sigma^2E'),$$

$$\int_{x}^{x_2} f(t)\,dt=u_1(y_2-y_2'u_1/2)+h^2\cdot\{\sigma a[(1+e)(\sigma^2b_1^2/2+1)-e_1-e_2-\sigma b_1a]d_2-(a[a(\sigma^2b_1^2/2+1)-e_1+e_2-(1+e)\sigma b_1]+\sigma[e_1e-e_2+2\sigma b_1e])(d_1+d_2)\}/(\sigma^2E'),$$

$$\int_{x_1}^{x_2} f(t)\,dt=(h/2)\{y_1+y_2-(h/\sigma^2)[\sigma(1+e)/a-2](d_1+d_2)\}.$$

Tests were conducted which demonstrated the above expressions to have smooth transitions (continuity with respect to σ) at $\sigma = 0$, $\sigma = .5$, and $\sigma = \sigma^c$ where $\exp(-\sigma^c)$ is equal to the machine precision (the floating point values of $1-e$ and $1+e$ are unity for $\sigma > \sigma^c$).

5. Nodal derivatives. This section is addressed to both local derivative-estimation procedures for a C^1 interpolant, and a global procedure in which the nodal derivatives y_i' are chosen to result in second-derivative continuity with some choice of end conditions. The C^2 interpolation method has an advantage in terms of accuracy and the additional smoothness, which may be necessary in some applications, but is less efficient in both the preprocessing and evaluation phases. In the preprocessing phase, a linear system must be solved for the derivatives; and when constraints are to be satisfied, an iteration must be placed around the computation of derivatives and tension factors. Also, this process must be repeated if an additional data point is added to the system, whereas local methods, in which y_i' depends only on a few nearby data points, allow for efficient updating of the data set. When constraints are imposed, the C^2 interpolant generally has nonzero tension factors in most of the intervals, whereas the C^1 method, with appropriate choice of derivatives, results in relatively few nonzero tension factors, and is therefore more efficient in the evaluation phase.

5.1. Local derivative estimates. A number of local methods have been developed for choosing nodal derivatives for the Hermite cubic interpolant, and these apply equally well in the more general situation treated here. The fourth-order finite difference method of Hyman [8] and the weighted least squares cubic fitting method of Ellis and McLain [4] are designed for accuracy, while the methods of Akima [1], Fritsch and Carlson [6], Fritsch and Butland [5], and the filtering methods of Hyman [8] are designed to preserve shape properties of the data.

Five methods were compared using the data sets listed in § 6 and tension factors chosen to preserve local convexity and monotonicity. On the basis of appearance of the interpolant, the shape-preserving methods of Akima, Fritsch and Butland, and Hyman were found to be comparable to each other and superior to McLain's method and the parabolic interpolation method. We therefore conclude that tension cannot overcome the defects of a method which is not designed to preserve shape properties, but any shape-preserving method for Hermite cubic interpolation provides an adequate means of selecting derivatives for C^1 tension splines.

On the basis of its simplicity and efficiency, we chose Hyman's monotonicity-constrained parabolic method as our primary derivative-estimation procedure. The derivative at x_i is initialized to the derivative of the quadratic interpolant defined by three data points; i.e.,

$$y_i' = \begin{cases} s_1 + h_1(s_1 - s_2)/(h_1 + h_2) & \text{if } i = 1, \\ (h_{i-1}s_i + h_i s_{i-1})/(h_{i-1} + h_i) & \text{if } 2 \leqq i \leqq n-1, \\ s_{n-1} + h_{n-1}(s_{n-1} - s_{n-2})/(h_{n-2} + h_{n-1}) & \text{if } i = n, \end{cases}$$

where $h_i = x_{i+1} - x_i$ and $s_i = (y_{i+1} - y_i)/h_i$. This value is then constrained by

$$y_i' = \begin{cases} \min\{\max\{0, y_i'\}, 3\min\{|s_{i-1}|, |s_i|\}\} & \text{if } SGN > 0, \\ \max\{\min\{0, y_i'\}, -3\min\{|s_{i-1}|, |s_i|\}\} & \text{if } SGN < 0, \end{cases}$$

where $SGN = \text{sign}(s_{i-1})$ if $|s_{i-1}| > |s_i|$, and $SGN = \text{sign}(s_i)$ otherwise. This is the method used to generate test results presented in § 6.

5.2. Global derivative estimates. For given tension factors, the equations for second-derivative continuity are obtained by equating the expression for f'' at the right

end of the interval $[x_{i-1}, x_i]$ with the expression for f'' at the left end of $[x_i, x_{i+1}]$, $i = 2, \cdots, n-1$. The notation must be modified with the addition of subscripts denoting the left end of an interval; i.e., $h_i = x_{i+1} - x_i$, $s_i = (y_{i+1} - y_i)/h_i$, σ_i is associated with $[x_i, x_{i+1}]$, and $E_i = E(\sigma_i)$. We obtain the following $n-2$ equations:

$$\sigma_{i-1}/(h_{i-1}E_{i-1})[\sigma_{i-1}\,\text{coshm}\,(\sigma_{i-1})(y_i' - s_{i-1}) - \text{sinhm}\,(\sigma_{i-1})(y_i' - y_{i-1}')]$$
$$= \sigma_i/(h_iE_i)[\sigma_i\,\text{coshm}\,(\sigma_i)(s_i - y_i') - \text{sinhm}\,(\sigma_i)(y_{i+1}' - y_i')];$$

or equivalently,

$$[g_2(\sigma_{i-1}) - g_1(\sigma_{i-1})]y_{i-1}' + [g_1(\sigma_{i-1}) + g_1(\sigma_i)]y_i' + [g_2(\sigma_i) - g_1(\sigma_i)]y_{i+1}'$$
$$= g_2(\sigma_{i-1})s_{i-1} + g_2(\sigma_i)s_i \quad \text{for } i = 2, \cdots, n-1$$

where

$$g_1(\sigma) = \begin{cases} \sigma/(hE)[\sigma * \text{coshm}\,(\sigma) - \text{sinhm}\,(\sigma)] & \text{if } \sigma > 0, \\ 4/h & \text{if } \sigma = 0, \end{cases}$$

and

$$g_2(\sigma) = \begin{cases} \sigma^2\,\text{coshm}\,(\sigma)/(hE) & \text{if } \sigma > 0, \\ 6/h & \text{if } \sigma = 0. \end{cases}$$

(The internal with which σ is associated has width h.)

The two additional equations required for a unique solution can be obtained by specifying first or second derivatives at the endpoints. In the latter case, we obtain

$$g_1(\sigma_1)y_1' + [g_2(\sigma_1) - g_1(\sigma_1)]y_2' = g_2(\sigma_1)s_1 - y_1''$$

and

$$[g_2(\sigma_{n-1}) - g_1(\sigma_{n-1})]y_{n-1}' + g_1(\sigma_{n-1})y_n' = g_2(\sigma_{n-1})s_{n-1} + y_n''$$

where y_1'' and y_n'' are the specified values. In either case, the linear system is symmetric and tridiagonal. In the case of natural boundary conditions, $f''(x_1) = f''(x_n) = 0$, the nodal derivatives are the unique solution to the problem of minimizing the functional

$$Q(y') = \int_{x_1}^{x_n} f''(x)^2\,dx + \sum_{i=1}^{n-1} (\sigma_i/h_i)^2 \int_{x_i}^{x_{i+1}} f'(x)^2\,dx.$$

If a pair of parameterized tension splines are to be used to fit a curve through a sequence of points in the plane, then, in the case of a closed curve, each spline must satisfy periodic end conditions. This is achieved by extending f to the appropriate value of $x_{n+1} > x_n$ such that $f(x_{n+1}) = y_1$, $f'(x_{n+1}) = y_1'$, and $f''(x_{n+1}) = f''(x_1)$. The corresponding equations, for $h_n = x_{n+1} - x_n$, $s_n = (y_1 - y_n)/h_n$, and σ_n associated with $[x_n, x_{n+1}]$ are

$$[g_2(\sigma_n) - g_1(\sigma_n)]y_n' + [g_1(\sigma_n) + g_1(\sigma_1)]y_1' + [g_2(\sigma_1) - g_1(\sigma_1)]y_2' = g_2(\sigma_n)s_n + g_2(\sigma_1)s_1$$

and

$$[g_2(\sigma_{n-1}) - g_1(\sigma_{n-1})]y_{n-1}' + [g_1(\sigma_{n-1}) + g_1(\sigma_n)]y_n' + [g_2(\sigma_n) - g_1(\sigma_n)]y_1'$$
$$= g_2(\sigma_{n-1})s_{n-1} + g_2(\sigma_n)s_n.$$

In this case the matrix has additional nonzero entries in the upper right and lower left corners.

Regardless of the choice of end conditions, the system is strictly diagonally dominant since

$$2g_1(\sigma) - g_2(\sigma) = \sigma/(hE)[\sigma * \text{coshm}(\sigma) - 2\,\text{sinhm}(\sigma)] > 0,$$

and thus $g_1(\sigma) > g_2(\sigma) - g_1(\sigma)$. Also, since the diagonal elements are positive, the system is positive definite and hence requires no pivoting for a stable solution. The work space required by the solution method is a single array of length $n-1$, or two such arrays in the case of periodic end conditions.

It may be of interest to note the effect of infinite tension on the choice of derivatives. For large σ, approximate values of g_1 and g_2 are $\sigma(\sigma-1)/(h(\sigma-2))$ and $\sigma^2/(h(\sigma-2))$, respectively. Thus both g_1 and g_2 approach σ/h, and as σ_{i-1} and σ_i approach infinity, y_i' approaches $(h_{i-1}s_i + h_i s_{i-1})/(h_{i-1}+h_i)$ for $2 \leqq i \leqq n-1$. This is the parabolic interpolation method.

We now consider the problem of choosing minimum tension factors required to satisfy constraints associated with the intervals when a C^2 interpolant is desired. In this case, the equations for tension factors cannot be separated, and a system of $2n-1$ nonlinear equations must be solved for the n nodal derivatives and $n-1$ tension factors. Newton's method might be a viable approach but would probably require finite difference approximations to Jacobian entries since the partial derivatives with respect to tension factors are extremely complex and difficult to formulate without cancellation error—particularly in the case of bounds on function values. Even in the case of a single uniform tension factor, a method of this type is quite cumbersome [9].

As an alternative we used a simple iterative procedure which, beginning with zero tension factors, alternates between solving the linear system for nodal derivatives and applying the methods of § 3 to choose optimal tension factors for the given derivative estimates. By updating a tension factor only if it increased from the previous iteration, and placing an upper bound (100) on tension factors, convergence is ensured (although not necessarily within an acceptable number of iterations) and the constraints are satisfied, but with more tension than necessary in some cases. This method has the advantages that no additional code is required for its implementation, it takes full advantage of the sparsity and linearity in the problem, and it provides for flexibility in choosing a balance between constraints and smoothness of the interpolant. The iterative procedure can be terminated at any point with either the constraints or second-derivative continuity satisfied, depending upon which step was last executed. Test results are presented in the next section.

6. Test results. This section presents plots of both the C^1 and C^2 interpolants on each of six data sets with tension factors chosen to preserve local convexity and monotonicity (see Figs. 1-12). All computing was performed on the DEC PDP-10 system at Oak Ridge National Laboratory, and the DISSPLA graphics package was used to produce the plots. Data points, nodal derivatives, and tension factors are listed in Tables 1-6. Since convexity of the interpolant implies monotonicity, tension factors were chosen for convexity except where monotonicity can be satisfied and convexity cannot. This occurs only in the second to last interval of data set 1. The endpoint derivatives y_1' and y_n' of the C^1 interpolant were used as boundary values for the C^2 method.

Table 7 lists the numbers of iterations used to generate the C^2 interpolants, along with the number of data points and the origin of each data set. Each iteration consists of computing both tension factors and derivatives, and the number of iterations was limited to 30. For data sets 1, 2, 4, and 5, this was sufficient for convergence—the last iteration resulted in no change in the tension factors. For data sets 3 and 6, the total

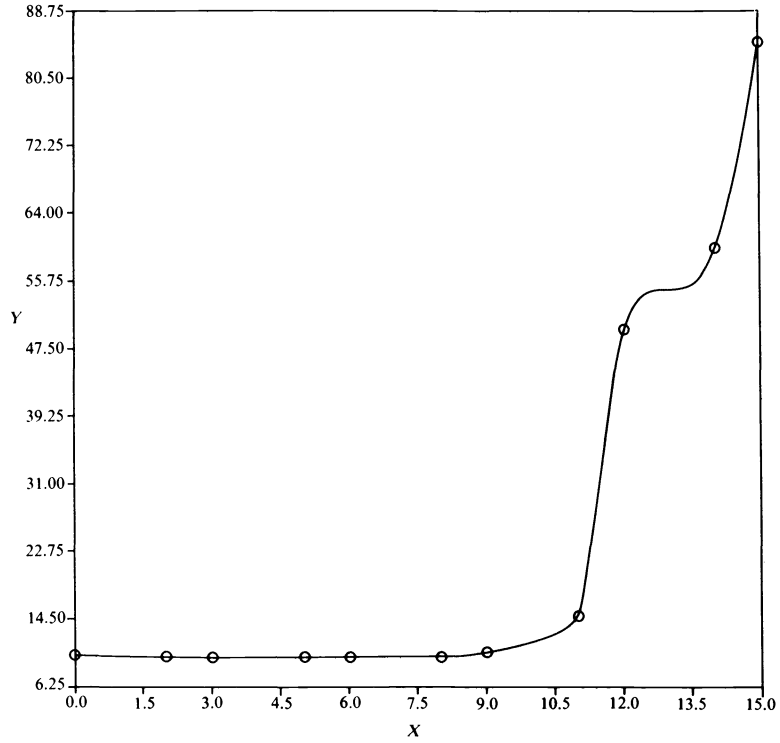


FIG. 1. C^1 *interpolant for data set* 1.

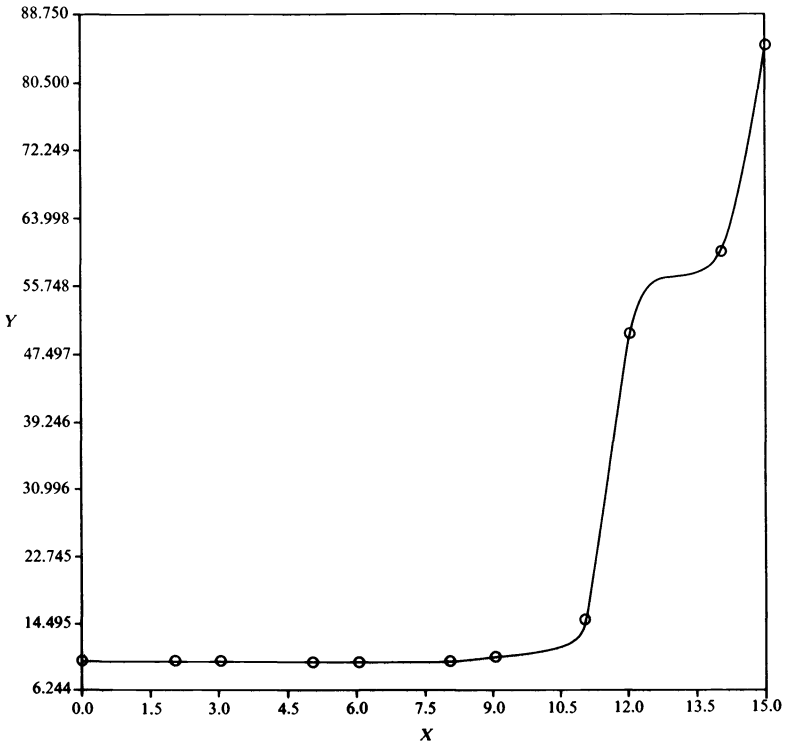


FIG. 2. C^2 *interpolant for data set* 1.

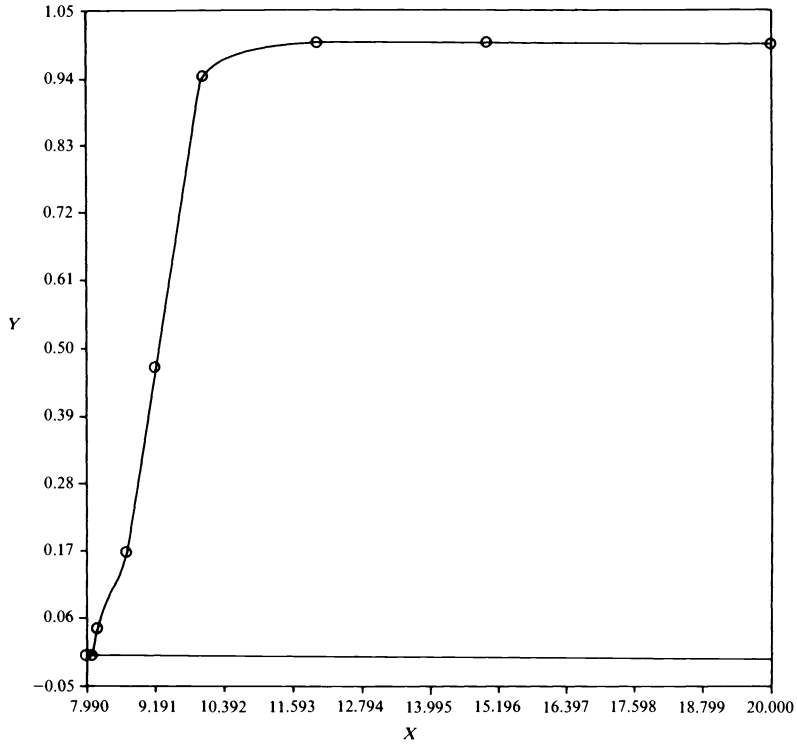


FIG. 3. C^1 *interpolant for data set* 2.

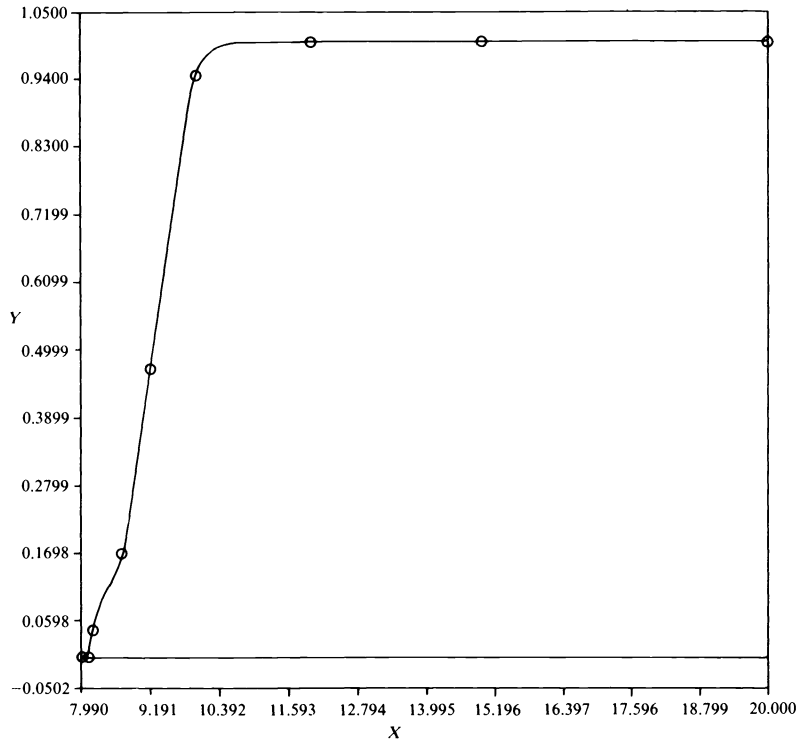


FIG. 4. C^2 *interpolant for data set* 2.

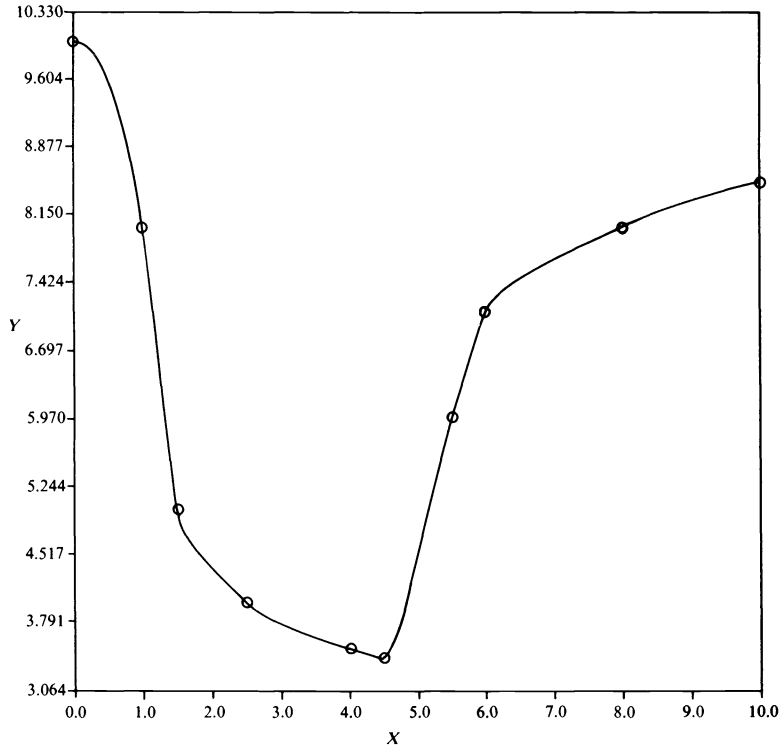


FIG. 5. C^1 *interpolant for data set* 3.

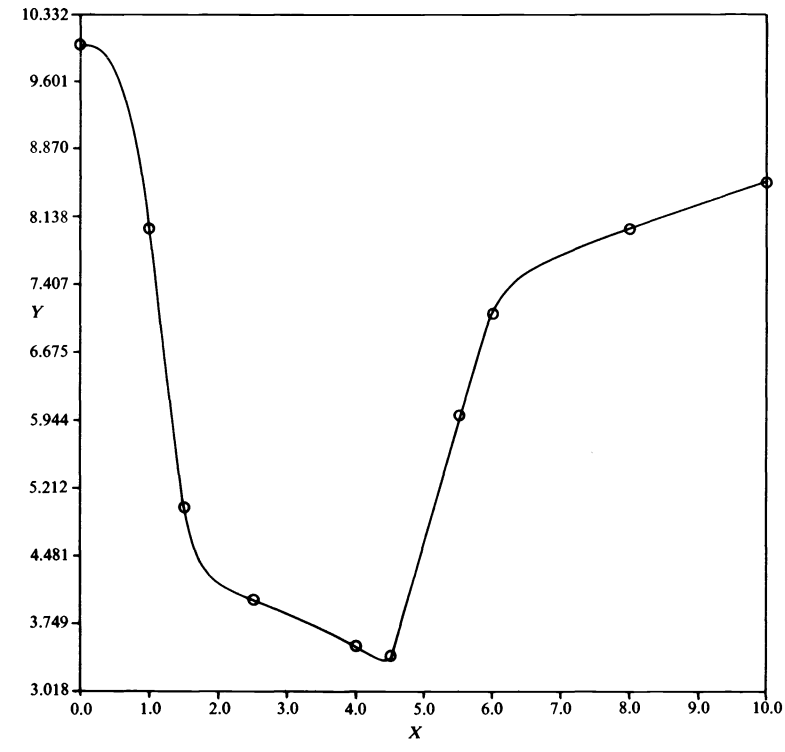


FIG. 6. C^2 *interpolant for data set* 3.

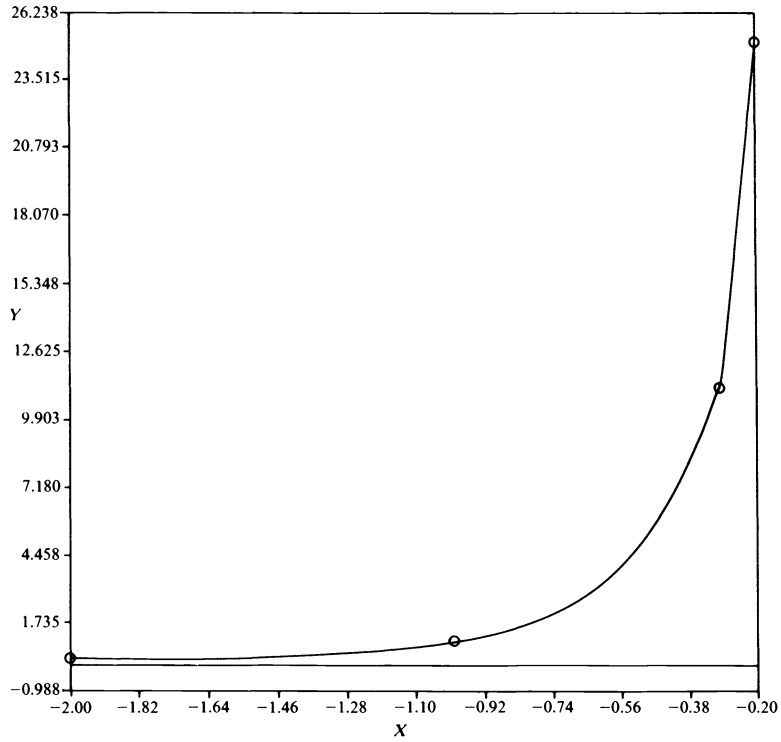


FIG. 7. C^1 *interpolant for data set* 4.

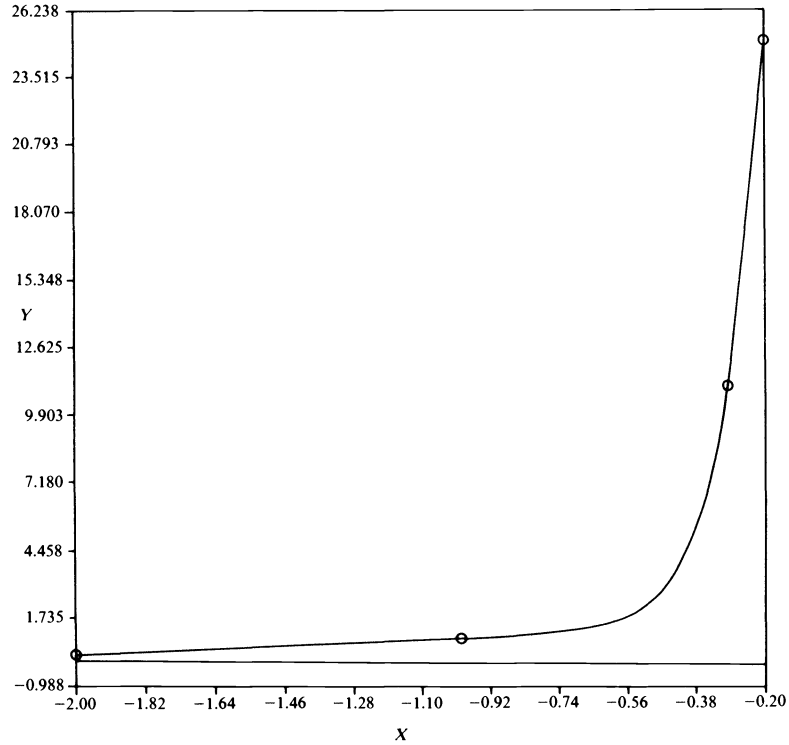


FIG. 8. C^2 *interpolant for data set* 4.

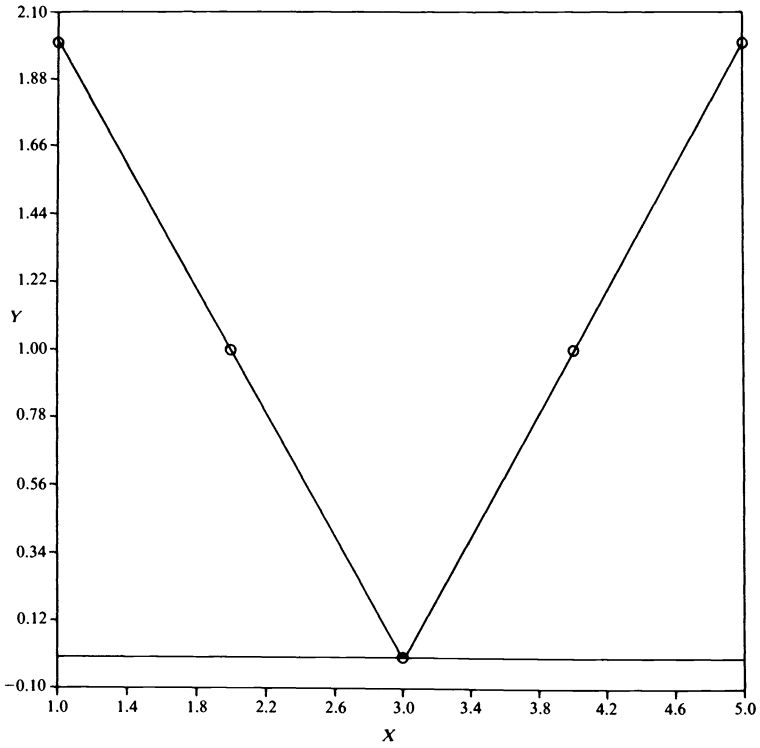


FIG. 9. C^1 *interpolant for data set* 5.

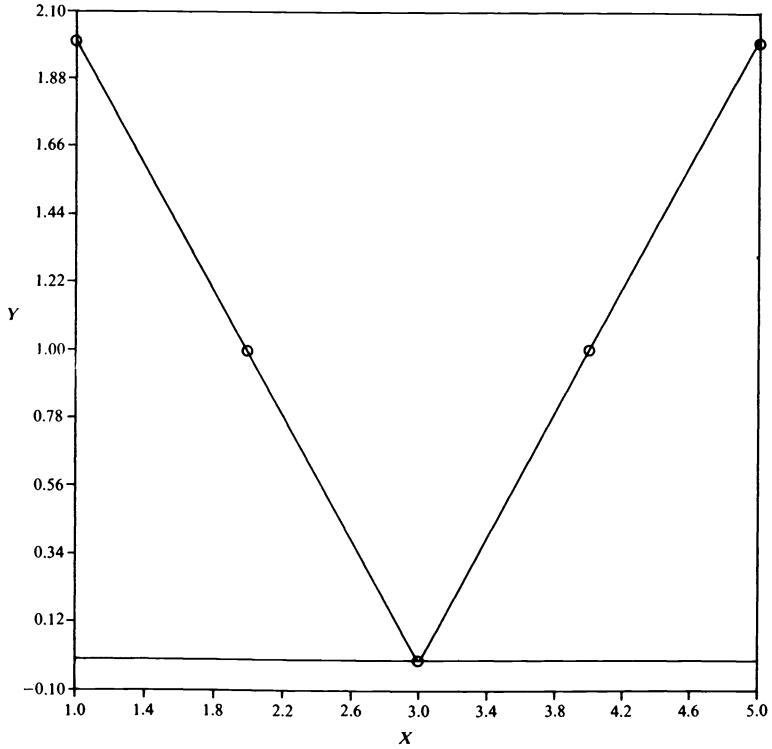


FIG. 10. C^2 *interpolant for data set* 5.

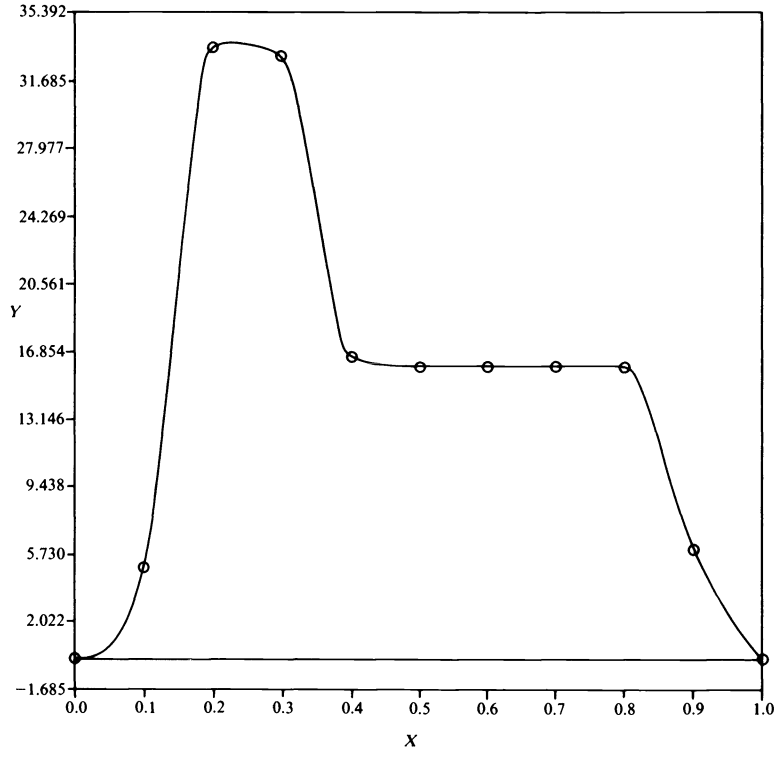


FIG. 11. C^1 *interpolant for data set* 6.

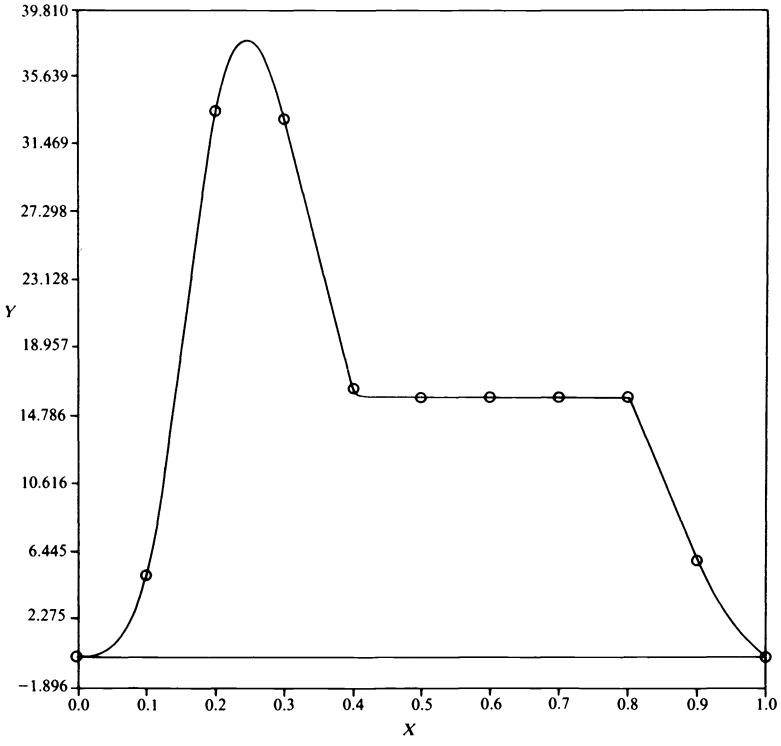


FIG. 12. C^2 *interpolant for data set* 6.

TABLE 1
Data set 1.

		C^1 Interpolant		C^2 Interpolant	
x	y	y'	σ	y'	σ
0.0	10.0	0.0	0.0	0.0	100.0
2.0	10.0	0.0	0.0	.1570498E−9	100.0
3.0	10.0	0.0	0.0	−.2332190E−7	100.0
5.0	10.0	0.0	0.0	.6926291E−5	100.0
6.0	10.0	0.0	0.0	−.1028543E−2	100.0
8.0	10.0	0.0	0.0	.3054633E0	.7836898E+2
9.0	10.5	.1083333E+1	.4457221E+1	.5163587E0	.1085806E+2
11.0	15.0	.6750000E+1	0.0	.1755330E+2	0.0
12.0	50.0	.1500000E+2	0.0	.2419420E+2	.6168116E+1
14.0	60.0	.1500000E+2	0.0	.1176266E+2	0.0
15.0	85.0	.3166667E+2		.3166667E+2	

TABLE 2
Data set 2.

		C^1 Interpolant		C^2 Interpolant	
x	y	y'	σ	y'	σ
7.99	0.0	0.0	.4882813E−3	0.0	100.0
8.09	.276429E−4	.8292868E−3	0.0	.2187491E0	100.0
8.19	.437498E−1	.4058651E0	0.0	.4374670E0	0.0
8.70	.169183E0	.4249739E0	0.0	.3681239E0	.1931883E+1
9.20	.469428E0	.5975669E0	100.0	.6985542E0	.2876000E+1
10.00	.943740E0	.8234400E−1	.9915005E0	.2962585E0	.1094519E+2
12.00	.998636E0	.1282997E−2	.1538315E+1	.4292283E−3	100.0
15.00	.999919E0	.4500300E−4	.4882813E−3	.2729635E−3	100.0
20.00	.999994E0	0.0		0.0	

TABLE 3
Data set 3.

		C^1 Interpolant		C^2 Interpolant	
x	y	y'	σ	y'	σ
0.0	10.0	0.0	0.0	0.0	0.0
1.0	8.0	−.4666667E+1	0.0	−.5695050E+1	0.0
1.5	5.0	−.3000000E+1	.8473443E+1	−.3914849E+1	.4883898E+1
2.5	4.0	−.7333333E0	.4640847E+1	−.3066378E0	.2007099E0
4.0	3.5	−.2333333E0	.2500000E+2	−.3792315E0	.1300812E+2
4.5	3.4	.6000000E0	0.0	.1989169E+1	100.0
5.5	6.0	.2333333E+1	.7312812E+1	.2602246E+1	0.0
6.0	7.1	.1350000E+1	.9991768E+1	.1494606E+1	.6547370E+1
8.0	8.0	.3500000E0	0.0	.2624173E0	.1051179E+2
10.0	8.5	.1500000E0		.1500000E0	

TABLE 4
Data set 4.

		C^1 Interpolant		C^2 Interpolant	
x	y	y'	σ	y'	σ
−2.0	0.500	0.0	0.0	0.0	.1293490E+2
−1.0	1.0000	.2250000E+1	.1930536E+1	.8150119E0	.7596293E+1
−0.3	11.1111	.4333333E+2	.7068929E+1	.1050236E+3	.1343148E+1
−0.2	25.0000	.1544444E+3		.1544444E+3	

TABLE 5
Data set 5.

		C^1 Interpolant		C^2 Interpolant	
x	y	y'	σ	y'	σ
1.0	2.0	−1.0	0.0	−1.000000	100.0
2.0	1.0	−1.0	100.0	−1.005051	100.0
3.0	0.0	0.0	100.0	0.0	100.0
4.0	1.0	1.0	0.0	1.005051	100.0
5.0	2.0	1.0		1.000000	

TABLE 6
Data set 6.

		C^1 Interpolant		C^2 Interpolant	
x	y	y'	σ	y'	σ
0.0	0.0	0.0	0.0	0.0	.3271173E+1
0.1	5.0	150.0	0.0	.1778152E+3	0.0
0.2	33.5	15.0	0.0	.2063615E+3	0.0
0.3	33.0	−15.0	0.0	−.1632611E+3	100.0
0.4	16.5	−15.0	.4882813E−3	−.1310362E+3	.2619704E+2
0.5	16.0	0.0	0.0	.1592003E−2	.7843300E+1
0.6	16.0	0.0	0.0	−.2134268E−2	.8390489E+2
0.7	16.0	0.0	0.0	.1940007E0	100.0
0.8	16.0	0.0	0.0	−.3535074E+2	.5418210E+2
0.9	6.0	−80.0	0.0	−.9904715E+2	.7815939E0
1.0	0.0	−40.0		−.4000000E+2	

TABLE 7
Data sets.

Data set	n	Source	No. iterations for C^2 method
1	11	Akima (third set) [1]	8
2	9	Fritsch and Carlson (RPN 14) [6]	19
3	10	Späth [18]	30
4	4	McAllister, Passow and Roulier ($1/x^2$) [10]	16
5	5	Renka	5
6	11	Pruess [13]	30

number of iterations required for convergence was 204 and 59, respectively. However, the additional iterations resulted in no discernible change in the plots, and thus, while convergence can be very slow, the slowness occurs only where there is little to be gained. Data set 5 was chosen to demonstrate that a first-derivative discontinuity can be simulated with large tension factors.

7. Conclusion. We have described an interpolation method which provides for automatically satisfying general bounds on function and derivative values, as well as preserving shape properties of the data. The method has been implemented in a software package written in a portable subset of 1966 ANSI Standard FORTRAN and available from the author. A primary characteristic of this package is the versatility derived from variable tension factors. Options range from highly efficient piecewise cubic interpolation to C^2 splines with user-specified and/or automatically computed tension factors. Work is under way on extending the methods and software to more general problems such as fitting a surface to points arbitrarily distributed in a two-dimensional domain.

Acknowledgments. The author is grateful for the thorough examination and discovery of several errors by the referees.

REFERENCES

[1] H. AKIMA, *A new method of interpolation and smooth curve fitting based on local procedures*, J. ACM, Comput. Math., 17 (1970), pp. 589-602.

[2] A. K. CLINE, *Scalar- and planar-valued curve fitting using splines under tension*, Comm. ACM, 17 (1974), pp. 218-223.

[3] C. DE BOOR, *A Practical Guide to Splines*, Springer-Verlag, New York, 1978.

[4] T. M. R. ELLIS AND D. H. MCLAIN, *Algorithm* 514. *A new method of cubic curve fitting using local data*, ACM Trans. Math. Software, 3 (1977), pp. 175-178.

[5] F. N. FRITSCH AND J. BUTLAND, *A method for constructing local monotone piecewise cubic interpolants*, this Journal, 5 (1984), pp. 300-304.

[6] F. N. FRITSCH AND R. E. CARLSON, *Monotone piecewise cubic interpolation*, SIAM J. Numer. Anal., 17 (1980), pp. 238-246.

[7] J. F. HART, E. W. CHENEY, C. L. LAWSON, H. J. MAEHLY, C. K. MESZTENYI, J. R. RICE, H. G. THACHER AND C. WITZGALL, *Computer Approximations*, John Wiley, New York, 1968.

[8] J. M. HYMAN, *Accurate Monotonicity-preserving Cubic Interpolation*, Report LA-8796-MS, Los Alamos National Laboratory, February 1982.

[9] R. W. LYNCH, *A method for choosing a tension factor for spline under tension interpolation*, M. A. Thesis, Univ. Texas, 1982.

[10] D. F. MCALLISTER, E. PASSOW AND J. A. ROULIER, *Algorithms for computing shape preserving spline interpolations to data*, Math. Comp., 31 (1977), pp. 717-725.

[11] D. F. MCALLISTER AND J. A. ROULIER, *An algorithm for computing a shape preserving osculatory quadratic spline*, ACM Trans. Math. Software, 17 (1981), pp. 331-347.

[12] S. PRUESS, *Properties of splines in tension*, J. Approx. Theory, 17 (1976), pp. 86-96.

[13] ———, *An algorithm for computing smoothing splines in tension*, Computing, 19 (1978), pp. 365-373.

[14] ———, *Alternatives to exponential splines in tension*, Math. Comp., 33 (1979), pp. 1273-1281.

[15] P. RENTROP, *An algorithm for the computation of the exponential spline*, Numer. Math., 35 (1980), pp. 81-93.

[16] L. L. SCHUMAKER, *On shape preserving quadratic spline interpolation*, SIAM J. Numer. Anal., 20 (1983), pp. 854-864.

[17] D. G. SCHWEIKERT, *An interpolation curve using a spline in tension*, J. Math. Phys., 45 (1966), pp. 312-317.

[18] H. SPÄTH, *Exponential spline interpolation*, Computing, 4 (1969), pp. 225-233.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 3, May 1987
© 1987 Society for Industrial and Applied Mathematics
013

ON LARGE SCALE NONLINEAR LEAST SQUARES CALCULATIONS*

PH. L. TOINT†

Abstract. The nonlinear model fitting problem is analyzed in this paper, with special emphasis on the practical solution techniques when the number of parameters in the model is large. Classical approaches to small dimensional least squares calculations are reviewed and an extension of them to problems involving many variables is proposed. This extension uses the context of partially separable structures, which has already proved its applicability for large scale optimization. An adaptable algorithm is discussed, which chooses between various possible models of the objective function. Preliminary numerical experience is also presented, which shows that actual numerical solution of a large class of fitting problems involving several hundreds of nonlinear parameters is possible at a reasonable cost.

Key words. large scale computations, nonlinear least squares, nonlinear estimation

AMS(MOS) subject classifications. 65D10, 65K05

1. Introduction: The nonlinear least squares problem. One of the most pervasive questions in the field of general scientific computing is that of fitting a given model, depending on several parameters, to a given set of observations. There is indeed a basic need for experimentalists and in so many different scientific subjects, that even a brief survey of them is impossible. In a fair number of these problems, the accuracy of the model predictions compared to the observations does not depend linearly on the model parameters; the fitting problem is nonlinear, which also means that its solution is likely to require a nontrivial calculation. The problem becomes even worse when the number of parameters present in the model is of the order of several hundred, because most of the available solution methods for this type of computation are designed for handling small dimensional models, and lose their applicability for larger ones.

If the interest in fitting procedures is important for many scientists outside the field of numerical analysis, it is also true that numerical analysts have, in recent years, devoted a substantial amount of work to the study and implementation of robust and efficient algorithms for this problem. Among recent developments, one can mention [3], [19], [9], [20], [2], [4], [23], [16], [1] and [6], to cite only a few. Although an important research effort has been produced in the context of the nonlinear fitting case, as can be seen from the reference list above, the case where the number of parameters in the model is large is still considered as a difficult and challenging topic.

It is the main purpose of this paper to show that a large class of these difficult large nonlinear fitting problems can be solved at a reasonable cost.

One of the natural ways to describe the accuracy of model predictions is to measure the "distance" between these predictions and the observations. The purpose of the fitting method is then to reduce this distance to a minimum. If the distance chosen is the usual Euclidean distance in the space of observations, the problem is then reduced to the mathematical formulation of a "least squares" optimization calculation. More precisely, if the model parameters, or variables, are denoted by a real n-dimensional vector x, the problem is to find a point x^* that minimizes the real valued function

$$f(x)=\tfrac{1}{2}\sum_{i=1}^{m} f_i^2(x), \tag{1}$$

* Received by the editors July 10, 1985, and in revised form December 5, 1985.
† Department of Mathematics, Facultés Universitaires ND de la Paix, B-5000 Namur, Belgium.

where the $f_i(x)$ represent the error in the model prediction for the ith observation. The origin of the name "nonlinear least squares" is now obvious. As mentioned above, we are interested in the case where these functions depend nonlinearly on x. We also emphasize the important practical case where second derivative matrices of the objective function (1) are either too costly to compute or simply unavailable. Finally, we consider the case where n is "large." In this context, the word "large" has to be somewhat qualified: we will consider the dimension n to be large when storing and revising an n by n matrix causes storage and/or processing time to become unduly expensive. A rule of thumb in many nonlinear least squares applications is to put the boundary somewhere between 100 and 200: most of the available codes will either fail on larger problems, or require an unacceptably long computing time.

The next section introduces more notation and definitions related to the problem, and briefly reviews some of the proposed solution methods. Section 3 is devoted to the presentation and motivation of a new procedure adapted to the large dimensional case. The purpose of § 4 is to shed light on some of the algorithmic choices by presenting numerical experience on both small and large problems. Finally, some conclusions will be drawn in § 5, and further questions of interest outlined.

2. Numerical approaches to the nonlinear least squares problem. As is usual in many other minimization problems, we have to study the structure of the derivatives of our objective function $f(x)$ in (1). If $g(x)$, $g_i(x)$, $H(x)$ and $H_i(x)$ denote the gradients and Hessian matrices of $f(x)$ and $f_i(x)$, respectively, it is easy to check that

$$g(x) = \sum_{i=1}^{m} f_i(x) g_i(x) \tag{2}$$

and that

$$H(x) = \sum_{i=1}^{m} g_i(x) g_i^T(x) + f_i(x) H_i(x), \tag{3}$$

where the superscript T denotes the transpose. As stated above, we assume that (3) cannot be computed exactly; and one of the major problems we discuss below is that of approximating this quantity in a reasonable and efficient way.

Let us now denote by $F(x)$ the vector function from $\mathbf{R}^n$ into $\mathbf{R}^m$ whose components are $f_i(x)$. Our problem can be rewritten as the minimization of the Euclidean norm of this vector function. If $J(x)$ is the Jacobian matrix of $F(\cdot)$ at x, one has that

$$g(x) = J(x)^T F(x) \tag{4}$$

and

$$H(x) = J(x)^T J(x) + \sum_{i=1}^{m} f_i(x) H_i(x). \tag{5}$$

In a neighbourhood of x, one can therefore write a second order local model for $f(\cdot)$ as

$$m(x+s) = f(x) + g(x)^T s + \tfrac{1}{2} s^T H s, \tag{6}$$

where the symmetric matrix H is an approximation to the Hessian $H(x)$.

A first classical approach, the Gauss-Newton method, uses a matrix H^k of the form

$$H^k = J(x^k)^T J(x^k) \tag{7}$$

to approximate $H(x^k)$. The corresponding model can be derived from an affine approximation of $F(x)$ around x^k. It is well known that this method performs well in theory and practice when the residuals $f_i(x^*)$ are zero or small compared to the singular values of $J(x^*)$ (see [10], for example). The ultimate rate of convergence can be shown to be Q-quadratic on zero residual problems and fast Q-linear on problems with small residuals. A further advantage of this approach is that linear problems (i.e. problems where the $f_i(x)$ are linear in x) are solved in just one iteration, H^k being, in this case, equal to $H(x^k)$. Various strategies have been designed to cope with the possible rank deficiency of $J(x^k)$, as the Levenberg-Marquardt, or trust region approach [20], which is also suitable to ensure good convergence even from starting points that are far from the solution x^*.

However, it is also clear that the performance of the Gauss-Newton method can be degraded substantially if the term $J(x)^TJ(x)$ is not dominant in $H(x)$. To overcome this difficulty, one could think of using a classical quasi-Newton algorithm on the original problem. A numerical procedure using the BFGS update to approximate $H(x^k)$ will typically be considered with the secant condition

$$H^{k+1}(x^{k+1}-x^k) = J(x^{k+1})^TF(x^{k+1}) - J(x^k)^TF(x^k) \tag{8}$$

where k denotes the iteration number. In this case, the matrices H^k are obtained by the famous BFGS updating formula

$$H^{k+1} = H^k - \frac{H^k s^k [s^k]^T H^k}{[s^k]^T H^k s^k} + \frac{y^k [y^k]^T}{[y^k]^T s^k} \tag{9}$$

starting from a suitable symmetric positive definite matrix H^0, and where

$$y^k = J(x^{k+1})^TF(x^{k+1}) - J(x^k)^TF(x^k) \tag{10}$$

and

$$s^k = x^{k+1} - x^k. \tag{11}$$

(The reader is referred to [5] for a more complete description of this classical method.) The resulting algorithm performs relatively well, especially if the second term in (5) is large compared to the first. However, the special structure of the nonlinear least squares problem is not used, which might be considered inefficient, and difficulties arise for small residual problems, because the BFGS update does not use the fact that $|f_i(x)|$ decrease sometimes very fast. This method is nevertheless quite popular among users in the case where the Gauss-Newton approach fails.

From this point of view, it seems attractive to design a method that could include the best of both worlds, that is one that would keep good convergence properties independent from the residual size. A proposal in this direction has been made by Dennis, Gay and Welsh in [4], where they consider the Hessian as

$$H(x^k) = J(x^k)^TJ(x^k) + S(x^k), \tag{12}$$

and suggest to approximate $S(x^k)$, and not $H(x^k)$, by a quasi-Newton updating scheme. The modified secant condition they use in this framework is

$$B^{k+1}(x^{k+1}-x^k) = J(x^{k+1})^TF(x^{k+1}) - J(x^k)^TF(x^{k+1}), \tag{13}$$

where B^k now approximates $S(x^k)$. The resulting approximation to the Hessian $H(x^k)$ is then

$$H^k = J(x^k)^TJ(x^k) + B^k. \tag{14}$$

The algorithm also considers two different models, depending on whether the approximation to $S(x)$ is included in the model or neglected, and chooses that one which best predicts the overall progress and change in $f(x)$ from iteration to iteration. This adaptive character is attractive, and experience shows it to be quite efficient: the code NL2SOL, implementing this strategy, is clearly among the best for solving small dimensional nonlinear least squares problems. The author finds it interesting that Dennis, Gay and Welsh thought it valuable to include a so-called "sizing" procedure in their method. This sizing premultiplies the DFP quasi-Newton approximation to

$$S(x^k) = \sum_{i=1}^{m} f_i(x^k) H_i(x^k) \tag{15}$$

by a scalar factor, whose purpose is to take into account the variations in the sizes of the $f_i(x^k)$, especially for problems with small residuals. Further work in the direction of adaptive quasi-Newton methods has also been published by Al-Baali [1], who recommends using a BFGS-type update instead of the DFP formula that is implemented in NL2SOL.

The last approach we wish to review is slightly older, and is due to Brown and Dennis in [3]. They consider building a model for each component function $f_i(x)$ that is of the form

$$m_i(x+s) = f_i(x) + g_i(x)^T s + \tfrac{1}{2} s^T B_i s, \tag{16}$$

where the matrix B_i is obtained as a quasi-Newton approximation to $H_i(x)$ by using the unsymmetric Broyden's formula. Their paper contains a proof of Q-quadratic convergence on zero residual problems as well as some encouraging numerical results. Unfortunately, this method never became popular, mainly because it requires storing m unsymmetric matrices B_i of dimension n; and this storage was judged prohibitive. As will be discussed below, we would like to question this perhaps too quick dismissal.

Let us now examine the extensions of these algorithms to large dimensional problems and the difficulties that arise in this context.

Using the Gauss–Newton approach for nonlinear least squares problems involving many variables is not too complicated, at least conceptually. Indeed, this method proceeds by solving a sequence of linear least squares calculations; and there are several strategies for extending this computation to large sparse problems (see [15], [8] for further details and references). Therefore, using these strategies, it is possible to exploit sparsity in the nonlinear framework also. Of particular interest are iterative solvers like LSQR [21], because they provide the possibility of obtaining only an inaccurate solution to the linear problem, when this problem appears in the early iterations of the nonlinear procedure, far from the solution. This type of "truncation" has already proven its practical value for other types of large scale nonlinear calculations ([7], [25], for example).

Extending quasi-Newton methods to large dimension has proven to be more challenging. The main difficulty lies in the fact that all classical quasi-Newton updating formulae for Hessian approximation are built from low rank corrections to an existing matrix, and that any sparsity present in this matrix would therefore soon be destroyed by the updating scheme. Efforts have been made to design sparse updating procedures ([18], [24], etc.), but the numerical performance of the resulting algorithms has been somewhat disappointing. A newer approach has, however, shown a substantial improvement in this field: it was proposed in [12] and uses the concepts of partial separability and partitioned updating.

A numerical function $f(x)$ from $\mathbf{R}^n$ into $\mathbf{R}$ is said to be partially separable if and only if it satisfies the two following conditions:

1) it can be written as the sum of the so-called "element functions," that is,

$$f(x) = \sum_{i=1}^{m} f_i(x), \tag{17}$$

2) for each of these element functions, there exists a linear subspace $\mathcal{N}_i$ in $\mathbf{R}^n$ such that, for all vectors w in $\mathcal{N}_i$, $f_i(\cdot)$ is invariant under translations along the direction w, that is,

$$f_i(x+w) = f_i(x), \tag{18}$$

for all w in $\mathcal{N}_i$ and all x in $\mathbf{R}^n$.

As was shown in [13], this type of function is extremely general and contains, among others, any sufficiently smooth function whose Hessian matrix is sparse. Moreover, sparsity is often detected in practice because one already has a decomposition of the form (17). This structure is especially interesting from the algorithmic point of view when the dimensions of the subspaces $\mathcal{N}_i$ are large, compared to n. A typical case is when every element function depends only on a small subset of the set of n variables: the subspaces $\mathcal{N}_i$ may then be spanned by a subset of the canonical basis vectors.

The partitioned updating technique, proposed in the context of general unconstrained calculations, consists of maintaining an approximating matrix to each one of the m "element Hessians" (that is, the Hessian matrices of the element functions $f_i(x)$) and revising these approximations using a classical quasi-Newton update with the secant condition

$$H_i^{k+1}(x^{k+1} - x^k) = g_i(x^{k+1}) - g_i(x^k), \tag{19}$$

where H_i^{k+1} is the approximation to the Hessian of the ith element function. Since this approximation should be identically zero on $\mathcal{N}_i$, one needs only to store the restriction of this matrix on the orthogonal complement of that subspace. Similarly, only this part of H_i^k should be updated. These considerations lead to large savings in storage when the dimension of $\mathcal{N}_i$ is large, which is the usual case.

The connection of the partitioned updating technique with the Brown and Dennis method is now clear: in both proposals, the objective function is decomposed in a sum and an approximation to the Hessian matrix of each term of this sum is maintained independently. Partial separability only implies that, for most large scale problems, the storage needed for these m symmetric matrices is far from being prohibitive, as opposed to the argument that was previously mentioned to dismiss the Brown and Dennis procedure.

Our purpose in what follows is to experiment with a technique that incorporates:

- separate models for each element function $f_i(x)$ appearing in (1),
- the storage scheme described in [14] for the element Hessian approximation,
- the BFGS formula to update element Hessian approximation for the convex element function, and the safeguarded symmetric rank one (RK1) formula for the others, as discussed in [14],
- an adaptive strategy, in the spirit of Dennis, Gay and Welsh, to choose, for each element, the most appropriate model to the Hessian matrix of $[f_i(x)]^2$.

The last three points distinguish the proposal from the Brown and Dennis approach.

A consequence of the possible inclusion of an approximation to the Hessian of the element function in the model is that the overall approximation to $H(x)$ need not

be convex. This problem also appears in NL2SOL, where a trust region scheme is used to overcome the difficulty. We will follow this strategy also, and refer the reader to [4] or [14] for further practical details about its implementation.

Let us now outline the specific algorithm we have in mind for the nonlinear least squares problem.

1. Assume a starting point x^0 is given, together with the element functions values $f_i(x^0)$ and element gradients $g_i(x^0)$, for $i=1,\cdots,m$. Assume that the initial $\{B_i^0\}_{i=1}^m$ are given on the orthogonal complements of the subspaces $\mathcal{N}_i$. Set $k=0$.
2. For each element ($i=1,\cdots,m$), choose a model for its Hessian matrix by setting

$$H_i^k = g_i(x^k)g_i(x^k)^T \tag{20}$$

or

$$H_i^k = g_i(x^k)g_i(x^k)^T + f_i(x^k)B_i^k. \tag{21}$$

(The choice (20) clearly corresponds to the Gauss-Newton approach, while (21) is a quasi-Newton type model in the spirit of Brown and Dennis.)
3. In order to define a search direction, use the linear conjugate gradient algorithm starting from the origin to solve the system

$$H^k d^k = \left(\sum_{i=1}^m H_i^k\right) d^k = -\sum_{i=1}^m g_i(x^k) \tag{22}$$

for d^k, as long as a direction of negative curvature is not encountered. If such a direction is detected, scale the corresponding iterate to the trust region radius to determine d^k.
4. Perform an inexact linesearch along the direction d^k, to ensure the two classical conditions

$$f(x^k + t^k d^k) \leqq f(x^k) + c_1 t^k g(x^k)^T d^k \tag{23}$$

and

$$g(x^k + t^k d^k)^T d^k \geqq c_2 g(x^k)^T d^k, \tag{24}$$

where

$$0 > c_1 > \tfrac{1}{2} \quad \text{and} \quad c_1 \leqq c_2 < 1. \tag{25}$$

Set

$$s^k = t^k d^k \tag{26}$$

and

$$x^{k+1} = x^k + s^k. \tag{27}$$

This step also provides the values of $f_i(x^{k+1})$ and $g_i(x^{k+1})$ for $i=1,\cdots,m$.
5. Test satisfaction of the stopping conditions.
6. Update the element Hessian approximations, for $i=1,\cdots,m$, by imposing the secant conditions (19) and using the BFGS update

$$B_i^{k+1} = B_i^k - \frac{B_i^k s^k [s^k]^T B_i^k}{[s^k]^T B_i^k s^k} + \frac{y_i^k [y_i^k]^T}{[y_i^k]^T s^k} \tag{28}$$

if

$$[s^j]^T y_i^j > 0 \tag{29}$$

for all $j = 1, \cdots, k$, where

$$y_i^k = g_i(x^{k+1}) - g_i(x^k), \tag{30}$$

or the symmetric rank one update

$$B_i^{k+1} = B_i^k + \frac{[y_i^k - B_i^k s^k][y_i^k - B_i^k s^k]^T}{[y_i^k - B_i^k s^k]^T s^k} \tag{31}$$

otherwise.

7. Update the trust region radius by doubling it if the decrease $f(x^k) - f(x^{k+1})$ is at least a tenth of its predicted value $-[g(x^k)]^T s^k - \frac{1}{2}[s^k]^T H^k s^k$, or halve it otherwise.
8. Increment k by one and go to step 2.

This brief outline immediately suggests some comments.

1. The presence and structure of the nullspaces $\mathcal{N}_i$ must be used in the practical implementation of the storage scheme of the matrices B_i^k and their updating. The reader is again referred to [14] for a complete description of this scheme. Although this technique allows for modest storage requirements, we will nevertheless examine later in this paper the case when the problem is so large that even this reduced storage becomes prohibitive or impossible.

2. In step 2, the method of choice between (21) and (20) is not specified. Different strategies for this choice will be compared below. However, we will always impose that (20) is chosen for $k = 0$.

3. In step 3, it is possible (and often extremely efficient) to allow for an inexact or "truncated" solution of the linear system (22). The iterative solution of this system is then stopped when

$$\|g(x^k) + H^k d^k\| \leqq \alpha_k \|g(x^k)\|, \tag{32}$$

for a suitable constant $\alpha_k \in [0, 1)$, or when a direction of negative curvature is encountered. The resulting strategy is then a standard truncated conjugate gradient technique as studied in [22].

4. In the case where the Gauss-Newton model (34) is selected for all elements, the linear problem of step 3 reduces to a classical linear least squares. It may therefore be appropriate to use an iterative technique adapted to this particular problem, as LSQR [21], instead of the more classical conjugate gradient on the normal equations. Other techniques for large scale linear least squares may also be of interest.

5. Although they could be used in this framework with some success, we do not comment here on the use of the linesearch procedures that are specifically designed for the nonlinear least squares problem (see [1], [16]) in step 4.

6. Many stopping rules can be designed for the nonlinear least squares problem. We refer the reader to [4] for further discussion. The rule we used in the tests below is discussed at the beginning of the next section.

7. The update (31) is skipped whenever

$$|(y_i^k - B_i^k s^k)^T s^k| \geqq 0.1 \|y_i^k - B_i^k s^k\| \cdot \|s^k\| \tag{33}$$

in order to avoid the inherent instability of the symmetric rank one formula. This safeguard was also used in [14]. The constant 0.1 is rather arbitrary and other choices could certainly be made. As an alternative, it is also possible to limit the norm of the correction in (31) to a suitably large constant.

8. It is highly useful to perform an initial scaling of the element Hessians at the first iteration, as is now recommended in most quasi-Newton procedures (see [5]).

A priori advantages of this scheme seem to be its adaptive modelling capability, the fact that it is well suited to deal with large scale applications, and also the observation that sizing in the sense of Dennis, Gay and Welsh [4] is no longer necessary to take into account the decrease (at varying speeds) of the $f_i(x^k)$ to zero, when applied on small residual problems. The reason for this last remark is the explicit and separate presence of these residual sizes in the Hessian approximation (21), as opposed to the globalization induced by the approximation to the whole of $S(x)$ in (15). Interesting questions concerning the proposed procedure are first its practical performance, but also the model choice strategy (among 2^m possible models at every iteration!), and storage considerations when m is large compared to n. These questions will be addressed in the next section, under the light of numerical experience.

3. Tests and their implications. In order to gain some insight about the behavior of the algorithm itself and about the specific questions that were left open above, the routine VE08AD (Harwell Library version of PSPMIN described in [14]), initially written for the general unconstrained partially separable problem, has been adapted to the nonlinear least squares setting. The resulting code then follows the outline discussed in the previous section.

All computations were performed on a DEC2060 in the Facultés Universitaires ND de la Paix in Namur, Belgium. The FORTRA compiler was used, with double precision arithmetic (the machine wordlength is 36 bits). Stopping rules were chosen as in VE08AD, where an extra test was included to detect that the objective function value was close enough to zero. In fact, all successful runs were stopped because $\|g(x)\| < 10^{-7}$ and the overall model for the objective function was convex and/or $f(x) < 10^{-14}$.

A battery of classical test problems has been used in the numerical tests that were performed with this code. Table 1 briefly summarizes them. In this table, the second column indicates whether or not the problem has a zero residual (marked by 0, as opposed to >0). The first seven of these problems are well known, so we do not describe them or provide reference for each one of them individually. The last eight, being less classical extensions of small dimensional test functions, are described in

TABLE 1
Test problems.

Problem	Res. size	Dimension	
		Small	Large
Tridia	0	10, 50, 100	500, 1000
Ext. Rosenbrock	0	10, 50, 100	
Min. Surface	>0	64	464
Nondia	0	10, 50, 100	500, 1000
Boundary value	0	10, 50, 100	
Broyden tridiag	0	10, 50, 100	500, 1000
Broyden banded	0	10, 50, 100	250
Powell singular	0	49, 100	502
Wrong Wood	>0	10, 50, 100	500, 1000
Gaussian	>0	10, 50, 100	500, 1000
Engvall	>0	10, 50, 100	500, 1000
Gragg-Levy	>0	10, 50, 100	500, 1000
Fruedenst.-Roth	>0	10, 50, 100	500, 1000
Schmvt	0	10, 50, 100	500, 1000
Cube	0	10	

the Appendix. An internal report [26] provides all supplementary information to any one that is interested.

This set thus includes 40 problems of dimension at most 100, and 21 larger ones, that is 61 tests in all, with a fair mix of small and larger residual cases.

The first point we wish to address is that of the model choice strategy: at step 2 of the above described procedure, one has to choose, on every iteration and for every element, between a Gauss-Newton type model (20) and a quasi-Newton model (21). Several questions immediately arise:

1. Should the choice depend on the element?
2. What criterion on the models should be used?
3. If a criterion is chosen, should one consider the model incorporating B_i^{k+1} (that is after updating the element Hessians) or the model incorporating B_i^k, as it was when the last step was chosen (that is before updating)?
4. What is the numerical behavior of methods where the Gauss-Newton or quasi-Newton model is systematically chosen? What is the sensitivity of these methods to the size of the residuals at the solution?

We first present results related to the last questions. Two methods are considered: the pure Gauss-Newton and the pure quasi-Newton. In the first, the Gauss-Newton model (20) is chosen for all elements and for each iteration, while in the second it is the model (21) that is always chosen. Table 2 reports the numerical tests made with these choices.

TABLE 2
Pure Gauss-Newton vs. pure quasi-Newton.

Pure Gauss-Newton					
Category	nprob	nsolved	av.dim.	av.it.	av.nfg
Small	40	33	52.2	26.7	37.2
Large	21	17	720.7	29.3	43.5
Pure quasi-Newton					
Category	nprob	nsolved	av.dim.	av.it.	av.nfg
Small	40	40	53.6	19.8	21.6
Large	21	21	701.7	12.1	11.7

In this table, and in those that follow, nprob is the number of problems that were tried with the considered method, nsolved is the number of those that have been solved successfully, av.dim is the average dimension of the solved problems and av.it. (av.nfg resp.) is the average number of iterations (average number of function and gradient evaluations, resp.) required for solution.

This first test set clearly shows that the pure quasi-Newton method is both more reliable and more efficient than the pure Gauss-Newton. The pure quasi-Newton has, in fact, an excellent performance.

It is interesting to observe that, for the pure quasi-Newton method on the large problems, the number of function and gradient calls is smaller than the number of iterations. This is because the code, as VE08AD, exploits the decomposition (17) to avoid recomputing parts of the objective function that are not modified from iteration

to iteration. However, this feature does not depend on the type of model that is chosen, and a similar behaviour could well have appeared for the Gauss–Newton case.

Table 3 shows the behaviour of the same two methods when the level of the residual at the solution ($\|r\|$) varies. The problem chosen in the SCHMVT problem with $n = 10$, whose residual was increased step by step. The sign - indicates that the problem was not run. This table shows the extraordinary sensitivity of the Gauss–Newton approach to the size of the residuals at the solution. It also shows the relative independence of the quasi-Newton method from this factor.

TABLE 3
Variations of residual size.

$\|r\|$	Gauss–Newton		quasi-Newton	
	it.	nfg.	it.	nfg.
0	31	39	25	29
0.0001	>100	>151	17	20
0.001	>100	>147	17	18
0.01	>100	>150	15	17
0.1	>100	>181	13	14
1	>100	>152	11	14
10	-	-	11	17
100	-	-	11	18
1000	-	-	11	18
10000	-	-	11	18

At this point, the quasi-Newton model (21) is clearly favored on average. However, if one looks at the detailed runs, it is soon apparent that the Gauss–Newton choice is surprisingly efficient for some problems although very inefficient for others. In the favorable cases, its efficiency is better than that of the quasi-Newton method. The search for an adaptive model choice strategy, which could take advantage of this fact, is therefore still of interest. However, one clearly has to measure the resulting algorithm against the pure quasi-Newton approach, an already fairly high standard.

We now turn our attention to the model choice problem in this perspective, and examine a possible criterion for this purpose. As discussed above, we now wish to consider a separate model for each element, and therefore we obtain, for the ith element, the models

$$m_i^{GN}(x^k+s^k) = \tfrac{1}{2}f_i^2(x^k) + f_i(x^k)g_i^T(x^k)s^k + \tfrac{1}{2}[g_i^T(x^k)s^k]^2, \tag{34}$$

and

$$m_i^{QN}(x^k+s^k) = m_i^{GN}(x^k+s) + \tfrac{1}{2}f_i(x^k)[s^k]^T B_i^{k+1}s^k, \tag{35}$$

corresponding to the choices (21) and (20), respectively. Due to the complexity of the problem (2^m possible choices for m elements), we propose to determine the preferred model for each element before any attempt to compute a reasonable step is made, and stick to this preference until the next iteration. Following Dennis, Gay and Welsh, we may choose the model for which the value (34) or (35) is closest to $f_i(x^k+s^k)$. This strategy will be referred to as "best fit elementwise." The chosen model (computed in x^{k+1} instead of x^k) is then minimized in the trust region to determine s^{k+1} in step 3 of the algorithm described above.

Observe also that the matrix B_i^{k+1} has been chosen as the Hessian of the local quasi-Newton model (35), instead of the more classical B_i^k. This slightly unorthodox choice is motivated by the following remark: it is the model incorporating B_i^{k+1}, and not B_i^k any more, that will be used in the computation of s^{k+1}, if one assumes that the quasi-Newton model has been selected for the ith element. Furthermore, since we expect this Hessian matrix to be nonsingular on a small dimensional subspace only, the updated B_i^{k+1} may differ substantially from B_i^k. Because of this choice, and because of the secant equation (19), we can now rewrite (35) as

$$m_i^{QN}(x^k+s^k) = m_i^{GN}(x^k+s)+\tfrac{1}{2}f_i(x^k)[s^k]^T y_i^k \tag{36}$$

and we obtain the surprising fact that all references to the quality of the approximation B_i^{k+1} to $S(x^k)$ or $S(x^{k+1})$ have completely disappeared; the preference test between (34) and (36) is now purely a test on the local behavior of the element function $f_i(\cdot)$ itself! This has the advantage of clarifying the respective roles of the trust region (control of the neighbourhood where the quality of the model is adequate) and model choice (determination of the type of model best suited to the considered function).

We now report the numerical results obtained with this best fit elementwise model switching technique: they are contained in Table 4.

It is immediately apparent that the resulting method is fairly robust, but somewhat less efficient than the pure quasi-Newton method. This fact is not caused by our choice of model in (35), as demonstrated by Table 5. This table is indeed typical of other tests that were run to estimate the effect of choosing B_i^{k+1} instead of B_i^k: the first choice has always been, in our experience, marginally more efficient. As a consequence, in the sequel we will use this choice only, and the other possibility will not be discussed further.

Another possible cause of the (relative) lack of success of the best fit elementwise strategy is the relatively frequent oscillation between models on successive iterations. This behaviour was indeed observed in many of the runs reported in Table 4. Thus, it is interesting to introduce some inertia in the model switching technique in the hope of limiting this oscillatory behavior. To this aim we introduced a minimum number of iterations where, for each element, switching from one model to the other is prevented once a new model has just been preferred. Table 6 reports the results obtained with this technique. In this table, the parameter "In." is this minimum number of iterations.

TABLE 4
Best fit elementwise rule.

Category	nprob	nsolved	av.dim.	av.it.	av.nfg
Small	40	40	53.6	21.1	27.0
Large	21	21	701.7	14.2	15.7

TABLE 5
Best fit elementwise rule (where the quasi-Newton models use B_i^k).

Category	nprob	nsolved	av.dim.	av.it.	av.nfg
Small	40	40	53.6	21.2	27.7
Large	21	21	701.7	15.0	19.6

TABLE 6
Effect of inertia on the best fit elementwise rule.

In.	Category	nprob	nsolved	av.it.	av.nfg
2	Small	40	40	20.3	26.6
	Large	21	21	11.7	13.3
3	Small	40	40	19.4	26.2
	Large	21	21	12.0	12.7
4	Small	40	40	19.1	26.0
	Large	21	21	15.7	19.0
5	Small	40	40	19.1	26.3
	Large	21	21	18.3	21.9

The results of Table 4 correspond to the choice In. = 1. The column reporting the average dimensions of the solved problem has been omitted because the orders of magnitude only are important and they have already appeared in the previous tables.

Although some improvement in efficiency is observed for values of the inertia around 3, this improvement is not very impressive and none of these methods performs as well as the pure quasi-Newton algorithm of Table 2.

This seems to indicate that one should probably make a single model choice for all the elements, and give up the freedom of preferring a Gauss-Newton model for one and a quasi-Newton model for the other. The (again relative) lack of success of mixing strategies could indeed well be due to a certain inconsistency in the overall model for the complete objective function.

Two different preference rules can be considered at this point:

1. The model whose prediction best fits the overall function value $f(x^k + s^k)$ is preferred. This rule is a direct extension of the "best fit elementwise," and will be referred to as the "best fit overall" rule. It is also the criterion that is used by Dennis, Gay and Welsh in their code NL2SOL.

2. A model is chosen for each element according to the best fit elementwise rule, and then the final model is chosen as the one that has been selected by a majority of elements. This rule is somewhat more arbitrary, but, as we will see, it is still interesting. It will be referred to as the "best fit majority" rule.

The numerical results using the first of these strategies appear in Table 7. In order to distinguish the influence of possible oscillations between the models, the tests also incorporate the inertia parameter discussed above.

The efficiency is better for this mthod, compared to the best fit elementwise rule of Tables 5 and 6. The robustness of the method is, however, a little bit decreased. Despite the clear improvement in performance on the solved problems, the method is still slightly inferior to the pure quasi-Newton algorithm. However, the difference appears only in a few problems where the Gauss-Newton model is preferred too often over the adaptive strategy.

Results for the best fit majority rule are reported in Table 8.

These last results still reduce the gap with the pure quasi-Newton rule, especially for In. = 1. In fact, the introduction of inertia does not seem to be of much interest when a single model is chosen for all the element, and this idea will therefore be abandoned in the sequel.

TABLE 7
Best fit overall rule.

In.	Category	nprob	nsolved	av.it.	av.nfg
1	Small	40	39	19.4	23.7
	Large	21	21	10.2	11.7
2	Small	40	39	19.3	23.4
	Large	21	21	10.8	11.8
3	Small	40	40	18.7	23.2
	Large	21	21	10.9	12.5
4	Small	40	40	19.4	24.3
	Large	21	21	11.4	12.8
5	Small	40	40	19.5	24.6
	Large	21	21	11.6	15.2

TABLE 8
Best fit majority rule.

In.	Category	nprob	nsolved	av.it.	av.nfg
1	Small	40	39	15.6	21.4
	Large	21	21	11.0	12.4
2	Small	40	38	15.1	21.5
	Large	21	21	11.3	11.8
3	Small	40	40	15.3	21.9
	Large	21	21	11.8	14.0
4	Small	40	40	15.9	22.8
	Large	21	21	11.4	13.7
5	Small	40	40	15.7	22.6
	Large	21	21	12.1	14.7

As for the best fit overall rule, it can be observed that convergence is slowed down on a few problems because the Gauss-Newton model is preferred too often. This suggests the idea of using the Gauss-Newton model only when it is clearly and substantially better than the quasi-Newton model. Hence, we choose to bias the model preference in favor of the quasi-Newton rule: a switch from the quasi-Newton to the Gauss-Newton model is only allowed if this last model is at least 10 times better than the current quasi-Newton choice. The rules corresponding to the best fit overall and best fit majority will be referred to as the "biased best fit overall" and "biased best fit majority" rules, respectively.

The numerical tests seem to indicate that this biasing technique may be advantageous. This conclusion can be drawn from the examination of Table 9, which reports the performance of the two biased adaptive methods.

These last two methods provide the best results up to now. The excellent robustness and efficiency that are demonstrated by the two algorithms are indeed very encouraging.

TABLE 9
Biased adaptive methods.

Biased best fit overall					
Category	nprob	nsolved	av.dim	av.it.	av.nfg
Small	40	40	53.6	18.4	20.6
Large	21	21	701.7	10.1	10.7
Biased best fit majority					
Category	nprob	nsolved	av.dim	av.it.	av.nfg
Small	40	40	53.6	18.0	20.0
Large	21	21	701.7	10.9	11.7

The results are better than those obtained for the pure quasi-Newton method; and this is because the latter methods allow for the use of the Gauss–Newton model when it is clearly the best choice, while offering the good convergence properties of the quasi-Newton technique in the other cases.

Because of the somewhat arbitrary character of the majority rule used in the second method, and because there is very little to say between the two from the efficiency point of view, we prefer the biased best fit overall (BBFO) rule for general use. It is therefore interesting to compare this method with NL2SOL, which is a standard in this field and also uses an adaptive strategy, and VE08AD, a routine designed for general large scale unconstrained optimization. This comparison has been realized on a smaller set of problems with $n = 10$. This dimension is small and does not reveal all the potential of both VE08AD and the new technique for solving large problems. However, it has the advantage of keeping the tests well in the range covered by NL2SOL. In running this comparison, the same accuracy requirements were imposed in VE08AD and the new method (BBFO), and care was taken in choosing the stopping conditions for NL2SOL, such that this last method would not produce much more accurate results. The results of the comparison is presented in Table 10.

In this table, the columns nf and ng (for NL2SOL) contain, respectively, the number of function and gradient calls, as these are reported separately by this code. As can be guessed, it stands for the number of iterations and nfg for the number of combined function and gradient calls needed by the respective methods to achieve the

TABLE 10
Comparison with NL2SOL *and* VE08AD.

Problem	VE08AD		NL2SOL			BBFO	
	it	nfg	it	nf	ng	it	nfg
Tridia	3	5	1	2	2	2	3
Nondia	20	31	7	10	8	5	6
Broyden banded	29	50	6	7	7	6	7
Wrong Wood	30	36	7	9	8	8	9
Gragg-Levy	71	76	19	20	20	15	17
Freudenst.-Roth	24	31	7	10	8	7	9
Cube	48	72	70	103	71	76	86
Ext. Rosenbrock	18	30	27	27	24	28	34

required accuracy. One has also to keep in mind, when looking at these numbers, that VE08AD is not designed to take advantage of the structure of the nonlinear least squares problem, in contrast with the two other algorithms.

One feels entitled to conclude from this comparison that, for the small dimensional problems in our test set, the performance of the new method BBFO is certainly comparable to that of NL2SOL. The expected advantage of using the least squares special structure is also apparent when comparing BBFO and VE08AD.

We now address the last remaining question raised above, namely that of storage costs. As mentioned, the new method(s) uses the storage scheme of partially separable problems and partitioned updating. The advantage and drawbacks of this technique already have been discussed in [12] and [13], and we will not reproduce this discussion here. However, the main conclusions are that this scheme allows the computations with problems involving several thousands of variables, and that it is comparable with the more classical sparse matrix technology. The fact that we solved problems with 1000 variables for the above tests, is a reasonable proof to this claim.

However, it happens in least squares applications that m is much larger than n, even if n is already large. These applications lead then to unacceptably important storage requirements, despite the use of advanced techniques. To overcome this problem, we know of practical cases where several observations are "merged" together to form an aggregate element in (1). More precisely, a set of element functions $\{f_i(x)\}_{i\in J}$ is replaced in the sum (1) by the new element function

$$f_J(x)\sqrt{\sum_{i\in J} f_i^2(x)}, \tag{37}$$

where the set J contains selected integers in the range $1, \cdots, m$. The size m may then be substantially reduced, but one can expect some deterioration in the efficiency of the algorithm that is used for the computation. This question is of particular importance when, as above, one uses a technique that separates the elements as much as possible and takes each residual specifically into account. In fact, this merging technique is related to the much more general problem of finding the "best" (or a "better") decomposition of the form (17), given a function $f(x)$. The word "best" can be interpreted here in the sense of "most efficient," in terms of the computational resources. As this problem is very general, and probably quite complicated, we restrict our attention here to the case where a compromise between storage and efficiency is obtained by merging the elements of the initial decomposition together in the simple manner just described in (37).

To gain some insight into the question, three problems were considered and tested at various levels of merging. They are described in the Appendix. It is enough at this point to know that the first involves a mix of trigonometric and exponential nonlinearities ($n = 50$), and that the second and third involve quadratic observations ($n = 50,200$). We also remark that their elements may be summed up to a certain point without affecting the row structure of the sparsity pattern associated with the problem. In the partial separability context, this is to say that, up to a certain point, the subspaces $\mathcal{N}_i$ are not reduced in dimension.

The results of these merging tests, presented in Tables 11 and 12, show the expected deterioration of performance as storage shrinks (this quantity is measured by the length of the double precision array used to store the successive B_i^k). In these examples, elements are always merged with their immediate neighbours in the ordering specified by the problem definition. It is clear that the tests are somewhat extreme: aggregating more than 1000 observations into about 50 is probably a doubtful practice. However,

TABLE 11
Performance versus storage for a trigonometric-exponential example.

n	m	it	nfg	storage	cpu(s)
50	572	14	15	16016	91
	308	12	13	8624	51
	220	12	13	6160	42
	132	23	24	3696	72
	44	79	105	1232	237
	22	83	114	792	233
	11	115	164	605	317
	7	151	198	582	398

TABLE 12
Performance versus storage for a quadratic example.

n	m	it	nfg	storage	cpu(s)
50	282	15	16	2820	56
	141	18	19	1410	39
	94	20	21	940	32
	47	34	36	470	37
	16	32	35	330	20
	8	44	60	351	23
200	1182	18	20	11820	453
	591	23	24	5910	283
	394	23	24	3940	196
	197	45	47	1970	346
	66	37	48	1380	93
	50	41	50	1382	104

in our examples at least, even these extreme cases do not behave too badly as storage requirements decrease sharply. Since the amount of internal linear algebra at each iteration also decreases with storage, we also report, in the column cpu(s), the runtimes of our algorithms.

Caution must be exercised when interpreting these measures: one has to remember that the cost of computing the objective function is negligible for the test cases, in contrast with many real world applications, and also that the code has not been optimized for dealing with the merging technique. However, one sees that it could be interesting, in applications where evaluating the $f_i(x)$ is cheap, to use this type of idea to reduce the total computing time. Our present experience is nevertheless too limited to discuss this point any further.

It is interesting to note that, before $m = 22$ for the first problem and $m = 16$ and 66 for the second problem, the merging causes no modification of the $\mathcal{N}_i$, in contrast to the following merging levels where these subspaces are reduced in dimension. One can see, in our tests, that this distinction is less relevant in practice than expected. The fact that storage increases again for small values of m is due to the deterioration of the sparsity pattern.

More sophisticated ways to deal with the merged elements obviously can be designed. For example, one can use a Dennis-Gay-Welsh type approximation (with sizing factor) for their Hessians. The question of the choice of elements to merge is

also worth some study. If some of them are linear with identical sparsity pattern, one could apply the ideas of Liu [17]. Postprocessing of linear elements as suggested by Golub and Pereyra [11] might also be considered. However, it is not our purpose to explore this topic in length, but we like to point out that there might be valuable developments in this direction.

4. Conclusion. In this paper, we showed how to extend derivative algorithms for small dimensional nonlinear least squares to large scale problems. In particular, ideas of Dennis, Gay and Welsh and Brown and Dennis have been combined with the concepts of partially separable optimization and partitioned updating into a proposal for adaptive methods suited for large scale computations.

Some new methods are proposed and tested on a set of test problems including cases that involve several hundreds of variables. At least one of these methods is shown to be efficient and reliable for these computations.

A preliminary study of merging procedures for the observations also shows the compromise between storage and efficiency in the cases where the number of observations by far exceeds that of parameters.

Further research in these merging techniques is finally suggested. Other topics for future work include, for example, a finer analysis of the involved linear algebra in its relation to conditioning and the study of a hierarchical partially separable function of the form

$$f(x) = \sum_{i=1}^{m} h_i\left(\sum_{j=1}^{s} f_{ij}(x)\right). \tag{38}$$

This last study could then lead, via the classical penalty or augmented Lagrangian techniques, to the solution of large scale nonlinear minimization problems with many nonlinear constraints.

Finally, we would like to mention the fact that the algorithms described above lend themselves very well to parallel computing, as already observed in [14]: every element has its separate model and the structure of the linear system (22) is naturally interesting for this type of computations. The lack of adequate hardware has unfortunately made a practical parallel implementation impossible, but one may clearly expect significant savings in computing time in this context.

Appendix A. Eight test problems with variable dimension.

A.1. Extended Powell singular function.

$$m = \tfrac{4}{3}(n-1),$$
$$f_{4i-3}(x) = x_{3i-2} + 10x_{3i-1},$$
$$f_{4i-2}(x) = \sqrt{5}(x_{3i} - x_{3i+1}),$$
$$f_{4i-1}(x) = (x_{3i-1} - 2x_{3i})^2,$$
$$f_{4i}(x) = \sqrt{10}(x_{3i-2} - x_{3i+1}),$$

for $i = 1, \cdots, (n-1)/3$, and with the starting point defined as

$$x_1 = 3,$$
$$x_{3i-1} = -1,$$
$$x_{3i} = 0,$$
$$x_{3i+1} = 1,$$

for $i = 1, \cdots, m$.

A.2. Wrong Wood function.

$$m = 3(n-2),$$
$$f_{6i-5}(x) = 10(x_{2i} - x_{2i-1}^2),$$
$$f_{6i-4}(x) = 1 - x_{2i-1},$$
$$f_{6i-3}(x) = 3\sqrt{10}(x_{2i+2} - x_{2i+1}^2),$$
$$f_{6i-2}(x) = 1 - x_{2i+1},$$
$$f_{6i-1}(x) = \sqrt{10}(x_{2i} + x_{2i+2} - 2),$$
$$f_{6i}(x) = \sqrt{10}(x_{2i} - x_{2i+2}),$$

for $i = 1, \cdots, n-2$ and with the starting point defined as

$$x_i = \begin{cases} -3 & \text{if mod } (i, 2) = 1, \\ -1 & \text{if mod } (i, 2) = 0, \end{cases}$$

for $i = 1, \cdots, n$.

A.3. Gaussian function.

$$m = n - 2,$$
$$f_i^2(x) = \left(\frac{10}{m} + x_{i+2}^2\right)\left(2 - \exp\left(\frac{-(x_i - x_{i+1})^2}{0.1 + x_{i+2}^2}\right)\right),$$

for $i = 1, \cdots, m$ and with all the components of the starting point being equal to 3.

A.4. Extended Engvall function.

$$m = n - 1,$$
$$f_i^2(x) = (x_i + x_{i+1}^2)^2 - 4x_i + 3,$$

for $i = 1, \cdots, m$ and with all the components of the starting point being equal to 2.

A.5. Extended Gragg–Levy function.

$$m = \tfrac{5}{2}(n-2),$$
$$f_{5i-4}(x) = (x_{2i} - e^{x_{2i-1}})^2,$$
$$f_{5i-3}(x) = 10(x_{2i} - x_{2i+1})^3,$$
$$f_{5i-2}(x) = \tan^2(x_{2i+1} - x_{2i+2}),$$
$$f_{5i-1}(x) = x_{2i-1}^4,$$
$$f_{5i}(x) = x_{2i+1} - 1,$$

for $i = 1, \cdots, (n-2)/2$, and with all the components of the starting point being equal to 2.

A.6. Extended Freudenstein and Roth function.

$$m = 2(n-1),$$
$$f_{2i-1}(x) = x_i + x_{i+1}((5 - x_{i+1})x_{i+1} - 2) - 13,$$
$$f_{2i}(x) = x_i + x_{i+1}((1 + x_{i+1})x_{i+1} - 14) - 29,$$

for $i = 1, \cdots, n-1$ and with all the components of the starting point being equal to -2.

A.7. Extended Schmvt function.

$$m = n-2,$$

$$f_i^2(x) = 3 - \frac{1}{1+(x_i - x_{i+1})^2} - \sin\left(\frac{x_{i+1}x + x_{i+2}}{2}\right) - \exp\left(\frac{x_i + x_{i+2}}{x_{i+1}} - 2\right),$$

for $i = 1, \cdots, m$ and with all the components of the starting point being equal to 0.5.

A.8. Extended Cube function.

$$m = 2(n-1),$$

$$f_{2i-1}(x) = 10(x_{i+1} - x_i^3),$$

$$f_{2i}(x) = 1 - x_i,$$

for $i = 1, \cdots, n-1$, and with all the components of the starting point being equal to -2.

Appendix B. Merging test problems.

B.1. Exponential-trigonometric.

$$m = 13(n-6),$$

$$s(i) = \min(\max[\text{mod}(i, 13) - 2, 1], 7),$$

$$r(i) = \lceil i/13 \rceil,$$

$$c(i) = 3s(i)/10,$$

$$q(i) = r(i) + s(i) - 1,$$

$$t(i) = \begin{cases} r(i)+s(i) & \text{if } s(i) < 7, \\ r(i) & \text{if } s(i) = 7, \end{cases}$$

$$f_i(x) = [1 + \cos(c(i))][x_{t(i)} - \sin(x_{q(i)}) - 1 + \sin(1)]^2$$

$$+ 5[x_{q(i)} - 1] \exp\{\sin(c(i))x_{t(i)}\} + \tfrac{1}{2} \sum_{j=r(i)}^{r(i)+6}{}^{*} [\sin(x_j) - \sin(1)],$$

for $i = 1, \cdots, m$, where $\lceil z \rceil$ is the smallest integer larger or equal to z, and where the $\sum^*$ is taken over all indices different from $q(i)$ and $t(i)$. All components of the starting vector are equal to 5.

B.2. Quadratic.

$$m = 6(n-3),$$

$$f_{6i-5}(x) = x_i + 3x_{i+1}(x_{i+2} - 1) + x_{i+3}^2 - 1,$$

$$f_{6i-4}(x) = (x_i + x_{i+1})^2 + (x_{i+2} - 1)^2 - x_{i+3} - 3,$$

$$f_{6i-3}(x) = x_i x_{i+1} - x_{i+2}x_{i+3},$$

$$f_{6i-2}(x) = 2x_i x_{i+2} + x_{i+1}x_{i+3} - 3,$$

$$f_{6i-1}(x) = (x_i + x_{i+1} + x_{i+2} + x_{i+3})^2 + (x_i - 1)^2,$$

$$f_{6i}(x) = x_i x_{i+1} x_{i+2} x_{i+3} + (x_{i+3} - 1)^2 - 1,$$

for $i = 1, \cdots, n-3$, and with all components of the starting point being equal to 5.

REFERENCES

[1] M. AL-BAALI, *Methods for nonlinear least squares*, Ph.D. dissertation, Univ. Dundee, 1984.

[2] M. C. BARTHOLOMEW-BIGGS, *The estimation of the Hessian matrix in nonlinear least-squares problems with zero residual*, Math. Programming, 12 (1977), pp. 67-80.

[3] K. M. BROWN AND J. E. DENNIS, *A new algorithm for nonlinear least squares curve fitting*, in Mathematical Software, J. R. Rice, ed., Academic Press, New York, 1971, pp. 391-396.

[4] J. E. DENNIS, D. M. GAY AND R. E. WELSH, *An adaptive nonlinear least squares algorithm*, ACM Trans. Math. Software, 7 (1981), pp. 348-368.

[5] J. E. DENNIS AND R. B. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[6] J. E. DENNIS, S. SONGBAI AND P. A. VU, *A memoryless augmented Gauss-Newton method for nonlinear least squares problems*, Technical Report 85-1, Dept. Math. Sci., Rice Univ., Houston, TX, 1985.

[7] R. S. DEMBO, S. C. EISENSTAT AND T. STEIHAUG, *Inexact Newton methods*, SIAM J. Numer. Anal., 19 (1982), pp. 400-408.

[8] J. A. GEORGE AND J. W. LIU, *Computer Solution of Large Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[9] P. E. GILL AND W. MURRAY, *Algorithms for the solution of the nonlinear least squares problem*, SIAM J. Numer. Anal., 15 (1978), pp. 977-992.

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

[11] G. H. GOLUB AND V. PEREYRA, *The differentiation of the pseudo-inverse and the non-linear least squares problems whose variables separate*, SIAM J. Numer. Anal., 10 (1973), pp. 413-432.

[12] A. GRIEWANK AND PH.L. TOINT, *Partitioned variable metric updates for large structured optimization problems*, Numer. Math., 39 (1982), pp. 119-137.

[13] ———, *On the unconstrained optimization of partially separable functions*, in Nonlinear Optimization 1981, M. J. D. Powell, ed., Academic Press, London, 1982.

[14] ———, *Numerical experiments with partially separable optimization problems*, in Numerical Analysis: Proceedings, Dundee, 1983, D. F. Griffiths, ed., Lecture Notes in Mathematics 1066, Springer-Verlag, Berlin, 1984, pp. 203-220.

[15] M. T. HEATH, *Numerical methods for large linear least squares problems*, this Journal, 5 (1984), pp. 497-513.

[16] P. LINDSTRÖM AND P. A. WEDIN, *A new linesearch algorithm for non-linear least squares problems*, Math. Programming, 29 (1984), pp. 268-296.

[17] J. W. LIU, *On general row merging schemes for sparse Givens transformations (revised)*, Technical Report 83-03, Dept. Comput. Sci., York Univ., Ontario, Canada, 1984.

[18] E. MARWIL, *Exploiting sparsity in Newton-like methods*, Ph.D. dissertation, Cornell Univ., Ithaca, NY, 1978.

[19] J. J. MCKEOWN, *Specialized versus general purpose algorithms for minimizing functions that are sums of squared terms*, Math. Programming, 9 (1975), pp. 57-68.

[20] J. J. MORÉ, *The Levenberg-Marquardt algorithm: implementation and theory*, in Proceedings, Dundee, 1977, G. A. Watson, ed., Lecture Notes in Mathematics, Springer-Verlag, Berlin, 1978.

[21] C. V. 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.

[22] T. STEIHAUG, *The conjugate gradient method and trust regions in large scale optimization*, SIAM J. Numer. Anal., 20 (1983), pp. 626-637.

[23] A. RUHE, *Accelerated Gauss-Newton algorithms for nonlinear least squares problems*, BIT (1979), pp. 356-367.

[24] PH.L. TOINT, *On sparse and symmetric matrix updating subject to a linear equation*, Math. Comp., 31 (1977), pp. 954-961.

[25] ———, *Numerical solution of large sets of algebraic nonlinear equations*, Math. Comp., 46 (1986), pp. 175-189.

[26] ———, *Test problems for partially separable optimization and results for the routine* PSPMIN, Technical Report 83/4, Dept. Math., FUNDP, Namur, Belgium, 1983.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 3, May 1987
© 1987 Society for Industrial and Applied Mathematics
014

A GENERALIZED PULSE-SPECTRUM TECHNIQUE (GPST) FOR DETERMINING TIME-DEPENDENT COEFFICIENTS OF ONE-DIMENSIONAL DIFFUSION EQUATIONS*

X. Y. LIU†‡ AND Y. M. CHEN†

Abstract. The applicability of the generalized pulse-spectrum technique (GPST) is demonstrated for the solutions of inverse problems of one-dimensional linear diffusion equations with time-dependent coefficients. Numerical simulations are carried out to test the feasibility and to study the general characteristics of this technique without real measurement data. It is found that the GPST also gives excellent results and is as robust as it is for solving the time-independent coefficient inverse problems.

Key words. generalized pulse-spectrum technique, time-dependent coefficient, inverse problems, diffusion equation

AMS(MOS) subject classifications. 35R30, 35R25

1. Introduction. The generalized pulse-spectrum technique (GPST) [1] is a versatile and efficient iterative numerical algorithm for solving inverse problems (to determine the unknown coefficients, sources, initial-boundary values, and geometries of the space-domain from additionally measured data or auxiliary conditions in the space-time domain or the space-frequency domain) of a system of nonlinear partial differential equations. The "pulse" and "spectrum" of GPST symbolize that GPST can solve inverse problems in the space-time domain and the space-frequency domain, respectively. In general, inverse problems of partial differential equations can be formulated as ill-posed nonlinear operator equations. It is also important to point out that the GPST is not a single narrowly defined iterative numerical algorithm but a broad class of iterative numerical algorithms based on the concept that either the nonlinear operator equation is first linearized by any one of the Newton-like iteration methods and then each iterate is solved by using any one of the smoothing methods to overcome the ill-posedness, e.g., the Tikhonov's regularization method [2], the Backus and Gilbert method [3], etc., the smoothing method is first applied to the nonlinear operator equation and then the smoothed nonlinear problem is solved by using a Newton-like iteration method. Hence different choices of various Newton-like iteration methods and smoothing methods lead to different special forms of GPST. The choice of a specific Newton-like iteration method and smoothing method and the question of whether to solve the inverse problem in the space-time domain or in the space-frequency domain depend mainly on the particular inverse problem under consideration. Other than the basic structure of GPST, the efficiency of GPST will then depend upon that how efficiently one can treat every minute step in numerical algorithm.

It has been demonstrated that the GPST iterative numerical algorithm does give excellent results in solving the time-dependent coefficient inverse problems of one-, two-, and three-dimensional second-order linear evolutional (parabolic and hyperbolic) partial differential equations in the space-complex frequency domain [4]-[10] and in

* Received by the editors October 17, 1984, and in revised form February 18, 1986.

† Department of Applied Mathematics and Statistics, State University of New York, Stony Brook, New York 11794.

‡ Permanent address: Department of Mathematics, Peking University, Beijing, People's Republic of China. The work of this author was supported by the Committee on Educational Exchange with China at the State University of New York at Stony Brook as the 1983-84 Fong Shu-chuen Fellow.

the space-time domain [11]. Similarly, time-independent coefficient inverse problems of a weakly nonlinear wave equation [12] and a fourth-order plate equation [13] can be solved by using the GPST with equal efficacy. Furthermore, the inverse problem of determination of the geometry of the space-domain of the Helmholtz equation has been solved efficiently by using the GPST [14]. On the other hand, many numerical and analytic methods for constructing approximate solutions of inverse problems of the diffusion equations have been developed by researchers in the past and present. In particular, the questions of existence and uniqueness of the solution, and techniques for constructing approximate solutions of the inverse problem of the one-dimensional diffusion equation, have been treated by Jones [15], Douglas and Jones [16], Cannon [17], [18], and Cannon and DuChateau [19]-[22]. For two-dimensional cases, history matching techniques based on the minimization of functionals of the differences between the observations and the computed solutions of the diffusion equation have been developed by Chen, Gavalas, Seinfeld and Wasserman [23], Chavent, DuPay and Lemonnier [24], and Chang and Yeh [25].

The main purpose of the present paper is to demonstrate the applicability of a special GPST to solve the inverse problems of a one-dimensional diffusion equation with time-dependent coefficients which have not been solved before. For simplicity in the next section the formulation of the inverse problem of a linear one-dimensional diffusion equation is presented and the basic numerical algorithm of GPST is given. Then numerical simulations are carried out to test the feasibility and to study the intrinsic characteristics of this numerical algorithm with artificially generated data. Finally, a discussion of the GPST and its performance is given.

2. Numerical algorithm (GPST). Consider the following initial-boundary value problem of a one-dimensional linear diffusion equation:

$$\partial[k(x,t)\partial u/\partial x]/\partial x - \partial u/\partial t = 0, \qquad 0<x<1, \quad 0<t\leqq T<\infty,$$

$$u(x,0)=\phi(x), \qquad 0\leqq x\leqq 1, \tag{1}$$

$$u(0,t)=f(t), \quad u(1,t)=g(t), \quad 0\leqq t\leqq T<\infty,$$

and the auxiliary conditions,

$$\partial u(0,t)/\partial x = h(t), \quad \partial u(1,t)/\partial x = \psi(t), \quad 0\leqq t\leqq T<\infty, \tag{2}$$

where the time-dependent coefficient

$$k(x,t)=A(x)\cdot B(t). \tag{3}$$

The inverse problem here is to determine the unknown coefficient $k(x,t)$ from the known data $\phi(x)$, $f(t)$, $g(t)$, $h(t)$ and $\psi(t)$. Because the unknown coefficient $k(x,t)$ is also a function of t, one cannot perform the Laplace or Fourier transformation of (1) and hence a GPST in the space-time domain will be used to solve this inverse problem. The reason for choosing this simple expression of $k(x,t)$ is purely economical, because for this simple case, only $(M+1)+(J+1)$ unknowns, $A(x_i)$, $B(t_j)$, $i=1,2,3,\cdots,M+1$, $j=1,2,3,\cdots,J+1$, need to be found, and $(M+1)\times(J+1)$ unknowns, $k(x_i,t_j)$, need to be found for the general case. Of course, the implementations for these two cases will be different.

GPST calls for setting

$$u_{n+1}=u_n+\delta u_n, \quad A_{n+1}=A_n+\delta A_n, \quad B_{n+1}=B_n+\delta B_n, \quad n=0,1,2,\cdots, \tag{4}$$

where $A_0(x)$ and $B_0(t)$ are the initial guesses of $A(x)$ and $B(t)$, respectively,

$$\|\delta u_n\|<\|u_n\|, \quad \|\delta A_n\|<\|A_n\|, \quad \|\delta B_n\|<\|B_n\|.$$

Upon substituting (4) into (1) and neglecting terms of order δ^2 and higher, one obtains a system for $u_n(x, t)$,

$$\begin{aligned} &\partial[A_n(x)B_n(t)\partial u_n/\partial x]/\partial x-\partial u_n/\partial t=0, \quad 0<x<1, \quad 0<t\leqq T,\\ &u_n(x,0)=\phi(x), \qquad 0\leqq x\leqq 1,\\ &u_n(0,t)=f(t), \quad u_n(1,t)=g(t), \quad 0\leqq t\leqq T, \end{aligned} \tag{5}$$

and a system for $\delta u_n(x, t)$, the δ-system,

$$\begin{aligned} &\partial[A_n(x)B_n(t)\partial\delta u_n/\partial x]/\partial x-\partial\delta u_n/\partial t\\ &\quad=-B_n(t)\partial[\delta A_n(x)\partial u_n/\partial x]/\partial x, \qquad 0<x<1,\\ &\quad -\delta B_n(t)\partial[A_n(x)\partial u_n/\partial x]/\partial x, \qquad 0<t\leqq T,\\ &\delta u_n(x,0)=0, \qquad 0\leqq x\leqq 1,\\ &\delta u_n(0,t)=\delta u_n(1,t)=0, \qquad 0\leqq t\leqq T. \end{aligned} \tag{6}$$

By using the method of Green's function, the δ-system (6) can be changed to an integral relation,

$$\begin{aligned} \delta u_n(x,t)=\int_0^t\int_0^1 G_n(x,t;x',t')[&B_n(t')\partial(\delta A_n(x')\partial u_n/\partial x')/\partial x'\\ &+\delta B_n(t')\partial(A_n(x')\partial u_n/\partial x')/\partial x']\,dx'\,dt',\\ &0\leqq t\leqq T, \end{aligned} \tag{7}$$

where $-G_n(x, t; x', t')$ is the Green's function of the δ-system (6). Upon differentiating (7) with respect to x, setting $x=0$ and $x=1$, replacing $\partial u_{n+1}/\partial x$ by $\partial u/\partial x$ for the purpose of accelerating the rate of convergence, and finally using the auxiliary conditions (2), one obtains a system of two Fredholm integral equations of the first kind for the two unknowns $\delta A_n(x)$ and $\delta B_n(t)$,

$$\begin{aligned} &\int_0^t\int_0^1 \partial G_n(0,t;x',t')/\partial x[B_n\partial(\delta A_n\partial u_n/\partial x')/\partial x'+\delta B_n\partial(A_n\partial u_n/\partial x')/\partial x']\,dx'\,dt'\\ &\qquad=h(t)-\partial u_n(0,t)/\partial x,\\ &\int_0^t\int_0^1 \partial G_n(1,t;x',t')/\partial x[B_n\partial(\delta A_n\partial u_n/\partial x')/\partial x'+\delta B_n\partial(A_n\partial u_n/\partial x')/\partial x']\,dx'\,dt'\\ &\qquad=\psi(t)-\partial u_n(1,t)/\partial x, \qquad 0\leqq t\leqq T. \end{aligned} \tag{8}$$

Equations (4), (5) and (8) form the basic structure for each iteration in the iterative numerical algorithm of GPST.

The initial-boundary value problem (5) can be solved by using the following second-order finite-difference method,

$$\begin{aligned} u_{n,i}^{j+1}-u_{n,i}^j=\frac{\Delta t}{2}\Bigg\{&A_{n,i}B_n^{j+1}\frac{(u_{n,i-1}^{j+1}-2u_{n,i}^{j+1}+u_{n,i+1}^{j+1})}{(\Delta x)^2}+A_{n,i}B_n^j\frac{(u_{n,i-1}^j-2u_{n,i}^j+u_{n,i+1}^j)}{(\Delta x)^2}\\ &+B_n^{j+1}\frac{(A_{n,i-1}-A_{n,i+1})}{2\Delta x}\,\frac{(u_{n,i-1}^{j+1}-u_{n,i+1}^{j+1})}{2\Delta x}\\ &+B_n^j\frac{(A_{n,i-1}-A_{n,i+1})}{2\Delta x}\,\frac{(u_{n,i-1}^j-u_{n,i+1}^j)}{2\Delta x}\Bigg\}, \end{aligned} \tag{9}$$

where $u^j_{n,i} = u_n(x_i, t_j)$, $A_{n,i} = A_n(x_i)$ and $B^j_n = B_n(t_j)$, $i = 2, \cdots, M$, $j = 1, \cdots, J$. Instead of solving for the Green's function of (6) by the above finite difference method and discretizing the Fredholm integral equations of the first kind (8), one can first discretize the δ-system (6) by using a finite difference method similar to (9) and then obtain a linear algebraic system for the unknowns $A_{n,i}$ and B^j_n, $i = 1, 2, \cdots, M+1$, $j = 1, 2, \cdots, J+1$. To summarize the discretized systems, one has for (5),

$$\begin{aligned} &\underline{\underline{C}}^j_n \cdot \underline{u}^{j+1}_n = \underline{\underline{D}}^j_n \cdot \underline{u}^j_n + \underline{H}^j_n, \qquad j = 1, 2, 3, \cdots, J, \\ &\underline{u}^1_n = \underline{\phi}, \end{aligned} \tag{10}$$

where

$$(\underline{u}^j_n)^T = (u^j_{n,2}, u^j_{n,3}, u^j_{n,4}, \cdots, u^j_{n,M}), \qquad (\underline{\phi})^T = (\phi(x_2), \phi(x_3), \phi(x_4), \cdots, \phi(x_M)),$$

and $\underline{H}^j_n$ is determined by the boundary values f^j and g^j, $\underline{\underline{C}}^j_n$ and $\underline{\underline{D}}^j_n$ are $(M-1)\times(M-1)$ tridiagonal matrices with the ith rows given by

$$(0, \cdots, 0, -(1+S_{n,i+1}), 2(1+R_{n,i+1}/B^{j+1}_n), -(1-S_{n,i+1}), 0, \cdots, 0)$$

and

$$(0, \cdots, 0, (1+S_{n,i+1})Q^j_n, -2(1-R_{n,i+1}/B^j_n)Q^j_n, (1-S_{n,i+1})Q^j_n, 0, \cdots, 0),$$

respectively, $Q^j_n = B^j_n/B^{j+1}_n$, $j = 1, 2, \cdots, J$, $R_{n,i} = (\Delta x)^2/\Delta t A_{n,i}$, and $S_{n,i} = (A_{n,i-1} - A_{n,i+1})/4A_{n,i}$, $i = 2, 3, \cdots, M$.

The following finite difference approximations for $\partial u_n/\partial x$ are used,

$$\begin{aligned} &-(3u^j_{n,1} - 4u^j_{n,2} + u^j_{n,3})/2\Delta x \sim \partial u_n(0, t)/\partial x, \qquad j = 1, 2, \cdots, J+1, \\ &(u^j_{n,M-1} - 4u^j_{n,M} + 3u^j_{n,M+1})/2\Delta x \sim \partial u_n(1, t)/\partial x. \end{aligned} \tag{11}$$

Similarly, the δ-system becomes

$$\begin{aligned} &\underline{\underline{C}}^j_n \cdot \delta\underline{u}^{j+1}_n = \underline{\underline{D}}^j_n \cdot \delta\underline{u}^j_n + \underline{\underline{E}}^j_n \cdot \begin{pmatrix} \delta\underline{A}_n \\ \delta\underline{B}_n \end{pmatrix}, \qquad j = 1, 2, \cdots, J, \\ &\delta\underline{u}^1_n = 0, \qquad \delta u^j_{n,1} = \delta u^j_{n,M+1} = 0, \quad j = 1, 2, \cdots, J+1, \end{aligned} \tag{12}$$

where $(\delta\underline{A}_n)^T = (\delta A_{n,1}, \delta A_{n,2}, \cdots, \delta A_{n,M+1})$, $(\delta\underline{B}_n)^T = (\delta B^2_n, \delta B^3_n, \cdots, \delta B^{J+1}_n)$,

$$\underline{\underline{E}}^j_n = (\underline{\underline{H}}^j_n \,\vdots\, \underline{\underline{K}}^j_n),$$

where $\underline{\underline{H}}^j_n$ is a $(M-1)\times(M+1)$ upper tridiagonal matrix with the ith row given by

$$(0, \cdots, 0, \underbrace{-\tfrac{1}{4}(b^{j+1}_{n,i+1} + Q^j_n b^j_{n,j+1})}_{i\text{th column}}, a^{j+1}_{n,i+1} + Q^j_n a^j_{n,i+1}, \tfrac{1}{4}(b^{j+1}_{n,i+1} + Q^j_n b^j_{n,i+1}), 0, \cdots, 0)$$

and $\underline{\underline{K}}^j_n$ is a $(M-1)\times J$ bi-column matrix with the nonzero $(j-1)$th and jth columns given by

$$(B^{j+1}_n)^{-1}((a^j_{n,2} + S_{n,2}b^j_{n,2}), (a^j_{n,3} + S_{n,3}b^j_{n,3}), \cdots, (a^j_{n,M} + S_{n,M}b^j_{n,M}))^T$$

and

$$(B^{j+1}_n)^{-1}((a^{j+1}_{n,2} + S_{n,2}b^{j+1}_{n,2}), (a^{j+1}_{n,3} + S_{n,3}b^{j+1}_{n,3}), \cdots, (a^{j+1}_{n,M} + S_{n,M}b^{j+1}_{n,M}))^T,$$

respectively, with

$$a^j_{n,i} = u^j_{n,i-1} - 2u^j_{n,i} + u^j_{n,i+1} \quad \text{and} \quad b^j_{n,i} = u^j_{n,i-1} - u^j_{n,i+1}.$$

In a straightforward manner, (12) can be reduced to

$$\delta \underline{u}_n^{j+1} = (\underline{\underline{C}}_n^j)^{-1} \cdot \underline{\underline{F}}_n^j \cdot \begin{pmatrix} \delta \underline{A}_n \\ \hline \delta \underline{B}_n \end{pmatrix}, \qquad j = 1, 2, \cdots, J,$$

$$\delta \underline{u}_n^1 = 0, \quad \delta u_{n,1}^j = \delta u_{n,M+1}^j = 0, \quad j = 1, 2, \cdots, J+1, \tag{13}$$

where

$$\underline{\underline{F}}_n^1 = \underline{\underline{E}}_n^1, \ \underline{\underline{F}}_n^{j+1} = \underline{\underline{D}}_n^{j+1} \cdot (\underline{\underline{C}}_n^{j+1})^{-1} \cdot \underline{\underline{F}}_n^j + \underline{\underline{E}}_n^{j+1}, \qquad j = 1, 2, \cdots, J-1.$$

It is clear that (13) is a de facto discretization of (7).

Upon incorporating (11) and (13) into (8), one obtains a linear algebraic system for the unknown parameters $\delta A_n(x)$ and $\delta B_n(t)$,

$$\underline{\underline{A}}_n \cdot \begin{pmatrix} \delta \underline{A}_n \\ \hline \delta \underline{B}_n \end{pmatrix} = \underline{V}_n \tag{14}$$

subject to the continuity condition of $B(t)$ at $t=0$, $\delta B_n^2 = \delta B_n^1 + B_n^1 - B_n^2$. By using Tikhonov's regularization method, the regularized solution of (14) satisfies

$$(\underline{\underline{A}}_n^T \cdot \underline{\underline{A}}_n + \underline{\underline{\Lambda}}_n) \cdot \begin{pmatrix} \delta \underline{A}_n \\ \hline \delta \underline{B}_n \end{pmatrix} = \underline{\underline{A}}_n^T \cdot \underline{V}_n \tag{15}$$

where $\underline{\underline{\Lambda}}_n$ is the nonnegative diagonal regularizing-parameter matrix.

Now, (4), (10) and (15) form the basic structure for each iteration in the iterative numerical algorithm of GPST.

3. Numerical simulation. In order to test the feasibility and to study the general characteristics of the GPST computational algorithm for solving the inverse problems of the one-dimensional diffusion equation with time-dependent coefficients without real measurement data, the following numerical simulation procedure is carried out. A direct problem is solved for a given coefficient $k^*(x, t) = A^*(x)B^*(t)$, and the GPST algorithm is used to see whether k^* is recovered or not. First, one chooses a $k^*(x, t)$ which represents the correct diffusion coefficient, and also one chooses the initial condition $\phi(x)$ and the boundary conditions $f(t)$ and $g(t)$ which represent a part of the measured data. Then the initial-boundary value problem (1) is solved by the finite difference method (9); thus one generates the rest of the data $h(t)$ and $\psi(t)$ for the GPST algorithm by the finite difference approximation (11). Next, $A_0(x)$ and $B_0(t)$ are chosen. Hence upon solving (4), (10) and (15) numerically, $A_1(x)$ and $B_1(t)$ are obtained. $A_2(x)$ and $B_2(t)$ can be obtained in a similar manner. One continues this procedure until finally a numerical limit $A_N(x)B_N(t)$ is reached. Other than the truncation, round-off, and finite difference approximation errors in both generating the numerical data and computing $k_N(x, t)$, any norm $\|k^*(x, t) - k_N(x, t)\|$ can be used as a criterion for evaluating the performance of the computational algorithm of GPST. As a matter of fact, due to the uncontrollable nature of these errors, one can consider them to be equivalent to some kind of pseudo-noise in the measured data.

The numerical simulation here is carried out for a large class of $A^*(x)$, $B^*(t)$, $A_0(x)$ and $B_0(t)$, e.g., constants, piecewise-linear continuous functions and oscillatory functions. The closed intervals $0 \leqq x \leqq 1$ and $0 \leqq t \leqq 2$ are divided into ten and twenty equal subintervals, respectively, i.e., $\Delta x = \Delta t = 0.1$, $\Delta x/L = 0.1$ and $\Delta t/T = 0.05$. For simplicity, the initial-boundary conditions $\phi(x) \equiv g(t) \equiv 0$ and $f(t) = 0.5\,(1 - \cos 2\pi t)$, $0 \leqq t \leqq 2$, are used.

The numerical results are plotted in Figs. 1-10. (Comparison of the calculated $k_N(x, t)$ · · · · · · and the exact $k^*(x, t)$ ———, with the initial guess $k_0(x, t)$ ------.) The ratios

$$\operatorname*{Max}_{x,t} |k^*(x, t) - k_0(x, t)| / \operatorname*{Max}_{x,t} |k^*(x, t)| \quad \text{and} \quad \operatorname*{Max}_{x,t} |k^*(x, t) - k_N(x, t)| / \operatorname*{Max}_{x,t} |k^*(x, t)|$$

of these examples are tabulated in Table 1.

The L_2 norms, $I_n = \|k^*(x, t) - k_n(x, t)\|_2$, $n = 0, N$, for all cases are tabulated in Table 2. Initially, the elements of $\underline{\Delta}_n$, $\lambda_{n,i} = 1 \times 10^{-4}$, $i = 1, 2, \cdots, M+1$, $\lambda_{n,j} = 1 \times 10^{-8}$, $j = 1, 2, \cdots, J$, are chosen and their values are reduced in the subsequent iterations.

4. Discussion. Although only a small number of computational zones are used in the numerical simulation here, the numerical results in Figs. 1-10 have demonstrated that the GPST iterative numerical algorithm does give excellent results in solving the inverse problems of the one-dimensional diffusion equation with time-dependent coefficients and it is as robust as for all of the time-independent coefficient inverse problems [4]-[14]. Moreover, as for all previous cases

$$\operatorname*{Max}_{x,t} |k^*(x, t) - k_N(x, t)| / \operatorname*{Max}_{x,t} |k^*(x, t)| \sim \operatorname{Max} (\Delta x / L, \Delta t / T).$$

Determination of the optimal or proper values of the regularization parameters $\lambda_{n,i}$'s is the most elusive but important problem. In spite of many methods for determining λ's that appear in the literature, there is still a lack of simple and effective methods to provide an a priori estimate of λ, because they are problem-dependent. Experience combined with a simple trial-and-error procedure in numerical simulation will probably be the most effective way to deal with this problem. This is analogous to the practical procedure for determining the "Q" in the method of artificial viscosity.

Mathematically, the inverse problem is a nonlinear problem regardless of whether the original partial differential equation is linear or nonlinear. Hence in general the solution of an inverse problem with a minimum number of constraints is not unique in the global sense. To be sure, the GPST iterative numerical algorithm is not a method for settling the question of the uniqueness of the solution of an inverse problem. Rather, it is a constructive method for constructing one of the approximate solutions of the inverse problem. This solution is unique locally in the sense of being the closest one to the initial guess in the L_2 norm. Moreover, it is clear from our numerical

TABLE 1

Fig. #	1	2	3	4	5	6	7	8	9	10
N	4	6	9	5	4	10	4	6	7	4
$\operatorname*{Max}_{x,t} \lvert k^* - k_0 \rvert / \operatorname*{Max}_{x,t} \lvert k^* \rvert$	1.25	0.81	1.63	1.25	1.25	0.56	0.56	0.56	0.56	0.56
$\operatorname*{Max}_{x,t} \lvert k^* - k_N \rvert / \operatorname*{Max}_{x,t} \lvert k^* \rvert$	0.08	0.09	0.12	0.13	0.08	0.14	0.05	0.07	0.13	0.12

TABLE 2

Fig. #	1	2	3	4	5	6	7	8	9	10
N	4	6	9	5	4	10	4	6	7	4
I_0	0.86	1.10	1.09	0.91	1.03	1.10	1.03	0.90	0.76	0.91
I_N	0.06	0.07	0.09	0.08	0.06	0.14	0.09	0.08	0.15	0.13

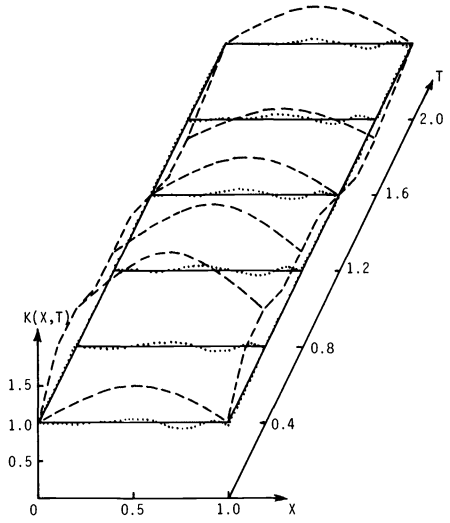


FIG. 1

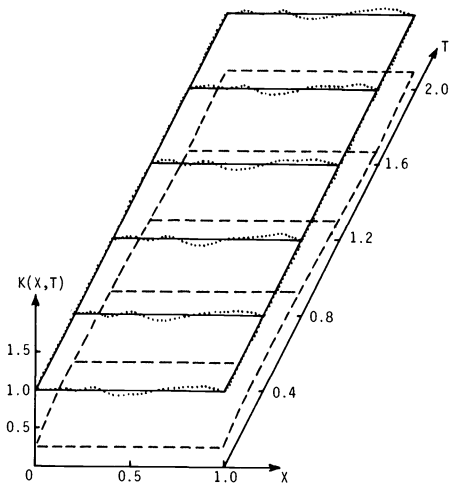


FIG. 2

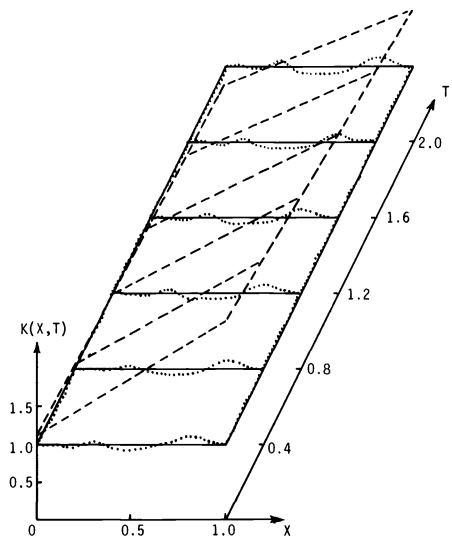


FIG. 3

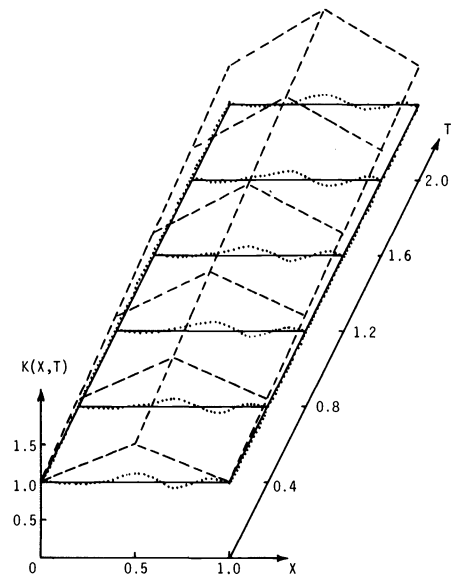


FIG. 4

FIGS. 1-10. *Comparison of the calculated* $k_N(x, t)$ ..... *and the exact* $k^*(x, t)$ ———, *with the initial guess* $k_0(x, t)$ -------.

simulation that for different initial guess $k_0(x, t)$ the iterations converge to slightly different $k_N(x, t)$'s. However, if this numerical algorithm is reasonably robust, then any one of the approximate solutions will be an acceptable approximation. This numerical computation phenomenon can be attributed to the accumulation of the

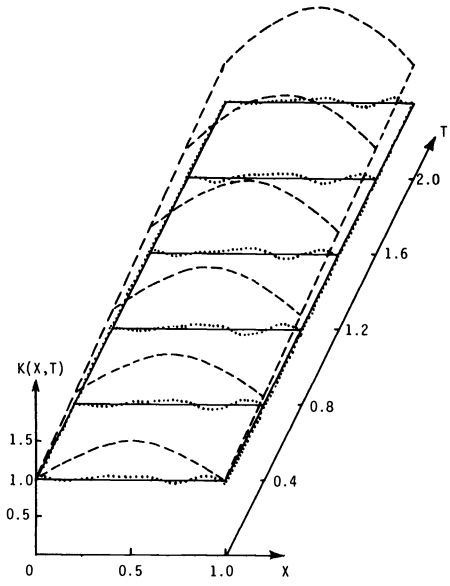


FIG. 5

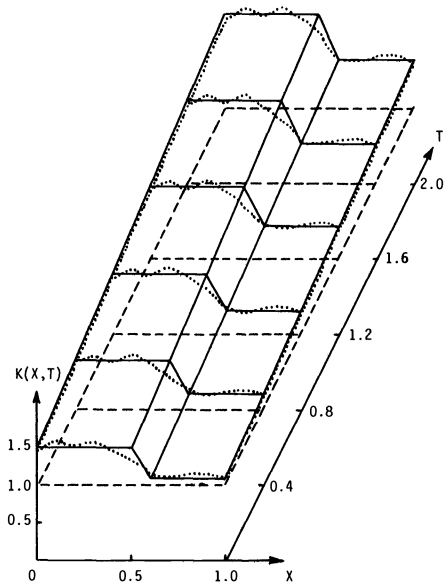


FIG. 6

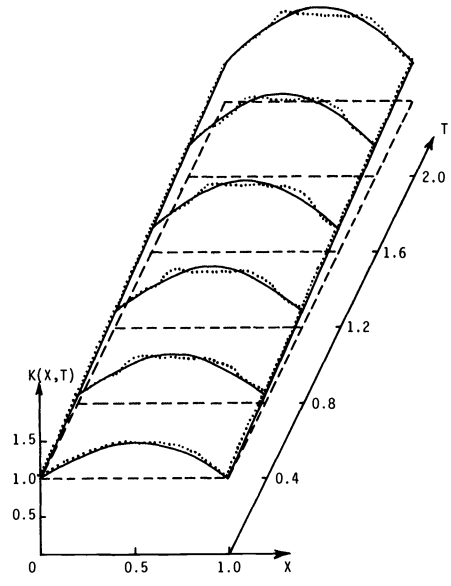


FIG. 7

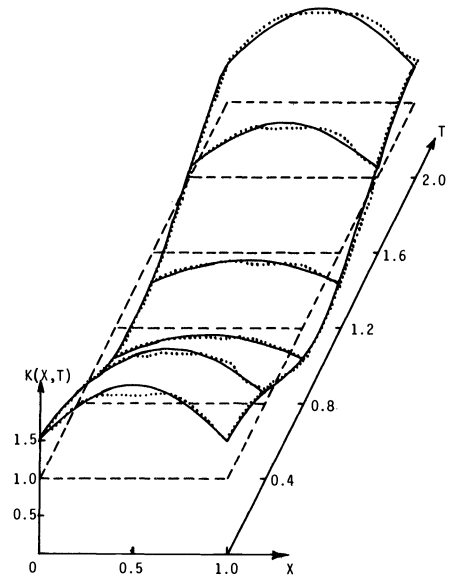


FIG. 8

nonnegligible errors in computing each iterate. Here, most of these computational errors come from the regularization procedure in solving the ill-posed system (8) or (14).

Efforts to carry out the generalization of this GPST iterative numerical algorithm to time-dependent multi-coefficients inverse problems of two- and three-dimensional diffusion equations are under way and their results will be reported in the near future.

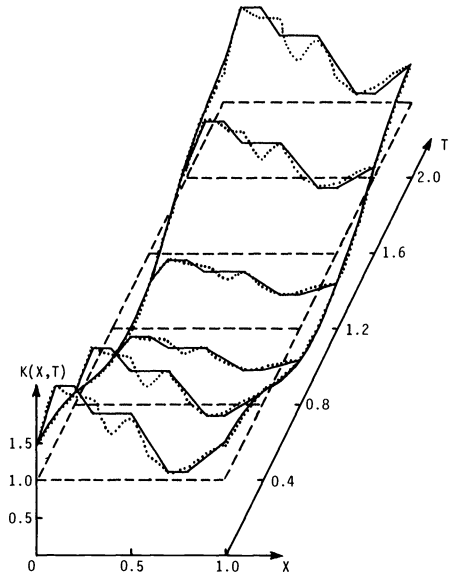


FIG. 9

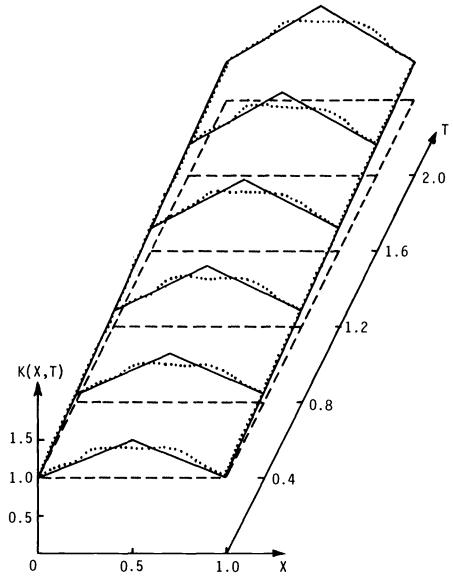


FIG. 10

REFERENCES

[1] Y. M. CHEN, *Generalized pulse-spectrum technique* (GPST) *for solving problems in parameter identification*, Proc. U.S.-China Workshop on Advances in Computational Engng. Mech., Dalian, China, September 5-9, 1983.

[2] A. N. TIKHONOV AND V. Y. ARSENIN, *Solutions of Ill-Posed Problems*, John Wiley, New York, 1977.

[3] G. BACKUS AND F. GILBERT, *The resolving power of gross earth data*, Geophys. J. R. Astron. Soc., 16 (1968), pp. 169-205.

[4] Y. M. CHEN AND D. S. TSIEN, *A numerical algorithm for remote sensing of density profiles of a simple ocean model by acoustic pulses*, J. Comput. Phys., 25 (1977), p. 366.

[5] Y. M. CHEN AND J. Q. LIU, *A numerical algorithm for remote sensing of thermal conductivity*, J. Comput. Phys., 43 (1981), pp. 315-326.

[6] ———, *A numerical algorithm for solving inverse problems of two-dimensional wave equations*, J. Comput. Phys., 50 (1983), pp. 193-208.

[7] J. Q. LIU AND Y. M. CHEN, *An iterative algorithm for solving inverse problems of two-dimensional diffusion equations*, this Journal, 5 (1984), pp. 255-269.

[8] Y. M. CHEN AND J. Q. LIU, *An iterative numerical algorithm for solving multi-parameter inverse problems of evolutional partial differential equations*, J. Comput. Phys., 53 (1984), pp. 429-442.

[9] Y. M. CHEN AND G. Q. XIE, *A numerical method for simultaneous determination of bulk modulus, shear modulus and density variations for nondestructive evaluation*, Nondestructive Testing Communication, 1 (1984), pp. 125-135.

[10] G. Q. XIE AND Y. M. CHEN, *A modified pulse-spectrum technique for solving inverse problems of two-dimensional elastic wave equation*, Appl. Numer. Math., 1 (1985), pp. 217-237.

[11] Y. N. TANG AND Y. M. CHEN, *Application of* GPST *algorithm to history matching of single-phase simulator models*, in Adv. in Computer Meth. for Part. Dif. Eqs., V, R. Vichnevetsky and R. Stepleman, eds., IMACS, 1984, pp. 433-439.

[12] R. P. HATCHER AND Y. M. CHEN, *An iterative method for solving inverse problems of a nonlinear wave equation*, this Journal, 4 (1983), pp. 149-163.

[13] C. H. QIU AND Y. M. CHEN, *Inverse problems for elastic plates with variable flexural rigidity*, Internat. J. Solids and Structures, to appear.

[14] Y. M. CHEN AND S. L. WANG, *An efficient numerical method for determination of shapes, sizes and orientations of flaws for nondestructive evaluation*, in Rev. of Progress in Quantitative Nondestructive Evaluation 4, D. O. Thompson and D. E. Chimenti, eds., Plenum Press, New York, 1985, pp. 543-549.

[15] B. F. JONES, JR., *The determination of a coefficient in a parabolic differential equation Part* I. *Existence and uniqueness*, J. Math. Mech., 11 (1962), pp. 907-918.

[16] J. DOUGLAS, JR. AND B. F. JONES, JR., *The determination of a coefficient in a parabolic differential equation Part* II. *Numerical approximation*, J. Math. Mech., 11 (1962), pp. 919-926.

[17] J. R. CANNON, *Determination of an unknown coefficient in a parabolic differential equation*, Duke Math. J., 30 (1963), pp. 313-323.

[18] ———, *Determination of certain parameters in heat conduction problems*, J. Math. Anal., 8 (1964), pp. 188-201.

[19] J. R. CANNON AND P. DUCHATEAU, *Determination of unknown physical properties in heat conduction problems*, Internat. J. Engng. Sci., 11 (1973), pp. 783-794.

[20] ———, *Determination of unknown coefficients in a nonlinear heat conduction problem*, SIAM J. Appl. Math., 24 (1973), pp. 298-314.

[21] ———, *Determination of unknown coefficients in parabolic operators from overspecified initial-boundary data*, J. Heat Transfer, 100 (1978), pp. 503-507.

[22] ———, *An inverse problem for a nonlinear diffusion equation*, SIAM J. Appl. Math., 39 (1980), pp. 272-289.

[23] W. H. CHEN, G. R. GAVALAS, J. H. SEINFELD AND M. L. WASSERMAN, *A new algorithm for automatic history matching*, Soc. Petrol. Engng. J., 257 (1974), pp. 593-608.

[24] M. G. CHAVENT, M. DUPAY AND L. LEMONNIER, *History matching by use of optimal theory*, Soc. Petrol. Engng. J., 259 (1975), pp. 74-86.

[25] S. CHANG AND W. YEH, *A proposed algorithm for the solution of the large-scale inverse problem in groundwater*, Water Resource Res., 12 (1976), pp. 365-374.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 3, May 1987

© 1987 Society for Industrial and Applied Mathematics

ESTIMATING NONLINEAR MODELS BY MAXIMUM LIKELIHOOD FOR THE EXPONENTIAL FAMILY*

M. R. OSBORNE†

Abstract. Many but not all attractive properties of generalized linear models associated with the exponential family of distributions are destroyed by nonlinearity. A consequence is that ensuring the stability of a computational process for maximizing the likelihood becomes relatively more important. Here it is shown that trust region methods for solving nonlinear least squares problems are readily adapted to maximize likelihoods based on the exponential family, and that the nice theoretical results available for the nonlinear least squares problem also generalize.

Key words. maximum likelihood, exponential family, GLIM, Gauss-Newton algorithm, method of scoring, law of large numbers, line search, trust region

AMS (MOS) subject classifications. 62, 65

1. Introduction. The 1972 paper of Nelder and Wedderburn, which formalized a general approach to the estimation of generalized linear models, followed by the implementation of their algorithm in the problem-solving language, GLIM, has had a major impact on statistical practice. One result has been that considerable ingenuity has been applied to fit problems into the GLIM framework. However, this approach may not always be profitable and one case where it can be worthwhile to consider convenient alternatives is presented here. The major restrictions in GLIM include the following.

(i) The likelihood assumes that the underlying probability distribution is a member of the exponential family. Specifically it is assumed that the model is characterized by

(a) A dependent variable z with distribution given by

$$p(z\,|\,\theta,\phi)=\exp\left(\alpha(\phi)\{z\theta-g(\theta)+h(z)\}+\eta(\phi,z)\right) \tag{1.1}$$

where θ is the parameter of the distribution, ϕ could be a nuisance parameter, and $\alpha(\phi)>0$;

(b) A linear prediction model

$$Y=\sum_{i=1}^{P} x_i y_i \tag{1.2}$$

where the x_i (the independent variables or covariates in the representation of Y) are functions of the independent variables of the modelling process (for example, time), and the y_i are the components of the parameter vector $\mathbf{y}$ that specifies the particular realization of the model;

(c) A link function (nonlinear, in general)

$$\theta=f(Y), \tag{1.3}$$

which connects the parameter θ of the distribution of z with the Y of the linear model. Restrictions on f are implied by the assumptions required for Theorem 2.2 and discussed in Remark 2.1. Typically f is monotonic. (From our point of view, what is restrictive here is the linearity of (1.2) rather than the specific distributional assumptions. In fact,

* Received by the editors April 1, 1985; accepted for publication (in revised form) March 8, 1986. This work was carried out while the author was visiting the Department of Mathematics and Statistics at the University of New Mexico, Albuquerque, New Mexico 87131. It was supported in part by the National Science Foundation under grant 127199.

† Department of Statistics, Institute of Advanced Studies, Australian National University, Canberra, Australia ACT 2601.

the weakening of the distributional assumptions to allow estimation procedures based on quantities called quasi-likelihoods has been considered by several authors (a useful survey is given in McCullagh and Nelder [1983]). However, it is not clear that this generalization is compatible with our requirements (we expand this cryptic comment in Remark 2.3), while a unified treatment is possible if the restriction to the exponential family is accepted.)

(ii) The algorithm implemented in GLIM is naive in the sense that it implements Fisher's method of scoring without recourse to such stabilizing devices as a line search to ensure that the increment in the current estimate of $\mathbf{y}$ does increase the likelihood. This may surprise the numerical analyst who recognizes the method of scoring as a generalization of the Gauss-Newton algorithm for nonlinear least squares and is well aware of the importance of such devices in this context. However, this may not be a problem with GLIM as originally envisaged for the following reasons:

(a) The log likelihood frequently has nice concavity properties (Pratt [1981]);

(b) It is often the case that the natural choice of the link function improves the numerical performance of the algorithm (acting as a linearizing transformation, at least locally (McCullagh and Nelder [1983, p. 33]);

(c) The direct generalization of results such as those of Jennrich [1969] for the nonlinear least squares problem guarantee convergence of the naive algorithm (perhaps from a good enough starting point) provided sufficient "appropriate" observations are made on the dependent variable z.

Relaxing the design constraints on GLIM to permit nonlinear models is unlikely to be kind to advantages (a) and (b) above, but (c) can be expected to be largely unaffected (the point is considered further in § 2). Thus, provided there is sufficient data (and there is an intimate connection between the conditions under which the method of scoring works well and the conditions under which satisfactory inferences can be made on the parameter estimates), we can expect to determine $\mathbf{y}$ satisfactorily given good enough starting values. But if we do not have such starting values, we are likely to be back in the situation where stabilizing devices are going to be needed.

In this paper, the problem of providing these stabilizing devices is attacked by giving a generalization of the Levenberg algorithm; this is a direct analogue of Fisher's generalization of the Gauss-Newton algorithm. The main result is that the good global convergence behaviour of the Levenberg algorithm is preserved for problems derived from the exponential family of distributions. Also, minimal changes need to be made to algorithms for solving nonlinear least squares problems to implement our generalization. To this extent, we appear to have gained by going outside the GLIM framework. Other papers in which the numerical properties of algorithms applied to the exponential family are considered include Dennis [1981] and Gay [1980]. I am indebted to a referee for these references.

That the ability of GLIM to handle nonlinear problems could be improved is hardly a new observation. However, the emphasis here is on the need to do rather better than the naive scoring method when estimating nonlinear models. One might argue also (as does Cox [1984]) for more appropriate organizational facilities within GLIM to help in setting up these problems, but that is a different point.

2. Modifying the method of scoring. In this section, Fisher's method of scoring is introduced and a standard modification to permit a line search to be used as a stabilizing device is summarized. Introducing the expectation operator E, we write the nonlinear model as

$$E\{z\}(t) = \mu(t, \mathbf{y}) \tag{2.1}$$

where t is the independent variable in the modelling process, and where the connection between the mean μ and variance V of the dependent variable z and the parameter of the distribution can be established by using the identities

$$E\left\{\frac{\partial L}{\partial \theta}\right\}=0 \tag{2.2}$$

and

$$E\left\{\frac{\partial^2 L}{\partial \theta^2}\right\}=-E\left\{\frac{\partial L}{\partial \theta}\right\}^2 \tag{2.3}$$

where $L(z, \theta, \phi)=\log p$. Differentiation of L gives

$$\frac{\partial L}{\partial \theta}=\alpha(\phi)(z-g'(\theta)) \tag{2.4}$$

so that, using (2.2),

$$\mu=g'(\theta). \tag{2.5}$$

Differentiating again and applying the second identity gives

$$\alpha(\phi)g''(\theta)=\alpha(\phi)^2(z-\mu)^2 \tag{2.6}$$

so that

$$V=\frac{g''(\theta)}{\alpha(\phi)}. \tag{2.7}$$

It is convenient to reparameterize in terms of μ, and this is possible provided V does no vanish. Here this gives

$$\frac{\partial L}{\partial \mu}=\frac{\partial L}{\partial \theta}\frac{d\theta}{d\mu}=\frac{\alpha(z-\mu)}{g''(\theta)}=\frac{z-\mu}{V(\mu)}. \tag{2.8}$$

Also

$$\frac{\partial^2 L}{\partial \mu^2}=\frac{\partial^2 L}{\partial \theta^2}\left(\frac{d\theta}{d\mu}\right)^2+\frac{\partial L}{\partial \theta}\frac{d^2\theta}{d\mu^2} \tag{2.9}$$

so that (taking expectations and using (2.2))

$$\begin{aligned} E\left\{\frac{\partial^2 L}{\partial \mu^2}\right\} &= E\left\{\frac{\partial^2 L}{\partial \theta^2}\right\}\left(\frac{d\theta}{d\mu}\right)^2=-E\left\{\frac{\partial L}{\partial \theta}\right\}^2\left(\frac{d\theta}{d\mu}\right)^2 \\ &= -E\left\{\frac{\partial L}{\partial \mu}\right\}^2=-\frac{1}{V}. \end{aligned} \tag{2.10}$$

Given data on z (say independent observations z_i made at time points t_i, $i=1, 2, \cdots, n$), then the method of maximum likelihood suggests that $\mathbf{y}$ be determined by minimizing

$$K(\mathbf{y})=-\sum_{i=1}^{n} L(z_i, \mu(t_i, \mathbf{y}))=-\sum_{i=1}^{n} L_i. \tag{2.11}$$

It will be convenient to assume that (2.11) has an isolated minimum $\hat{\mathbf{y}}_n$ where the subscript will be suppressed unless the number of data points is relevant. We denote the true parameter vector by $\mathbf{y}^*$.

Newton's method for minimizing $K(\mathbf{y})$ leads to the iteration

$$\mathbf{y}^{(k+1)} = \mathbf{y}^{(k)} - \nabla^2 K(\mathbf{y}^{(k)})^{-1} \nabla K(\mathbf{y}^{(k)})^T. \tag{2.12}$$

This is modified in the method of scoring to give the iteration

$$\mathbf{y}^{(k+1)} = \mathbf{y}^{(k)} - M(\mathbf{y}^{(k)})^{-1} \nabla K(\mathbf{y}^{(k)})^T \tag{2.13}$$

where

$$\nabla K = -\sum_{i=1}^{n} \frac{\partial L_i}{\partial \mu} \nabla \mu_i \tag{2.14}$$

and

$$M(\mathbf{y}) = \sum_{i=1}^{n} \frac{1}{V(\mu_i)} \nabla \mu_i^T \nabla \mu_i \tag{2.15}$$

so that, in particular,

$$M(\mathbf{y}^*) = E\{\nabla^2 K(\mathbf{y}^*)\}. \tag{2.16}$$

This replacement of the Hessian of K by its expectation is the characteristic feature of the method of scoring.

If we now define the matrix $A(\mathbf{y}): R^p \to R^n$ (with $A_k = A(\mathbf{y}^{(k)})$) by

$$\mathbf{e}_i^T A = -\frac{1}{V(\mu_i)^{1/2}} \nabla \mu_i, \qquad i = 1, 2, \cdots, n, \tag{2.17}$$

and the vector $\mathbf{f}(\mathbf{y}) \in R^n$ by

$$\mathbf{e}_i^T \mathbf{f} = \frac{1}{V(\mu_i)^{1/2}} (z_i - \mu_i), \tag{2.18}$$

then (2.13) is equivalent to updating $\mathbf{y}$ using the solution $\mathbf{h}$ to the linear least squares problem

$$\min_{h} \mathbf{r}^T \mathbf{r}: \mathbf{r} = \mathbf{f} + A\mathbf{h}. \tag{2.19}$$

To stabilize either Newton's method or the method of scoring, it is usual not to accept $\mathbf{h}^{(k)}$, but rather to use it to define a direction of descent in which K can be decreased. The calculation which provides the basis for this in the method of scoring is summarized below (here the norm is the euclidean norm and cond (A) is the spectral condition number of A).

LEMMA 2.1.

$$\frac{\nabla K \mathbf{h}}{\|\nabla K\| \, \|\mathbf{h}\|} \leqq \frac{-1}{\text{cond}\,(A)^2}. \tag{2.20}$$

This result shows that the method of scoring will produce satisfactory descent directions under reasonable assumptions on the problem. Thus we consider the iteration

$$\mathbf{y}^{(k+1)} = \mathbf{y}^{(k)} + \lambda_k \mathbf{h}^{(k)} \tag{2.21}$$

where $\lambda_k > 0$ is chosen to make a useful reduction in $K(\mathbf{y})$. One convenient way to achieve this is by use of the Goldstein test which chooses λ_k to satisfy

$$0 < \sigma \leqq \frac{K(\mathbf{y}^{(k)}) - K(\mathbf{y}^{(k)} + \lambda \mathbf{h}^{(k)})}{-\lambda \nabla K(\mathbf{y}^{(k)})\mathbf{h}^{(k)}} \leqq 1 - \sigma \tag{2.22}$$

where $\sigma < .5$. Under reasonable smoothness conditions on K, this condition can be satisfied always, and it is now comparatively straightforward to obtain convergence results (for example, by following the arguments of Goldstein [1965]).

THEOREM 2.1. *Let successive iterates* (2.21) *satisfy* (2.22) *and be confined to a compact region of R^p in which A has full rank and the second derivatives of K are continuous. Then limit points of the iteration are stationary points of $K(\mathbf{y})$.*

This is quite a satisfactory result as it shows how the use of a line search can exorcise some of the problems of a poor initial guess. However, convergence obtained by restricting the size of λ to ensure small steps in $\mathbf{y}$ proves to be unsatisfactory. Thus it is significant that an excellent rate of convergence can be anticipated under reasonable conditions on the t_i and μ provided n is chosen large enough (this is advantage (c) referred to in the previous section). The idea is to treat the method of scoring as a perturbation of Newton's method and to show that the perturbation is small almost surely for large n. Differentiability and boundedness of deterministic quantities and independence and boundedness of the variance of random variables are assumed.

LEMMA 2.2. *Let the iterative process*

$$F(\mathbf{y}_i)\mathbf{h}_i = \mathbf{w}(\mathbf{y}_i), \qquad \mathbf{y}_{i+1} = \mathbf{y}_i - \mathbf{h}_i \tag{2.23}$$

converge to $\hat{\mathbf{y}}$ with second order rate of convergence and consider the perturbed process

$$G(\mathbf{y}_i)\mathbf{k}_i = \mathbf{w}(\mathbf{y}_i), \qquad \mathbf{y}_{i+1} = \mathbf{y}_i - \mathbf{k}_i. \tag{2.24}$$

It is assumed that $F(\mathbf{y})$, $G(\mathbf{y}): R^p \to R^p$ have full rank. Then $\hat{\mathbf{y}}$ is a point of attraction for the perturbed process provided

$$\pi\{G(\hat{\mathbf{y}})^{-1}(F(\hat{\mathbf{y}}) - G(\hat{\mathbf{y}}))\} < 1 \tag{2.25}$$

where π is the spectral radius of the indicated matrix. The convergence rate is first order with multiplier equal to π.

Proof. This follows by considering both iterations started from a displacement from $\hat{\mathbf{y}}$ small enough that the second order process converges in a single step to working accuracy but is otherwise arbitrary. Now if F, G are nonsingular then $\mathbf{w}(\hat{\mathbf{y}}) = 0$. It follows that under this small displacement ($\boldsymbol{\delta}$, say)

$$F(\hat{\mathbf{y}})\mathbf{h} = \nabla \mathbf{w}\boldsymbol{\delta} + O(\|\boldsymbol{\delta}\|^2),$$

and

$$G(\hat{\mathbf{y}})\mathbf{k} = \nabla \mathbf{w}\boldsymbol{\delta} + O(\|\boldsymbol{\delta}\|^2).$$

But, by assumption,

$$\mathbf{h} = \boldsymbol{\delta} + O(\|\boldsymbol{\delta}\|^2)$$

so that

$$\begin{aligned} F\mathbf{h} &= F\boldsymbol{\delta} + O(\|\boldsymbol{\delta}\|^2) \\ &= \nabla \mathbf{w}\boldsymbol{\delta} + O(\|\boldsymbol{\delta}\|^2). \end{aligned}$$

Thus the amount by which (2.24) fails to correct for the perturbation $\boldsymbol{\delta}$ is

$$\begin{aligned}\mathbf{k}-\boldsymbol{\delta} &= \{G^{-1}\nabla\mathbf{w}-I\}\boldsymbol{\delta}+O(\|\boldsymbol{\delta}\|^2)\\ &= G^{-1}(F-G)\boldsymbol{\delta}+O(\|\boldsymbol{\delta}\|^2)\end{aligned}$$

and $\boldsymbol{\delta}$ small enough is otherwise arbitrary.

Comparing the method of scoring with Newton's method, we see that

$$G^{-1}\{F-G\} = M^{-1}\{\nabla^2 K - M\} = \left(\left(\frac{1}{n}M\right)^{-1}\left\{\frac{1}{n}(\nabla^2 K - M)\right\}\right). \tag{2.26}$$

To estimate these terms, we make the following assumptions:

(a) The maximum likelihood estimates are consistent so that

$$\hat{\mathbf{y}}_n \to \mathbf{y}^* \text{ a.s.}, \qquad n\to\infty; \tag{2.27}$$

(b) $M(\hat{\mathbf{y}}_n)/n \to$ positive definite limit, $n\to\infty$.

For a discussion of these conditions and their interrelations in the context of nonlinear least squares, see Jennrich [1969] and Wu [1981].

THEOREM 2.2. *The spectral radius* π *in* (2.25) *satisfies*

$$\pi \to 0 \text{ a.s.}, \qquad n\to\infty. \tag{2.28}$$

Proof. As $(M/n)^{-1}$ behaves itself by assumption as $n\to\infty$ it is necessary to show that the remaining terms in (2.23) become small. We have

$$\begin{aligned}\frac{1}{n}(\nabla^2 K - M)(\hat{\mathbf{y}}_n) &= \frac{1}{n}(\nabla^2 K - M)(\mathbf{y}^*)+O(\|\hat{\mathbf{y}}_n-\mathbf{y}^*\|)\\ &= \frac{1}{n}\sum_{i=1}^{n}-\left(\frac{\partial^2 L_i}{\partial\mu^2}-E\left\{\frac{\partial^2 L_i}{\partial\mu^2}\right\}\right)\nabla\mu_i^T\nabla\mu_i+O(\|\hat{\mathbf{y}}_n-\mathbf{y}^*\|) \qquad (2.29)\\ &\to 0 \text{ a.s.}\end{aligned}$$

as a consequence of the assumed independence of the z_i by using consistency and the strong law of large numbers.

Remark 2.1. This result shows that the ultimate rate of convergence of the method of scoring will be very satisfactory provided n is large enough. Thus it is worth stressing the connection already mentioned between the satisfactory performance of the method of scoring and satisfactory inferences concerning the parameter values. This comes about because both depend on π, which has a geometric interpretation in terms of the intrinsic curvature of the likelihood surface. This suggests that ill conditioning in whatever form is a signal indicating trouble in a wider context, so that if the iterative method has to struggle to obtain convergence the results may not be worth having anyway. Here this is offered as an excuse for strengthening assumption (b) by assuming that M is well conditioned in the vicinity of $\mathbf{y}^*$ provided n is large enough.

Remark 2.2. Theorem 2.2 also permits us to show that the line search test will be satisfied by choosing $\lambda = 1$ provided the current iterate is close to the solution in the sense that $|\nabla K\mathbf{h}|$ is small enough and n is large enough. The argument expands the

test functional in Taylor series ignoring terms of higher order than the second. This gives

$$\frac{K(\mathbf{y})-K(\mathbf{y}+\mathbf{h})}{-\nabla K(\mathbf{y})\mathbf{h}}=1-\frac{\frac{1}{2}\mathbf{h}^T\nabla^2 K\mathbf{h}}{-\nabla K\mathbf{h}}$$

$$=1-\frac{1}{2}\frac{\nabla KM^{-1}\nabla^2 KM^{-1}\nabla K^T}{\nabla KM^{-1}\nabla K^T}$$

$$=1-\frac{1}{2}\frac{\nabla K(I+M^{-1}(\nabla^2 K-M))M^{-1}\nabla K^T}{\nabla KM^{-1}\nabla K^T}$$

$$=\frac{1}{2}(1-\gamma)$$

where

$$\gamma=\frac{\mathbf{w}^T M^{-1/2}(\nabla^2 K-M)M^{-1/2}\mathbf{w}}{\mathbf{w}^T\mathbf{w}}$$

and

$$\mathbf{w}=M^{-1/2}\nabla K^T.$$

Thus γ is in the span of the eigenvalues of $M^{-1/2}(\nabla^2 K-M)M^{-1/2}$ so that $|\gamma|\leqq \pi+O(\|\mathbf{y}-\hat{\mathbf{y}}\|)$.

Remark 2.3. The method of scoring in the form developed here is capable of wider applicability. For example, the basic formula (2.8) is used in Wedderburn [1974] as a point of departure for developing quasi-likelihoods. However, the problem of starting with a derivative condition such as this is that extra work must be done to obtain a suitable objective function if a line search is needed to stabilize the computation.

3. A Levenberg-like modification. It is probably not controversial to suggest that the modified Gauss-Newton algorithm has been largely replaced for nonlinear least squares problems by variants of a class of methods that have become known as trust region methods. These methods modify the linear subproblem (2.17) by introducing a constraint on the size of $\mathbf{h}$. This has the following advantages:

(a) The resulting problem for $\mathbf{h}$ can often be better conditioned;

(b) By adjusting the constraint, an alternative to the line search is obtained for stabilizing the algorithm.

Trust region methods appear to lend themselves readily to nice treatments of convergence behaviour under a minimum of local assumptions (Moré [1982], Powell [1983]). Here an adaptation of perhaps the earliest of these methods (due originally to Levenberg [1943]) is described. This choice was dictated by the availability of the program (Osborne [1976]) that was used to implement the example discussed in the next section, but there appear to be no major problems in adapting alternative procedures (for example, the method of Moré [1978]).

The starting point is (2.19), modified to include the bound on the length of $\mathbf{h}$. This gives

$$\min_{\mathbf{h}} \mathbf{r}^T\mathbf{r}\colon \mathbf{r}=\mathbf{f}+A\mathbf{h}, \qquad \mathbf{h}^T\mathbf{h}\leqq\kappa. \tag{3.1}$$

A direct application of the Kuhn-Tucker conditions shows that this problem is equivalent to the unconstrained problem

$$\min_{\mathbf{h}} \mathbf{r}^T\mathbf{r}: \mathbf{r} = \begin{bmatrix} \mathbf{f} \\ 0 \end{bmatrix} + \begin{bmatrix} A \\ \lambda^{1/2} I \end{bmatrix} \mathbf{h} \tag{3.1a}$$

where λ corresponds to the multiplier associated with the inequality constraint, and that the tentative correction to the current $\mathbf{y}$ is given by

$$\mathbf{h} = -(A^TA + \lambda I)^{-1} A^T\mathbf{f}. \tag{3.2}$$

The following properties are immediate:

(i) $\mathbf{h}$ is defined for all $\lambda > 0$ irrespective of the rank of A;

(ii) $\mathbf{h}$ is downhill for minimizing K ($\nabla K^T\mathbf{h} < 0$);

(iii) $\mathbf{h} \sim -(1/\lambda)\nabla K$, $\lambda \to \infty$. In particular, $\|\mathbf{h}\| \to 0$, $\lambda \to \infty$.

Property (iii) above shows that λ can be used to control $\|\mathbf{h}\|$, and here it is suggested that $\lambda \geqq 0$ be chosen to satisfy the inequality

$$0 < \sigma \leqq \frac{K(\mathbf{y}) - K(\mathbf{y}+\mathbf{h})}{-\nabla K(\mathbf{y})\mathbf{h}}. \tag{3.3}$$

Because $\mathbf{h}$ is downhill, this inequality can always be satisfied for σ small enough because the right-hand side tends to 1 as $\lambda \to \infty$ (forcing $\mathbf{h}$ to tend to zero).

Remark 3.1. The satisfactory convergence rate predicted by Theorem 2.2 when n is large enough suggests that λ be kept as small as possible in (3.1) compatible with satisfying (3.3), and the calculation carried out in Remark 2.2 shows that there is no conflict between the requirements of fast and stable convergence when $\|\mathbf{y} - \hat{\mathbf{y}}\|$ is small and n large. One suitable strategy for ajdusting λ is the following:

(i) Choose the initial λ (the choice $\lambda = \|A\|_F^2/np$ has proved adequate, but is often conservative).

(ii) If the current $\mathbf{h}$ does not satisfy (3.3), increase λ

$$\lambda := \lambda * \rho, \qquad \rho > 1 \tag{3.4}$$

and recompute $\mathbf{h}$ from (3.1).

(iii) If in the current iteration $\mathbf{h}$ is accepted without needing to increase λ, then decrease λ:

$$\lambda := \lambda * \delta, \qquad \delta < 1. \tag{3.5}$$

(iv) Update $\mathbf{y}$:

$$\mathbf{y} := \mathbf{y} + \mathbf{h}, \tag{3.6}$$

and commence next iteration.

To favour decreasing λ, the adjustment parameters ρ, δ should be chosen so that

$$\rho * \delta < 1. \tag{3.7}$$

Otherwise the choice of these parameters does not seem to be critical (say ranging from $\rho = 2$, $\delta = .25$ for difficult problems up to $\rho = 4$, $\delta = .01$ for problems where fast eventual convergence is anticipated).

A convergence theorem is now given. It carries over to the maximum likelihood problem the main result given in Osborne [1976] in the nonlinear least squares case.

THEOREM 3.1. *Limit points of the iteration sketched in Remark* 3.1 *are stationary points of the* log *likelihood provided the sequence of values of the smoothing parameter* $\{\lambda_k\}$ *is bounded. If this sequence is unbounded, then so is* $\nabla^2 K(\hat{\mathbf{y}})$ *provided an unbounded* (*sub*)*sequence* $\{\lambda_k\}$ *is associated with the* (*sub*)*sequence* $\mathbf{y}^{(k)} \to \hat{\mathbf{y}}$.

Proof. Assume first that the sequence $\{\lambda_k\}$ is bounded. Then it follows from (3.3) that

$$\nabla K(A_k^T A_k + \lambda_k I)^{-1} \nabla K^T \leqq \frac{K(\mathbf{y}^{(k)}) - K(\mathbf{y}^{(k)} + \mathbf{h}^{(k)})}{\sigma}. \tag{3.8}$$

Provided the likelihood is bounded, it follows from the descent property of $\mathbf{h}$ that $\{K(\mathbf{y}^{(k)})\}$ is decreasing and bounded below so that it is convergent. It now follows from (3.8) that

$$\|\nabla K(\mathbf{y}^{(k)})\| \to 0, \qquad k \to \infty, \tag{3.9}$$

so that limit points are stationary points of K.

If $\{\lambda_k\}$ is not bounded, then it contains an infinite subsequence with the property that the step (3.4) must be applied. Thus there is an unbounded subsequence $\{\lambda'_k = \lambda_k/\rho\}$ such that (3.3) is violated. Assume now that $\nabla^2 K$ is bounded and continuous in a neighbourhood of $\mathbf{y}^{(k)}$ for each k. Then the mean value theorem gives (here $\mathbf{h}'_k$ is the predicted correction to $\mathbf{y}^{(k)}$ corresponding to $\lambda = \lambda'_k$ and $\to 0$ as $k \to \infty$)

$$\begin{aligned} \sigma &< \frac{K(\mathbf{y}^{(k)}) - K(\mathbf{y}^{(k)} + \mathbf{h}'_k)}{-\nabla K(\mathbf{y}^{(k)})\mathbf{h}'_k} \\ &= -\frac{\nabla K(\mathbf{y}^{(k)})\mathbf{h}'_k - \frac{1}{2}\mathbf{h}'_k \overline{\nabla^2 K}\,\mathbf{h}'_k}{-\nabla K(\mathbf{y}^{(k)})\mathbf{h}'_k} \end{aligned} \tag{3.10}$$

where the bar indicates mean values as appropriate, so that

$$\frac{\mathbf{h}_k'^T \overline{(\nabla^2 K)}\,\mathbf{h}'_k}{\mathbf{h}_k'^T (A^T A + \lambda'_k I)\mathbf{h}'_k} > 2(1-\sigma) \tag{3.11}$$

and, in particular,

$$\max_{t, \|t\|=1} \mathbf{t}^T \overline{(\nabla^2 K)}\,\mathbf{t} > 2(1-\sigma)\lambda'_k. \tag{3.12}$$

This contradicts the assumption that $\nabla^2 K$ is bounded and continuous in a neighbourhood of each $\mathbf{y}^{(k)}$, and the desired result is an immediate consequence.

Remark 3.2. As nonlinear approximating families are, in general, noncompact, this result is about as good as can be expected. For example, if $\mu(t, \mathbf{y})$ is a sum of exponentials, it says that finite limit points of the iteration are stationary points of K and are associated with bounded $\{\lambda_k\}$ as the Hessian of K is bounded at finite points. The reason we cannot do better is that without a compactness result, divergence to ∞ is very much in the nature of things (consider, for example, the problem of approximating t by a sum of exponentials).

4. An example. The example that provided the basis for the development of this paper had to do with estimating the mean of a Poisson process, where

$$\mu(t, \mathbf{y}) = y_1 + y_2 e^{-y_3 t} \tag{4.1}$$

from observed counts z_i, $i = 1, 2, \cdots, n$. Here the distribution is discrete and

$$p(z = n \mid \mu) = \frac{\mu^n e^{-\mu}}{n!}. \tag{4.2}$$

For inference purposes it is convenient to use likelihood ratio statistics where the basis of comparison is the saturated model (McCullagh and Nelder [1983]). This leads us to consider

$$L = -z \ln\left(\frac{z}{\mu}\right) + (z - \mu). \tag{4.3}$$

The nonlinear least squares program used accepts sums of squares of the general form $\sum_{i=1}^{n} w_i(\mathbf{y})^2$ and requires a driver subroutine to provide the objective function, the individual w_i, and their gradients. In this case, the driver subroutine required modification as follows:

$$\begin{aligned} \sum_{i=1}^{n} w_i(\mathbf{y})^2 \leftarrow K(\mathbf{y}) &= \sum_{i=1}^{n} \left\{ z_i \ln\left(\frac{z_i}{\mu(t_i, \mathbf{y})}\right) - (z_i - \mu(t_i, \mathbf{y})) \right\}, \\ w_i \leftarrow f_i &= \frac{(z_i - \mu(t_i, \mathbf{y}))}{s_i} \end{aligned} \tag{4.4}$$

and

$$\nabla w_i \leftarrow e_i^T A = \frac{(-1, -e^{-y_3 t_i}, t_i y_2 e^{-y_3 t_i})}{s_i} \tag{4.5}$$

where $s_i = \mu(t_i, \mathbf{y})^{1/2}$, $i = 1, 2, \cdots, n$.

The only modification to the main subroutine was necessary because of the need to provide $\nabla K\mathbf{h}$ in the numerator of the line search test (3.3) (in the least squares case, this information could be obtained conveniently as part of the solution of the linear subproblem).

Remark 4.1. It is easy to verify that L given by (4.3) has a unique maximum of zero as a function of μ which is attained when $\mu = z$. Thus $-L$ can be represented as a square, say

$$w = \theta\{-L\}^{1/2} \tag{4.6}$$

where $\theta = \pm 1$ is chosen so that w is differentiable. Formally this device can be used to represent K as a sum of squares. However, the direct application of methods such as the Gauss–Newton algorithm to this sum of squares has proved unsatisfactory by comparison with the approach advocated here. This underlines the importance of results such as Theorem 2.2 to the satisfactory performance of these methods in problems of data analysis.

To show the performance of the algorithm on the problem discussed in this section, calculations have been carried out using simulated data obtained by setting $y_1 = 1$, $y_2 = 5$, $y_3 = 10$ in (4.1) and generating z_i by sampling a Poisson random number generator with mean $\mu(t_i)$, $t_i = i/n$, $i = 1, 2, \cdots, n$. For $n = 2^i$, $i = 5, 6, \cdots, 10$, ten replications of the data were generated and the algorithm used to estimate the parameter values for each data set. Initial values of the parameters were given by

$$x_i = y_i(1 + .2(\mathrm{RND} - .5)), \qquad i = 1, 2, 3$$

where RND is a uniform random number generator on $[0, 1]$, the parameters for adjusting λ were $\delta = .1$, $\rho = 2$, and the computations were terminated when $|\nabla K\mathbf{h}|/K <$

TABLE 1
Numerical results for simulated data.

n	Average no. of iterations	Mean of parameter estimates
32	11.2	1.026, 5.035, 9.819
64	6.6	1.111, 7.385, 15.321
128	5.7	1.079, 5.176, 11.013
256	5.3	1.017, 5.625, 11.129
512	4	1.018, 5.063, 10.315
1024	3.9	1.018, 4.960, 10.161

10^{-8}. Results are summarized in Table 1, which gives the mean number of iterations and mean parameter values for each value of n. They show clearly enough the improved rate of convergence as n increases. They show also the quite large amounts of data needed in order to be reasonably certain that the parameters have been obtained with modest accuracy.

Acknowledgments. This paper has been improved significantly as a consequence of the helpful comments of two careful referees.

REFERENCES

C. COX (1984), *Generalised linear models—the missing link*, Appl. Statist., 33, pp. 18–24.

J. E. DENNIS, JR. (1981), *Algorithms for nonlinear fitting*, in Nonlinear Optimization 1981, M. J. D. Powell, ed., Academic Press, New York, pp. 67–78.

D. M. GAY (1980), *On solving robust and generalised regression problems*, in Ottimazzazione non lineare e applicazioni, S. Incerti and G. Treccani, eds., Quad. Unione Mat. Italiana, 17, pp. 55–83.

A. GOLDSTEIN (1965), *On steepest descent*, SIAM J. Control, 3, pp. 147–151.

R. I. JENNRICH (1969), *Asymptotic properties of nonlinear least squares estimators*, Ann. Math. Statist., 40, pp. 633–643.

K. LEVENBERG (1943), *A method for the solution of certain nonlinear problems in least squares*, Quart. Appl. Math., 2, pp. 164–168.

P. MCCULLAGH AND J. A. NELDER (1983), *Generalized Linear Models*, Chapman and Hall, London.

J. J. MORÉ (1978), *The Levenberg-Marquardt algorithm: implementation and theory*, in Proc. Dundee Conference on Numerical Analysis, G. A. Watson, ed., Springer-Verlag, Berlin, New York.

——— (1982), *Recent Developments in Algorithms and Software for Trust Region Methods*, Argonne National Laboratory Report ANL/MCS-TM2.

J. A. NELDER AND R. W. M. WEDDERBURN (1972), *Generalized linear models*, J. Roy. Statist. Soc. Ser. A, 135, pp. 370–384.

M. R. OSBORNE (1976), *Nonlinear least squares—the Levenberg algorithm revisited*, J. Austral. Math. Soc. Ser. B, 19, pp. 348–357.

M. J. D. POWELL (1983), *General Algorithms for Discrete Nonlinear Approximation Calculations*, Cambridge Univ. Report DAMTP 1983/NA2.

J. W. PRATT (1981), *Concavity of the log likelihood*, J. Amer. Statist. Assoc., 76, pp. 103–106.

R. W. M. WEDDERBURN (1974), *Quasi-likelihood functions, generalized linear models, and the Gauss-Newton method*, Biometrika, 61, pp. 439–447.

CHIEN-FU WU (1981), *Asymptotic theory of nonlinear least squares estimation*, Ann. Statist., 9, pp. 501–513.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 3, May 1987

© 1987 Society for Industrial and Applied Mathematics
016

ERRATA: A DOMAIN-DECOMPOSED FAST POISSON SOLVER ON A RECTANGLE*

TONY F. CHAN† AND DIANA C. RESASCO†

Figs. 1 and 4 should be replaced by the following figures.

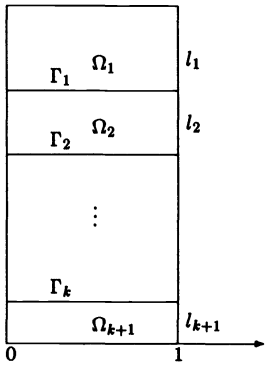


FIG. 1. *The domain* Ω *and its partition.*

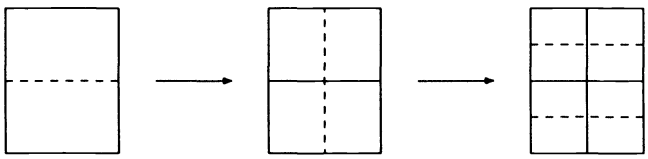


FIG. 4. *An example of nested dissection.*

The last line of page s15 should read as follows:
interfaces between Ω_i and Ω_{i+1}, by Γ_i, $i = 1, \cdots, k$. We use a uniform mesh with grid

* This Journal, 8(1987), pp. s14–s26.

† Department of Computer Science, Research Center for Scientific Computing, Yale University, New Haven, Connecticut 06520.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 4, July 1987
© 1987 Society for Industrial and Applied Mathematics

CONVERGENCE OF DYNAMIC ITERATION METHODS FOR INITIAL VALUE PROBLEMS*

ULLA MIEKKALA† AND OLAVI NEVANLINNA†

Abstract. In VLSI-simulation there has recently been interest in an iterative technique called the waveform relaxation method. In this paper we set up a theoretical framework to analyze the convergence of such methods. We restrict the discussion to linear systems and do not consider the effects of time discretization, but assume that the initial value problems are solved exactly.

Key words. initial value problems, iteration methods, waveform relaxation

AMS (MOS) subject classifications. 65L05, 65F10

1. Introduction. In large dynamical systems there are likely to be time constants of several different magnitudes. This stiffness forces one in practice to use implicit time discretization methods, and hence a system of equations is formed to be solved at each time level. In the usual way of implementing implicit methods one solves these equations once and for all at each time level before advancing to the next level.

A different approach is to solve the equations only partially but to do several iterative time sweeps to improve the results. Recently there has been interest in such an approach in VLSI-simulation; see the survey article [7]. The idea of the approach is to decompose the large system into subsystems and to integrate each subsystem independently by reading inputs from other subsystems from their values on the previous iteration.

This approach has its obvious virtues and drawbacks. It suits excellently for parallel computation, especially when the original problem to be simulated consists of subprocesses with many internal and few external variables. Known time histories of nearby problems can be efficiently utilized. Large savings in the computation are to be expected, due to the independent time stepping in subsystems, especially if a relatively large but unknown portion of the subsystems is most of the time inactive. The desynchronization of the time marching might add some robustness characteristic to the original problem. On the other hand, one has to save the time histories of all external variables and to introduce an interpolation procedure to provide the values at correct time instances.

In VLSI-simulation this type of approach, called waveform relaxation method, seems to result in considerable savings in computing time [7]. It would not be surprising if efficient implementations of the idea appeared also to applications outside circuit equations.

In this paper we start a study of the convergence properties of such iteration methods. In the waveform relaxation literature there is one general convergence result, which essentially says that such iterations converge uniformly on bounded time intervals, provided the equations determining the dynamics of the system are Lipschitz continuous [6]. Such a result can be proved in a similar way as the corresponding result on Picard–Lindelöf iteration, but then no distinction is made between stable and unstable problems and we do not get any insight about how the decomposition affects the actual rate of convergence.

* Received by the editors July 15, 1985; accepted for publication (in revised form) March 21, 1986. This work was partially supported by the Academy of Finland.

† Helsinki University of Technology, Institute of Mathematics, 02150 Espoo, Finland.

Here we shall concentrate on linear autonomous systems and try to establish a formalism in which we could in a meaningful manner discuss the rate of convergence and its dependence on the different ways to decompose the system. We intend later to discuss also the interaction between the decomposition of the system and the stability properties of the time discretization method. Nonlinear systems will be discussed when a satisfactory understanding of linear problems is reached.

In § 2 we present the set up and show how the convergence is connected to an iteration matrix obtained e.g. by Laplace transforming the problem. The section ends with an existence theorem which says in particular that any asymptotically stable linear system can be decomposed into a set of forced scalar equations so that the iteration converges uniformly for $t \geqq 0$.

It is instructive to compare the convergence for the dynamic problem with that one obtained for the linear static equation $Ax = b$ using the same decomposition of A. We refer to the iteration methods for solving $Ax = b$ as *static iteration* methods; this leads us to use the term *dynamic iteration* for the iteration schemes we introduce for time-dependent initial value problems (the scheme (2.2) below). In § 3 we show that in many cases the dynamic iteration can be discussed analogously with the static one. In particular this is the case if A is an H-matrix or both A and the decomposed system are symmetric.

One of the well-known results for the static equation is the convergence of SOR for all relaxation parameters $\omega \in (0, 2)$ when the matrix A is symmetric and positive definite. A similar result for the dynamic iteration does not hold. In § 4 we give a detailed discussion on the dynamic SOR for the so-called consistently ordered matrices.

Difficulties with the convergence for the dynamic SOR are connected with the asymmetric splitting of the matrix. It is natural therefore also to study the symmetric SOR iteration and we do that in § 5.

Finally § 6 contains a numerical experiment. We discretize the heat equation in space, which results in a system with a consistently ordered symmetric matrix. We apply the dynamic SOR iteration by discretizing the time variable by the trapezoidal rule. Although the convergence is estimated from a relatively short time interval in our experiment, the results show essentially the same behavior as in § 4 where uniform convergence on $t \geqq 0$ is concerned.

2. Basic concepts. Let A be an $n \times n$ complex matrix. We want to solve

$$\dot{x} + Ax = f, \quad t > 0, \qquad x(0) = x_0 \tag{2.1}$$

using an iteration scheme that is based on the splitting $A = M - N$:

$$\dot{x}^{(k)} + Mx^{(k)} = Nx^{(k-1)} + f, \qquad x^{(k)}(0) = x_0. \tag{2.2}$$

Introducing the linear integral operator $\mathcal{K}$: $u \mapsto x$

$$\mathcal{K}u(t) := \int_0^t e^{(s-t)M} Nu(s)\, ds, \tag{2.3}$$

we can write the solution of (2.2) in the form

$$x^{(k)} = \mathcal{K}x^{(k-1)} + \varphi \tag{2.4}$$

where

$$\varphi(t) = e^{-tM} x_0 + \int_0^t e^{(s-t)M} f(s)\, ds. \tag{2.5}$$

Let X be a Banach space and assume that $\mathcal{K}$ is a bounded operator in X. Then for the convergence we need that the spectral radius of $\mathcal{K}$,

$$\rho(\mathcal{K}) := \lim_{k\to\infty} \|\mathcal{K}^k\|^{1/k} \tag{2.6}$$

satisfies $\rho(\mathcal{K}) < 1$, since typically $f \in X$ would imply $\varphi \in X$.

For practical purposes one is perhaps mostly interested in uniform convergence on a long but bounded time interval. However, then one would always have $\rho(\mathcal{K}) = 0$, and we would learn nothing about the effect of the splitting on the actual rate of convergence. Although for any splitting the rate on bounded intervals will ultimately be faster than geometric, the splitting strongly effects the speed of convergence on the early phases of the iteration.

One possibility to see the dependence on the splitting would be to look after a weighted norm of the form

$$\|x\| := \sup_{[0,T]} |e^{-\alpha t} x(t)|$$

in which $\mathcal{K}$ would become a contraction. For the smaller α, one could take the better practical convergence properties one could expect. Notice, however, that the "best" α would depend on the particular choice of norm in $\mathbf{C}^n$.

To see that $\rho(\mathcal{K}) = 0$ in the space of uniform convergence on $[0, T]$ observe that

$$|\mathcal{K}\varphi(t)| \leqq C \int_0^t |\varphi(s)|\, ds$$

with $C = e^{T|M|}|N|$, where $|\cdot|$ denotes the norm in $\mathbf{C}^n$. Hence

$$|\mathcal{K}^j\varphi(t)| \leqq C^j \frac{t^j}{j!} \|\varphi\|_\infty$$

and in particular

$$\|\mathcal{K}^j\| \leqq \frac{(CT)^j}{j!}.$$

Thus

$$\left\| \left(1 - \frac{1}{\lambda}\mathcal{K}\right)^{-1} \right\| \leqq e^{CT/|\lambda|}$$

for all $\lambda \neq 0$. A similar estimate holds naturally for the weighted spaces above.

In what follows we shall take a different approach. We rather ask directly for results which would be independent of the length of the interval. It is therefore natural to consider Banach spaces on the interval $[0, \infty)$.

Let X denote any usual L^p-space of $\mathbf{C}^n$-valued functions defined on $[0, \infty)$. We see from (2.5) that then $\varphi \in X$ is implied by $f \in X$ if and only if all eigenvalues of M have positive real parts. A similar statement holds for $\mathcal{K}$.

THEOREM 2.1. *Let $X = L^p(\mathbf{R}_+, \mathbf{C}^n)$ with $1 \leqq p \leqq \infty$. If all eigenvalues of A have positive real parts then $\mathcal{K}$ is a bounded operator in X if and only if the eigenvalues of M have positive real parts.*

Proof. The sufficiency part of the theorem is, of course, obvious. Suppose S transforms M into Jordan canonical form: $M = SJS^{-1}$. Setting $Q := S^{-1}NS$ we can define an integral operator $\mathcal{K}_S$ as in (2.3), using the kernel $e^{(s-t)J}Q$. By construction the operator $\mathcal{K}_S$ is bounded in X if and only if $\mathcal{K}$ is.

Let us write after a possible permutation of coordinates

$$J = \begin{bmatrix} J_+ & \\ & J_0 \end{bmatrix}$$

where J_+ contains all Jordan blocks whose diagonal elements have positive real parts. We have to show that if $\mathcal{K}_S$ is bounded in X then J_+ actually equals J.

Let P denote the projection onto the subspace of $\mathbf{C}^n$ where J_0 acts. Setting

$$v_+(t) := \int_0^t e^{(s-t)J_+}(I-P)Qu(s)\,ds,$$

$$v_0(t) := \int_0^t e^{(s-t)J_0}PQu(s)\,ds,$$

we have $\mathcal{K}_S u = v_+ \oplus v_0$ with $Pv_+ = 0$, $Pv_0 = v_0$. The mapping $u \mapsto v_+$ is clearly bounded in X while the mapping $u \mapsto v_0$ is never bounded in X unless $PQ = 0$, which is easily seen using the special structure of e^{-tJ_0}. Since $PQ = 0$ we can write $J - Q$ in block form:

$$J - Q = \begin{bmatrix} J_+ - Q_1 & Q_2 \\ & J_+ \end{bmatrix}.$$

Therefore eigenvalues of J_0 are also eigenvalues of $J - Q$. This contradicts the assumption that all eigenvalues of A have positive real parts, since $J - Q = S^{-1}AS$. □

Because of Theorem 2.1 we shall usually formulate our results by assuming explicitly that the eigenvalues of M have positive real parts.

Remark 2.1. Assume now that the eigenvalues of M have positive real parts so that in particular $\mathcal{K}$ is bounded in L^p-spaces and $\rho(\mathcal{K}) < \infty$. *If* $\rho(\mathcal{K}) < 1$ *then all eigenvalues of* A *must have positive real parts* as well. In fact, if $\rho(\mathcal{K}) < 1$ then the equation $x - \mathcal{K}x = \varphi$ can be boundedly solved for any φ in L^p. Transforming this back to (2.1) yields the result.

THEOREM 2.2. *Let* $X = L^p(\mathbf{R}_+, \mathbf{C}^n)$ *with* $1 \leqq p \leqq \infty$, *and assume that all eigenvalues of* M *have positive real parts. Then*

$$\rho(\mathcal{K}) = \max_{\xi \in \mathbf{R}} \rho((i\xi I + M)^{-1}N). \tag{2.7}$$

In (2.7) $\rho(\mathcal{K})$ is defined in (2.6) while the ρ on the right-hand side denotes the spectral radius of the matrix. Notice that $(zI + M)^{-1}N$ is the Laplace transform of the kernel of $\mathcal{K}$.

Proof. The spectral radius satisfies the following property: $\rho(\mathcal{K})$ equals the smallest such ρ for which $|\lambda| > \rho$ implies that $\lambda - \mathcal{K}$ has a bounded inverse in X. The result would then follow directly from a matrix valued version of a result of Wiener and Levy. We shall indicate that later in the proof of Theorem 5.1. Here we rather present an elementary proof.

Suppose that $\lambda \neq 0$,

$$\lambda u - \mathcal{K}u = f \tag{2.8}$$

and f is smooth so that by differentiating we obtain

$$\dot{u} + \left(M - \frac{1}{\lambda}N\right)u = \frac{1}{\lambda}(\dot{f} + Mf). \tag{2.9}$$

This yields immediately

$$u(t) = \frac{1}{\lambda}f(t) + \frac{1}{\lambda^2}\int_0^t e^{(s-t)(M-(1/\lambda)N)}Nf(s)\,ds. \tag{2.10}$$

From this we see that $\lambda - \mathcal{K}$ has a bounded inverse at least when all eigenvalues of $M-(1/\lambda)N$ have positive real parts.

On the other hand, suppose that $M-(1/\lambda)N$ would have an eigenvalue, say μ, with a negative real part. Denoting by x the corresponding eigenvector, let f be the solution of $\dot{f}+Mf=0$ with $f(0)=x$. By the assumption on the spectrum of M, $f \in X$. However, it is clear from (2.9) that then $u(t)=e^{-\mu t}x$ and hence $u \notin X$. Suppose finally that $M-(1/\lambda)N$ would have a purely imaginary eigenvalue at some $\lambda=\lambda_0$. Then, since $M-(1/\lambda)N$ is analytic in $(1/\lambda)$, in every neighbourhood of λ_0 $M-(1/\lambda)N$ would have at least one eigenvalue with negative real part, unless the eigenvalue is constant. In that case, however, M would have a purely imaginary eigenvalue contradicting one hypothesis. We conclude that $\rho(\mathcal{K})$ is the smallest such ρ for which all eigenvalues of $M-(1/\lambda)N$ have positive real parts whenever $|\lambda|>\rho$.

Let $-z$ be an eigenvalue of $M-(1/\lambda)N$ and $y \in \mathbf{C}^n$ be the corresponding eigenvector. Then

$$\left(M-\frac{1}{\lambda}N\right)y=-zy$$

and $|\lambda|>\rho$ together imply $\operatorname{Re} z<0$. This further implies λ with $|\lambda|>\rho$ can only be an eigenvalue of $(zI+M)^{-1}N$ if $\operatorname{Re} z<0$. The smallest such ρ clearly satisfies

$$\rho(\mathcal{K})=\sup_{\operatorname{Re} z \geqq 0} \rho((zI+M)^{-1}N).$$

Now the eigenvalues of M have positive real parts, and $(zI+M)^{-1}N$ is therefore analytic in an open set containing the closed right half plane. Since the spectral radius satisfies the maximum principle, we conclude that the supremum is attained along the imaginary axis. □

By Theorem 2.2 we observe that the spectral radius is independent of the particular L^p-space where we consider the problem. If we would use exponential scaling in our norm, e.g.

$$\|x\| := \|e^{-\alpha t}x(t)\|_\infty \tag{2.11}$$

then Theorem 2.2 would hold unchanged if we replace M by $\alpha I+M$.

DEFINITION 2.1. For a splitting $A=M-N$ the convergence abscissa β is the infimum of all α's for which $\rho(\mathcal{K})<1$ in the weighted space normed by (2.11).

THEOREM 2.3. *Either $N=0$ and $\beta=-\infty$, or $\beta=\max\{\beta_1,\beta_2\}$ where β_1 is determined by*

$$1=\sup_{\operatorname{Re} z>\beta_1} \rho((zI+M)^{-1}N) \tag{2.12}$$

and β_2 is the smallest number such that $K(z):=(zI+M)^{-1}N$ is analytic for $\operatorname{Re} z>\beta_2$.

Proof. In (2.12), $(zI+M)^{-1}N$ is analytic except for possible poles at eigenvalues of $-M$. Let μ_j be the eigenvalues of $-M$, ordered so that $\operatorname{Re}\mu_j \geqq \operatorname{Re}\mu_{j+1}$. If (2.12) holds with $\beta_1>\operatorname{Re}\mu_1$ then $(zI+M)^{-1}N$ is not constant, and for any $\hat{\beta}$ with $\beta_1>\hat{\beta}>\operatorname{Re}\mu_1$

$$\sup_{\operatorname{Re} z>\hat{\beta}} \rho((zI+M)^{-1}N)>1.$$

The conclusion then follows from Theorem 2.2, applied to the matrix $\beta_1 I+M$.

If, on the other hand, such a $\beta_1>\operatorname{Re}\mu_1$ does not exist but $\beta_2=\operatorname{Re}\mu_1$, then $\beta=\beta_2$. In fact, if $\mathcal{K}$ is bounded in a space weighted with $e^{-\alpha t}$ then $K(z)$ is analytic for $\operatorname{Re} z>\alpha$.

Assume therefore that $\beta_2 < \operatorname{Re}\mu_1$ and $\rho(K(z)) \leqq 1$ for $\operatorname{Re} z \geqq \operatorname{Re}\mu_1$. Now μ_1 is not a pole and $K(z)$ stays bounded near μ_1. Transform M into Jordan canonical form: $M = SJS^{-1}$ and set $Q := S^{-1}NS$. Let now X_1 stand for the invariant eigenspace of J corresponding to the eigenvalue $-\mu_1$. Take an arbitrary $x \in R(Q)$, where $R(Q)$ denotes the range of Q, and decompose it as $x = x_1 + x_2$ with $x_1 \in X_1$, $x_2 \in X_1^{\perp}$. If y is such that $Qy = x$, then

$$S^{-1}K(z)Sy = (zI+J)^{-1}x_1 + (zI+J)^{-1}x_2$$

where the left-hand side stays bounded as $z \to \mu_1$. Since $x_2 \in X_1^{\perp}$ $(zI+J)^{-1}x_2$ is bounded, too, and hence $(zI+J)^{-1}x_1$. However, in X_1 $zI+J$ acts like $(z-\mu_1)I - U$ with U nilpotent, and therefore $(zI+J)^{-1}$ acts in X_1 as

$$\sum_{j=0}^{l} \left(\frac{1}{z-\mu_1}\right)^{j+1} U^j.$$

From this it is clear that $(zI+J)^{-1}x_1$ is bounded as $z \to \mu_1$ only if $x_1 = 0$. Thus we conclude that the range of Q and the eigenspace X_1 are orthogonal. Decomposing $\mathbf{C}^n$ accordingly we can proceed by considering a lower dimensional matrix.

The process either finds a $\beta > -\infty$ or Q must eventually be orthogonal against the whole $\mathbf{C}^n$. □

In practical applications on intervals of finite length, we could be satisfied if the convergence abscissa were of the size corresponding to the slowest or "nonstiff" time scales of the original problem. Since the effect of exponential scaling is fully characterized by Theorem 2.3, we can, in the following, concentrate on just L^p-spaces. For simplicity we shall adopt the following terminology.

DEFINITION 2.2. The dynamic iteration, specified by the splitting $M - N$, converges, if the eigenvalues of M have positive real parts and

$$\max_{\xi \in \mathbf{R}} \rho((i\xi I + M)^{-1}N) < 1. \tag{2.13}$$

Similarly, the static iteration converges if M is nonsingular, and

$$\rho(M^{-1}N) < 1.$$

Setting $f(t) \equiv b$ in (2.1) we see that the static iteration for solving $Ax = b$ is obtained from the dynamic iteration as $t \to \infty$. Thus, $x^{(k)}(t)$ converges pointwise with fixed t faster than any power, while $x^{(k)}(\infty)$ converges with the rate $\rho(M^{-1}N)$.

Since the dynamic iteration cannot be faster than the static one, we might expect that in many cases the rates would actually equal, i.e. that the maximum in (2.13) would be attained at the origin. It turns out that there are important situations where this does happen in addition to ones where it is not the case.

We shall complete this section by showing the existence of a convergent splitting. Recall, see Remark 2.1, that then all eigenvalues of A must have positive real parts.

THEOREM 2.4. *If all eigenvalues μ_j of A have positive real parts, set*

$$\omega^* := 2 \min_j \frac{\operatorname{Re}\mu_j}{|\mu_j|^2}.$$

Then the dynamic iteration specified by $M_\omega := (1/\omega)I$ converges if and only if $\omega \in (0, \omega^*)$. In particular, $\rho(\mathcal{K}_\omega)$ decreases with ω for $\omega \in (0, \omega_{\text{opt}})$ and increases for $\omega \in (\omega_{\text{opt}}, \omega^*)$, where $\omega_{\text{opt}} \in [\frac{1}{2}\omega^*, \omega^*)$.

Proof. Set $K_\omega(z) := ((1/\omega)+z)^{-1}((1/\omega)I - A)$ and suppose that $\lambda_j(z)$ is an eigenvalue of $K_\omega(z)$. Then $\mu_j := (1/\omega) - \lambda_j(z)((1/\omega)+z)$ must be an eigenvalue of A. Hence for Re $z \geqq 0$

$$|\lambda_j(z)|^2 \leqq 1 - 2(\text{Re } \mu_j)\omega + |\mu_j|^2\omega^2 \tag{2.14}$$

with equality for $z=0$. By Theorem 2.2 $\rho(\mathcal{K}_\omega) = \max_j |\lambda_j(0)|$ and the conclusion follows. □

If A has real eigenvalues we have $|\lambda_j(0)| = |1-\mu_j\omega|$ in (2.14) and the spectral radius of $\mathcal{K}_\omega$ may be determined graphically as in Fig. 2.1.

We can state

COROLLARY 2.1. *Let A have real positive eigenvalues $0 < \mu_1 \leqq \cdots \leqq \mu_n$. Then the dynamic iteration specified by $M_\omega = (1/\omega)I$ converges if and only if $\omega \in (0, (2/\mu_n))$. The spectral radius of the iteration method is*

$$\rho(\mathcal{K}_\omega) = \begin{cases} 1-\omega\mu_1, & \omega \in (0, \omega_{\text{opt}}], \\ -1+\omega\mu_n, & \omega \in (\omega_{\text{opt}}, 2/\mu_n), \end{cases}$$

where $\omega_{\text{opt}} = 2/(\mu_1+\mu_n)$.

Notice that if A has constant diagonal $D = dI$, the iteration in Theorem 2.4 and its corollary is the Jacobi overrelaxation (JOR) method.

3. Convergence results. In this section we show that some methods have the same convergence properties for the dynamic iteration as for the static one. We study the convergence of SOR and JOR for the special class of H-matrices, and also methods using symmetric splitting for the class of Hermitian positive definite matrices.

We recall the definitions of H- and M-matrices. The notation $C \geqq 0$ means that the matrix C is nonnegative, i.e., every element $c_{ij} \geqq 0$, $1 \leqq i, j \leqq n$. By matrix notation $|\ |$ we, on the other hand, mean that we take absolute value of the matrix elementwise.

DEFINITION 3.1. A real $n \times n$ matrix B is an M-matrix if it can be expressed in the form $B = \tau I - C$ where $\tau > 0$, $C \geqq 0$ and the spectral radius $\rho(C) \leqq \tau$.

We recall that another way to define an M-matrix is to demand that B be nonsingular, have nonnegative off-diagonal elements and have a positive inverse. We refer to [1] to see the equivalence of these two definitions.

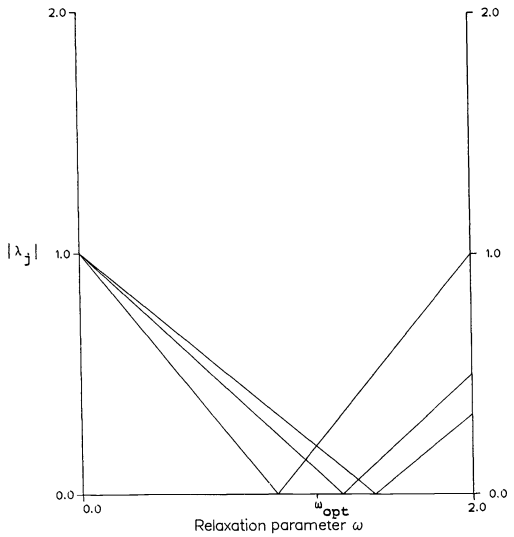


FIG. 2.1. *$|\lambda_j| = |1-\mu_j\omega|$ for different values of μ_j. The spectral radius is the maximum over $|\lambda_j|$.*

We set $\mathcal{M}(A)=(m_{ij})$ to be the comparison matrix of a complex $n\times n$ matrix A, such that

$$m_{ii}=|a_{ii}|, \qquad m_{ij}=-|a_{ij}|, \quad i\neq j.$$

DEFINITION 3.2. A complex matrix A is an H-matrix if its comparison matrix $\mathcal{M}(A)$ is an M-matrix.

Let us use the decomposition of the form $A=D-L-U$ where D is a diagonal, L a lower triangular and U an upper triangular matrix. We shall also use the notation $J(0)=D^{-1}(L+U)$; this matrix is often called Jacobi matrix for an obvious reason.

We have a convergence result for the Jacobi overrelaxation (JOR) method, which uses the splitting

$$A=M-N=\frac{1}{\omega}D-\left(\frac{1-\omega}{\omega}D+L+U\right). \tag{3.1}$$

THEOREM 3.1. *For any H-matrix A with $D>0$ and $\mathcal{M}(A)$ a nonsingular M-matrix the dynamic* JOR *method converges whenever*

$$0<\omega<\frac{2}{1+\rho(|J(0)|)}. \tag{3.2}$$

We omit the proof, since it is quite similar to the proof of Theorem 3.2 if we substitute $M_\omega=(1/\omega)(D+zI)$, $N_\omega=((1-\omega)/\omega)D+L+U$, $\tilde{M}_\omega=(1/\omega)|D|$ and $\tilde{N}_\omega=(|1-\omega|)/\omega|D|+|L|+|U|$. Notice that for symmetric A with constant diagonal even the optimal relaxation parameter can be determined as is explained after Corollary 2.1 in § 2.

THEOREM 3.2. *For any H-matrix A, with $D>0$ and $\mathcal{M}(A)$ a nonsingular M-matrix, the dynamic* SOR *converges whenever*

$$0<\omega<\frac{2}{1+\rho(|J(0)|)}. \tag{3.3}$$

Proof. From the definition of the comparison matrix it is clear that

$$|J(0)|=J(\mathcal{M}(A)).$$

We know that $\rho(J(\mathcal{M}(A)))<1$ because $\mathcal{M}(A)$ is a nonsingular M-matrix, see e.g. [5], so that

$$\rho(|J(0)|)<1. \tag{3.4}$$

If we use the composition $A=D-L-U$, as usual, the iteration matrix for dynamic SOR becomes

$$H_\omega(z)=M_\omega^{-1}N_\omega, \quad \text{where } M_\omega=\frac{1}{\omega}(D+zI)-L, \qquad N_\omega=\frac{1-\omega}{\omega}D+U \tag{3.5}$$

and $\omega>0$ is the relaxation parameter. Let us now consider the matrices

$$\begin{aligned}
\tilde{M}_\omega&=\frac{1}{\omega}|D|-|L|,\\
\tilde{N}_\omega&=\frac{|1-\omega|}{\omega}|D|+|U|,\\
\tilde{A}_\omega&=\tilde{M}_\omega-\tilde{N}_\omega=\frac{1-|1-\omega|}{\omega}|D|-|L|-|U|,\\
\tilde{H}_\omega&=\tilde{M}_\omega^{-1}\tilde{N}_\omega.
\end{aligned} \tag{3.6}$$

We want to show that $\tilde{M}_\omega - \tilde{N}_\omega$ is a regular splitting of $\tilde{A}_\omega$, which, according to definition [9], is equivalent to showing that $\tilde{N}_\omega \geqq 0$ and $\tilde{M}_\omega^{-1} \geqq 0$ with $\tilde{M}_\omega$ nonsingular. The former condition clearly holds and it is also true that $|N_\omega| = \tilde{N}_\omega$. To show the latter of the conditions and to obtain an upper bound for M_ω^{-1} we write

$$\begin{aligned}|M_\omega^{-1}| &= \left|\left(\frac{1}{\omega}(D+zI)-L\right)^{-1}\right| = \left|\left[\frac{1}{\omega}(D+zI)(I-\omega(D+zI)^{-1}L)\right]^{-1}\right| \\ &= |(I-\omega(D+zI)^{-1}L)^{-1}\omega(D+zI)^{-1}| \\ &= |I+\omega(D+zI)^{-1}L+(\omega(D+zI)^{-1}L)^2 \\ &\quad +\cdots+(\omega(D+zI)^{-1}L)^{n-1}|\cdot\omega\cdot|D+zI|^{-1}.\end{aligned}$$

The series is finite because the matrix $(D+zI)^{-1}L$ is a strictly lower triangular matrix and its nth power is 0. We obtain the inequality

$$|M_\omega^{-1}| \leqq [I+\omega|(D+zI)^{-1}L|+\cdots+\omega^{n-1}|(D+zI)^{-1}L|^{n-1}]\omega|D+zI|^{-1},$$

which further becomes

$$\begin{aligned}|M_\omega^{-1}| &\leqq (I-\omega|D+zI|^{-1}|L|)^{-1}\left(\frac{1}{\omega}|D+zI|\right)^{-1} \\ &= \left[\frac{1}{\omega}|D+zI|(I-\omega|D+zI|^{-1}|L|)\right]^{-1} \\ &= \left(\frac{1}{\omega}|D+zI-|L|\right)^{-1}.\end{aligned}$$

This inequality also implies that the matrix $(1/\omega)|D+zI|-|L|$ is a nonsingular M-matrix because its inverse is positive and its off-diagonal elements $\leqq 0$. Further the inequality above, after setting $z=0$, implies that $\tilde{M}_\omega$ is an M-matrix. In particular we have found $\tilde{A}_\omega = \tilde{M}_\omega - \tilde{N}_\omega$ to be a regular splitting.

Because for $\operatorname{Re} z \geqq 0$ $(1/\omega)|D+zI|-|L| \geqq \tilde{M}_\omega$ and differs from $\tilde{M}_\omega$ only in its diagonal elements we can make use of Proposition 2.4.11 in [8] which gives an upper bound

$$|M_\omega^{-1}| \leqq \left(\frac{1}{\omega}|D+zI|-|L|\right)^{-1} \leqq \tilde{M}_\omega^{-1}. \tag{3.7}$$

To show that $\tilde{A}_\omega^{-1} \geqq 0$ we first express $\tilde{A}_\omega$ as

$$\tilde{A}_\omega = |D|\left[\frac{1-|1-\omega|}{\omega} I - |J(0)|\right]. \tag{3.8}$$

Since $|D|^{-1} > 0$ according to our assumptions, it follows from Theorem 3.8 in [9] that $\tilde{A}_\omega^{-1} \geqq 0$ if

$$\rho(|J(0)|) < \frac{1-|1-\omega|}{\omega}. \tag{3.9}$$

For $0 < \omega \leqq 1$ this becomes $\rho(|J(0)|) < 1$, which we know to be true.

For $\omega > 1$ the condition becomes

$$\rho(|J(0)|) < \frac{2-\omega}{\omega},$$

equivalent to the inequality for ω

$$\omega < \frac{2}{1+\rho(|J(0)|)}.$$

From the theory of regular splittings we now know that

$$\rho(\tilde{M}_\omega^{-1}\tilde{N}_\omega) < 1.$$

For the iteration matrix $H_\omega(z)$ we have using (3.7)

$$|H_\omega(z)| = |M_\omega^{-1}N_\omega| \leqq |M_\omega^{-1}||N_\omega| \leqq \tilde{M}_\omega^{-1}\tilde{N}_\omega. \tag{3.10}$$

We finally obtain the convergence result

$$\rho(\mathscr{H}_\omega) = \sup_{z\in\mathbf{C}_+} \rho(H_\omega(z)) \leqq \rho(\tilde{M}_\omega^{-1}\tilde{N}_\omega) < 1, \tag{3.11}$$

with the aid of Theorem 2.8 in [9]. □

COROLLARY 3.1. *For any H-matrix A with $D>0$ and $\mathscr{M}(A)$ a nonsingular M-matrix the dynamic Gauss–Seidel iteration converges.*

We now turn to studying symmetric splittings in dynamic iteration. By positive definiteness of a complex matrix A we mean that

$$\operatorname{Re}(Ax, x) > 0 \quad \text{for all } x \in \mathbf{C}^n,\ x \neq 0$$

where the inner product is defined by

$$(x, y) = \sum_i x_i y_i^*.$$

We state

THEOREM 3.3. *Suppose that a positive definite Hermitian matrix A is split into symmetric parts, that is $A = M - N$ where M and N are also Hermitian and M is positive definite. Then the dynamic iteration method converges if and only if $2M - A$ is positive definite.*

Proof. The iteration matrix for this symmetric iteration becomes

$$K(z) = (M + zI)^{-1}N. \tag{3.12}$$

If we call its eigenvalues λ we have an eigenvalue problem

$$Nx = \lambda(M + zI)x, \qquad x \in \mathbf{C}^n,\ z \in \mathbf{C}_+,$$

and furthermore

$$(Nx, x) = \lambda(Mx, x) + \lambda z(x, x).$$

We can solve for $\lambda = \alpha + i\beta$ for we know that the inner products above are real since M and N are Hermitian. We can also take $(x, x) = 1$ and use, for simplicity, the notation

$$(Mx, x) = r \in \mathbf{R}, \qquad (Nx, x) = s \in \mathbf{R}.$$

The positive definiteness of M implies that $r > 0$. If we write $z = \eta + i\xi$, $\eta \geqq 0$, we obtain equations

$$s = \alpha r + \alpha\eta - \beta\xi, \qquad 0 = \beta r + \alpha\xi + \beta\eta.$$

And after solving for α and β we get

$$|\lambda|^2 = \alpha^2 + \beta^2 = \frac{s^2}{(r+\eta)^2 + \xi^2}. \tag{3.13}$$

The convergence is obtained—since r and η have the same sign—if

$$|s|<r \quad \text{or} \quad |(Mx, x)-(Ax, x)|<(Mx, x),$$

which is satisfied if $2M-A$ is positive definite.

On the other hand, it is known from the static iterations [5] that $\rho(K(0))<1$ only if $2M-A$ is positive definite. This completes the proof since $\rho(\mathcal{K})\geqq\rho(K(0))$. One sees easily, that here in fact $\rho(\mathcal{K})=\rho(K(0))$. □

Specifying, for a Hermitian positive definite A, $M:=(1/\omega)D$ we obtain convergence for the dynamic JOR for $\omega\in(0, \omega^*)$ where ω^* is the supremum of all ω's for which $(2/\omega)D-A$ is positive definite.

4. SOR for consistently ordered matrices. For a certain class of matrices it is possible to develop a relation between the eigenvalues of the SOR iteration matrix $H_\omega(z)$ and those of the associated Jacobi matrix $J(0)$. This is due to the fact that these matrices, the consistently ordered matrices, are characterized by one parameter only. We define consistently ordered matrices after Young.

DEFINITION 4.1. A matrix A of order n is consistently ordered if for some t there exist disjoint subsets $S_1, S_2, \cdots, S_t$ of $W=\{1, 2, \cdots, n\}$, where $\cup_{i=1}^{t} S_i = W$, such that if i and j are such that $|a_{ij}|+|a_{ji}|>0$, then $j\in S_{k+1}$ if $j>i$ and $j\in S_{k-1}$ if $j<i$, where S_k is the subset containing i.

It can be shown that every block tridiagonal matrix of the form

$$\begin{bmatrix} D_1 & U_1 & & & \\ L_1 & D_2 & U_2 & & \\ & L_2 & D_3 & \ddots & \\ & & \ddots & \ddots & U_{N-1} \\ & & & L_{N-1} & D_N \end{bmatrix}$$

where D_i are diagonal matrices, is consistently ordered. In practice it is sufficient that the matrix can be permuted into this form. Forsythe and Wasow [4] used this to define matrices with property A.

The following result holds (see [10, p. 147]):

PROPOSITION 4.1. *If $A=D-L-U$ is a consistently ordered matrix, then*

$$\det\left(\alpha L+\frac{1}{\alpha}U-\gamma D\right)$$

is independent of α for $\alpha\neq 0$ and for all $\gamma\in\mathbf{C}$.

Notice that the conclusion in the proposition may also be written in the form: the matrix

$$B(\alpha)=D^{-1}\left(\alpha L+\frac{1}{\alpha}U\right) \tag{4.1}$$

has eigenvalues independent of α for $\alpha\neq 0$. Varga has used this as a definition for consistently ordered matrices and for certain classes of matrices these two properties are equal. For information about their relations see [2].

To find the expression for $\rho(\mathcal{H}_\omega)$ we first give the relation between the eigenvalues of the two matrices $H_\omega(z)$ and $J(0)$.

THEOREM 4.1. *Let A be a consistently ordered matrix with constant positive diagonal $D=dI>0$ and $\omega\neq 0$. If μ is an eigenvalue of $J(0)$ and λ satisfies*

$$(\lambda(1+z)+\omega-1)^2=\lambda(\omega\mu)^2 \tag{4.2}$$

then λ is an eigenvalue of $H_\omega(z)$. Vice versa, if $\lambda \neq 0$ is an eigenvalue of $H_\omega(z)$ and satisfies (4.2), *then μ is an eigenvalue of $J(0)$.*

Proof. The iteration matrix for dynamic SOR is

$$\begin{aligned} H_\omega(z) &= (D+zI-\omega L)^{-1}((1-\omega)D+\omega U) \\ &= (I+zD^{-1}-\omega\hat{L})^{-1}((1-\omega)I+\omega\hat{U}), \end{aligned}$$

where we have used $\hat{L}=D^{-1}L$, $\hat{U}=D^{-1}U$. Because of the assumption $D=dI$ we have $\det(I+zD^{-1}-\omega\hat{L})=(1+z/d)^n$ so that

$$\begin{aligned} \det(\lambda I-H_\omega(z)) &= \frac{1}{(1+z/d)^n}\det[((1+z/d)I-\omega\hat{L})(\lambda I-H_\omega(z))] \\ &= \frac{1}{(1+z/d)^n}\det[((1+z/d)\lambda+\omega-1)I-\omega(\lambda\hat{L}+\hat{U})]. \end{aligned}$$

Before advancing further, notice that (4.1) implies that $B(1)=J(0)$ and $B(-1)=-J(0)$ must have the same eigenvalues, so that if μ is an eigenvalue of $J(0)$ then also $-\mu$ is an eigenvalue of $J(0)$.

Therefore, if μ is an eigenvalue of $J(0)$ and $\lambda \neq 0$ satisfies (4.2) we can choose

$$(1+z/d)\lambda+\omega-1=\sqrt{\lambda}\,\omega\mu \tag{4.3}$$

and obtain

$$\begin{aligned} \det(\lambda I-H_\omega(z)) &= \frac{1}{(1+z/d)^n}\det[\sqrt{\lambda}\,\omega\mu I-\omega(\lambda\hat{L}+\hat{U})] \\ &= \left(\frac{\omega\sqrt{\lambda}}{1+z/d}\right)^n \det\left[\mu I-\left(\sqrt{\lambda}\,\hat{L}+\frac{1}{\sqrt{\lambda}}\,\hat{U}\right)\right] \\ &= \left(\frac{\omega\sqrt{\lambda}}{1+z/d}\right)^n \det[\mu I-(\hat{L}+\hat{U})] \\ &= 0. \end{aligned}$$

If $\lambda=0$ in (4.2) we must have $\omega=1$, i.e. Gauss–Seidel iteration has always the eigenvalue 0. The deduction above may also be utilized to prove the second part of the theorem. □

The spectral radius of the dynamic SOR iteration method is given by the following theorem.

THEOREM 4.2. *Let A be a consistently ordered matrix with constant positive diagonal $D=dI>0$ and assume that the eigenvalues of the corresponding Jacobi matrix $J(0)=D^{-1}(L+U)$ are real with $\mu=\rho(J(0))$. Then the spectral radius for dynamic* SOR *iteration method $\rho(\mathscr{H}_\omega)$, $0<\omega<2$ is determined by the smooth curve*

$$\rho(\mathscr{H}_\omega)=\begin{cases} 1-\omega+\frac{1}{2}(\omega\mu)^2+\omega\mu\sqrt{1-\omega+\frac{1}{4}(\omega\mu)^2}, & \omega\leqq\omega_d, \\ \dfrac{8(\omega-1)^2}{8(\omega-1)-(\omega\mu)^2}, & \omega>\omega_d \end{cases} \tag{4.4}$$

where $\omega_d=(4/3)(2-\sqrt{4-3\mu^2})/\mu^2$.

Further, the curve has the following properties

(i) *$\rho(\mathscr{H}_\omega)$ has its minimum at*

$$\omega_{\text{opt}}=\frac{4}{4-\mu^2}, \qquad 0<\mu<1.$$

(ii) $\rho(\mathscr{H}_\omega)<1$, *when*

$$0<\omega<\frac{3+\sqrt{1-\mu^2}}{2+(\mu/2)^2}, \qquad 0<\mu<1.$$

(iii) *For* $0<\mu<1$, $\omega_d>1$ *and* $\omega_{\text{opt}}\geqq\omega_d$. *The limit values are*

$$\omega_{\text{opt}}\to\frac{4}{3} \quad \text{as } \mu\to 1, \qquad \omega_d\to\frac{4}{3} \quad \text{as } \mu\to 1.$$

Remark 4.1. $\rho(\mathscr{H}_\omega)$ given by (4.4) is an increasing function of μ. See Figs. 4.1 and 4.2.

Proof. Let us denote any of the real eigenvalues of $J(0)$ by μ_i. From (4.3) we may solve

$$\frac{z}{d}=-1+\frac{1-\omega}{\lambda}+\frac{\omega\mu_i}{\sqrt{\lambda}},$$

or with a more convenient notation $\lambda=|\lambda|\,e^{it}$ where $t\in[0,2\pi)$

$$\frac{z}{d}=-1+\frac{1-\omega}{|\lambda|}\,e^{-it}+\frac{\omega\mu_i}{\sqrt{|\lambda|}}\,e^{-i(t/2)}.$$

If we consider t as a parameter, we obtain equilibrium lines for $|\lambda|$ by letting t vary $0\to 4\pi$. As $\rho(\mathscr{H}_\omega)=\max_{\xi\in\mathbf{R}}\rho(H_\omega(i\xi))$ by Theorem 2.2 we observe that $\rho(\mathscr{H}_\omega)$ can also be obtained by determining the value of $|\lambda|$ for the equilibrium line that osculates the imaginary axis. From the conditions $\operatorname{Re} z(t)=\operatorname{Re} z'(t)=0$ we deduce that either the osculation point is at the origin giving the same value for $|\lambda|$ as the well-known static iteration method (the first part of (4.4)), or the osculation occurs at a certain point $(0, i\xi)$, $\xi\neq 0$ (and by symmetry at $(0,-i\xi)$) giving

$$|\lambda|=\frac{8(\omega-1)^2}{8(\omega-1)-(\omega\mu_i)^2}.$$

Since $|\lambda|$ clearly is an increasing function of μ in both expressions, we obtain (4.4). The value for ω_d may be obtained by determining the least value of ω for which the osculation point with the imaginary axis moves away from the origin. The properties (i)-(iii) may be directly calculated from (4.4). □

We obtain immediately a result for Gauss-Seidel iteration method by putting $\omega=1$ in (4.4).

COROLLARY 4.1. *Let A be a consistently ordered matrix with positive constant diagonal dI. Then*

$$\rho(\mathscr{H}_1)=\sup_{z\in\mathbf{C}_+}\rho(H_1(z))=\rho(H_1(0))=(\rho(J(0)))^2=(\rho(\mathscr{J}))^2.$$

Proof. The last equality follows since the dynamic Jacobi matrix is

$$J(z)=(D+z)^{-1}(L+U)=(1+z/d)^{-1}J(0)$$

and, on the other hand $\rho(\mathscr{J})=\sup_{z\in\mathbf{C}_+}\rho(J(z))$. □

We can remove the assumption on $D=dI$ at the expense of requiring that A be additionally symmetric. Observe that in Theorem 4.1 we only used Proposition 4.1, while in Theorem 4.3 we need to go back to the definition of consistently ordered matrices.

THEOREM 4.3. *Suppose A is a real symmetric positive definite consistently ordered matrix and let $\mu = \rho(J(0))$. Then*

$$\rho(\mathscr{H}_\omega) = 1 - \omega + \tfrac{1}{2}(\omega\mu)^2 + \omega\mu\sqrt{1 - \omega + \tfrac{1}{4}(\omega\mu)^2} \quad \text{for } \omega \leqq \omega_d,$$

$$\rho(\mathscr{H}_\omega) \leqq \frac{8(\omega-1)^2}{8(\omega-1) - (\omega\mu)^2} \quad \text{for } \omega > \omega_d.$$

Proof. We shall need the following stronger version of Proposition 4.1.

LEMMA 4.1. *Eigenvalues of the matrix $\beta D + \gamma I - \alpha L - (1/\alpha)U$ are independent of $\alpha \neq 0$ for any complex β and γ.*

We postpone the proof of the lemma.

To prove Theorem 4.3, we proceed first as in the proof of Theorem 4.1 to obtain

$$\det(\lambda I - H_\omega(z)) = \frac{1}{(d_1+z)\cdots(d_n+z)} \cdot \det\left[(\lambda+\omega-1)D + \lambda z I - \omega\sqrt{\lambda}\left(\sqrt{\lambda}L + \frac{1}{\sqrt{\lambda}}U\right)\right].$$

The determinant above becomes

$$\left(\frac{1}{\omega\sqrt{\lambda}}\right)^n \det\left[\frac{\lambda+\omega-1}{\omega\sqrt{\lambda}}D + \frac{\lambda z}{\omega\sqrt{\lambda}}I - \left(\sqrt{\lambda}L + \frac{1}{\sqrt{\lambda}}U\right)\right],$$

which according to Lemma 4.1 has the same value, if we replace $\sqrt{\lambda}L + (1/\sqrt{\lambda})U$ by $L+U$. If λ is an eigenvalue of $H_\omega(z)$, then we must have

$$\det((\lambda+\omega-1)D + \lambda z I - \omega\sqrt{\lambda}(L+U)) = 0.$$

If so, then $\exists x \in \mathbf{C}^n$ such that $|x| = 1$ and

$$(\lambda+\omega-1)Dx + \lambda z x - \omega\sqrt{\lambda}(L+U)x = 0$$

$$\Rightarrow (\lambda+\omega-1)(x, Dx) + \lambda z - \omega\sqrt{\lambda}(x, (L+U)x) = 0.$$

Defining $d := (x, Dx)$ and $\mu d := (x, (L+U)x)$ the assumptions imply that both d and μ are real and $d > 0$ also. After dividing the equation with d we obtain

$$\lambda + \omega - 1 + \lambda\frac{z}{d} - \omega\sqrt{\lambda}\mu = 0.$$

This is just the equation (4.3) and because of Remark 4.1 the conclusion for "$\leqq$" of Theorem 4.3 follows from $\mu \leqq \rho(J(0))$. To see this, we observe that μ satisfies

$$\mu = \frac{(x, (L+U)x)}{(x, Dx)} \leqq \max_y x \frac{(y, (L+U)y)}{(y, Dy)}.$$

Using the symmetry of A, we can find the maximum value from

$$\operatorname{grad}\frac{(y, (L+U)y)}{(y, Dy)} = \frac{1}{(y, Dy)^2}[(y, Dy)2(L+U)y - (y, (L+U)y)2Dy] = 0,$$

$$\Rightarrow (L+U)y = \frac{(y, (L+U)y)}{(y, Dy)}Dy,$$

$$\Rightarrow J(0)y = \frac{(y, (L+U)y)}{(y, Dy)}y,$$

$$\Rightarrow \mu \leqq \rho(J(0)).$$

On the other hand, $\rho(\mathscr{H}_\omega) \geqq \rho(H_\omega(0))$ which equals the right-hand side of (4.4) for $\omega \leqq \omega_d$, so that we must have equality for at least $\omega \leqq \omega_d$. □

Proof of Lemma 4.1. Using Theorem 3.2 on p. 146 in [10] one sees that if A is consistently ordered so is $A+\eta I$ for any $\eta \in \mathbf{C}$, since the same compatible ordering vector works for both A and $A+\eta I$. The result then follows from Proposition 4.1. □

The spectral radius for dynamic SOR iteration with nonconstant diagonal of A is shown in Fig. 4.3 and compared with $\rho(\mathscr{H}_\omega)$ given by (4.4).

COROLLARY 4.2. *If A is a real symmetric positive definite consistently ordered matrix, then*

$$\rho(\mathscr{H}_1) = (\rho(J(0)))^2 = (\rho(\mathscr{J}))^2 < 1.$$

Proof. The second equality can be proved using the proof of Theorem 3.3. The inequality is a well-known result from the theory of static iterations. □

If we solve systems of very large dimensions, it is clear that the parameter $\mu = \rho(J(0))$ is near unity. For example, solving equation of heat conduction type $u_t = \Delta u$ with central differences for space discretization results in a system of linear differential equations where the coefficient matrix is consistently ordered. If we use N discretization points in each space direction (our problem may be one-, two- or three-dimensional) we find that all the eigenvalues μ_i of $J(0)$ are real and $\mu = \rho(J(0)) = \cos \pi/(N+1) = \cos \pi h$, where h is the mesh size.

We suggest one should always choose $\omega = \frac{4}{3}$ to obtain fast convergence with dynamic SOR for consistently ordered matrices. In Fig. 4.4 we show the difference between $\rho(\mathscr{H}_{4/3})$ and $\rho(\mathscr{H}_{\omega_{\text{opt}}})$. We remind that in the example above $\mu = 0.9$ means that we use only 6 discretization points while for 100 points the parameter $\mu \approx 0.9995$. We find that the two curves are almost equal for $\mu > 0.98$. The asymptotic behavior can be approximated from (4.4)

$$\rho(\mathscr{H}_{\omega_{\text{opt}}}) \approx 1 - 2\pi^2 h^2, \qquad \omega_{\text{opt}} \approx \frac{4}{3} - \frac{4}{9}\pi^2 h^2.$$

We recall that the asymptotic behavior of optimal static SOR is approximately

$$\rho(H_{\omega_{\text{opt}}}(0)) \approx 1 - 2\pi h,$$

with $\omega_{\text{opt}} \approx 2 - 2\pi h$. Because $\rho(\mathscr{H}_1) = (\rho(\mathscr{J}))^2$ for Gauss-Seidel method we have

$$\rho(\mathscr{H}_1) \approx 1 - \pi^2 h^2.$$

Also the minimum point in the static SOR curve is very sharp, making it difficult to obtain the minimum spectral radius in practice. On the other hand, the minimum in spectral radius of dynamic SOR is quite flat.

For consistently ordered matrices we have seen that the optimal relaxation parameter can vary between $\omega_{\text{opt}} = 1$ for $\mu = 0$ and $\omega_{\text{opt}} = \frac{4}{3}$ for $\mu = 1$. Also the range of ω for which dynamic SOR converges varies from $\omega \in (0, 2)$ to $\omega \in (0, \frac{4}{3})$, respectively. To show that, in general, ω_{opt} fails to fit between the above extremal values, we present the following examples.

Positive definite matrices have a special position in the static iteration theory. For example, the matrix

$$\begin{pmatrix} 1 & a & a \\ a & 1 & a \\ a & a & 1 \end{pmatrix} \tag{4.5}$$

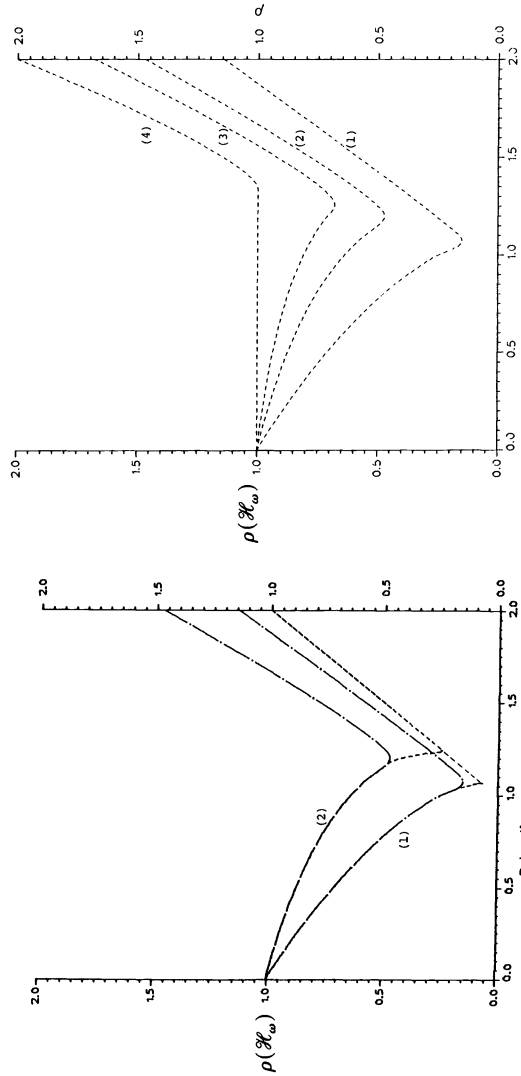


FIG. 4.1. *$\rho(\mathscr{H}_\omega)$ as a function of ω with $\mu = 0.5$ (1) and $\mu = 0.8$ (2). The dashed lines indicate the corresponding curves for static iteration, i.e. $\rho(H_\omega(0))$.*

FIG. 4.2. *$\rho(\mathscr{H}_\omega)$ as a function of ω with $\mu = 0.5$ (1), $\mu = 0.8$ (2), $\mu = 0.9$ (3) and $\mu = 1.0$ (4).*

FIG. 4.3. *The spectral radius of dynamic* SOR *for the matrix*

$$\begin{pmatrix} 2 & -1 \\ -1 & 0.781 \end{pmatrix}$$

(dashed line). For this matrix $\rho(J(0)) = 0.8$ and the chaindotted line gives the corresponding curve $\rho(\mathscr{H}_\omega)$.

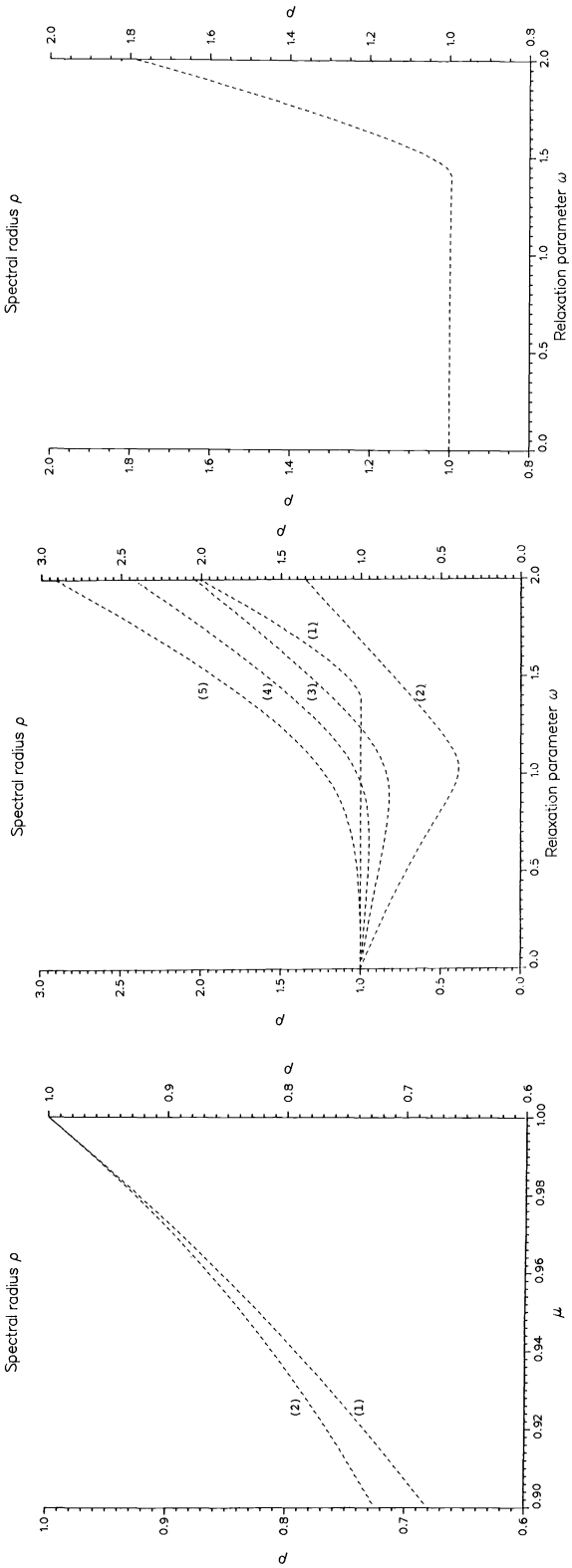


FIG. 4.4. *The spectral radius of dynamic* SOR *for consistently ordered matrices when the optimal relaxation parameter is used* (1) *and* $\omega = \frac{4}{3}$ *is used* (2).

FIG. 4.5. *The spectral radius of dynamic* SOR *iteration matrix for the matrix* (4.5). *The curves correspond to* $a = -0.5$ (1), $a = 0.5$ (2), $a = 0.8$ (3), $a = 0.9$ (4) *and* $a = 1.0$ (5).

FIG. 4.6. *The spectral radius of dynamic* SOR *for the matrix* (4.6) *with* $\varepsilon = 10$.

has eigenvalues $\mu_{1,2}=1-a$, $\mu_3=1+2a$ and is thus positive definite if $-\frac{1}{2}<a<1$. The spectral radius of the dynamic SOR iteration matrix is drawn in Fig. 4.5 for several values of a. We find the convergent range of ω go to zero as $a\to 1$, so that also $\omega_{\text{opt}}\to 0$ in this example.

Consider the positive definite matrix

$$\begin{pmatrix} \varepsilon & 0.999 \\ 0.999 & 1/\varepsilon \end{pmatrix} \tag{4.6}$$

with $\varepsilon=10$. The spectral radius of $\mathscr{H}_\omega$ for this matrix is drawn in Fig. 4.6, which shows that $\omega_{\text{opt}}\approx 1.41$, so that the limit value $\frac{4}{3}$ for consistently ordered matrices with constant diagonal may be exceeded.

5. Convergence of symmetric SOR. Instead of studying iteration method of the form (2.2) to solve (2.1) we now turn to methods that involve two different phases of the form (2.2)

$$\begin{aligned} &\dot{x}^{(k-(1/2))}+M_1x^{(k-(1/2))}=N_1x^{(k-1)}+g_1,\\ &\dot{x}^{(k)}+M_2x^{(k)}=N_2x^{(k-(1/2))}+g_2,\\ &x^{(k)}(0)=x^{(k-(1/2))}(0)=x_0. \end{aligned} \tag{5.1}$$

Actually we are interested in the symmetric SOR method where

$$\begin{aligned} &M_1=\frac{1}{\omega}D-L, \qquad N_1=\frac{1-\omega}{\omega}D+U,\\ &M_2=\frac{1}{\omega}D-U, \qquad N_2=\frac{1-\omega}{\omega}D+L, \end{aligned} \tag{5.2}$$

and want to prove an analogue of Theorem 2.2 for SSOR.

The solution of (5.1) may be written using integral operator $\mathscr{S}_\omega$, which is defined as

$$\mathscr{S}_\omega u(t):=\int_0^t \Sigma_2(t-s)\left(\int_0^s \Sigma_1(s-r)u(r)\,dr\right)ds \tag{5.3}$$

with

$$\Sigma_1(t)=e^{-tM_1}N_1, \qquad \Sigma_2(t)=e^{-tM_2}N_2.$$

Setting $\Sigma(t):=\int_0^t \Sigma_2(t-s)\Sigma_1(s)\,ds$, we also have

$$\mathscr{S}_\omega u(t)=\int_0^t \Sigma(t-s)u(s)\,ds. \tag{5.4}$$

The solution becomes

$$x^{(k)}=\mathscr{S}_\omega x^{(k-1)}+\varphi$$

where φ is a function of g_1, g_2 and x_0.

Taking the Laplace transform of (5.1) and letting, for simplicity, $g_1=g_2=x_0=0$, we obtain

$$(M_1+zI)\hat{x}^{(k-(1/2))}=N_1\hat{x}^{(k-1)}, \qquad (M_2+zI)\hat{x}^{(k)}=N_2\hat{x}^{(k-(1/2))}$$

which reduces to

$$\hat{x}^{(k)}=(M_2+zI)^{-1}N_2(M_1+zI)^{-1}N_1\hat{x}^{(k-1)}. \tag{5.5}$$

We call the transform matrix of (5.5) $S_\omega(z)$ so that

$$\hat{x}^{(k)} = S_\omega(z)\hat{x}^{(k-1)}. \tag{5.6}$$

Since we assume $D>0$ $S_\omega(z)$ is for SSOR method analytic and bounded for Re $z > -\eta$ with some $\eta > 0$. We conclude $S_\omega(z)$ to be a Laplace transform of a matrix-valued L^1-function.

In order for λ to be regular for $\mathcal{S}_\omega$ we need to be able to solve $(\lambda - \mathcal{S}_\omega)u = f$ boundedly for $f \in L^p$. Since this is a convolution-type Volterra equation, the solution is of the form $u = (1/\lambda)f + \mathcal{R}f$ where $\mathcal{R} = R*$ is a Volterra convolution operator. By a matrix valued version of a result of Wiener and Levy [3, p. 68] $R \in L^1$ if and only if $\det[\lambda I - S_\omega(z)] \neq 0$ for Re $z \geqq 0$.

If $\det[\lambda I - S_\omega(z_0)] = 0$ then $\hat{f}(z_0)$ cannot be freely chosen in the associated eigendirection, and λ is not regular for $\mathcal{S}_\omega$. On the other hand, if $\det[\lambda I - S_\omega(z)] \neq 0$ then $\mathcal{R} = R*$ where $R \in L^1$. But then $(1/\lambda) + \mathcal{R}$ is bounded, which implies λ to be regular.

Thus we conclude that

THEOREM 5.1. *If $D>0$, then for the dynamic* SSOR

$$\rho(\mathcal{S}_\omega) = \max_{\xi \in \mathbf{R}} \rho(S_\omega(i\xi))$$

in any L^p-space with $1 \leqq p \leqq \infty$.

Again, we say that the dynamic SSOR converges if $D>0$ and $\sup_{z \in \mathbf{C}_+} \rho(S_\omega(z)) < 1$. We can now state a convergence theorem for H-matrices.

THEOREM 5.2. *For any H-matrix A with $D>0$ and $\mathcal{M}(A)$ a nonsingular M-matrix the dynamic* SSOR *converges whenever*

$$0 < \omega < \frac{2}{1+\rho(|J(0)|)}.$$

Proof. As in Theorem 3.2 the assumptions imply that $\rho(|J(0)|) < 1$. The iteration matrix for dynamic SSOR becomes

$$\begin{aligned} S_\omega(z) &= \left(\frac{1}{\omega}(D+zI) - U\right)^{-1}\left(\frac{1-\omega}{\omega}D + L\right)\left(\frac{1}{\omega}(D+zI) - L\right)^{-1}\left(\frac{1-\omega}{\omega}D + U\right) \\ &= \left(\frac{1}{\omega}(I+zD^{-1}) - \hat{U}\right)^{-1}\left(\frac{1-\omega}{\omega}I + \hat{L}\right)\left(\frac{1}{\omega}(I+zD^{-1}) - \hat{L}\right)^{-1}\left(\frac{1-\omega}{\omega}I + \hat{U}\right) \end{aligned}$$

where $\hat{L} = D^{-1}L$ and $\hat{U} = D^{-1}U$.

$|S_\omega(z)|$ can be estimated as

$$\begin{aligned} |S_\omega(z)| \leqq &\left|\left(\frac{1}{\omega}(I+zD^{-1}) - \hat{U}\right)^{-1}\right|\left(\frac{|1-\omega|}{\omega}I + |\hat{L}|\right) \\ &\cdot \left|\left(\frac{1}{\omega}(I+zD^{-1}) - \hat{L}\right)^{-1}\right|\left(\frac{|1-\omega|}{\omega}I + |\hat{U}|\right). \end{aligned}$$

In the proof of Theorem 3.2 we have shown that

$$\left|\left(\frac{1}{\omega}(D+zI) - L\right)^{-1}\right| \leqq \left(\frac{1}{\omega}|D+zI| - |L|\right)^{-1} \leqq \left(\frac{1}{\omega}|D| - |L|\right)^{-1} \quad \forall z \in \mathbf{C}_+.$$

We use these inequalities to deduce

$$\left|\left(\frac{1}{\omega}(I+zD^{-1})-\hat{L}\right)^{-1}\right| = \left|\left(\frac{1}{\omega}(D+zI)-L\right)^{-1}D\right|$$

$$= \left|\left(\frac{1}{\omega}(D+zI)-L\right)^{-1}\right|\cdot|D| \leqq \left(\frac{1}{\omega}|D|-|L|\right)^{-1}|D|$$

$$= \left(\frac{1}{\omega}I-|\hat{L}|\right)^{-1}.$$

In the same manner

$$\left|\left(\frac{1}{\omega}(I+zD^{-1})-\hat{U}\right)^{-1}\right| \leqq \left(\frac{1}{\omega}I-|\hat{U}|\right)^{-1}.$$

We have thus an upper bound independent of z for $|S_\omega(z)|$

$$|S_\omega(z)| \leqq \left(\frac{1}{\omega}I-|\hat{U}|\right)^{-1}\left(\frac{|1-\omega|}{\omega}I+|\hat{L}|\right)\left(\frac{1}{\omega}I-|\hat{L}|\right)^{-1}\left(\frac{|1-\omega|}{\omega}I+|\hat{U}|\right)$$

$$= \left[\left(\frac{1}{\omega}I-|\hat{L}|\right)\left(\frac{1}{\omega}I-|\hat{U}|\right)\right]^{-1}\left[\left(\frac{|1-\omega|}{\omega}I+|\hat{L}|\right)\left(\frac{|1-\omega|}{\omega}I+|\hat{U}|\right)\right]$$

where the latter form follows because of the commutation of the second and the third factor in the former expression.

If we choose

$$\tilde{M}_\omega = \frac{\omega}{1+|1-\omega|}\left(\frac{1}{\omega}I-|\hat{L}|\right)\cdot\left(\frac{1}{\omega}I-|\hat{U}|\right),$$

$$\tilde{N}_\omega = \frac{\omega}{1+|1-\omega|}\left(\frac{|1-\omega|}{\omega}I+|\hat{L}|\right)\cdot\left(\frac{|1-\omega|}{\omega}I+|\hat{U}|\right),$$

we find

$$\tilde{A}_\omega = \tilde{M}_\omega - \tilde{N}_\omega = \frac{1-|1-\omega|}{\omega}I-(|\hat{L}|+|\hat{U}|) = \frac{1-|1-\omega|}{\omega}I-|J(0)|$$

to be a regular splitting because $\tilde{M}_\omega^{-1} \geqq 0$ and $\tilde{N}_\omega \geqq 0$. Notice that we have

$$|S_\omega(z)| \leqq \tilde{M}_\omega^{-1}\tilde{N}_\omega = \tilde{S}_\omega$$

and the rest of the proof is just the same as in Theorem 3.2. □

The above theorem gives the same range of ω to assure convergence for both dynamic SSOR and SOR methods, but does not tell which one is better—or converges faster. In order to compare these two methods with each other, we study the SSOR method in connection with consistently ordered matrices.

The eigenvalues of the iteration matrix $S_\omega(z)$ cannot be expressed as a function of only one parameter as was the case for $H_\omega(z)$ in § 4. Consequently, one has not been able to deduce a general expression for optimal relaxation parameter even with static SSOR method. We are, however, interested in ω_{opt} for dynamic SSOR, and want especially to find out whether it has an asymptotic behavior similar to ω_{opt} of dynamic SOR.

We computed the spectral radii for $\mathscr{S}_\omega$ in the example that is obtained, for instance, from one-dimensional heat conduction equation $u_t = u_{xx}$ using central differences for spatial discretization. The matrix A becomes

$$\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{pmatrix}. \tag{5.7}$$

Remark that A is not only consistently ordered but also an H-matrix. The spectral radius of the corresponding Jacobi matrix, or μ, is given by $\mu = \cos(\pi/(N+1))$, where N is the dimension of A.

As we increased μ we observed that the dynamic ω_{opt} does not follow the static ω_{opt} but approaches a certain asymptotic value, which is approximately 1.79 for the matrix (5.7). The spectral radius of $\mathscr{S}_\omega$ is drawn in Fig. 5.1 with several values of N. The limit value of ω_{opt} is, of course, not a function of $\mu = \mu(N)$ only, so that the above-mentioned number is no general asymptotic value for SSOR method. Anyway, it is obvious that we again obtain ω_{opt} whose asymptotic value stays less than 2, as it does not in the static method.

In Fig. 5.2 we compare dynamic SOR and SSOR methods when the example above is computed with the dimension of A in (5.7) being $N = 20$, 80. SSOR method takes roughly two times more computational work than SOR method, so it should converge at least twice as fast as SOR method to be more efficient. That is why we have drawn the square root of $\rho(\mathscr{S}_\omega)$ in Fig. 5.2. We observe that SSOR method gives faster convergence in this example. For smaller dimensions of matrix (5.7) SOR method is, however, faster as shown in Fig. 5.3. The curves $\rho(\mathscr{H}_{\omega_{\text{opt}}})$ and $\sqrt{\rho(\mathscr{S}_{\omega_{\text{opt}}})}$ intersect at $\mu \approx 0.96$ which corresponds to the dimension $N = 10$. Figure 5.3 indicates that asymptotically SSOR gives faster convergence.

On the contrary, for the matrix (4.5) the results in Fig. 5.4 show that SOR method gives better convergence than SSOR method.

6. Numerical experiment. We have in this paper considered the dynamic iteration method applied to a system continuous in time. The effect of time discretization to the convergence properties will be a subject of future studies. Anyway, we present here a numerical experiment whose results seem to be in excellent agreement with the convergence speed of the time continuous problem studied. Our example is a heat conduction equation is one space variable

$$\begin{aligned} &u_t = u_{xx}, \qquad 0 < x < 1, \\ &u(t, 0) = u(t, 1) = 0, \qquad t > 0, \\ &u(0, x) = 10x. \end{aligned} \tag{6.1}$$

We used central differences for space derivative $u_{xx} \approx (u_{j+1} - 2u_j + u_{j-1})/(\Delta x)^2$, so that the coefficient matrix A in the time continuous system $u_t = Au$ became consistently ordered. The mesh size in the calculation was $\Delta x = 0.1$ which means that $\mu = \cos(\pi/10) \approx 0.95$ as explained in § 4.

For time discretization we used trapezoidal rule as a kind of safe choice because we were, in fact, interested in the properties of relaxation method applied to the time continuous problem. The time domain was $0 < t \leqq 1.0$ and we had $\Delta t = 0.05$.

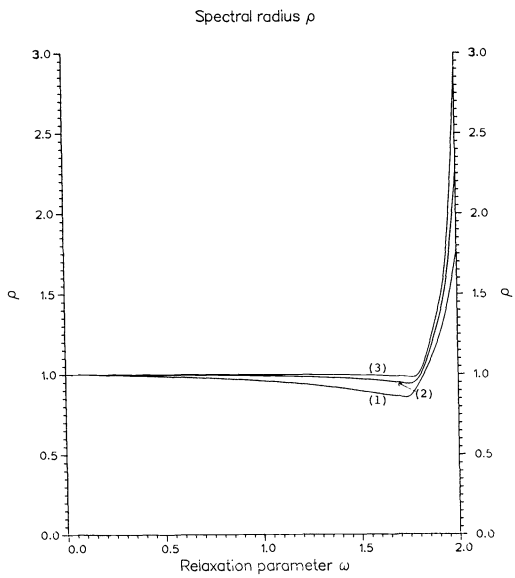


FIG. 5.1. *The spectral radius of dynamic* SSOR *iteration matrix for the matrix* (5.7) *with* $N = 20$ (1), $N = 40$ (2) *and* $N = 80$ (3).

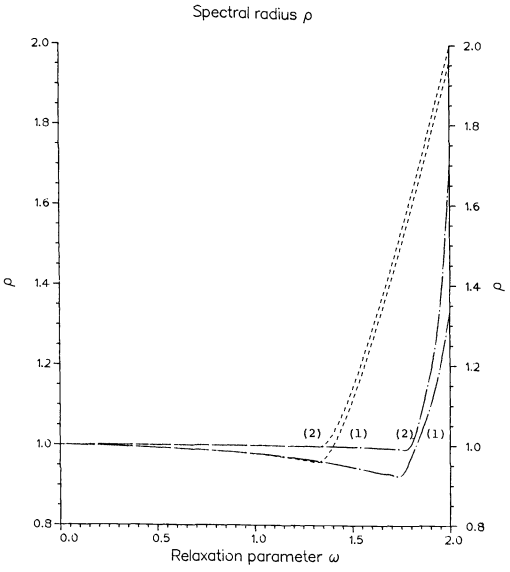


FIG. 5.2. *Comparison between the spectral radii of dynamic* SOR (*dashed line*) *and* SSOR (*chaindotted line*) *for the matrix* (5.7) *with* $N = 20$ (1) *and* $N = 80$ (2). *The curves for* SSOR *method are the square roots of the originals.*

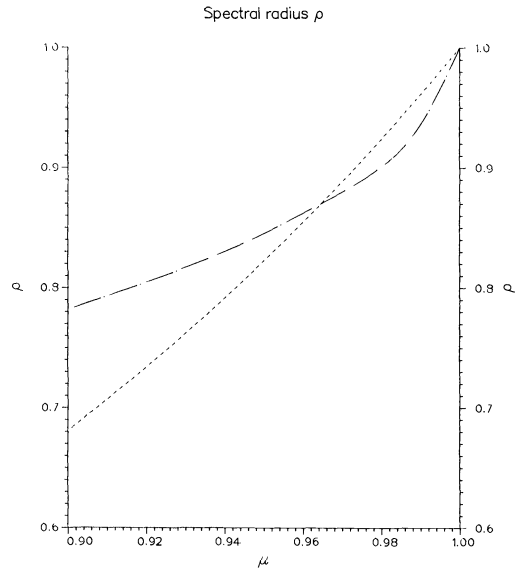


FIG. 5.3. *The spectral radius of dynamic* SOR (*dashed line*) *and* SSOR (*chaindotted line*) *for the matrix* (5.7). *The parameter* $\mu = \cos(\pi/(N+1))$ *where* N *is the dimension of the matrix* (5.7).

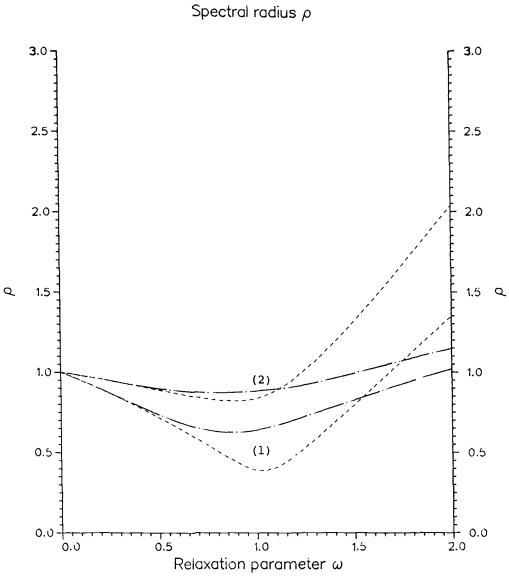


FIG. 5.4. *Comparison between the spectral radii of dynamic* SOR (*dashed line*) *and* SSOR (*chaindotted line*) *for the matrix* (4.5). *The curves* (1) *and* (2) *correspond to* $a = 0.5$ *and* $a = 0.8$, *respectively. Square roots have been taken from the values for the* SSOR *method.*

The iteration started with $u(t_i, x_j) = 0$ for $t_i > 0$ and during the iterations we measured the maximum change in both time and space between two subsequent iterations

$$\varepsilon_k = \max_{i,j} |u_{ij}^k - u_{ij}^{k-1}|. \tag{6.2}$$

The maximum change ε_k was found to travel in time with the iterations. So we saved ε_{k_1} when it occurred at the timestep $t = 0.2$ and ε_{k_2} when maximum error reached the last timestep $t = 1.0$ to approximate the spectral radius by

$$\bar{\rho} = \left(\frac{\varepsilon_{k_2}}{\varepsilon_{k_1}}\right)^{1/(k_2 - k_1)}. \tag{6.3}$$

The theoretical spectral radius for the dynamic SOR method in this example is obtained from (4.2) with $\mu \approx 0.95$. This curve is drawn in Fig. 6.1 as a function of relaxation parameter ω. The discrete points in the figure are experimental results for the spectral radius using (6.3). The numerical results agree very well with the theoretical values and even minimum ρ is found with approximately the same ω_{opt}. The spectral radius for static SOR method is drawn in Fig. 6.1 for comparison.

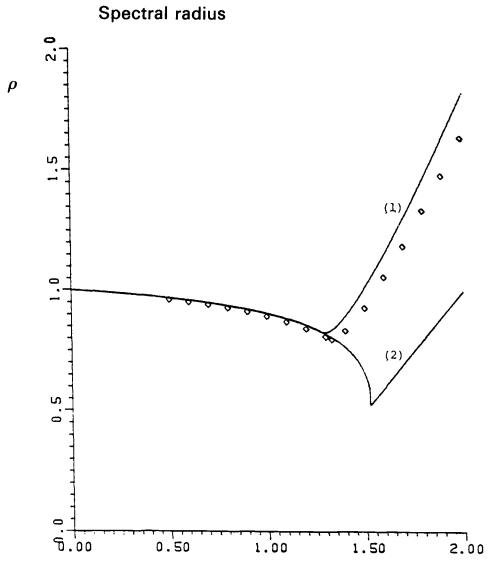


FIG. 6.1. *The spectral radius of dynamic* SOR *for the example in* § 6 *calculated theoretically* (1) *and from numerical experiment* (*discrete points*). *The spectral radius for the corresponding static* SOR *method is also shown* (2).

We detected also that during the iteration ε_k travelled in time faster if relaxation parameter ω was increased. For example, we found $k_2 = 91$ for Gauss-Seidel method and $k_2 = 41$ for $\omega = \frac{4}{3}$, while with underrelaxation $\omega = 0.6$, $k_2 = 210$.

Problem (6.1) was also solved with another initial condition, namely

$$u(0, x) = \begin{cases} 20x, & 0 < x \leqq \frac{1}{2}; \\ 20(1-x), & \frac{1}{2} < x < 1. \end{cases}$$

The numerically obtained spectral radius $\bar{\rho}$ behaved in an exactly same way as in the first example.

REFERENCES

[1] A. BERMAN AND R. J. PLEMMONS, *Nonnegative Matrices in the Mathematical Sciences*, Academic Press, New York, 1979.

[2] C. G. BROYDEN, *Some aspects of consistent ordering*, Numer. Math., 12 (1968), pp. 47–56.

[3] C. CORDUNEANU, *Integral Equations and Stability of Feedback Systems*, Academic Press, New York, 1973.

[4] G. E. FORSYTHE AND W. R. WASOW, *Finite-Difference Methods for Partial Differential Equations*, John Wiley, New York, 1960.

[5] G. H. GOLUB AND G. A. MEURANT, *Résolution numérique des grands systèmes linéaires*, Eyrolles, Paris, 1983.

[6] 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 of ICAS, Vol. CAD-1, No. 3, 1982, pp. 131–145.

[7] A. R. NEWTON AND A. L. SANGIOVANNI-VINCENTELLI, *Relaxation-based electrical simulation*, IEEE Trans. Computer-Aided Design, Vol. CAD-3, No. 4, 1984.

[8] J. M. ORTEGA AND W. C. RHEINBOLDT, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.

[9] R. S. VARGA, *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1962.

[10] D. M. YOUNG, *Iterative Solution of Large Linear Systems*, Academic Press, New York, 1971.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 4, July 1987
© 1987 Society for Industrial and Applied Mathematics

A NEW BASIS IMPLEMENTATION FOR A MIXED ORDER BOUNDARY VALUE ODE SOLVER*

G. BADER† AND U. ASCHER‡

Abstract. The numerical approximation of mixed order systems of multipoint value ordinary differential equations by collocation requires appropriate representation of the piecewise polynomial solutions. B-splines were originally implemented in the general purpose code COLSYS, but better alternatives exist. One promising alternative was proposed by Osborne and discussed by Ascher, Pruess and Russell.

In this paper we analyze the performance of the latter solution representation for cases not previously covered, where the mesh is not necessarily dense. This analysis and other considerations have led us to implement a basis replacement in COLSYS, and we discuss some implementation details. Numerical results are given which demonstrate the improvement in performance of the code.

Key words. collocation, new implementation

AMS (MOS) subject classifications. 65L10, 65L20

1. Introduction. When considering the numerical solution of mixed order systems of multipoint value ordinary differential equations (ODEs), one promising method is that of collocation. Piecewise polynomial collocation at Gaussian points was implemented in a general purpose code COLSYS (Ascher, Christiansen and Russell [6]) which has proved successful in many applications. Still, improvements to this code can be made in a number of aspects.

One such aspect is the representation used for the piecewise polynomial solutions. The B-spline basis (de Boor [9]) used for this purpose in COLSYS was shown by Ascher, Pruess and Russell [7] to be inferior, both in operation count and in conditioning, to a monomial representation proposed by M. Osborne [19], when applied to a linear, higher order ODE. Recently, parts of the collocation theory of de Boor and Swartz [10] for a two-point boundary value problem (BVP) for a higher order ODE were worked out using such a monomial basis (Ascher [1]), enabling a much simpler presentation as well as sharpening of some results. It was shown that the stability constant of the numerical scheme relates to that of the associated first order BVP and to the number of mesh elements alone, for any mesh. Thus, a more effective control of a damped Newton iteration for some difficult nonlinear problems can be applied, because the BVP dependent variables are more directly controlled and certain superfluous ill-conditioning effects which are present in extreme cases when using B-splines are removed.

The analysis in [19], [7], [1] was performed for small h, in case of one higher order ODE. In this paper we discuss extensions of this analysis to stiff BVPs, for a higher order ODE as well as for a first order system of ODEs, and describe an implementation, imbedded in COLSYS, of a monomial solution representation for multipoint value mixed order systems of ODEs. We give numerical examples comparing COLSYS before and after the change of basis. These examples show a gain in efficiency and robustness. The coding of the new representation is simpler as well.

* Received by the editors September 3, 1985; accepted for publication (in revised form) May 14, 1986.

† Institut Angewandt. Mathematik, Universitat Heidelberg, D-6900 Heidelberg, Germany. The work of this author was supported in part by the Natural Sciences and Engineering Research Council of Canada under grant A4306 and the German Research Council.

‡ Department of Computer Science, University of British Columbia, Vancouver, British Columbia, Canada V6T 1W5. The work of this author was supported in part by the Natural Sciences and Engineering Research Council of Canada under grant A4306.

When making this change in COLSYS, we have decided to keep using collocation at Gaussian points. One reason for this choice is that for first order systems we have algebraic stability, see Ascher and Bader [2]. In the analysis here we therefore consider mainly Gaussian points.

2. Solution representation. For the presentation here, it is sufficient to consider two linear differential ODEs: a first order system of order n,

$$D\mathbf{y} = L(x)\mathbf{y} + \mathbf{q}(x), \qquad a < x < b \tag{2.1}$$

where $D \equiv d/dx$, $\mathbf{y}(x)$, $\mathbf{q}(x) \in \mathbf{R}^n$, $L(x) \in \mathbf{R}^{n\times n}$, and a higher order ODE of order $m \geqq 1$,

$$D^m u = \sum_{l=1}^{m} c_l(x) D^{l-1} u + q(x), \qquad a < x < b \tag{2.2}$$

where D^l stands for l-fold differentiation. Of course, (2.2) can be considered as a special case of (2.1) with $n := m$ and

$$\mathbf{y}(x) := (u(x), Du(x), \cdots, D^{m-1}u(x)), \tag{2.3}$$

but in practice we apply collocation directly to the form (2.2).

Consider a k-stage collocation scheme for (2.2), $k \geqq m$. Thus, there is a partition π of $[a, b]$

$$\pi : a = x_1 < x_2 < \cdots < x_N < x_{N+1} = b, \tag{2.4}$$

$$h_i := x_{i+1} - x_i, \qquad h := \max_{1\leqq i\leqq N} h_i$$

and k points

$$0 \leqq \rho_1 < \rho_2 < \cdots < \rho_k \leqq 1 \tag{2.5}$$

which we later choose to be Gaussian points. A piecewise polynomial function $u_\pi(x)$ is sought such that $u_\pi \in \mathbf{P}_{k+m,\pi} \cap C^{m-1}[\mathrm{a}, \mathrm{b}]$;[1] $u_\pi(x)$ satisfies m side conditions associated with (2.2) and given on a subset of π; and $u_\pi(x)$ satisfies the ODE (2.2) at the collocation points

$$x_{ij} = x_i + h_i\rho_j, \qquad 1 \leqq j \leqq k, \quad 1 \leqq i \leqq N. \tag{2.6}$$

This collocation procedure can of course be applied to systems. In particular, for (2.1) we have a piecewise polynomial vector function which satisfies (componentwise) $\mathbf{y}_\pi \in \mathbf{P}_{k+1,\pi} \cap C[a, b]$.

Let us now recall the monomial Runge-Kutta basis representation [7], [1]. For a fixed mesh element $[x_i, x_{i+1}]$ the approximate solution is written as

$$u_\pi(x) = \sum_{j=1}^{m} \frac{(x - x_i)^{j-1}}{(j-1)!} y_{ij} + h_i^m \sum_{j=1}^{k} \psi_{mj}\left(\frac{x - x_i}{h_i}\right) z_{ij} \tag{2.7}$$

where $\psi_{mj} \in \mathbf{P}_{k+m}$ are defined by

$$D^{l-1}\psi_{mj}(0) = 0, \qquad 1 \leqq l \leqq m, \tag{2.8a}$$

$$D^m\psi_{mj}(\rho_l) = \delta_{jl}, \qquad 1 \leqq l \leqq k, \tag{2.8b}$$

$$1 \leqq j \leqq k.$$

[1] We say that v is in $\mathbf{P}_{k+m}$ if $v(x)$ is a polynomial of order $k+m$ (degree $<k+m$) on an appropriate interval, and that v is in $\mathbf{P}_{k,m,\pi}$ if $v(x)$ is a piecewise polynomial which is in $\mathbf{P}_{k+m}$ on each subinterval of the mesh π.

Thus

$$y_{ij} = D^{j-1}u_\pi(x_i), \qquad 1 \leqq j \leqq m, \tag{2.9a}$$

$$z_{ij} = D^m u_\pi(x_{ij}), \qquad 1 \leqq j \leqq k. \tag{2.9b}$$

Note that the polynomials ψ_{mj} satisfy

$$\psi_{rj} \equiv D^{m-r}\psi_{mj}, \qquad 1 \leqq r \leqq m, \quad 1 \leqq j \leqq k \tag{2.10}$$

which is rather useful when collocating for mixed order systems of ODEs. Note also that these functions are independent of the mesh element i.

Application of the collocation equations now yields k equations for $\mathbf{y}_i := (y_{i1}, \cdots, y_{im})^T$ and $\mathbf{z}_i := (z_{i1}, \cdots, z_{ik})^T$,

$$-V_i\mathbf{y}_i + W_i\mathbf{z}_i = \mathbf{q}_i, \qquad 1 \leqq i \leqq N \tag{2.11}$$

where V_i and W_i are matrices of order $k \times m$ and $k \times k$ respectively, with entries

$$V^i_{rj} = \sum_{l=1}^{j} \frac{c_l(x_{ir})(h_i\rho_r)^{j-l}}{(j-l)!}, \qquad 1 \leqq r \leqq k, \quad 1 \leqq j \leqq m, \tag{2.12}$$

$$W^i_{rj} = \delta_{rj} - \sum_{l=1}^{m} c_l(x_{ir})h_i^{m+1-l}D^{l-1}\psi_{mj}(\rho_r), \qquad 1 \leqq r, j \leqq k, \tag{2.13}$$

and

$$\mathbf{q}_i := (q(x_{i1}), \cdots, q(x_{ik}))^T. \tag{2.14}$$

Defining the $k \times k$ matrices $A^{(l)}$ by

$$A^{(m-l)}_{rj} := D^l\psi_{mj}(\rho_r) = \psi_{m-l,j}(\rho_r), \qquad 1 \leqq r, \quad j \leqq k, \quad 0 \leqq l \leqq m \tag{2.15}$$

(note $A^{(0)} = I$) and the constants

$$\xi_l := c_l(x_i)h_i^{m+1-l}, \qquad 1 \leqq l \leqq m, \tag{2.16}$$

we can write W_i, upon freezing $c_l(x)$ along the ith element $[x_i, x_{i+1})$, as

$$W_i = I - \sum_{l=1}^{m} \xi_l A^{(m+1-l)}. \tag{2.17}$$

Next, we obtain m additional relations from the global continuity requirements,

$$C_i\mathbf{y}_i + D_i\mathbf{z}_i = \mathbf{y}_{i+1}, \qquad 1 \leqq i \leqq N \tag{2.18}$$

where C_i is an $m \times m$ upper triangular matrix with entries

$$C^i_{rj} = \frac{h_i^{j-r}}{(j-r)!}, \qquad j \geqq r \tag{2.19}$$

and D_i is an $m \times k$ matrix with entries

$$D^i_{rj} = h_i^{m+1-r}D^{r-1}\psi_{mj}(1), \qquad 1 \leqq r \leqq m, \quad 1 \leqq j \leqq k. \tag{2.20}$$

Defining the k-vectors $\mathbf{b}^{(l)}$ by

$$b^{(m-l)}_j := D^l\psi_{mj}(1), \qquad 1 \leqq j \leqq k, \quad 0 \leqq l \leqq m-1 \tag{2.21}$$

and $\mathbf{b} := \mathbf{b}^{(1)}$, (note that $\mathbf{b}^{(1)}$ and $A^{(1)}$ are the usual Runge–Kutta coefficients for a first order ODE; see e.g. [2]) we claim the following.

LEMMA 1. *Assume that the k points of* (2.5) *satisfy the orthogonality conditions*

$$\int_0^1 \phi(t) \prod_{l=1}^{k} (t-\rho_l)\, dt = 0, \qquad \phi \in \mathbf{P}_m \tag{2.22a}$$

(*this certainly holds for Gaussian points and* $k \geqq m$). *Then*

$$(\mathbf{b}^{(m+1-l)})^T = \mathbf{b}^T A^{(m-l)}, \qquad 1 \leqq l \leqq m. \tag{2.22b}$$

Hence

$$D_i = \operatorname{diag}\{h_i^m, \cdots, h_i\} \begin{pmatrix} \mathbf{b}^T A^{(m-1)} \\ \vdots \\ \mathbf{b}^T A^{(1)} \\ \mathbf{b}^T A^{(0)} \end{pmatrix}. \tag{2.22c}$$

Proof. The precision of the quadrature formula with $\mathbf{b}$ is guaranteed by (2.22a) to be at least $k+m$. Hence

$$\sum_{r=1}^{k} b_r D^l \psi_{mj}(\rho_r) = \int_0^1 D^l \psi_{mj}(t)\, dt = D^{l-1}\psi_{mj}(1). \qquad \square$$

The linear relations (2.11) and (2.18) are now used to eliminate $\mathbf{z}_i$ and express a relation between the solution values at the end points of the ith element,

$$\Delta_i \mathbf{y}_{i+1} = \Gamma_i \mathbf{y}_i + \mathbf{r}_i, \qquad 1 \leqq i \leqq N. \tag{2.23}$$

If W_i is nonsingular (which clearly holds for h_i sufficiently small) then

$$\Delta_i = I, \quad \Gamma_i = C_i + D_i W_i^{-1} V_i, \quad \mathbf{r}_i = D_i W_i^{-1} \mathbf{q}_i. \tag{2.24}$$

However, exceptional cases may occur if some ξ_l of (2.16) are not small. This is considered in the next section. The relations (2.23) together with m boundary conditions form a linear algebraic system of a familiar form for $u_\pi(x)$ and its first $m-1$ derivatives at mesh points. The solution of this system has been discussed elsewhere, see [4], [7], [11] and references therein.

The above description can be applied also for the first order system (2.1), taking $m = 1$. An implicit Runge-Kutta scheme results.

In this paper we concentrate on Gaussian points. One property of these points is that

$$\rho_r = 1 - \rho_{k+1-r}, \qquad 1 \leqq r \leqq k \tag{2.25}$$

and the obtained schemes are symmetric. This implies that, for $0 \leqq t \leqq 1$,

$$D^m \psi_{mj}(t) = D^m \psi_{m,k+1-j}(1-t), \qquad 1 \leqq j \leqq k. \tag{2.26}$$

Also,

$$\rho_1 > 0 \tag{2.27}$$

(unlike Lobatto schemes). Then it easily follows that the matrices $A^{(l)}$ are nonsingular, $0 \leqq l \leqq m$.

3. Local parameter elimination. From (2.16), (2.17) it follows immediately that for h_i sufficiently small, W_i is nonsingular and

$$W_i^{-1} = I + O(h_i).$$

Hence, the elimination of $\mathbf{z}_i$ as in (2.24) is sound and the obtained method is stable, with a stability constant comparable to that of the BVP being solved [1].

However, in some (stiff) cases, the step size h_i taken may be large, for efficiency purposes, so that ξ_l in (2.16) are not all small. Then it is not clear that W_i^{-1} exists and is bounded. Indeed, Osborne [19] proposed to apply (a more expensive) QR decomposition to $\binom{W_i}{D_i}$ to eliminate $\mathbf{z}_i$. In order to show that the construction of the difference scheme (2.23) is then sound, it is necessary to show that this augmented $(k+m)\times k$ matrix has a full rank. Otherwise, this results in an unexplained redundancy in the method.

3.1. First-order systems. The collocation schemes based on Gaussian points are known to be A-stable. Since $A^{(1)}$ is nonsingular, they are also D-stable, see Hairer [13].

Consider at first one equation

$$Dy = \lambda y. \tag{3.1}$$

We write the resulting equations (2.11), (2.18) as

$$\left[\begin{array}{c|c|c} -\lambda\mathbf{1} & I-\xi A^{(1)} & 0 \\ \hline 1 & h\mathbf{b}^T & -1 \end{array}\right]\begin{bmatrix} y_i \\ z_i \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} 0 \\ \hline 0 \end{bmatrix} \tag{3.2}$$

where $\xi := \xi_1 = \lambda h$, $h := h_i$ and $\mathbf{1} := (1, \cdots, 1)^T \in \mathbf{R}^k$. Formally then,

$$h\mathbf{z}_i = (I - \xi A^{(1)})^{-1}\xi\mathbf{1}.$$

Now, if $\mathrm{re}\,(\lambda) < 0$ then the A-stability of the method and the nonsingularity of $A^{(1)}$ imply that $I - \xi A^{(1)}$ has a bounded inverse and that $(I-\xi A^{(1)})^{-1}\xi\mathbf{1}$ is bounded. But for BVPs we must also consider the case $\mathrm{re}\,(\lambda) > 0$. If $|\xi| \gg 1$ then $I - \xi A^{(1)}$ again has a bounded inverse as above, and $(I-\xi A^{(1)})^{-1}\xi\mathbf{1} \approx (-A^{(1)})^{-1}\mathbf{1}$ is bounded. But if $|\xi| = O(1)$ then a singularity may occur. For example, the midpoint scheme ($k=1$) yields

$$y_{i+1} = \frac{1+\xi/2}{1-\xi/2} y_i$$

and a singularity occurs at $\xi = 2$. The remedy is to write

$$y_i = \frac{1-\xi/2}{1+\xi/2} y_{i+1}$$

corresponding to integration in the stable direction $-x$.

Similarly for higher order schemes. If $\mathrm{re}\,(\xi) > 0$ and $I - \xi A^{(1)}$ is singular (which may occur when $|\xi| = O(1)$) then we multiply the last row in (3.2) by λ and add to each of the first k, obtaining the matrix

$$\left[\begin{array}{c|c|c} 0 & I-\xi A^{(1)} + \xi\mathbf{1}\mathbf{b}^T & -\lambda\mathbf{1} \\ \hline 1 & h\mathbf{b}^T & -1 \end{array}\right]. \tag{3.3}$$

Because of the symmetry of the points (see [2, eq. (10)]),

$$A^{(1)} + EA^{(1)}E = \mathbf{1}\mathbf{b}^T \tag{3.4}$$

where $E \in \mathbf{R}^{k\times k}$ is defined as

$$E := \begin{pmatrix} & & & 1 \\ & & 1 & \\ & \iddots & & \\ 1 & & & \end{pmatrix}. \tag{3.5}$$

Hence in (3.3) we have

$$I - \xi A^{(1)} + \xi \mathbf{1}\mathbf{b}^T = I + \xi E A^{(1)} E.$$

But $EA^{(1)}E$ is easily seen to have the same eigenvalues as $A^{(1)}$, whence $I + \xi EA^{(1)}E$ has a bounded inverse. Moreover, the first k rows of (3.3) clearly correspond to a reversal in the direction of integration.

This shows, in particular, that the matrix $\binom{W_i}{D_i}$ has full rank and a sound construction of (2.23) is possible. The results can be easily extended to a first order system of equations (2.1) where L is a constant matrix, assuming that there exists a well-conditioned transformation matrix S such that the following hold:

$$S^{-1}LS = \begin{pmatrix} \Lambda_- & & \\ & \Lambda_0 & \\ & & \Lambda_+ \end{pmatrix} \equiv \Lambda \tag{3.6a}$$

where Λ_- is a diagonal block with eigenvalues each satisfying

$$\mathrm{re}\,(\lambda) < 0, \quad |\lambda h| \gtrsim 1, \quad |\mathrm{im}\,(\lambda)| \leqq \mathrm{const}\,|\mathrm{re}\,(\lambda)|, \tag{3.6b}$$

Λ_0 is a square block satisfying

$$h\|\Lambda_0\| \ll 1 \tag{3.6c}$$

and Λ_+ is a diagonal block with eigenvalues satisfying

$$\mathrm{re}\,(\lambda) > 0, \quad |\lambda h| \gtrsim 1, \quad |\mathrm{im}\,(\lambda)| \leqq \mathrm{const}\,|\mathrm{re}\,(\lambda)|. \tag{3.6d}$$

Generally for a system with constant coefficients, the $(k+1)n \times (k+2)n$ matrix corresponding to (3.2) has the form

$$\left(\begin{array}{c|c|c} -L & & \\ \vdots & I - hA^{(1)} \otimes L & 0 \\ -L & & \\ \hline I & hb_1 I \cdots hb_k I & -I \end{array}\right). \tag{3.7}$$

Multiplying each row block in (3.7) by S^{-1} and performing the change of variables

$$\mathbf{w}_i := S^{-1}\mathbf{y}_i, \qquad \mathbf{w}_{i+1} := S^{-1}\mathbf{y}_{i+1}, \quad \mathbf{w}_{ij} := S^{-1}\mathbf{z}_{ij} \tag{3.8}$$

we obtain the matrix

$$\left(\begin{array}{c|c|c} -\Lambda & & \\ \vdots & I - hA^{(1)} \otimes \Lambda & 0 \\ -\Lambda & & \\ \hline I & hb_1 I \cdots hb_k I & -I \end{array}\right). \tag{3.9}$$

Now, for the block Λ_+ in each Λ we can again change the direction of integration by multiplying the last rows in (3.9) by Λ_+ and adding to the bottom portions of each of the first k blocks of n rows. This results in a nonsingular matrix, showing that the original $\binom{W_i}{D_i}$ matrix is of full rank. Moreover, we can rescale each of the row-blocks corresponding to Λ_- and Λ_+, obtaining a sound construction (2.23) for all values λh. Note further that the diagonality assumption on the blocks Λ_- and Λ_+ can be weakened, requiring only that they be diagonally dominated (cf. Kreiss, Nichols and Brown [16]).

Finally, we allow the matrix $L(x)$ to vary, but assume that there exists a *smooth* transformation $S(x)$ such that

$$S^{-1}(x)L(x)S(x) = \begin{pmatrix} \Lambda_-(x) & & \\ & \Lambda_0(x) & \\ & & \Lambda_+(x) \end{pmatrix} \equiv \Lambda(x), \tag{3.10}$$

$$\|S(x)\|\,\|S^{-1}(t)\| \leqq \text{const}, \qquad x_i \leqq x,\ t \leqq x_{i+1}, \tag{3.11}$$

with Λ_-, Λ_0 and Λ_+ blocks as in (3.6), Λ_- and Λ_+ are diagonally dominated and smoothly varying, and the partition is independent of x (cf. [16]). In place of (3.7) we now get

$$\left(\begin{array}{c|c|c} -L(x_{i1}) & & \\ \vdots & I - h\ \mathrm{diag}\,\{L(x_{ij})\}A^{(1)} \otimes I & 0 \\ -L(x_{ik}) & & \\ \hline I & hb_1 I \quad \cdots \quad hb_k I & -I \end{array}\right)$$

so, we multiply the jth block of n rows by $S^{-1}(x_{ij})$ and define

$$\mathbf{w}_i := S^{-1}(x_i)\mathbf{y}_i, \quad \mathbf{w}_{ij} := S^{-1}(x_{ij})\mathbf{z}_{ij}, \quad 1 \leqq j \leqq k, \tag{3.12}$$

$$\mathbf{w}_{i+1} := S^{-1}(x_{i+1})\mathbf{y}_{i+1}.$$

This gives, in place of (3.9), the matrix

$$\left(\begin{array}{c|c|c} -\Lambda(x_{i1})S^{-1}(x_{i1})S(x_i) & & \\ \vdots & I - h\Omega & 0 \\ -\Lambda(x_{ik})S^{-1}(x_{ik})S(x_i) & & \\ \hline S(x_i) & hb_1 S(x_{i1}) \cdots hb_k S(x_{ik}) & -S(x_{i+1}) \end{array}\right),$$

with the blocks of Ω given by $\Omega_{jl} = \Lambda(x_{ij})S^{-1}(x_{ij})S(x_{il})a_{jl}$, $1 \leqq j,\ l \leqq k$. Multiplication of the last n rows by

$$\begin{pmatrix} 0 & \\ & \Lambda_+(x_{ij}) \end{pmatrix} S^{-1}(x_{ij})$$

and adding to the jth block of n rows $1 \leqq j \leqq k$, again yields the desired result for h_i sufficiently small (without requiring that $h_i\|L(x_i)\|$ be small !). This is because

$$S^{-1}(x_{ij})S(x_{il}) = I + O(h_i), \qquad 1 \leqq j,\ l \leqq k,$$

$$\Lambda_\pm(x_{ij}) = \Lambda_\pm(x_i)(I + O(h_i)), \qquad 1 \leqq j \leqq k,$$

and because rows corresponding to Λ_- and to Λ_+ are not mixed in the process. Thus, the variable coefficient case can be treated as a perturbation of the constant coefficient one, as e.g. in [13]. We have proved the following.

THEOREM 1. *If $L(x)$ can be decomposed as described above in* (3.10), (3.11), (3.6), *then for h_i small enough the matrix $\binom{W_i}{D_i}$ has a full rank, and the local parameters $\mathbf{z}_i$ can be stably eliminated and retrieved in terms of the nodal values $\mathbf{y}_i$, $\mathbf{y}_{i+1}$.* □

Remark. The decoupling (upwinding) transformation above has been used for purposes of proof only. Note also that usually the matrix W_i is nonsingular. This matrix is guaranteed to be nonsingular if, under the conditions of Theorem 1, all eigenvalues satisfy

$$\text{either} \quad |\lambda h| \ll 1 \quad \text{or} \quad |\lambda h| \gg 1. \qquad \square$$

3.2. Higher order ODE. Our results for the case $m > 1$ are slightly less complete than those summarized in Theorem 1 for $m = 1$. We state them below.

THEOREM 2. *Consider the k-stage collocation equations* (2.11), (2.18) *for the* ODE (2.2).

(a) *If h_i is so small that*

$$|\xi_l| \ll 1 =: \xi_{m+1}, \qquad 1 \leqq l \leqq m,$$

then W_i is nonsingular. More generally, if there is an index $\tilde{l}$, $1 \leqq \tilde{l} \leqq m+1$, such that,

$$|\xi_l| \ll |\xi_{\tilde{l}}|, \qquad 1 \leqq l \leqq m+1,$$

then (for h sufficiently small, but $\xi_{\tilde{l}}$ not necessarily small) W_i is nonsingular.

(b) *If*

$$|\xi_l| \ll |\xi_m|, \qquad 1 \leqq l < m,$$

then (for h sufficiently small, but ξ_m not necessarily small) $\binom{W_i}{D_i}$ is nonsingular.

Proof. To show part (a), consider first the case where $c_l(x)$ are constant along $[x_i, x_{i+1})$. Then we can write W_i as in (2.17) and the result follows from the fact that $A^{(m+1-\tilde{l})}$ is nonsingular, $1 \leqq \tilde{l} \leqq m+1$. Then, for the variable coefficient case, we use a perturbation argument for

$$c_l(x) = c_l(x_i)(1 + O(h_i)).$$

To see part (b), note that W_i is an $O(h_i)$ perturbation of the corresponding matrix for the first order case

$$I - h_i c_m A^{(1)}$$

and that the last row of D_i is just $h_i \mathbf{b}^T$. Hence the result follows from Theorem 1. □

The results above cover, for instance, the ODE

$$\varepsilon D^2 u = Du + u$$

for any combination of (sufficiently small values of) h_i and ε, but it does not cover some cases for the ODE

$$\varepsilon D^2 u = u.$$

Nonetheless, practical experience indicates that it is rather rare that W_i be singular. We feel that it is hardly justifiable in a general purpose code to do more than decompose W_i itself and change h_i (say by halving the subinterval) in the unlikely event that W_i becomes singular.

4. Symmetric representation. One striking feature in the solution representation (2.7) is that it is not symmetric with respect to $\mathbf{y}_i$ and $\mathbf{y}_{i+1}$, even though the collocation scheme itself is. This results in the matrix Δ_i of (2.24) being the identity. The simplicity of Δ_i is usually not taken into account when solving the global linear system (i.e., (2.23) plus side conditions), so the idea of averaging the representation with respect to $\mathbf{y}_i$ and a corresponding one with respect to $\mathbf{y}_{i+1}$ naturally arises.

Thus we write, corresponding to (2.7)

(4.1)

$$u_\pi(x) = \sum_{j=1}^{m} \frac{(x - x_{i+1})^{j-1}}{(j-1)!} y_{i+1,j} + (-h_i)^m \sum_{j=1}^{k} \psi_{m,k+1-j}\left(\frac{x_{i+1} - x}{h_i}\right) z_{ij}, \qquad x_i \leqq x \leqq x_{i+1}.$$

The collocation equations then yield

(4.2)
$$-\hat{V}_i \mathbf{y}_{i+1} + \hat{W}_i \mathbf{z}_i = \mathbf{q}_i$$

where $\hat{V}_i$ and $\hat{W}_i$ are $k \times m$ and $k \times k$ matrices, respectively, given by

$$\hat{V}^i_{rj} = \sum_{l=1}^{j} \frac{c_l(x_{ir})(-h_i\rho_{k+1-r})^{j-l}}{(j-l)!}, \qquad 1 \leqq r \leqq k, \quad 1 \leqq l \leqq m, \tag{4.3}$$

$$\hat{W}^i_{rj} = \delta_{ri} - \sum_{l=1}^{m} c_l(x_{ir}) h_i^{m+1-l} D^{l-1} \psi_{m,k+1-j}(\rho_{k+1-r}), \qquad 1 \leqq r, j \leqq k. \tag{4.4}$$

Averaging (4.2) with (2.11) we obtain

$$\tilde{W}_i \mathbf{z}_i = \mathbf{q}_i + \tfrac{1}{2}(V_i \mathbf{y}_i + \hat{V}_i \mathbf{y}_{i+1}), \tag{4.5a}$$

$$\tilde{W}_i := \tfrac{1}{2}(W_i + \hat{W}_i). \tag{4.5b}$$

Assuming that $\tilde{W}_i$ is nonsingular, we can eliminate $\mathbf{z}_i$ and substitute in (2.18) (which can actually be symmetrized too, yielding no advantage). This gives the form (2.23) with

$$\Delta_i = I - \tfrac{1}{2} D_i \tilde{W}_i^{-1} \hat{V}_i, \quad \Gamma_i = C_i + \tfrac{1}{2} D_i \tilde{W}_i^{-1} V_i, \quad \mathbf{r}_i = D_i \tilde{W}_i^{-1} \mathbf{q}_i. \tag{4.6}$$

The symmetry introduced into $\tilde{W}_i$ by the above process can be seen by writing, similarly to (2.17),

$$\tilde{W}_i = I - \sum_{l=1}^{m} \xi_l \tilde{A}^{(m+1-l)} \tag{4.7}$$

with

$$\tilde{A}^{(m-l)}_{rj} := \tfrac{1}{2}[D^l \psi_{mj}(\rho_r) + (-1)^{m-l} D^l \psi_{m,k+1-j}(\rho_{k+1-r})], \qquad 1 \leqq r, j \leqq k, \quad 0 \leqq l \leqq m. \tag{4.8}$$

It is apparent from (4.8) that

$$\tilde{A}^{(m-l)}_{k+1-r,k+1-j} = (-1)^{m-l} \tilde{A}^{(m-l)}_{rj} \tag{4.9a}$$

or, in matrix notation (cf. (3.5), (3.4))

$$E\tilde{A}^{(m-l)}E = (-1)^{m-l} \tilde{A}^{(m-l)}. \tag{4.9b}$$

However, for $m > 1$ we cannot recommend the symmetrization idea: The simplicity gained in $\tilde{W}_i$ is not seen to compensate for the fact that we now have two matrices, V_i and $\hat{V}_i$, to generate, solve for and possibly store, in place of the one matrix V_i previously encountered.

The case is different for $m = 1$. Here V_i and $\hat{V}_i$ coincide (as do C_i and I). Moreover, we now prove the following.

THEOREM 3. *For collocation at k Gaussian points with* $m = 1$, *not only*

$$E\tilde{A}^{(1)}E = \tilde{A}^{(1)}, \tag{4.10a}$$

but also

$$\tilde{A}^{(1)}_{jj} = 0, \qquad 1 \leqq j \leqq k. \tag{4.10b}$$

Furthermore, all the eigenvalues of $\tilde{A}^{(1)}$ *are on the imaginary axis.*

Proof. Note that (4.10a) follows directly from (4.9b) and that, in general, for any $m \geqq 1$

$$D^{m-1}\psi_{mj}(\rho_r) + D^{m-1}\psi_{m,k+1-j}(\rho_{k+1-r}) = D^{m-1}\psi_{mj}(1) = b_j, \qquad 1 \leqq r, j \leqq k.$$

Hence we can write

$$\tilde{A}^{(1)} = A^{(1)} - \tfrac{1}{2} 1 \mathbf{b}^T. \tag{4.11}$$

Now, since collocation at Gaussian points yields a symmetric, algebraically stable scheme, we have

$$A^{(1)}_{jj} = \tfrac{1}{2} b_j, \qquad 1 \leqq j \leqq k$$

(see [2, eq. (17)]) and (4.10b) follows.

Further, let X be a $k \times k$ matrix defined as in [2, eq. (12)] (following Hairer and Wanner [14]). Thus

$$X = W^{(1)^{-1}} A^{(1)} W^{(1)}$$

for a certain matrix $W^{(1)}$ and, since the scheme is symmetric and algebraically stable,

$$X + X^T = \mathbf{e}_1 \mathbf{e}_1^T, \qquad \mathbf{e}_1 := (1, 0, \cdots, 0)^T = W^{(1)^T} \mathbf{b}.$$

Defining similarly

$$\tilde{X} := W^{(1)^{-1}} \tilde{A}^{(1)} W^{(1)}$$

we obtain from (4.11)

$$\tilde{X} = X - \tfrac{1}{2} W^{(1)^T} \mathbf{b} \mathbf{b}^T W^{(1)} = X - \tfrac{1}{2} \mathbf{e}_1 \mathbf{e}_1^T.$$

Hence

$$\tilde{X} + \tilde{X}^T = 0.$$

The fact that $\tilde{A}^{(1)}$ is similar to a skew-symmetric matrix establishes our claim regarding its eigenvalues. □

The above proof of Theorem 3 obviously generalizes for any symmetric, algebraically stable Runge-Kutta scheme, cf. [2].

Thus we see for the first order ODE (3.1) that the matrix $\tilde{W}_i = I - \xi \tilde{A}^{(1)}$ (unlike W_i) has a bounded inverse for any $|\xi| = O(1)$, provided that either $|\text{im}\,(\xi)| \leqq \text{const}\,|\text{re}\,(\xi)|$ or $|\xi| \ll 1$. If λ is imaginary then $|\xi|$ has to be taken small for approximation purposes anyway.

On the other hand, when k is odd the matrix $\tilde{A}^{(1)}$ is clearly singular. For instance, the midpoint scheme ($k = 1$) yields $\tilde{A}^{(1)} = 0$, and (4.6) gives ($\tilde{W}_i = 1$)

$$(1 - \xi/2) y_{i+1} = (1 + \xi/2) y_i.$$

This equation has to be rescaled when $|\xi| \gg 1$.

These results extend to the general system considered in Theorem 1. In particular, if $L(x)$ can be decomposed as described in (3.6), (3.10), (3.11) then the corresponding matrix $\tilde{W}_i$ has a bounded inverse. The resulting difference equations (2.23) may have to be rescaled (row-wise) in case that $|\xi| \gg 1$ and k is odd.

We conclude that the symmetric implementation considered here is worthwhile for first order ODEs, but not for higher order ones.

5. Mixed order systems. The class of problems treated by COLSYS has the following general form (see [6]): A system of d (generally) nonlinear ODEs of orders m_n, $1 \leqq n \leqq d$,

$$\begin{aligned} D^{m_n} u_n(x) &= F_n(x; u_1, Du_1, \cdots, D^{m_1 - 1} u_1, u_2, \cdots, D^{m_d - 1} u_d) \\ &\equiv F_n(x; \mathbf{v}(\mathbf{u})), \qquad a < x < b \end{aligned} \tag{5.1}$$

is subject to nonlinear side conditions

$$g_j(\zeta_i; \mathbf{v}(\mathbf{u})) = 0, \quad \zeta_1 \leqq \zeta_2 \leqq \cdots \leqq \zeta_{m^*}, \quad \zeta_j \in [a, b], \quad j = 1, \cdots, m^* \tag{5.2}$$

where $m^* := \sum_{n=1}^{d} m_n$. For the nonlinearities, some techniques based on Newton's method in quasilinearization form are applied, see Ascher, Christiansen and Russell [4], [5]. Thus, for solution representation it suffices to consider the linear ODEs

$$D^{m_n} u_n(x) = \sum_{\mu=1}^{m^*} c_{n\mu}(x) v_\mu(x) + q_n(x), \qquad a < x < b, \tag{5.3}$$

$$c_{n\mu} \equiv \frac{\partial F_n}{\partial v_\mu}. \tag{5.4}$$

With a mesh π of (2.4) so constructed that $\zeta_j \in \pi$, $1 \leqq j \leqq m^*$, we seek approximate solutions

$$\mathbf{u}_\pi = (u_{1\pi}, \cdots, u_{d\pi})^T, \quad u_{n\pi} \in \mathbf{P}_{k+m_n,\pi} \cap C^{m_n-1}[a, b], \quad 1 \leqq n \leqq d \tag{5.5}$$

by collocation at Gaussian points. In view of the considerations in the two previous sections (and in [7], [2], [1]), we have implemented an extension of the solution representation of §2 which is now described.

We write as in (2.7), (2.8)

$$u_{n\pi}(x) = \sum_{j=1}^{m_n} \frac{(x - x_i)^{j-1}}{(j-1)!} y_{nij} + h_i^{m_n} \sum_{j=1}^{k} \psi_{m_n,j}\left(\frac{x - x_i}{h_i}\right) z_{nij}, \qquad x_i \leqq x \leqq x_{i+1} \tag{5.6}$$

where for each n, $1 \leqq n \leqq d$,

$$y_{nij} = D^{j-1} u_{n\pi}(x_i), \qquad 1 \leqq j \leqq m_n, \tag{5.7a}$$

$$z_{nij} = D^{m_n} u_{n\pi}(x_{ij}), \qquad 1 \leqq j \leqq k, \tag{5.7b}$$

and let $\mathbf{y}_i \in \mathbf{R}^{m^*}$ and $\mathbf{z}_i \in \mathbf{R}^{kd}$ be defined by

$$\mathbf{y}_i := (y_{1i1}, \cdots, y_{1im_1}, y_{2i1}, \cdots, y_{\dim_d})^T \equiv \mathbf{v}_\pi(x_i), \tag{5.8a}$$

$$\mathbf{z}_i := (z_{1i1}, \cdots, z_{1ik}, z_{2i1}, \cdots, z_{2ik}, \cdots, z_{di1}, \cdots, z_{dik})^T. \tag{5.8b}$$

The side conditions (5.2) can now be written in a straightforward manner as

$$g_j(\zeta_j; \mathbf{y}_{i_j}) = 0, \qquad 1 \leqq j \leqq m^* \tag{5.9}$$

where for each j, i_j is the index such that $\zeta_j = x_{i_j}$. (But, like the ODEs, the side conditions are linearized as well.)

Next we write down the collocation and the continuity equations on the ith subinterval of the mesh π. We use the same notation as in (2.11), (2.18), but now

$$C_i = \operatorname{diag}\{C_i^{(n)}\} \in \mathbf{R}^{m^* \times m^*} \tag{5.10}$$

where $C_i^{(n)}$, $1 \leqq n \leqq d$, is an $m_n \times m_n$ upper triangular matrix with entries given in (2.19). Similarly, $D_i \in \mathbf{R}^{m^* \times kd}$ is given by

$$D_i = \begin{pmatrix} D_i^{(1)} & & & \\ & D_i^{(2)} & & \\ & & \ddots & \\ & & & D_i^{(d)} \end{pmatrix} \tag{5.11}$$

where $D_i^{(n)} \in \mathbf{R}^{m_n \times k}$ is given by (2.20) with m replaced by m_n. Also $W_i \in \mathbf{R}^{kd \times kd}$ is given by

$$W_i = (W^{(n,\nu)})_{n,\nu=1}^{d} \tag{5.12a}$$

where each $W^{(n,\nu)}$ *is a* $k \times k$ matrix given by

$$W_{rj}^{(n,\nu)} = \delta_{n\nu}\delta_{rj} - \sum_{l=1}^{m_\nu} c_{n\mu}(x_{ir}) h_i^{m_\nu+1-l} D^{l-1}\psi_{m_\nu j}(\rho_r), \qquad 1 \leqq r, j \leqq k \tag{5.12b}$$

with

$$\mu \equiv \sum_{\eta=1}^{\nu-1} m_\eta + l. \tag{5.12c}$$

The advantage of using functions ψ_{mj} which satisfy (2.10) is now apparent, because the only from in which these functions appear in W_i and D_i is

$$h_i^\xi \psi_{\xi j}(t) := h_i^\xi D^{m-\xi}\psi_{mj}(t), \qquad 0 \leqq \xi \leqq m \quad (\xi := m_\nu + 1 - l) \tag{5.13a}$$

where m is a predetermined constant (e.g., $m = 4$) satisfying

$$m \geqq \max\{m_n;\ 1 \leqq n \leqq d\}. \tag{5.13b}$$

Thus, the mesh independent constants ρ_r, $\psi_{\xi j}(\rho_r)$ and $\psi_{\xi j}(1)$ are stored in the code for $1 \leqq k \leqq 7$, $m := \min(k, 4)$, $1 \leqq \xi \leqq m$, $1 \leqq j \leqq k$, $1 \leqq r \leqq k$. Using these, the assembly of W_i and D_i is straightforward.

Finally, for V_i of (2.11) we have a $kd \times m^*$ matrix given by

$$V_i = (V^{(n,\nu)})_{n,\nu=1}^{d} \tag{5.14a}$$

where $V^{(n,\nu)} \in \mathbf{R}^{k,m_\nu}$ is given by

$$V_{rj}^{(n,\nu)} = \sum_{l=1}^{j} \frac{c_{n\mu}(x_{ir})(h_i\rho_r)^{j-l}}{(j-l)!}, \quad 1 \leqq r \leqq k, \quad 1 \leqq j \leqq m_\nu \tag{5.14b}$$

with μ related to l by (5.12c), and

$$\mathbf{q}_i := (q_1(x_{i1}), \cdots, q_1(x_{ik}), \cdots, q_d(x_{i1}), \cdots, q_d(x_{ik}))^T. \tag{5.15}$$

Remark. A natural extension of this implementation can be made to a special case of *differential-algebraic* ODEs, where the ODEs in (5.1) depend also on $\hat{m}$ unknown functions $\mathbf{w}(x) \equiv (w_1(x), \cdots, w_{\hat{m}}(x))^T$ and are supplemented by $\hat{m}$ algebraic equations, tying up $\mathbf{v}(x)$ and $\mathbf{w}(x)$, $a \leqq x \leqq b$. These algebraic equations can be considered as ODEs of order 0 and, correspondingly, the collocation approximations is in the space

$$\mathbf{w}_\pi \in \mathbf{P}_{k,\pi}$$

(which admits discontinuities at mesh points). The obtained representation (5.6) for $w_{n\pi}(x)$ is just a Lagrange interpolant form. □

In our implementation we eliminate for each i, $1 \leqq i \leqq N$, the local unknowns $\mathbf{z}_i$ from (2.18), and form (2.23), (2.24). If W_i is deemed too ill-conditioned, the interval $[x_i, x_{i+1}]$ is halved. The obtained equations for $\mathbf{y}_i$, $1 \leqq i \leqq N+1$ are merged with those resulting from the side conditions to form a large, sparse system of order $(N+1)m^*$. The resulting matrix is almost block diagonal (see [6, §4] and references therein), with the ith block having $m^* + l_i$ rows (and l_i is the number of side conditions already encountered), $2m^*$ columns and an offset of m^* columns with respect to its neighboring blocks.

6. Implementation and numerical results. When considering a general purpose collocation code, the following aspects arise:

(1) Basis representation;
(2) Linear system solution;
(3) Error estimation;
(4) Mesh selection;
(5) Nonlinear problem solution.

Our aim when replacing the basis representation in COLSYS has been to make as little change as possible in the modules dealing with the other aspects, thus allowing for a meaningful assessment of the performed replacement. Nonetheless, some considerations are needed which may not seem at a first glance to be directly related to the basis representation. Thus, while error estimation and mesh selection are essentially not affected by changing the basis, the linear and nonlinear equation solving procedures are, as described below.

Using the monomial basis representation, the linear algebra part of the solution process appears in two places: in the local parameter elimination and in the solution of the resulting, global linear system for $\{\mathbf{y}_i\}$. For the latter, the code SOLVEBLOK [11] is used in COLSYS, see [5], [6]. When using B-splines, the block sizes of the global linear system are larger and no additional parameter elimination is done. Thus, a more extensive use of SOLVEBLOK is made in COLSYS than in its replacement. Consequently, if SOLVEBLOK can be improved then the comparison between the old and the new bases is affected.

And SOLVEBLOK can indeed be improved. As written by de Boor and Weiss [11], it is row-oriented in its handling of matrices. Since it is written in FORTRAN, multidimensional arrays are stored columnwise, so a column orientation is preferable, i.e., the innermost loops should attempt to access consecutive memory locations as much as possible, see Moler [18]. For the reasons mentioned above, we have not replaced SOLVEBLOK by another linear system solver altogether, but we have rewritten its loops and reduced the number of array shifts. The resulting savings in computing time are machine and compiler dependent, and can be significant when the block sizes which SOLVEBLOK has to handle are large, as in Example 5 below. Hay and Gladwell [15] have considered these questions, especially for vector machines.

The dense linear system solver used for the local parameter elimination has also been efficiently implemented, following the LINPACK routines [12]. An additional advantage of this is that it allows an easy adaptation of the code to vector machines or the use of a sparse system solver when appropriate.

The nonlinear system solution algorithm also has to be changed when changing the basis representation, because the unknowns are now scaled in a different way. This has an effect on the control of the damped Newton iteration used in COLSYS, and is felt only in highly (nonlinearly) sensitive situations. Our first instinct has been to use only the variables $\{\mathbf{y}_i\}$ for controlling the iteration. This, however, proved to be insufficient in some applications, and the variables $\{\mathbf{z}_i\}$ are now used as well. Consulting (5.6), (5.8), we use a relative scaling of $h_i^j/j!(|y_{n,i+1,1}|+1)$ for a variable representing the jth derivative of the nth solution component on the ith subinterval. Note that this scaling is independent of a stretching transformation to the independent variable. The rest of the damped Newton strategy in COLSYS has not been changed.

In the course of implementation of the monomial basis (we refer to the resulting program as COLNEW in this paper) and the inevitable comparison to COLSYS which followed, we have become once again painfully aware of two known facts in mathematical software: That a simple comparison of two proposed schemes can be completely different than a comparison of their implementation in a general purpose setting, and that a comparison of two general purpose codes, even fairly similar ones, can be complicated, depending on the machine, the precision and the compiler used, as well as on the examples chosen.

Our first goal during implementation has been not to compromise on any feature of COLSYS, i.e., that COLNEW be comparable or better than COLSYS for any BVP. We believe we have achieved this; however, the amount of improvement in COLNEW

over COLSYS in our experiments is varied. We have run our experiments on an IBM 3081 using, unless otherwise noted, double precision (56 bit mantissa) and occasionally also single precision (24 bit mantissa). Note that in [3] an example is constructed where COLSYS fails while COLNEW solves the problem very easily. But here we test the codes on some problems arising in applications.

Example 1. From [7] we know that COLNEW should be faster than COLSYS for a linear ODE of a high order. Indeed, consider Example 1 in [7] and [6],

$$(x^3u'')'' = x^3u'''' + 6x^2u''' + 6xu'' = 1, \qquad 1 \leqq x \leqq 2, \quad u = u'' = 0 \quad \text{at } x = 1, 2.$$

For this fourth order ODE, with tolerances of 10^{-6} and 10^{-10}, the run time of COLSYS is almost halved by COLNEW. The actual maximum error resulting from the runs with the stiffer tolerance is also more accurate when using COLNEW by about two digits (cf. [7]). □

Note that for linear problems, the monomial representation also offers an implementation which is cheaper in terms of storage, because the W_i matrices do not have to be saved. For nonlinear problems, however, we do save these matrices in order to be able to restore the full solution, and the total storage requirements are about 10% higher than with the B-spline implementation.

When using the monomial basis to solve a BVP, solution values at mesh points are readily available. For solution values elsewhere, the values of $\mathbf{z}_i$ are used as well, in a straightforward manner. However, the latter solution evaluations are not cheaper than when using B-splines. They become a more important component of the total cost in nonlinear problems and when more error estimation and mesh selection is done in COLSYS. Also, since the same number of Jacobian evaluations is required regardless of the basis (given that the nonlinear iteration control is as effective), the performance difference between COLSYS and COLNEW tends to be less significant the more expensive these Jacobian evaluations are.

Example 2. The BVP

$$\varepsilon^4 \Big/ \mu\left[\phi'' + \frac{1}{x}\phi' - \frac{1}{x^2}\phi\right] + \psi\left(1 - \frac{1}{x}\phi\right) - \phi = -\gamma x\left(1 - \frac{1}{2}x^2\right), \qquad 0 < x < 1,$$

$$\mu\left[\psi'' + \frac{1}{x}\psi' - \frac{1}{x^2}\psi\right] - \phi\left(1 - \frac{1}{2x}\phi\right) = 0,$$

$$\phi = x\psi' - 0.3\psi + 0.7x = 0 \qquad \text{at } x = 0, 1,$$

$$\varepsilon = \mu = 10^{-3}, \qquad \gamma = 1.1,$$

is used as Example 2 in [6], featuring boundary and interior layers. Here the two codes perform similarly, COLSYS being slightly faster. But COLSYS is more sensitive than COLNEW to a good compiler optimization. When using the FTN compiler with a lower level of optimization, COLNEW is about 20% faster. This is important when no high level optimizing compiler is available in a particular computing environment.

Another singularly perturbed BVP which involves a fourth order and a second order ODE is considered in [4, Example 3]. The BVP is

$$\varepsilon G'' + HG' - H'G = 0, \qquad -1 \leqq x \leqq 1,$$

$$\varepsilon H'''' + HH''' + GG' = 0,$$

and

$$G(-1) = -1, \quad G(1) = 1, \quad H(-1) = H'(-1) = H(1) = H'(1) = 0.$$

Here COLNEW is faster than COLSYS by about 25% using the optimized object codes. □

Example 3. The third example in [6] consists of three continuation steps for a coupled pair of second and third order ODEs. This is also a sensitive nonlinear BVP given by

$$G''+\frac{3-n}{2}HG'+(n-1)H'G-s(G-1)=0,$$

$$H'''+\frac{3-n}{2}HH''+n(H')^2+G^2-1-sH'=0,$$

$$G(0)=H(0)=H'(0)=0, \quad G(\infty)=1, \quad H'(\infty)=0.$$

Here COLNEW performs marginally better when using the optimized codes and by about 20% better with the low-level compiler optimization.

This BVP contains a solution component G which tends to 1 as the independent variable $x\to\infty$. If we now make the change of variable $G:=G-1$ then the runs with COLNEW are not seriously affected, but the COLSYS run times get significantly worse, because the scaling in the nonlinear iteration control is affected by the tail noise in G. This is unusual, though, and may be considered as a bug in COLSYS which is eliminated in COLNEW. □

The other, probably more important, theoretical improvement which the monomial basis has to offer is the improved condition number of the linear systems arising, in case of higher order ODEs. See [7] for an extended discussion on this point. This effect becomes apparent in our setting here only when the error tolerances are close to the machine precision, which normally is more practically relevant when a short word length is used.

Example 4. Consider Ascher and Russell [8, Example 7.2], rewritten here,

$$u''+\frac{4}{x}u'+(xu-1)u=0, \qquad 0<x<L,$$

$$u'(0)=0, \qquad u(L)+u'(L)=0.$$

Using double precision, the results in [8, Table 1] for $L=20$ are easily recovered with both codes, COLNEW being about 10% faster. But when switching to single precision, COLNEW is still able to produce the desired results, while the nonlinear iteration in COLSYS on a mesh with 32 elements fails, because the nonlinear iteration control cannot handle a case where roundoff error dominates.

A similar effect occurs when attempting to solve Example 3 in single precision. With COLNEW, the first two continuation steps are successfully completed (but the last step fails), while with COLSYS only the first continuation step is completed and the second one fails. The onset of roundoff error domination occurs sooner in COLSYS than in COLNEW! □

The advantage of COLNEW demonstrated in the last example above may be important in certain computing environments. If computations need to be performed with short word length then COLNEW should be clearly preferred, being more robust.

The comments made earlier regarding the linear system solver are not of major importance in the above examples. But the efficient implementation of the linear system solver becomes significant in the following.

Example 5. This problem is described in Stephenson et al. [20], Mejia and Stephenson [17]. Our numerical solution procedure for it using COLSYS has not appeared anywhere before, so we briefly describe it here. 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 J_{ik}—the outward transmural flux per unit length of the kth solute from the ith tube, there are 18 ODEs 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 [20].

Now, it can be verified that this BVP can be simplified to have only 14 ODEs with separated BC, as follows:

$$C'_{12} = 200(C_{12})^2[C_{41} + C_{42} - 21C_{12}],$$

$$C_1 = 20C_{12}, \quad F_{1v} = \frac{0.05}{C_{12}}, \quad F_{2v} = -\frac{0.05}{C_{22}},$$

$$C'_{21} = 20C_{22}J_{21}, \qquad C'_{22} = 0,$$

$$C'_{31} = \frac{10}{K_1}(C_{31})^2[C_{41} + C_{42} - C_{31} - C_{32}],$$

$$C'_{32} = \frac{C_{31}}{K_1}[J_{3v}C_{32} - J_{32}],$$

$$K'_1 = 0, \quad K'_2 = 0, \quad F'_{4v} = -J_{4v},$$

$$K_1 = \frac{C_{31}(0)}{2C_{32}(0)}, \quad F_{3v} = \frac{K_1}{C_{31}}, \quad F_{5v} = 5, \quad F_{6v} = \frac{0.05}{C_{62}},$$

$$C'_{41} = \frac{1}{F_{4v}}[J_{4v}C_{41} - J_{41}],$$

$$C'_{42} = \frac{1}{F_{4v}}[J_{4v}C_{42} - J_{42}],$$

$$C'_{51} = -200(C_{51} - C_{41}), \qquad C'_{52} = -200(C_{52} - C_{42}),$$

$$C'_{61} = 20C_{62}[J_{6v}C_{61} - J_{61}],$$

$$C'_{62} = 20(C_{62})^2 J_{6v},$$

$$C_{12}(0) = 0.05, \quad C_{51}(0) = 1, \quad C_{52}(0) = 0.05, \quad F_{4v}(1) = -5,$$

$$C_{31}(0) - 20K_1(0)C_{32}(0) = 0, \quad C_{22}(0) = C_{62}(0), \quad C_{61}(0) = C_{21}(0),$$

$$C_{12}(1) = C_{22}(1), \quad C_{21}(1) = 20C_{12}(1), \quad C_{41}(1) = C_{51}(1), \quad C_{42}(1) = C_{52}(1),$$

$$C_{61}(1) - 20K_1(1)C_{62}(1) = 0, \quad C_{31}(0) = K_2(0), \quad C_{61}(1) = K_2(1).$$

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:

$$J_{iv} = h_{iv} \sum_{k=1}^{2} (C_{4k} - C_{ik}), \qquad i = 1, 2, 3, 5,$$

$$J_{iv} = (1. - C_{i1}) + (0.05 - C_{i2}), \qquad i = 6,$$

where $h_{1v} = h_{3v} = 10$ and $h_{iv} = 0$ for $i = 2, 5$.

$$J_{4v} = -\sum_{i \neq 4,6} J_{iv}.$$

Transmural solute fluxes are:

$$\begin{aligned} J_{i1} &= 0, && i = 1, 3, \\ &= 0.75C_{i1}/(1. + C_{i1}), && i = 6, \\ &= 1000(C_{i1} - C_{41}), && i = 4. \end{aligned}$$

$$J_{21}(x)=\begin{cases}1.8, & 0\leqq x\leqq 0.4,\\ 1.8+[-18+100(C_{21}(x)-C_{41}(x))]\cdot(x-0.4), & 0.4<x<0.5,\\ 10[C_{21}(x)-C_{41}(x)], & 0.5\leqq x\leqq 1.\end{cases}$$

$$J_{i2}=0, \qquad i=1,2,6,$$

$$=1000(C_{i2}-C_{42}), \qquad i=5.$$

$$J_{32}(x)=\begin{cases}0, & 0\leqq x\leqq 0.4,\\ 0.1[C_{32}(x)-C_{42}(x)]\cdot(x-0.4), & 0.4<x<0.5,\\ 0.01[C_{32}(x)-C_{42}(x)], & 0.5\leqq x\leqq 1.\end{cases}$$

$$J_{4k}(x)=-\sum_{i\neq 4,6} J_{ik}(x), \qquad 0\leqq x\leqq 1, \quad 1\leqq k\leqq 2.$$

This completes the specification of the problem.

For the solution of this BVP, beginning with the simple initial guess that all solution components are identically equal to 1, we apply a chain of simple continuation steps. Thus we use COLSYS or COLNEW to solve the BVP defined above with $\lambda = h_{1v} = h_{3v}$ taking the values 0, 0.8, 1.6, 2.4, 4, 6, 8, 10, using the solution obtained with each λ as the initial guess for the BVP with the next λ in the sequence.

Since this is a first order system, we expect the difference between the codes, if any, to be mainly due to the linear system solver. The resulting run using COLSYS to complete the entire sequence of continuation steps, takes about 93 cpu seconds, while COLNEW needs only about 58 seconds. When linking COLSYS to the improved SOLVEBLOK as described above, the COLSYS run time is reduced to 70 seconds.

A similar effect can be expected when running other examples with many ODEs, because these give rise to large blocks in the Jacobian of the nonlinear system resulting from collocation discretization. In Examples 1–4 above these blocks are not very large and other considerations dominate the codes performance. □

In conclusion, the implementation of the monomial basis, as described here, provides a cleaner solution representation which has generally resulted in a somewhat more robust code than COLSYS.

REFERENCES

[1] U. ASCHER, *Collocation for two-point boundary value problems revisited*, SIAM J. Numer. Anal., to appear.

[2] U. ASCHER AND G. BADER, *Stability of collocation at Gaussian points*, SIAM J. Numer. Anal., 23 (1986), pp. 412–422.

[3] ———, *A Note on conditioning, stability and collocation matrices*, Technical Report CMA, Australian National University, Canberra, 1986, submitted.

[4] 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.

[5] ———, COLSYS—*A collocation code for boundary value problems*, in Springer Lecture Notes in Computer Science 76, B. Childs et al., eds., New York, 1979.

[6] ———, *Collocation software for boundary value* ODEs, ACM Trans. Math. Software, 7 (1981), pp. 209–222.

[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] C. DE BOOR, *A Practical Guide to Splines*, Applied Math. Sciences, Vol. 27, Springer, New York, 1978.

[10] C. DE BOOR AND B. SWARTZ, *Collocation at Gaussian points*, SIAM J. Numer. Anal., 10 (1973), pp. 582–606.

[11] 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.

[12]. J. J. DONGARRA, J. R. BUNCH, C. B. MOLER AND G. W. STEWART, LINPACK *User's Guide*, Society for Industrial and Applied Mathematics Publications, Philadelphia, 1979.

[13] E. HAIRER, *A note on D-stability*, BIT, 24 (1984), pp. 383-386.

[14] E. HAIRER AND G. WANNER, *Characterization of non-linearly stable implicit Runge-Kutta methods*, in Springer Lecture Notes in Math., 968, Springer, New York, 1982, pp. 207-219.

[15] R. I. HAY AND I. GLADWELL, *Solving almost block diagonal linear equations on the CDC Cyber* 205, Numerical Analysis Report 98, Dept. Mathematics, Univ. of Manchester, 1985.

[16] H. KREISS, N. NICHOLS AND D. BROWN, *Numerical methods for stiff two-point boundary value problems*, SIAM J. Numer. Anal., 23 (1986), pp. 325-368.

[17] R. MEJIA AND J. L. STEPHENSON, *Numerical solution of multinephron kidney equations*, J. Comp. Phys., 32 (1979), pp. 235-246.

[18] C. MOLER, *Algorithm* 423, *Linear equation solver*, Comm. ACM, 15 (1972), p. 274.

[19] M. R. OSBORNE, *Collocation, difference equations, and stitched function representations*, in Topics in Numerical Analysis II, J. Miller, ed., Academic Press, New York, 1975, pp. 121-132.

[20] 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. 8, No. 4, July 1987
© 1987 Society for Industrial and Applied Mathematics
003

A SIMPLE COMPUTATIONAL SCHEME FOR DETERMINING THE SOUND SPEED OF AN ACOUSTIC MEDIUM FROM ITS SURFACE IMPULSE RESPONSE*

PAUL SACKS† AND FADIL SANTOSA‡

Abstract. We describe an algorithm for recovering the sound speed of a one-dimensional half space from its impulse response at the boundary. Similar techniques for some related optimization problems are also discussed, and results of numerical experiments are given.

Key words. inverse problem, acoustic medium, wave equation

AMS(MOS) subject classifications. 35L, 86

Introduction. In this article, we will discuss a constructive approach to the solution of some coefficient determination problems which arise in geophysics. For definiteness, let us consider the following one-dimensional velocity inversion problem.

A velocity profile $c(z)$ is defined for $z \geqq 0$ with $0 < c_- \leqq c(z) \leqq c_+ < \infty$. The wavefield function $u = u(z, t)$ satisfies the hyperbolic equation

$$u_{tt} = c^2(z) u_{zz} \tag{0.1}$$

for $z > 0$. We assume that

$$u \equiv 0 \text{ for } t < 0 \tag{0.2}$$

and the surface $\{z = 0\}$ is traction free except at $t = 0$ when an impulse is applied

$$u_z(0, t) = -\delta(t) \tag{0.3}$$

where δ is Dirac's delta function.

We wish to recover the velocity $c(z)$ from measurement of the surface response data

$$g(t) = u(0, t), \qquad t > 0. \tag{0.4}$$

See for example Bamberger, Chavent and Lailly [1], Yagle [15], Symes [12] or Gray [7] for some previous approaches to this problem, and Bleistein et al. [2] for work on multidimensional analogues.

Let us denote by F the nonlinear map which associates to the velocity $c(z)$ the corresponding surface trace $u(0, t)$. By standard theory of hyperbolic equations (e.g., Courant and Hilbert [6]) F is defined at least for sufficiently smooth c's, which are bounded away from zero.

The inverse problem becomes that of solving the equation

$$F(c) = g \tag{0.5}$$

where g is a given function of t.

* Received by the editors September 30, 1985; accepted for publication May 20, 1986. This research was supported by the Office of Naval Research under an SRO-III grant contract N-000-14-83-K0051.

† Center for Applied Mathematics, Cornell University, Ithaca, New York 14853. Permanent address, Department of Mathematics, Iowa State University, Ames, Iowa 50010. The work of this author was supported in part by the Air Force Office of Scientific Research, United States Air Force, under grant AFOSR 84-0252.

‡ Department of Theoretical and Applied Mathematics, Cornell University, Ithaca, New York 14853. Present address, Department of Mathematical Science, University of Delaware, Newark, Delaware 19717.

In practice the data g will not be known exactly, while on the other hand some other information about the solution c may be available. One might, therefore, wish to replace the equation (0.5) by a best fit problem

$$\min \|F(c)-g\| \tag{0.6}$$

for some choice of norm $\|\cdot\|$ and where the minimization is done over a set of c's which incorporates some desired constraints. Furthermore, in cases when the inverse problem is not well posed, one might wish to include penalty terms in the objective functional, in order to stabilize the problem.

For constructive solution of (0.5) or (0.6), a variety of well-known methods suggest themselves, such as Newton's method, steepest descent or nonlinear least square procedures. These are all iterative methods which require for their implementation the ability to compute (besides F itself) the derivative $DF(c)$, its inverse, adjoint or some combination. The difficulty, of course, is that these linear operators may be difficult and expensive to compute. For example, it may be the case that computing $(DF(c))^{-1}$ is as difficult as finding F^{-1}, i.e., solving equation (0.5) directly.

What we propose here is to replace $DF(c)$ by a certain approximation $E(c)$ which permits simple (in some cases trivial) calculation of the inverse $E(c)^{-1}$, adjoint $E(c)^*$, etc. The special features of the problem at hand are such that if we make this replacement in the iterative methods just mentioned, we still obtain good approximation to the desired solution.

Let us illustrate with the case of Newton's method. In its usual form we should iterate according to the rule

$$c_{k+1} = c_k - (DF(c_k))^{-1}(F(c_k)-g). \tag{0.7}$$

The approximation to $DF(c)$ which will be justified in the next section is

$$(DF(c)\zeta)(t) \equiv (E(c)\zeta)(t) = \frac{\zeta}{c}\left(\psi\left(\frac{t}{2}\right)\right) \tag{0.8}$$

where $\psi = \phi^{-1}$ and

$$\phi(z) = \int_0^z \frac{1}{c(s)}\,ds = \phi(z;\, c). \tag{0.9}$$

Thus,

$$(E(c)^{-1}\chi)(z) = c(z)\chi(2\phi(z))$$

and the modified version of (0.7) is

$$c_{k+1}(z) = c_k(z)(1-(F(c_k)-g)(2\phi_k(z))) \tag{0.10}$$

with $\phi_k(z) = \phi(z;\, c_k)$.

If we assume that $F(c)$ can be easily calculated (eventually at the discrete level, of course), then use of the iteration scheme (0.10) is reasonably simple and inexpensive.

Without loss of generality we may assume that $c(0) = 1$; it then turns out that if the data g is consistent (i.e., in the range of F) there must be satisfied $g(0) = 1$. Assuming this and choosing an initial guess c_0 also consistent with the known surface value of the velocity, we find that the sequence c_k defined by (0.10) (with some minor technical modifications to be discussed in § 1) converges to the solution c of (0.5). The rate of convergence is fairly rapid if c is slowly varying, and less so if c has steep gradients. The depth to which an accurate reconstruction is obtained increases directly with the number of iterations.

In § 1 we will derive the approximation (0.8) and give heuristic arguments concerning the expected behavior of the algorithm (0.10). In § 3 we describe the results of numerical experiments involving this technique.

The method (0.10) for solving equation (0.5) in this case has no particular advantage over other known methods for solving this inverse problem, except perhaps for some conceptual simplicity. Indeed, one can make some changes in variables to transform to the impedance profile inversion problem (Bamberger, Chavent and Lailly [1], Symes [14]) for which very efficient numerical methods are known (Bube and Burridge [3], Santosa and Schwetlick [10], Yagle [15]). However, this would not be likely to lead to more insight into the resolution of more complicated coefficient determination problems, since exact impedance inversion has some very special properties.

Furthermore, if one wishes to solve the best fit problem (0.6) incorporating constraints and/or penalizations there would seem to be some advantages to the approach we develop here. See § 2 for some model problems of this type.

These ideas will also generalize to be useful in both the analysis and numerical solution of some more complicated inverse problems involving coefficients depending on a single variable, such as determining the material coefficients in a layered elastic medium (Sacks and Symes [9]), or problems in which a source function is to be determined along with the coefficients [4].

Finally, at a certain level, the ideas here do not rely in an essential way on the one-dimensional nature of the problem. That is to say, it seems possible that some generalization to higher-dimensional inverse problems may be possible. At present, however, this is the purest speculation.

1. In practice we are only interested in recovering the velocity $c(z)$ on a finite interval $0 \leqq z \leqq Z$. Since $c(z)$ is the speed of propagation of disturbances in the medium, it follows that the value of $c(z)$ for $z > Z$ does not affect $u(0, t)$ for $t < T$, where T is the two way travel time to depth Z,

$$T = T(Z; c) = 2\int_0^Z \frac{1}{c(s)}\, ds. \tag{1.1}$$

We may thus regard the trace map F defined earlier as mapping $H_1(0, Z)$ to $H_2(0, T(Z; c))$ where H_1 and H_2 are suitably chosen function (usually Hilbert) spaces. It is known that if the members of H_1 and H_2 are sufficiently differentiable, then F is one-to-one, i.e., given a trace $g(t) = u(0, t) \leqq t \leqq T$ there is uniquely determined a depth Z and a coefficient $c(z)$ defined on $[0, Z]$ so that $F(c)(t) = g(t)$ on $[0, T]$. See [1], [12] for details. Note that we have *assumed* g to belong to the range of F; no assertion about the existence question is made here.

Since the space on which $F(c)$ is defined itself depends on c, some care must be taken in doing calculus with F; indeed, to find the derivative $DF(c)$ according to the usual rules would require F to be a map between fixed spaces. However, any reasonable idea of the derivative leads to the same conclusion as to what the correct definition should be. This difficulty will reappear in a more significant way in § 2.

Let c be the point of $H_1(0, Z)$ at which the derivative is to be found, and let $u(z, t)$ be the corresponding solution of (0.1), (0.2), (0.3). Then $DF(c)$ is a linear map from $H_1(0, Z)$ to $H_2(0, T(Z; c))$, and if $\delta c \in H_1(0, Z)$ then

$$(DF(c)\delta c)(t) = v(0, t) \tag{1.2}$$

where $v(z, t)$ is the solution of

$$v_{tt} - c^2(z)v_{zz} = 2c(z)\delta c(z)u_{zz}, \qquad z > 0, \tag{1.3}$$

$$v_z(0, t) = 0 \quad \text{for all } t, \tag{1.4}$$

$$v = v_t = 0, \qquad t < 0. \tag{1.5}$$

Note that once the reference coefficient c is fixed, so is the range space $H_2(0, T(Z; c))$.

Recall that our goal is to find a simple approximation to $DF(c)$. Here we just indicate the formal argument. See Santosa and Symes [11] for proper justification and a more general setting.

The solution $u(z, t)$ has the progressing wave expansion (see Courant and Hilbert [6])

$$u(z, t) = \sqrt{c(z)}\, H(t - \phi(z)) + J(z, t), \tag{1.6}$$

where $H(s)$ is the Heaviside function, ϕ is the travel time function (0.9), and the function J has bounded derivatives, $J \equiv 0$ for $t < \phi(z)$ and $J(z, t) = O(|t - \phi(z)|)$ near the wavefront curve $\{t = \phi(z)\}$. We are still using the normalization $c(0) = 1$.

Next define the operator $\Box_c$ by

$$(\Box_c \eta)(z, t) = \frac{1}{c^2(z)} \eta_{tt}(z, t) - \eta_{zz}(z, t). \tag{1.7}$$

Some straightforward integration by parts yields the Greens identity

$$\int_0^T \int_0^\infty (w \Box_c v - v \Box_c w)\, dz\, dt = \int_0^T \int_0^\infty \left(\frac{1}{c^2}(v w_t - w v_t)_t - (v w_z - v_z w)_z \right) dz\, dt. \tag{1.8}$$

We choose v to be the solution of (1.3), (1.4), (1.5) and $w(z, t) = u(z, \tau - t)$ for some fixed $\tau \in (0, T)$, to obtain

$$2 \int_0^T \int_0^\infty u(z, \tau - t) \frac{\delta c(z)}{c(z)} u_{zz}(z, t) = \int_0^T v(0, t) \delta(\tau - t)\, dt = v(0, \tau). \tag{1.9}$$

We now substitute from (1.6) into (1.9) to obtain a representation for $DF(c)\delta c$, namely

$$(DF(c)\delta c)(\tau) = (E(c)\delta c)(\tau) + (K(c)\delta c)(\tau) \tag{1.10}$$

with, by definition

$$(E(c)\delta c)(\tau) = 2 \int_0^\tau \int_0^\infty \frac{\delta c(z)}{c^2(z)} H(\tau - t - \phi(z)) \delta'(t - \phi(z))\, dz\, dt \tag{1.11}$$

and K is everything else,

$$\begin{aligned} (K(c)\delta c)(\tau) = 2 \int_0^\tau \int_0^\infty & \frac{\delta c(z)}{c(z)} \\ & \times [J(z, \tau - t)(c(z)^{-3/2} \delta'(t - \phi(z)) + (c^{1/2})''(z) H(t - \phi(z)) + J_{zz}(z, t)) \\ & + c(z)^{1/2} H(t - \phi(z))((c^{1/2})''(z) H(\tau - t - \phi(z)) + J_{zz}(z, \tau - t))]\, dz\, dt. \end{aligned} \tag{1.12}$$

We have defined E by taking the most singular terms in the integral for $v(0, t)$, which amounts to calculating the high frequency asymptotics of $DF(c)$. This idea has been used before in other settings [5], [7].

We may evaluate the integral for E explicitly. Make the change of variables

$$p = t - \phi(z), \qquad q = \tau - t - \phi(z).$$

We have $|\partial(p, q)/\partial(t, z)| = 2/c$. Let ψ denote the inverse function to ϕ. We compute

$$\begin{aligned}(E(c)\delta c)(\tau) &= \int_0^\tau \int_{q-s}^{q+s} \frac{\delta c}{c}\left(\psi\left(\frac{\tau-(p+q)}{2}\right)\right)\delta'(p)\,dp\,dq \\ &= -\int_0^\tau \frac{\partial}{\partial p}\left(\frac{\delta c}{c}\left(\psi\left(\frac{\tau-(p+q)}{2}\right)\right)\right)\Bigg|_{p=0} dq \\ &= -\int_0^\tau \frac{\partial}{\partial q}\left(\frac{\delta c}{c}\left(\psi\left(\frac{\tau-q}{2}\right)\right)\right) dq \\ &= -\frac{\delta c}{c}(0) + \frac{\delta c}{c}\left(\psi\left(\frac{\tau}{2}\right)\right).\end{aligned}$$

Since we have fixed the surface value, we should have $\delta c(0) = 0$. We obtain (0.8).

The expression for K is of course much more complicated but it can be shown (see Santosa and Symes [11]) that it may be written

$$(K(c)\,dc)(\tau) = \int_0^{\psi(\tau/2)} G(z, t)\delta c(z)\,dz \tag{1.13}$$

where the kernel G is bounded as long as the reference coefficient $c(z)$ is sufficiently regular. We remark that if $c(z) \equiv 1$ then $K = 0$. The main point of interest here is that the operator norm of $K(c)$, considered as a map from $L^2(0, Z)$ to $L^2(0, T(Z; c))$ tends to zero with the depth interval Z. These norms of course vary with the base point c, but may be estimated, for example, in terms of $\|(\log c)'\|_{L^2(0,Z)}$.

As discussed in the introduction, we now wish to use $E(c)$ as a replacement for $DF(c)$ in Newton's method, that is, we try to solve $F(c) = g$ by iterating

$$c_{k+1} = c_k - E(c_k)^{-1}(F(c_k) - g). \tag{1.14}$$

Observe first that if the sequence c_k converges, $c_k \to c^*$ for some admissible c^* (i.e., $0 < c^*(z) < \infty$), then necessarily

$$E(c^*)^{-1}(F(c^*) - g)(z) = c^*(z)(F(c^*) - g)(2\phi^*(z)) = 0$$

with $\phi^*(z) = \phi(z; c^*)$, so that c^* must be the unique solution of the inverse problem. In particular, there can be no such admissible limit if the data g is not in the range of F. The question then remains whether we can expect convergence of the sequence $\{c_k\}$ in the case of consistent data.

Let

$$\Phi(c) = c - E(c)^{-1}(F(c) - g).$$

Then Φ is a map from $H_1(0, Z)$ into $H_1(0, Z)$ and we are looking for fixed points of Φ. It is natural to hope that Φ is a contraction, at least near the correct solution, and the space H_1 is chosen wisely.

We compute the derivative of Φ. After tedious but straightforward calculation we arrive at

$$\begin{aligned}(D\Phi(c)\zeta)(z) = &-(E(c)^{-1}K(c)\zeta)(z) \\ &+2c(z)(F(c) - g)'(2\phi(z))\int_0^z \frac{\zeta(s)}{c^2(s)}\,ds \\ &-(F(c) - g)(2\phi(z))\zeta(z)\end{aligned} \tag{1.15}$$

at least formally.

Let us suppose that all the action takes place on a set of c's satisfying

$$\|(\log c)'\|_{L^2(0,Z)} \leqq M \tag{1.16}$$

for some M. This may be justified a posteriori by the fact that we do not observe the iterates c_k blowing up when the data g is consistent. Furthermore, the iterative procedure is giving approximate descent directions for the functional $\|(F(c)-g)'\|_{L^2}$, and this should correspond to a decrease of $\|(\log c - \log c^*)'\|_{L^2}$ when $g = F(c^*)$.

A bound like (1.16) implies in particular that the c's are bounded and bounded away from zero.

Under such conditions the first two terms in (1.15) represent operators with the property that their norms tend to zero with the length of interval Z, uniformly in c. The third term has a small norm provided the residual $F(c)-g$ is small.

It follows that if the depth interval Z is sufficiently small and the initial guess c_0 is close enough, the functional Φ is contraction for all relevant c's, and so the sequence c_k converges to the correct solution.

We now observe that the restrictions on Z and c_0 may be dispensed with. Let us suppose that the data g satisfies $g(0)=1$ to be consistent with $c(0)=1$, and the initial guess c_0 also satisfies $c_0(0)=1$. Then the residual $F(c_0)-g$ is small on some depth interval $[0, Z_1]$ just by continuity. For Z_1 small enough, we are then in the situation just discussed, so we obtain convergence on $[0, Z_1]$. Now the convergence takes place essentially in just a few steps, at which point we may regard the solution as known on $[0, Z_1]$ and unknown for $Z > Z_1$. Again, the residual is small on $[Z_1, Z_2]$, some $Z_2 > Z_1$ by continuity, so the same argument implies that we will have convergence on $[Z_1, Z_2]$ in a few more steps. Continuing in this way we get a iterates $Z_1, Z_2, Z_3, \cdots$ so that the sequence $\{c_k\}$ converges on $[0, Z_n]$ for any n.

Now in principle it is possible that $Z_n \leqq Z^* < Z$ for all n so that reconstruction on the entire interval $[0, Z]$ cannot be achieved. Indeed if the data g is out of the range of F this is precisely what happens.

However, for consistent data, the iterates are eventually close to the solution $c(z)$ on $[0, Z_n]$ for any n, and the uniformity of the estimates on the operators in (1.14) implies that one can obtain a lower bound for the incremental depth interval $(Z_{n+1} - Z_n)$ on which convergence can be expected further in the sequence. Hence $Z^* < Z$ is not possible. Since these estimates depend essentially on the bound M in (1.15), we expect that if the desired solution c has steep gradients, then $(Z_{n+1} - Z_n)$ will be small, while for slowly varying $c(z)$ the distance will be larger. Thus there should be rapid convergence in the latter case and slow convergence in the former.

All of our numerical experiments yield behavior which is consistent with the above discussion. See § 3.

We have indicated that the iterates $\{c_k\}$ are each defined on the whole interval $[0, Z]$, and this may lead to some difficulties for values of Z which are significantly greater than those where the sequence is close to the correct solution. For example, in the first few steps the value of $c_k(z)$ for z near Z could actually go to zero, in which case the procedure could not be continued.

One could cope with this problem by working on successively larger subintervals of $[0, Z]$, which are small enough at each stage to guarantee that the above problem does not occur. In practice, an easier and more efficient approach is to constrain all c_k's to lie between given bounds, $0 < c_- \leqq c_k(z) \leqq c_+ < \infty$, where c_- and c_+ are upper and lower bounds for the true solution, which may typically be easily guessed. That is to say, we redefine $c_k(z) = c_-$ whenever $c_k(z) < c_-$, etc. If the limit solution c lies strictly between c_- and c_+ then it is still necessarily the solution of $F(c) = g$.

Finally, if we choose a depth interval Z on which we want to find $c(z)$ we do not know until we have found c exactly how large T should be so that $g(t)$, $0 \leqq t \leqq T$ determines $c(z)$. Here also then we need some preliminary guess of upper and lower bounds for $c(z)$, and from this we can calculate a maximum T necessary to be sure that enough information is present in the data.

On the other hand, if the time interval T is given, one could modify the above procedure to produce the solution $c(z)$ to the maximum depth Z which is possible with the given data. These two alternative viewpoints will be discussed further in the next section.

2. Let us now consider the inverse problem $F(c) = g$ as a best fit problem. In the simplest model problem of this type we attempt to minimize the L^2 norm of the residual $F(c) - g$; see Bamberger et al. [1] for more on this point of view. Given data $g(t)$ defined on $[0, T]$ we seek a profile $c(z)$ defined on $[0, Z]$ which best fits the data in this sense. In the exact formulation of the least square problem we must take account of the fact that the relation between T and Z depends on c. It is natural to regard either Z or T as given data with the other related via the constraint $T = 2\int_0^Z (1/c(s))\, ds$. Viewed in this way we are either trying to reconstruct c to the maximum possible depth for given length at data g, or else trying to recover c to a given depth Z using the least possible data. It would be interesting to determine whether or not redundancy in the data could be used to good effect in this problem.

We assume as always the normalization $c(0) = 1$; some regularity of the data is also necessary, say $g \in H^1(0, T)$. Here then are two possible forms of the problem.

(LS1) Given $T > 0$, $g \in H^1(0, T)$,

$$\text{solve} \quad \min_{\substack{c, Z \\ c>0,\, c\in H^1(0,Z) \\ c(0)=1 \\ 2\int_0^Z \frac{1}{c(s)}\, ds = T}} \|F(c) - g\|_{L^2(0,T)}.$$

(LS2) Given $Z > 0$, $g \in H^1_{\text{loc}}(0, \infty)$,

$$\text{solve} \quad \min_{\substack{c, T \\ c>0,\, c\in H^1(0,Z) \\ c(0)=1 \\ 2\int_0^Z \frac{1}{c(s)}\, ds = T}} \|F(c) - g\|_{L^2(0,T)}.$$

In the case of consistent data, then clearly both of these problems have a unique solution, namely, the exact solution of $F(c) = g$. For g not in the range of F, the question remains whether a best fit is possible. Furthermore, in considering optimization problems of this type, it is of interest to understand whether there can be other local minima or even stationary points of the functional, distinct from the global minimum.

We proceed by computing the Euler equations for these two problems via the usual methods of the calculus of variations. The functions c which enter may be assumed always to be defined on whatever interval is necessary so that $F(c)(t)$ is defined on $[0, T]$.

We obtain for (LS1)

$$DF(c)^*(F(c) - g) = 0, \tag{2.1}$$

while for (LS2) we get

$$DF(c)^*(F(c) - g) - \frac{(F(c) - g)^2(2\phi(Z, c))}{c^2} = 0. \tag{2.2}$$

From the results in Santosa and Symes [11] it may be shown that $DF(c)$ is onto, hence $DF(c)^*$ is one-to-one. Therefore, if (2.1) holds then necessarily, $F(c) = g$. Thus (LS1) cannot have a solution for g not in the range of F, while for consistent data g, the exact solution is the only possible stationary point for the objective functional.

For (LS2) the situation is less clear. For inconsistent data the Euler equation (2.2) may have a solution. For example, if $g(t) \equiv -1$ then $c(z) \equiv 1$ satisfies (2.2). However, this is a saddle point, not a minimum. We may obtain further information from the decomposition (1.10). By simple computations

$$(E(c)^*\chi)(z) = \frac{2}{c^2(z)}\chi(2\phi(z; c)) \tag{2.3}$$

and

$$(K(c)^*\chi)(z) = \int_{2\phi(z;c)}^{T} G(z, t)\chi(t)\, dt. \tag{2.4}$$

Hence, if we set $z = Z$ in (2.2), we get

$$2(F(c) - g)(T) = (F(c) - g)^2(T) \tag{2.5}$$

so that either $(F(c) - g)(T) = 0$ or 2. If the value is zero, we are back to equation (2.1) and the previous comments apply.

In the case that $(F(c) - g)(T) = 2$, we evaluate equation (2.2) at $z = 0$, recalling $c(0) = 1$ to obtain

$$2(1 - g(0)) + \int_0^T G(z, t)(F(c) - g)(t)\, dt = 4. \tag{2.6}$$

If the data is consistent with the surface value, then $g(0) = 1$; using the Schwarz inequality, we then get

$$\|F(c) - g\|_{L^2(0,T)} \geqq \frac{4}{K\sqrt{T}} \tag{2.7}$$

where K is an upper bound to $|G(z, t)|$. Thus we can be sure at least that (LS2) has no stationary points, aside from the exact solution, for which the residual is very small. We conjecture that there are no local minima aside from the exact solution.

Let us now consider the situation when a penalty term is added to the objective functional. Again there are two distinct problems, one with T fixed and the other with Z fixed.

(LS3) Given $T > 0$, $g \in H^1(0, T)$,

$$\text{solve} \quad \min_{\substack{c, Z \\ c(0) = 1 \\ 2\int_0^Z \frac{1}{c(s)}\, ds = T}} \left[\|F(c) - g\|^2_{L^2(0,T)} + \lambda^2 \|(\log c)'\|^2_{L^2(0,Z)}\right],$$

(LS4) Given $Z > 0$, $g \in H^1_{\text{loc}}[0, \infty]$,

$$\text{solve} \quad \min_{\substack{c, T \\ c(0) = 1 \\ 2\int_0^Z \frac{1}{c(s)}\, ds = T}} \left[\|F(c) - g\|^2_{L^2(0,T)} + \lambda^2 \|(\log c)'\|^2_{L^2(0,Z)}\right],$$

The use of such a penalty term will be justified by its performance later. We remark at this point that (LS3) or (LS4) has at least one solution for any data g, whether or not g belongs to the range of F, and any solution is necessarily bounded

away from zero. The proof of this may be achieved by the use of results from Symes [14] and standard arguments from variational methods in P.D.E. Uniqueness of the minimizer remains an open question; the results of numerical experiments support the conjecture that there is only one minimum, even locally. Saddle points, of course, will not be seen numerically even if they are there.

We expect that for small $\lambda > 0$ the solution of (LS3) or (LS4) will be a smooth approximation to the exact solution of the inverse problem, and furthermore in the penalized problem the solution will be more continuously dependent on the data.

We proceed to calculate the Euler equations for (LS3) and (LS4).

In the case that T is fixed, we regard Z as a functional of c and T and set

$$\frac{d}{d\varepsilon}[\|F(c+\varepsilon\zeta)-g\|^2_{L^2(0,T)}+\lambda^2\|(\log(c+\varepsilon\zeta))'\|^2_{L^2(0,Z(c+\varepsilon\zeta,T))}]_{\varepsilon=0}=0 \tag{2.8}$$

to obtain the condition for a stationary point. After some calculation we arrive at

$$2\langle DF(c)\zeta, F(c)-g\rangle+2\lambda^2\left\langle\frac{c'}{c},\left(\frac{\zeta}{c}\right)'\right\rangle+\lambda^2 c(Z)\left(\frac{c'}{c}\right)^2(Z)\left\langle\frac{1}{c^2},\zeta\right\rangle=0 \tag{2.9}$$

which is valid for $\zeta\in H^1(0,\infty)$, $\zeta(0)=0$. Choosing $\zeta\in C_0^\infty(0,Z)$ we obtain the Euler equation

$$2DF(c)^*(F(c)-g)-\frac{2\lambda^2}{c}\left(\frac{c'}{c}\right)'+\frac{\lambda^2}{c^2}c(Z)\left(\frac{c'}{c}\right)^2(Z)=0. \tag{2.10}$$

It follows that for arbitrary $\zeta\in H^1(0,\infty)$ we must have

$$\frac{c'}{c}\frac{\zeta}{c}\Big|_0^Z=0.$$

Since $\zeta(0)=0$ and $c(Z)\neq 0$ this implies $c'(Z)=0$. We conclude that any stationary point for the minimization problem (LS3) must satisfy

$$\begin{aligned} &DF(C)^*(F(c)-g)-\frac{\lambda^2}{c}\left(\frac{c'}{c}\right)'=0, \qquad 0\leqq z\leqq Z,\\ &c(0)=1, \qquad c'(Z)=0. \end{aligned} \tag{2.11}$$

In the case that Z is fixed, some analogous calculations lead to

$$\begin{aligned} &DF(c)^*(F(c)-g)-\frac{\lambda^2}{c}\left(\frac{c'}{c}\right)'+\frac{(F(c)-g)^2(2\phi(Z,c))}{c^2}=0, \qquad 0<z<Z,\\ &c(0)=1, \qquad c'(Z)=0. \end{aligned} \tag{2.12}$$

We now wish to derive a constructive method for approximation solutions to (LS3) or (LS4), analogous to our treatment of Newton's method. It is most convenient to regard Z as fixed in the problem, even though (2.11) has a simpler look to it. In the end, our procedure will yield a function c which may be regarded as a good approximation to the solution of either (LS3) or (LS4).

We shall try a gradient method, based on the approximation

$$J(c+\zeta)=J(c)+\langle DJ(c),\zeta\rangle$$

where

$$J(c)=\|F(c)-g\|^2_{L^2(0,T(c,Z))}+\lambda^2\|(\log c)'\|^2_{L^2(0,Z)}.$$

By a calculation analogous to those which give (2.9) we have

$$\langle DJ(c), \zeta\rangle = \langle F(c)-g, DF(c)\zeta\rangle + \lambda^2\left\langle \frac{c'}{c}, \left(\frac{\zeta}{c}\right)'\right\rangle + (F(c)-g)^2(2\phi(Z,c)\left\langle \frac{1}{c^2}, \zeta\right\rangle. \tag{2.13}$$

As we are interested only in small λ, we expect c to be near the exact solution of $F(c)=g$. Hence in particular the third term on the right in (2.13) will be quadratically small, so we drop it.

Now if the direction ζ is defined by

$$\begin{aligned} &DF(c)^*DF(c)\zeta - \frac{\lambda^2}{c}\left(\frac{\zeta}{c}\right)'' = DF(c)^*(F(c)-g) - \frac{\lambda^2}{c}\left(\frac{c'}{c}\right)', \\ &\zeta(0)=0, \qquad \zeta'(Z)=0, \end{aligned} \tag{2.14}$$

then $-\zeta$ is a descent direction for J, and a line search based on the assumption that $DF(c)$ does not vary near c reveals the $c-\zeta$ is locally the minimum for J to this order of approximation.

Now as in Newton's method we do not want to deal with $DF(c)$ itself, so we replace $DF(c)$ by $E(c)$, hopefully making a small error. Recalling the formula for $E(c)$ and $E(c)^*$ we arrive at the following algorithm:

$$c_{k+1} = c_k - \zeta, \tag{2.15}$$

where ζ is defined by

$$\frac{-\lambda^2}{c}\left(\frac{\zeta}{c}\right)'' + \frac{2}{c^3}\zeta = \frac{2}{c^2}(F(c)-g)(2\phi(\cdot\,; c)) - \frac{\lambda^2}{c}\left(\frac{c'}{c}\right)', \tag{2.16}$$

$$\zeta(0)=0, \qquad \zeta'(Z)=0. \tag{2.17}$$

The equations for ζ constitute a two point boundary value problem, which is easily seen to have a unique solution, for any $\lambda > 0$. When discretized, it becomes a strictly diagonally dominant tridiagonal system, which may be easily and inexpensively solved, even for very small λ; see § 3.

Note that as $\lambda \to 0$ the algorithm (2.15)-(2.16)-(2.17) goes over to the algorithm of § 1.

Now if this iterative method converges at all, $c_k \to c$, then necessarily the limit satisfies

$$\begin{aligned} &(F(c)-g)(2\phi(\cdot\,; c)) = \lambda^2 c\left(\frac{c'}{c}\right)', \\ &c(0)=1, \qquad c'(Z)=0, \end{aligned} \tag{2.18}$$

which may be regarded as close to either (2.11) or (2.12). Nevertheless, the function c is not an exact solution of (2.11) or (2.12) except in some special cases. By comparison, the algorithm of § 1 yielded the exact solution of $F(c)=g$, despite the fact that some approximations had been made.

In any case, the equation (2.18) simply says that the residual at c is of order λ^2, and hence is a reasonable candidate for an approximate solution of the inverse problem, without reference to how it was arrived at.

The derivation of the algorithm (2.15)-(2.16)-(2.17) is of course important in understanding its behavior. We may make some heuristic arguments similar to those in § 1, namely, on sufficiently small depth intervals this method is a small perturbation of a standard gradient method for least squares problems, and hence the sequence

$\{c_k\}$ will converge. As in the case of Newton's method, the iterates c_k should then converge to the solution c of (2.18) on larger and larger depth intervals as k increases.

3. In this section, we will describe how one can implement the optimization algorithm proposed in the previous section. We will also present some results of numerical simulations with our method. The numerical experiments were designed to display various features of the problem which were discussed in the previous sections.

Recall that we want to find the minimizer of the functional

$$J(c) = \|F(c) - g\|^2_{L^2(0,T(c;Z))} + \lambda^2 \|(\log c)'\|^2_{L^2(0,Z)}$$

where Z is fixed, and the data $g(t)$ is given on a sufficiently large interval so that all the information about $c(z)$ up to depth Z is contained in it.

We will seek the minimizer iteratively starting with some initial given $c_0(z)$. The updates, $c_{k+1}(z)$ are obtained from the previous values $c_k(z)$ by first calculating $\zeta(z)$, which solves

$$\text{(a)} \qquad \frac{-\lambda^2}{c_k(z)}\left(\frac{\zeta(z)}{c_k(z)}\right)'' + \frac{2}{c_k^3(z)}\zeta(z) = \frac{2}{c_k^2(z)}(F(c_k) - g)|_{t=2\phi(z,c_k)} - \frac{\lambda^2}{c_k(z)}\left(\frac{c_k'(z)}{c_k(z)}\right)'$$

$$\text{with } \zeta(0) = 0,\ \zeta'(Z) = 0, \text{ and } \phi(z, c_k) = \int_0^z \frac{1}{c_k}.$$

Once $\zeta(z)$ is found, we set

$$\text{(b)} \qquad c_{k+1}(z) = c_k(z) - \zeta(z).$$

Convergence is checked by

(c) if $\|c_{k+1}(z) - c_k(z)\| < \varepsilon$, stop;

else return to (a).

The stopping rule in (c) can be based on various choices of norms, such as $L^2[0, Z]$ for some tolerance $\varepsilon \ll 1$.

To find $\zeta(z)$, we need to first evaluate the right-hand side, and second, to solve the ODE in (a). Evaluating the right-hand side involves computing the forward map $F(c_k)$. Solving the two point boundary value problem for ζ will be done by making finite difference approximations to the derivative on the left-hand side and then solving the difference equations.

We will first discuss in some detail the computation of the forward map. We work with the discretized version of our problem. A function of depth z and time t will be represented by their discrete counterpart, which will be samples of its value at equal depth increments h and equal time increments k. Hence, for example, our data is now the sequence $g(t = mk)$ for $m = 0, 1, 2 \cdots M$, and our sound speed profile is $c(z = nk)$ for $n = 0, 1, 2 \cdots N$. Recall now that $F(c)$ is defined to be the solution $u(z, t)$ to (0.1), (0.2) and (0.3) restricted to $z = 0$, i.e., $F(c) := u(0, t)$. To obtain the discrete forward map, we will use a finite difference scheme to solve the PDE satisfied by $u(z, t)$, and evaluate $u(0, t = mk)$.

To obtain an accurate approximate solution to our PDE, we will first make a change of variable to "straighten" the characteristics of the operator in (0.1), then use a characteristic finite difference to compute the solution.

The first step involves introducing travel time $\phi(z) = \int_0^z 1/c(s)\, ds$ to represent depth. Substituting

$$\frac{d}{dz} = \frac{d}{d\phi}\frac{d\phi}{dz} = \frac{1}{c}\frac{d}{d\phi}$$

in (0.1), we get

$$\frac{1}{c}u_{tt} = \left(\frac{1}{c}u_\phi\right)_\phi \quad \text{for } \phi > 0. \tag{3.1}$$

Now, we can regard $c = c(z(\phi))$ and $u(z(\phi), t)$ as functions of ϕ. Equation (3.1) has straight characteristics: $dt/d\phi = \pm 1$. The initial condition and the boundary condition remain unchanged if we assume $c(0) = 1$. So, we solve (3.1) with

$$u \equiv 0 \quad \text{for } t < 0, \tag{3.2}$$

$$u_\phi(0, t) = -\delta(t). \tag{3.3}$$

Suppose we are given the sequences $c(z = nh)$ for $n = 0, 1, 2, 3 \cdots N$. The travel time function $\phi(z)$ is computed by trapezoidal rule:

$$\phi(z = nh) = h\left[\frac{1}{2c(0)} + \sum_{j=1}^{n-1}\frac{1}{c(jh)} + \frac{1}{2c(nh)}\right] \tag{3.4}$$

with $\phi(0) = 0$. We will want to use the characteristic finite differences as described in Bube and Burridge [3], and Santosa and Schwetlick [10]; hence, we will need samples of $1/c$ at equal increments. If we have chosen k, we will need $1/c(Z(\phi))$ at increments of $k/2$ in ϕ. We accomplish this by first inverting the relation in (3.4) for z as a function of ϕ sampled at the rate of $k/2$, i.e., $z(\phi = (m+\frac{1}{2})k/2)$, then finding $1/c$ at the sample points by linear interpolation. The inversion of (3.4) is done also by linear interpolation; hence we have committed an aggregate error of order h^2. Now since the class of $c(z)$ under consideration is a class of smooth functions, the error will be small.

Once we have the sequence $1/c(z(\phi = (m+\frac{1}{2})k/2))$, the application of the finite difference scheme to solve (3.1)-(3.3) is straightforward, and we refer the reader to [3] and [10]. The finite difference scheme returns the values $u(0, t = mk)$, which are second order accurate in k. The computation of $F(c)$ then can be expected to be second order accurate overall.

We have now the tools to evaluate the right-hand side of (a) for a given discrete $c_k(z)$. To solve for $\zeta(z)$, we replace the derivative on the left-hand side by the finite differences. This results in a tridiagonal system involving the vector $\zeta(z = nh)$, $n = 0, 1, 2 \cdots N$. The tridiagonal matrix equation is solved using standard linear algebra software. Our version of the systems of equations solver is stable; in particular, we can set $\lambda \to 0$ (a singular perturbation) and obtain very good results.

We have coded the algorithm proposed above and will next show some results of inversion simulations with this procedure. For our problem, we took the interval of reconstruction to be $[0, 1]$, the spatial sampling h is equal to the time sampling k, which is set to 0.01. To generate the data for the experiments, we created sound speed profiles $c(z)$ defined on larger intervals, and made certain that the data trace $g(t)$, defined on the interval $[0, 4]$, contains information about the profile $c(z)$ in the reconstruction interval $[0, 1]$. The synthetic data is computed by the procedure we described earlier.

The aim of our simulations are to study:

(a) the behavior of the algorithm as the smoothness of the unknown profiles $c(z)$ and penalty coefficient λ are varied,

(b) the properties of the reconstruction when the data is noisy,

(c) the properties of the solution as the penalty term is varied.

To this end, we created four sound speed profiles, Profiles A, B, C and D. Profiles A, B and C are displayed in Figs. 1(a), 2(a) and 3(a), respectively. Their impulse responses $u(0, t)$ are displayed in Figs. 1(b), 2(b) and 3(b). The fourth profile is created by passing two smoothing operations, using an averager (averages over 3 points) over Profile C. Profiles A and B represent "realistic" rough and smooth sound speed profiles respectively. Profiles C and D are pathological and will be used to illustrate our claims.

The first set of experiments is to display the behavior of the algorithm when the "target" profiles are of various degrees of smoothness and variation, and the penalty coefficient $\lambda = 0$. In Fig. 4, we show the iterative process of finding the solution (Profile A) given the impulse response. The convergence is rapid for this case, even when the starting profile was $c(z) = 1$. As predicted, the algorithm proceeds by correctly finding the first part of the profile, then progresses downward as iterations are continued.

We expect that for Profile B, a smoother profile, the algorithm will find the solution more quickly. This can be seen in Fig. 5. For very rough and large variation profiles, such as Profile C, the algorithm will need to perform many more iterations before converging to the correct solution. This is displayed in Fig. 6.

To assess the dependence of the rate of convergence on the profile and the value of the penalty parameter λ, we conducted a series of tests, the results of which are shown in Table 1. The iterative process is stopped when for the kth iterate, $\|c_k - c_{k-1}\|_{L^2} / \|c_{k-1}\|_{L^2}$ is less than 1%. Also listed in the table is the data residual, $\|F(c) - g\|_{L^2}^2$, and the penalty term, $\lambda^2 \|(\log c)'\|_{L^2}^2$ at convergence.

It is interesting to note that for profile of moderate variation (A) and smooth profile (B), the convergence is so rapid that we see little dependence of the convergence rate on the penalty parameter λ. However, for the rough profile, Profile C, we see a trend of increased rate of convergence for increasing λ's. Notice that for the smoothed versions of Profile C, Profile D, convergence is consistently more rapid than their rough counterpart. Therefore, we conclude that the convergence rate of the algorithm appears to be inversely proportional to the variation of $c(z)$ and proportional to the degree of smoothness of $c(z)$.

The residuals in the data fit and the penalty terms typically are monotonic functions of λ for all profiles. Convergence rate of the cost functional $J(c)$ for the case when g is in the range of $F(\cdot)$ can be estimated from the values of $\|F(c) - g\|^2$ and $\lambda^2 \|(\log c)'\|^2$ from the table.

In Figs. 7 and 8, we show the dependence of the solution on the value of λ. As one would expect, the smoothness of the solution is proportional to λ. It is interesting to note that while for Profile A, the solutions for increasing values of λ are roughly smoother and smoother copies of Profile A, for Profile C, we observe that the predictive capability of the method degrades with increasing λ. Note also the decay of the resolution power of the scheme as a function of depth. Recall that the reason for including the penalty term was to guarantee stability, i.e., continuous dependence of the solution on the data. We will elaborate on this later.

Next we study the dependence of the solution on noise in the data. To simulate noisy data, we simply added a small pseudo-random component to the impulse response traces. The noisy impulse response to Profile A is shown in Fig. 9(a) (compare with Fig. 1(b) for a measure of noise).

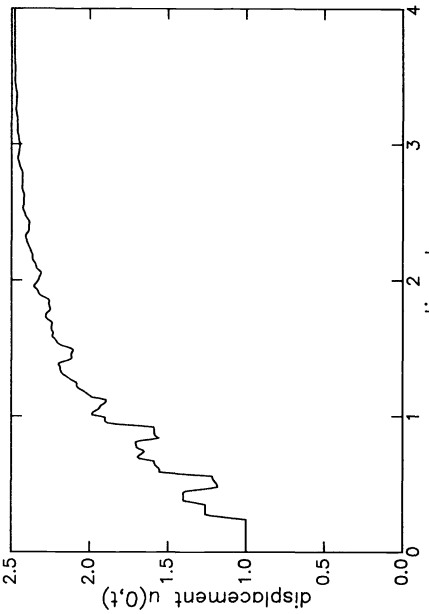


FIG. 1(b). *Impulse response of Profile* A.

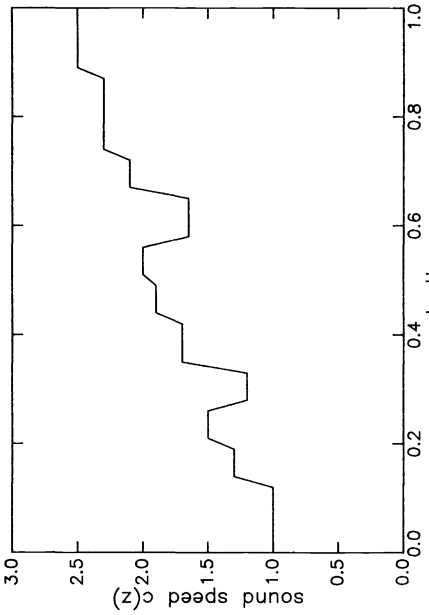


FIG. 1(a). *Sound speed Profile* A.

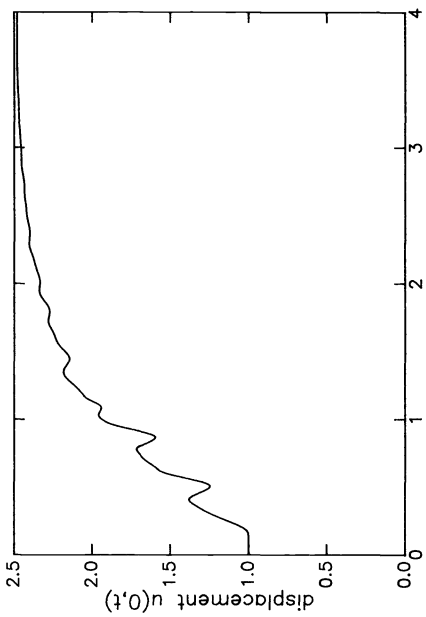


FIG. 2(b). *Impulse response of Profile* B.

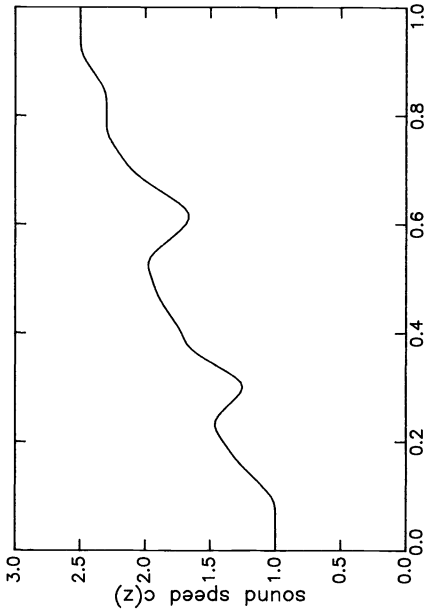


FIG. 2(a). *Sound speed Profile* B.

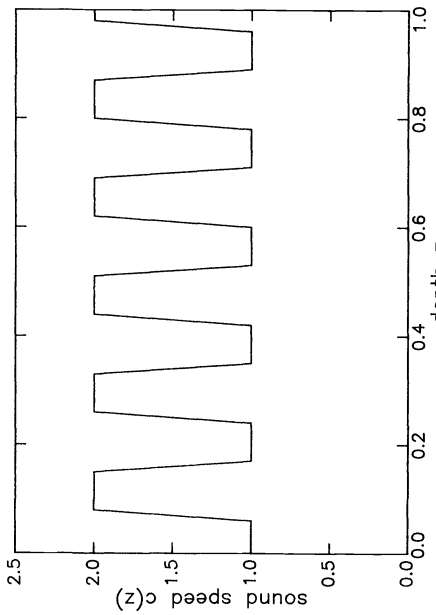


FIG. 3(a). *Sound speed Profile* C.

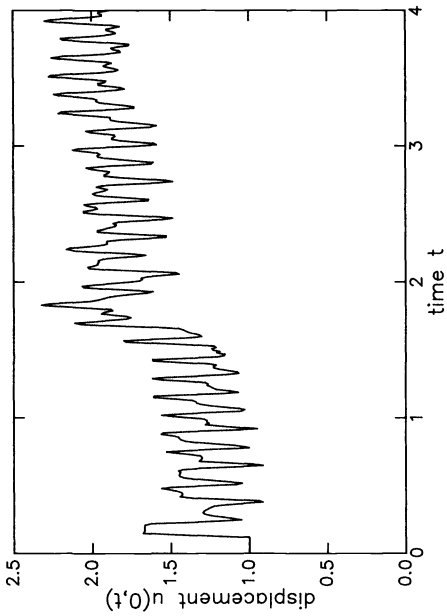


FIG. 3(b). *Impulse response of Profile* C.

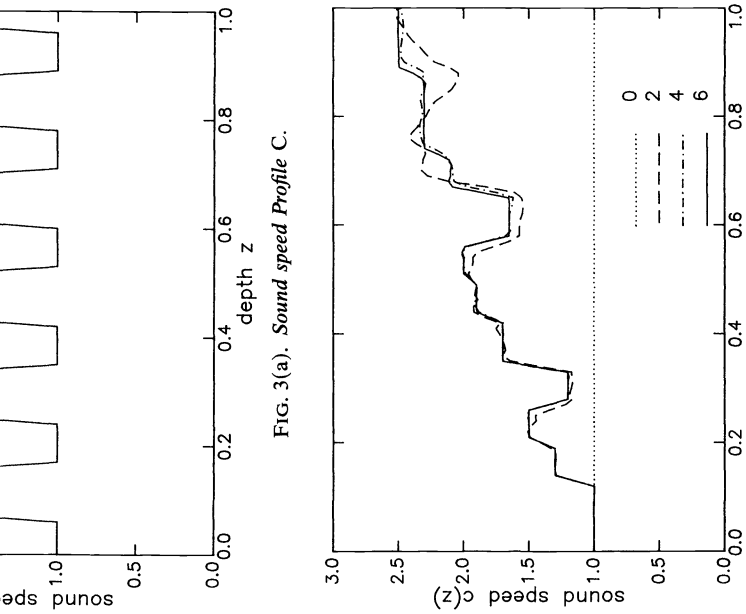


FIG 4. *The iterative process of seeking the unknown sound speed profile from the impulse response in Fig.* 1(b). *The initial guess is* $c(z) = 1$. *The penalty parameter* $\lambda = 0$.

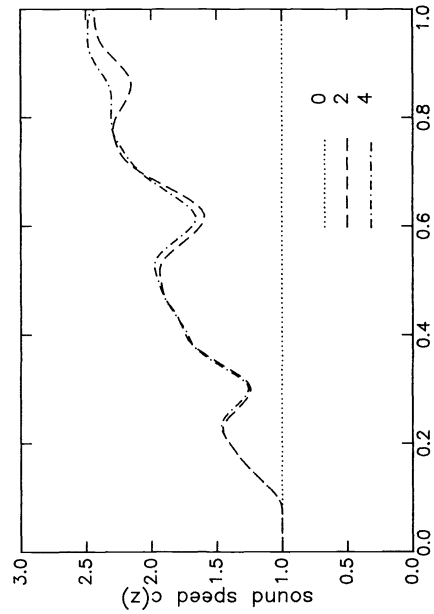


FIG. 5. *The iterative process of seeking the unknown sound speed profile from the impulse response in Fig.* 2(b); $\lambda = 0$.

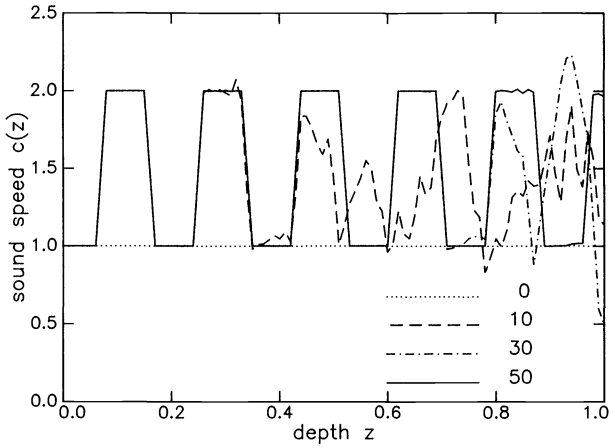


FIG. 6. *The iterative process of seeking the unknown sound speed profile from the impulse response in Fig.* 3(b); $\lambda = 0$.

For Profile A, which has moderate variation, we observe that the solutions are rather insensitive to noise even when $\lambda = 0$. Symes [13], however, has predicted that the loss of resolution is *necessary* to stabilize the sound speed inverse problem. To demonstrate this, we refer to the results of the next set of experiments.

We took the impulse response to Profile C and smoothed it using a 7-point averaging kernel. Then we apply our algorithm to the smoothed data for $\lambda = 0$ and $\lambda = 0.01$. The results are compared with the solutions when the data is unperturbed. In Figs. 10(a) and 10(b), we see that smoothing the data dramatically affects the solution when $\lambda = 0$, and not so severely when $\lambda = 0.01$. The effects are much less pronounced for impedance inversions; see Pao et al. [8]. This experiment suggests that the continuity depends on the smoothness and the variation of the profile.

Computing cost is very inexpensive. For our model problem involving 100 points in depth, it took the Prime 750 only 6 seconds of CPU/iteration, which translates to 5 cents/iteration. A quick operation count tells us that the increase in costs per iteration is proportional to the square of the number of points.

Finally, we note that it is not very difficult to alter the impulse response in such a way that $F(c) = g$ has no solution. That is, it is possible to alter a synthetically generated data so that it is no longer in the range of $F(c)$ for a reasonable (positive and bounded) $c(z)$. This can be done, for example, by shifting the $t = 0$ point in the trace. However, for $\lambda \neq 0$, we can still expect to find a minimizer (hence a solution of the inverse problem) to the optimization problem even when $g(t)$ is not in the range of $F(c)$.

4. Conclusions. We have presented a simple method for finding a sound speed profile $c(z)$ of a medium given the impulse response of the medium at the boundary. The method is iterative and uses the high frequency asymptotics of the linearized relation between sound speeds and impulse response. The method can be easily extended to solve the minimization problem of finding the least-squares solution with an additional penalty involving the logarithmic derivative of $c(z)$.

From our study, we made several observations about the behavior of the method and properties of the problem. We have seen that the proposed solution algorithm is relatively simple to implement and operationally efficient. Some aspects of the problem and the solution scheme are demonstrated in our numerical experiments.

TABLE 1

PROFILE A

λ	Data residual	Penalty term	Number of iterations
0	1.37×10^{-6}	0	7
0.005	4.95×10^{-5}	2.89×10^{-4}	7
0.01	2.84×10^{-4}	8.30×10^{-4}	7
0.03	2.15×10^{-3}	2.96×10^{-3}	7
0.05	4.45×10^{-3}	4.55×10^{-3}	6

PROFILE B

λ	Data residual	Penalty term	Number of iterations
0	8.55×10^{-7}	0	6
0.005	2.39×10^{-6}	9.93×10^{-5}	6
0.01	2.27×10^{-5}	3.65×10^{-4}	6
0.03	6.20×10^{-4}	1.98×10^{-3}	6
0.05	1.77×10^{-3}	3.60×10^{-3}	7

PROFILE C

λ	Data residual	Penalty	Number of iterations
0	8.20×10^{-7}	0	52
0.005	7.97×10^{-4}	2.64×10^{-3}	33
0.01	4.32×10^{-3}	5.94×10^{-3}	40
0.03	2.53×10^{-2}	9.12×10^{-3}	8
0.05	3.78×10^{-2}	7.55×10^{-3}	10

PROFILE D

λ	Data residual	Penalty term	Number of iterations
0	4.17×10^{-7}	0	44
0.005	9.57×10^{-5}	1.36×10^{-3}	28
0.01	7.32×10^{-4}	3.26×10^{-3}	19
0.03	7.39×10^{-3}	5.91×10^{-3}	7
0.05	1.31×10^{-2}	5.97×10^{-3}	7

We note here that the method presented generalizes to inverse problems for one-dimensional hyperbolic systems. Moreover, problems where responses other than those due to an impulse are given can also be treated in the same spirit. All that is needed is to compose all the relevant maps from coefficients of the PDE to the responses with the convolution operator involving the exciting source; the added complexity is minimal at the computational level. Although this will not solve the difficulty which arises when solving inverse problems from responses to sources which are low frequency band limited, the high frequency band limitation in the source should present no difficulty when smoothness penalties are included.

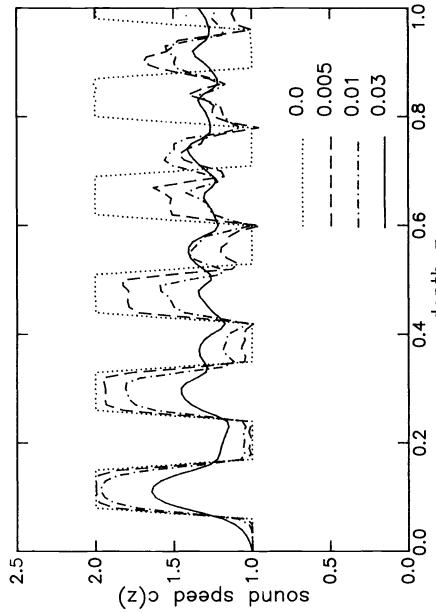


FIG. 8. *The solution of the inverse problem for Profile C for various choices of penalty parameter.*

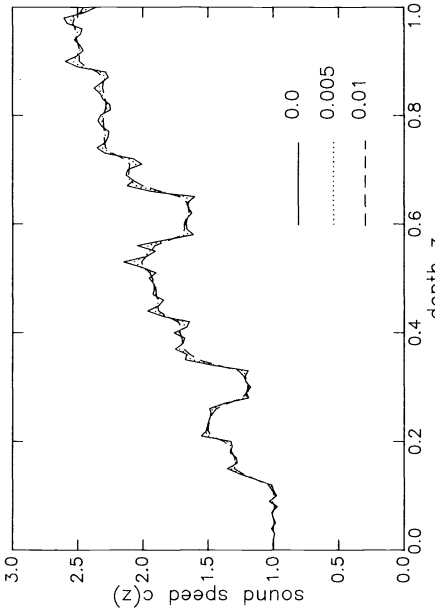


FIG. 9(b). *The solution of the inverse problem from noisy impulse response to Profile* A *for various choices of penalty parameters.*

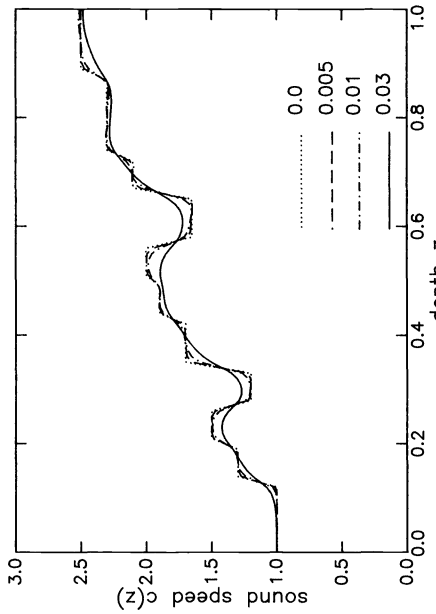


FIG. 7. *The solutions of the inverse problem for Profile* A *for various choices of penalty parameter.*

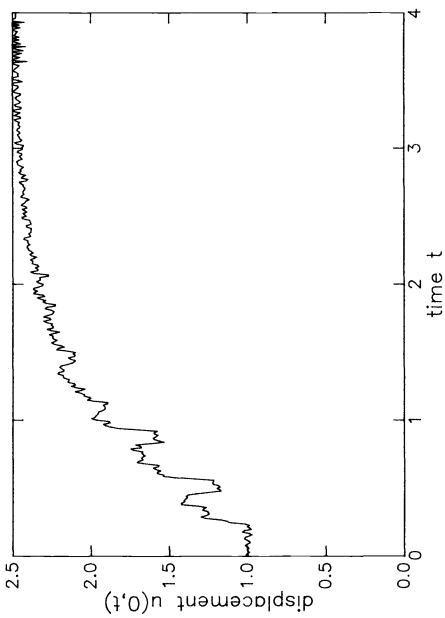


FIG. 9(a). *The noisy impulse response to Profile* A, *compare with Fig.* 1(b).

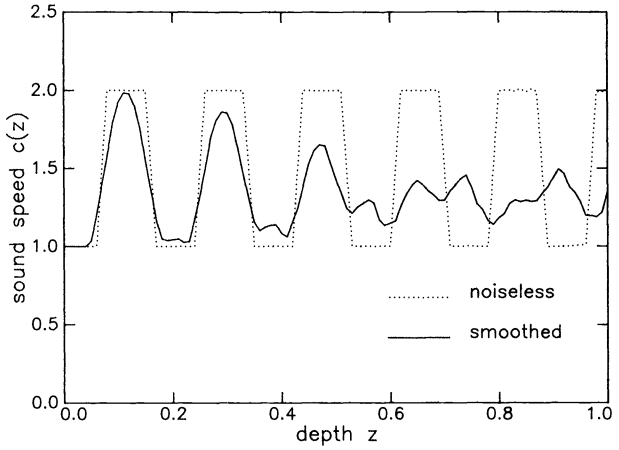


FIG. 10(a). *The results of inversion using the correct data (Fig. 3(b)), and smoothed data when the penalty parameter* $\lambda = 0$. *The smoothing simulates the absence of high frequency components and is done by a "box-car" averager of width 7 points.*

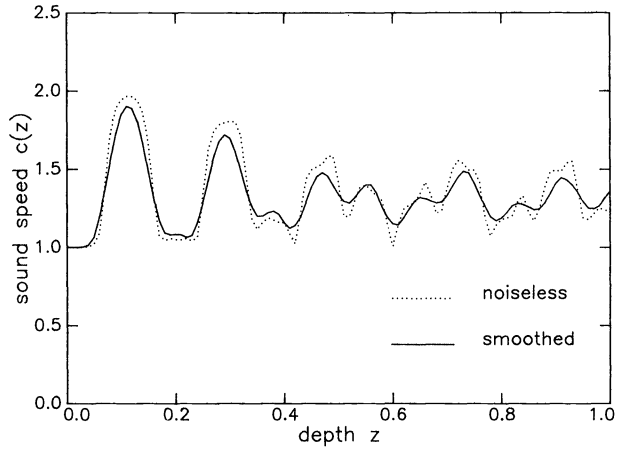


FIG. 10(b). *The results of inversion using the correct and smoothed data when* $\lambda = 0.01$. *Observe that the two results are closer to each other than the previous results.*

Acknowledgments. We thank Bill Symes, Ken Bube and Patrick Lailly for their participation in discussions on this work.

REFERENCES

[1] A. BAMBERGER, G. CHAVENT AND P. LAILLY, *About the stability of the inverse problem in* 1-D *wave equations*, Appl. Math. Optim., 5 (1979), pp. 1-87.

[2] N. BLEISTEIN, J. COHEN, J. DESANTO AND F. HAGIN, *Project review on geophysical and ocean sound speed profile inversion*, in Inverse Problems of Elastic and Acoustic Waves, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1984.

[3] K. BUBE AND R. BURRIDGE, *The one-dimensional inverse problem of reflection seismology*, SIAM Rev., 25 (1983), pp. 497-559.

[4] K. BUBE, P. LAILLY, P. SACKS, F. SANTOSA AND W. SYMES, *The determination of perturbations in velocity and the source wavelet from plane wave responses for stratified media and nonimpulsive sources*, preprint.

[5] R. CLAYTON AND R. STOLT, *A Born-WKBJ inversion method for acoustic reflection data*, Geophysics, 46 (1981), pp. 1559-1567.

[6] R. COURANT AND D. HILBERT, *Methods of Mathematical Physics*, Vol. 2, Interscience, John Wiley, New York, 1962.

[7] S. GRAY, *A second order procedure for one-dimensional velocity inversion*, SIAM J. Appl. Math., 39 (1980), pp. 456–462.

[8] Y. H. PAO, F. SANTOSA AND W. SYMES, *Inverse problems of acoustic and elastic waves*, in Inverse Problems of Acoustic and Elastic Waves, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1984.

[9] P. SACKS AND W. SYMES, *Recovery of the elastic parameters of a layered half space*, Geophys. J. R. Astr. Soc., to appear.

[10] F. SANTOSA AND H. SCHWETLICK, *The inversion of acoustical impedance profile by methods of characteristics*, Wave Motion, 4 (1982), pp. 99–118.

[11] F. SANTOSA AND W. SYMES, *High frequency perturbational analysis of the surface point-source response of a layered fluid*, J. Comp. Phys., to appear.

[12] W. SYMES, *Stable solution of the inverse reflection problem for a smoothly stratified elastic medium*, SIAM J. Math. Anal., 12 (1981), pp. 421–453.

[13] ———, *Stability properties for the velocity inversion problem*, in Mathematical and Computational Methods in Seismic Exploration and Reservoir Modeling, W. E. Fitzgibbon, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1986.

[14] ———, *On the relation between coefficients and boundary values in Webser's Horn equation*, SIAM J. Math. Anal., 17 (1986), pp. 1400–1420.

[15] A. YAGLE, *Layer stripping solutions of inverse seismic problems*, Ph.D. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1985.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 4, July 1987

© 1987 Society for Industrial and Applied Mathematics
004

A COMPARISON OF ITERATIVE METHODS AS APPLIED TO THE SOLUTION OF THE NONLINEAR THREE-DIMENSIONAL GROUNDWATER FLOW EQUATION*

LOGAN K. KUIPER†

Abstract. This paper compares the efficiency of seventeen different methods for the solution of the simultaneous nonlinear finite difference approximating equations for groundwater flow in a watertable aquifer in three dimensions. Solution methods include those of Newton, Picard, and generalized linear methods. The Newton and Picard iteration methods were implemented by making use of several different linear methods including successive overrelaxation, the strongly implicit procedure, and eight different preconditioned conjugate gradient methods. The best methods were found to be those using Picard iteration implemented with the preconditioned conjugate gradient method.

Key words. iterative solution of nonlinear simultaneous equations, groundwater flow

1. Introduction. The approximating equations to groundwater flow are commonly solved with iterative methods. This is true not only for linear confined aquifer flow equations, but also for nonlinear watertable aquifer flow equations. The most commonly used procedure for solving nonlinear flow equations is that of Picard iteration (Ortega and Rheinboldt [7]; Remson et al. [9]) in conjunction with a linear iterative method such as the alternating direction implicit procedure (ADI), successive overrelaxation (SOR), or the strongly implicit procedure (SIP) (Trescott and Larson [10]). Investigations of the performance of the many other methods for the solution of simultaneous nonlinear equations, applied to watertable aquifer simulations, are scarce in the literature. The author (Kuiper [6]) made a comparison of Picard iteration in conjunction with SIP (Picard-SIP), and Picard iteration in conjunction with the incomplete Cholesky conjugate gradient method (Picard-ICCG), applied to a two-dimensional watertable aquifer. Comments by Remson et al. [9] praise a method called one-step point SOR-Newton by Ortega and Rheinboldt [7], whose naming and classification procedures will also be used in this work. An investigation by Concus [2] of a nonlinear magnetostatic field problem in two dimensions compares one-step Newton-point SOR, one-step Newton-line SOR, and one-step point SOR-Newton. In his comparison, one-step point SOR-Newton compares favorably with the other two methods. However, these results may not apply directly to a watertable aquifer because the nonlinearity of the two problems differs significantly. Ortega and Rockoff [8] present a comparison between several generalized linear methods.

The comparisons made in this paper are perhaps intended to answer a different set of questions than the usual comparisons of numerical methods. The nonlinearity to be dealt with is of a specific type found in the approximating equations for groundwater flow. Also, the geometry of the system and the type of boundary conditions are specific. The numerical tests are designed to compare the various methods on a range of test problems that are inclusive with respect to groundwater flow but nevertheless are very restrictive with respect to the solution of simultaneous nonlinear approximating equations in general. Results are considered to be useful even if the methods compare fairly closely.

* Received by the editors June 7, 1984; accepted for publication (in revised form) September 8, 1986.
† Water Resources Division, U.S. Geological Survey, Austin, Texas 78701.

Of the various Newton and Picard iteration and generalized linear methods tested, Picard-ICCG was found to be the method usually requiring the least amount of computational effort for a solution.

2. Groundwater flow equation. The partial differential equation for steady-state groundwater flow in an isotropic confined aquifer in three dimensions may be written as

$$\frac{\partial}{\partial x}\left(K\frac{\partial h}{\partial x}\right)+\frac{\partial}{\partial y}\left(K\frac{\partial h}{\partial y}\right)+\frac{\partial}{\partial z}\left(K\frac{\partial h}{\partial z}\right)=W \tag{1}$$

where $K(x, y, z)$ is hydraulic conductivity, $h(x, y, z)$ is hydraulic head, and $W(x, y, z)$ is the volumetric rate of water injection per unit volume of the aquifer from wells or other sources. For the purposes of this paper, K and W are assumed to be given throughout some region of solution R. Either h or $\partial h/\partial n$ is given on the closed surface surrounding R.

3. Integrated finite-difference approximating equations. Figure 1 shows in cross section a common type of geologic setting in which the flow of groundwater occurs. The straight vertical lines in Fig. 1 denote the sides of imaginary grid elements into which the solution region R is discretized. Each grid element has horizontal dimensions Δx and Δy along the x and y cartesian coordinate directions shown. The thicknesses Δz of the grid elements vary and are measured at grid element centered values of x and y. Values for Δz are typically much smaller than Δx and Δy. The dots in Fig. 1 show the node points at which the hydraulic head $h(x, y, z)$ is to be considered. These node points are centered both horizontally and vertically within the aquifer grid elements. The flow rate f_{12} from aquifer grid element 1 into aquifer grid element 2 may be approximated by

$$f_{12}\cong(h_1-h_2)[\Delta x(\Delta z_1+\Delta z_2)/2][(\Delta y/2K_1)+(\Delta y/2K_2)]^{-1}. \tag{2}$$

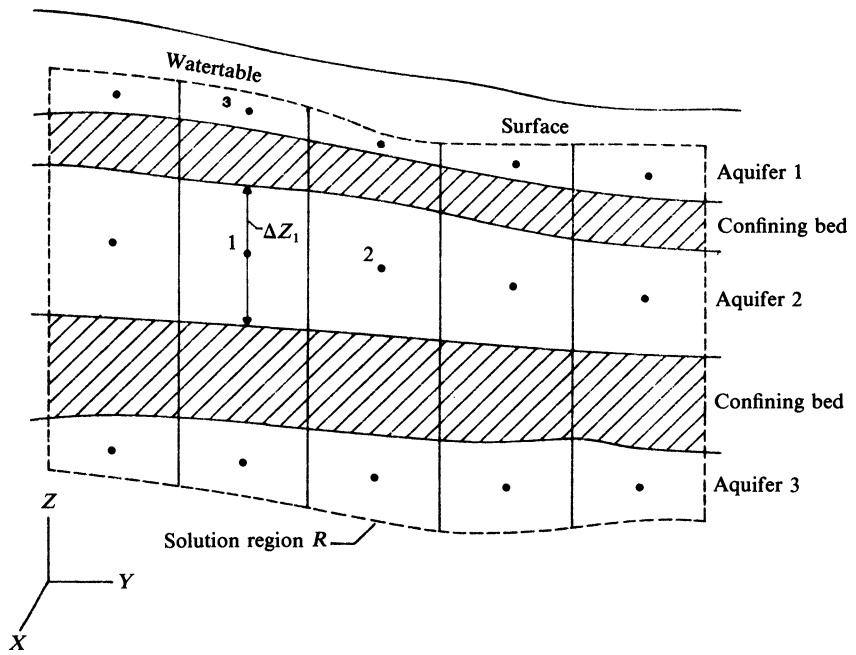


FIG. 1. *Geologic setting with integrated finite difference elements in solution region R.*

The expression for the vertical flow rate upward from aquifer grid element 1 into aquifer grid element 3 is

$$f_{13} \cong (h_1 - h_3)[\Delta x \Delta y][(\Delta z_1/2K_1) + (\Delta z_c/K_c) + (\Delta z_3/2K_3)]^{-1}. \tag{3}$$

Here, Δz_c is the vertical thickness of the confining bed between grid elements 1 and 3. The hydraulic conductivity K of this confining bed is denoted by K_c.

The nonlinearity arises because for the top aquifer, aquifer 1, Δz_i is equal to $h_i - v_i$ where v_i is the elevation z of the center point of the bottom surface of grid element i. Using the usual numbering procedure for the N_g grid elements in region R, let the order N_g matrix equation $Ah = b$ express the approximating equations to (1). Matrix A is symmetric positive definite and has three left of center diagonals $A_{i,i-1}$, $A_{i,i-r}$, and $A_{i,i-t}$, where r is the number of grid elements along the x direction of region R, which we shall suppose is rectangular in the x, y plane, and t is the number of grid elements in one layer. From (2) and (3), and for i corresponding to grid elements in aquifer 1, $A_{i,i-1}$ and $A_{i,i-r}$ are linear in $(h_i + h_{i-1})$ and $(h_i + h_{i-r})$ respectively, but $A_{i,i-t}$ is proportional to $(h_i + C)^{-1}$, where the positive constant C is usually much larger in magnitude than is h_i. Matrix element $A_{i,i-1}$ ($A_{i,i-r}$) increases as h_i or h_{i-1} (h_{i-r}) increase, but $A_{i,i-t}$ decreases when h_i increases. Values for $\Delta z_i = h_i - v_i \leqq 0$, corresponding to the dewatering of grid element i, are not allowed, so that there may not be a solution to $Ah = b$. An interesting feature of this nonlinear problem is that when water discharge occurs from some grid element i in aquifer 1 and h_i is thereby lowered, this lowering of h_i causes the conductance between grid element i with horizontally adjacent grid elements to decrease, thus causing an even further lowering of h_i. Thus for aquifer 1, grid elements which have discharge or are located close to those with discharge, may exhibit strongly nonlinear but local effects.

Let us replace h with x and rewrite matrix equation $Ah = b$ thus:

$$A(x)x = b. \tag{4}$$

Let $F = A(x)x - b$, then $F_i = \sum_s A_{is}x_s - b_i$, and $(F')_{ij} = \partial F_i/\partial x_j = A_{ij} + e_{ij}$ where $e_{ij} = \sum_s (\partial A_{is}/\partial x_j)x_s$. Here, although matrix A is symmetric positive definite, e_{ij} is not symmetric. For both i and j corresponding to grid elements in aquifer 1, $e_{ij} = -e_{ji}$. When i corresponds to a grid element in aquifer 1 and j corresponds to a grid element not in aquifer 1, $e_{ij} = 0$, but $e_{ji} \neq 0$.

Steady-state groundwater flow approximating equations similar to (2) and (3), but different, can be found in the literature. The type of nonlinearity exhibited by (2) and (3) will however be present in any set of approximating equations to the groundwater flow equation when one or more of the aquifers present can experience dewatering or in other words have watertable conditions.

4. Iterative methods. The iterative methods used to solve (4) may be grouped into two basic types.

4.1. Type 1. The first type consists in part of those methods which use the Newton method for their basic structure (Ortega and Rheinboldt [7]):

$$x^{n+1} - n^n = -[F'(x^n)]^{-1}F(x^n). \tag{5}$$

Here, $F = A(x)x - b$, $(F')_{ij} = \partial F_i/\partial x_j$, and n is the iteration index. Depending upon the method used to approximate x^{n+1} from (5), and also upon the number of inner iterations $m = 1, 2, \cdots, M$ that might be used to arrive at this approximation, (5) represents a variety of Newton methods. The remainder of the methods in type 1 consist of those which have as their basic structure:

$$x^{n+1} - x^n = -[B(x^n)]^{-1}F(x^n) \tag{6}$$

where B is an approximation to F' (Ortega and Rheinboldt [7]). Since $F = A(x)x - b$, $(F')_{ij} = A_{ij} + e_{ij}$ where $e_{ij} = \sum_s (\partial A_{is}/\partial x_j)x_s$. When $e_{ij} \ll A_{ij}$, $(F')_{ij} \cong A_{ij}$ so that an appropriate approximation for F' is the matrix A. With $B = A$, (6) becomes:

$$x^{n+1} = [A(x^n)]^{-1}b, \tag{7}$$

the method of Picard, an example of those methods whose basic structure is given by (6), where $B \cong F'$. As with (5), (6) and (7) represent a variety of methods depending upon the method used to approximate x^{n+1}.

Equation (5) may be written as:

$$F'(x^n)x^{n+1} = F'(x^n)x^n - F(x^n) \tag{8}$$

or

$$Cy = q \tag{9}$$

where $C = F'(x^n)$, $y = x^{n+1}$, and $q = F'(x^n)x^n - F(x^n)$. Let y in (9) be approximated by taking M inner iterations of

$$y^{m+1} - y^m = -Q^{-1}(Cy^m - q), \qquad m = 1, 2, \cdots, M \tag{10}$$

where $y' = x^n$, $y^{M+1} = x^{n+1}$ and Q is the splitting matrix of $C = F'(x^n)$ for one of the linear iterative methods such as line SOR, SIP, point Jacobi (pt. J.), etc. Then the composite method represented by (8)-(10) is called Newton-line SOR, Newton-SIP, Newton-point Jacobi, etc. When (7) is used instead of (5) the methods are called Picard-line SOR, Picard-SIP, etc. Ortega and Rheinboldt [7] have preceded each of these names with "M-step."

4.2. Type 2. The second type consists of generalizations (Ortega and Rheinboldt [7]) of certain of the linear iterative methods used for solving $Ax = b$, where A is a constant matrix. Each of these generalized linear methods reduces to its respective linear parent method when $A(x)$ reduces to a constant matrix. The basic structure for some of these linear iterative methods, several of which are mentioned above for the approximation of y in (9), is given by

$$x^{n+1} - x^n = -Q^{-1}(Ax^n - b) \tag{11}$$

or equivalently by

$$Qx^{n+1} + (A - Q)x^n = b. \tag{12}$$

The generalization of the point SOR method, for example, is:

$$\sum_j Q_{ij}(\tilde{x})x_j^{n+1} + \sum_j [A(\tilde{x}) - Q(\tilde{x})]_{ij}x_j^n = b_i, \qquad i = 1, 2, \cdots, N_g \tag{13}$$

where $(\tilde{x})$ denotes $(x_1^{n+1}, x_2^{n+1}, \cdots, x_{i-1}^{n+1}, x_i^{n+1}, x_{i+1}^n, \cdots, x_{N_g}^n)$ and $Q = L + \omega^{-1}D$. Note that when $A(x)$, and consequently $Q(x)$, reduce to constant matrices, that (13) reduce to (12). Let x_i^{n+1}, as determined by the ith member of (13), $G_i(x_i^{n+1}) = 0$, be approximated by M iterations of the Newton method for a single variable:

$$y^{m+1} - y^m = -[G_i'(y^m)]G_i(y^m), \qquad m = 1, 2, \cdots, M \tag{14}$$

where $G_i' = \partial G_i(x_i^{n+1})/\partial x_i^{n+1}$, y' is taken to be x_i^n, and x_i^{n+1} is set equal to y^{M+1}. The composite method as represented by (13) and (14) is called point SOR-Newton.

For a given value of n, note that the elements of the matrices Q and A are updated N times as the quantities x_i^{n+1} are determined and become available for the replacement of x_i^n, $i = 1, 2, \cdots, N_g$. Equation (14) is only one of several ways to approximate x_i^{n+1} from the ith member of (13). Another method is simply to use $(\tilde{x}) = (x_1^{n+1}, x_2^{n+1}, \cdots, x_{i-1}^{n+1}, x_i^n, x_{N_g}^n)$ in (13). This simple method of determining the quantities

x_i^{n+1} for updating Q and A is very likely less accurate than using (14), but it also requires less computation.

5. Methods used. Seventeen different methods were compared and applied to the solution of the test problems. Type 1 methods used were:

(1) Newton-[ ], where [ ] denotes line SOR or SIP with M in (10) equal to 1 or 5.

(2) Picard-[ ], where [ ] denotes line SOR or SIP with M in (10) equal to 1 or 5.

(3) Picard-[ ], where [ ] denotes one of the 8 preconditioned conjugate gradient methods: ICCG (0), ICCG (1), ICCG (2), RFCG, pt. J. CG, blk. J. CG, SFPCG, or SIPCG.

Since the preconditioned conjugate gradient method applies only to the inversion of symmetric positive definite matrices, it cannot be used with the Newton method because, as mentioned before, matrix F in (5) is not symmetric. It can, however, be used with the Picard method, (7), because A is a positive definite symmetric matrix.

The only type 2 method used was generalized line SOR, with the simple method for updating Q and A described previously. More precise methods for updating were not used because of the belief that they would only lead to more computation per iteration n, and that the convergence rate as a function of n would not be improved. This belief is supported by the fact that, for the test problems chosen, generalized line SOR, with updating only once per iteration n with $\tilde{x} = x^n$ in (13), which is equivalent to Picard-line SOR with $M = 1$, converged just as fast as generalized line SOR with the simple method of updating.

Generalized Jacobi and point SOR were not used because their respective linear parent methods, for solving $Ax = b$ where A is a constant matrix, are usually slower for solving confined aquifer groundwater flow equations than linear line SOR. Generalized SIP, incomplete Cholesky, and others having $Q = LU$, were not used because of their problem with updating.

Generalized preconditioned conjugate gradient was not attempted due to the difficulty of knowing how to correctly formulate such a method. Updating Q and A only once per iteration gives a method which is equivalent to Picard-preconditioned conjugate gradient with $M = 1$. This method, however, was found to be a total failure, suggesting perhaps that more sophisticated forms of updating would also perform poorly.

5.1. The Picard-preconditioned conjugate gradient methods. As mentioned above, eight of the type 1 methods used were Picard-[ ], where [ ] denotes one of the eight preconditioned conjugate gradient methods: ICCG (0), ICCG (1), ICCG (2), RFCG, pt. J. CG, blk. J. CG, SFPCG, or SIPCG. These conjugate gradient methods for solving $Ax = b$ are defined by giving the preconditioning or splitting matrices Q (Van der Vorst [11]; Hageman and Young [4]) associated with them. For RFCG (Hageman and Young [4]), $Q = I$, the identity matrix. For pt, J. CG, $Q_{ij} = \delta_{ij} A_{ij}$. For blk. J. CG, $Q_{ij} = A_{ij}$ when $|i-j| \leqq 1$, but zero otherwise. For the three incomplete Cholesky conjugate gradient methods ICCG (0), ICCG (1), and ICCG (2), $Q = LDL^T$ (Kershaw [5]), where the lower triangular matrix L and diagonal matrix D are related by $D_{ii} = L_{ii}^{-1}$. For SFPCG, $Q = LU$, where L and U are obtained by means of the symmetric factorization procedure described by Bracha-Barak and Saylor [1]. The matrix L is lower triangular and the matrix U is upper triangular.

For SIPCG, the strongly implicit procedure (SIP) is used to obtain Q. SIP, which is very similar to the symmetric factorization procedure described above, uses $Q = LU$ (Stone [12]), where the matrix L is lower triangular and the matrix U is

upper triangular and has the value unity on the center diagonal. Both L and U have three off-center diagonals. Although $Q = LU$ approximates the matrix A, Q is not symmetric positive definite and thus cannot be used as a preconditioning or splitting matrix for the preconditioned conjugate gradient method. To overcome this impasse, one can use a symmetric matrix $Q' \cong LU$ in place of Q. A possible choice for Q' is $Q' = LDL^T$, where $D_{ii} = L_{ii}^{-1}$. A second choice for Q' is $Q' = lDl^T$, where $Dl^T = U$, where $D_{ii} = L_{ii}^{-1}$. The first choice simply takes the needed three off-center diagonals from L, the second choice, in effect, takes them from U. Both choices use the center diagonal from L. Numerical experiments using weighted averages of these two choices for Q show that the second choice is better than the first with regard to the number of iterations required to obtain a solution. In this work, the second choice for Q' is used.

6. Test problems. The test problems used were chosen to be inclusive with regard to the various common confined and watertable groundwater flow situations that occur in practice.

Test problem 1 had the following: $i = 1, 10$, $j = 1, 10$, and $k = 1, 5$ for a total of $N = 500$ grid elements. Aquifer 1 has a thickness Δz of 500 and a top elevation of zero. Hydraulic head x was fixed at zero for the node points $i = 1$, $j = 1, 10$, and $k = 5$. Test problem 2 was similar to test problem 1, but had $i = 1, 5$, $j = 1, 10$, and $k = 1, 7$ for a total of 350 grid elements. Test problem 3 was also similar to test problem 1 but was two-dimensional, consisting of the top layer of test problem 1. Test problem 4 was the same as test problem 1, but linear. The value for Δz in aquifer 1, 500, was fixed and not decreased according to $\Delta z_i = h_i - v_i$. Test problem 5 was similar to test problem 1 except that it had $i = 1, 35$, $j = 1, 20$, and $k = 1, 7$ for a total of 4900 grid elements. As with test problem 4, values for Δz in aquifer 1 were fixed so that the problem was linear.

Other test problems were modifications of test problem 1. These modifications consisted of:

(1) Increasing the size of the wells.

(2) Removing certain of the grid elements from the region of solution, so that the solution region was irregularly shaped instead of rectangular.

(3) Varying the hydraulic conductivity K_c, of the confining bed beneath the top aquifer thereby causing h in the top aquifer to vary.

(4) Changing the location and value of the fixed head values.

(5) Introducing isotropy into (1).

(6) Setting $\Delta z_i = h_i - v_i$ in each aquifer instead of only the top aquifer, thereby increasing the nonlinearity of the problem.

7. Results.

7.1. Test problem 1. Table 1 shows the results for the 17 solution methods applied to test problem 1.

The numbers 1 and 5 in the parenthesis by SOR and SIP denote the number of steps M in (10). Newton line SOR (5), for example, has $m = 1, 2, \cdots, M$, with $M = 5$. For the 8 Picard-preconditioned conjugate gradient methods, $M(n)$ was selected to minimize the total number of iterations needed to arrive at solution. To this end and on the basis of several experiments using different procedures for selecting $M(n)$ values, iteration in m was discontinued when $\max(i)|(Cy^m - q)_i|$, from (9) became less than a value $r(n)$, where the value of $\ln[r(n)]$ decreased linearly with n. This linear relationship between $\ln[r(n)]$ and n was adjusted to minimize the total number of iterations. Using this procedure, the values selected for $r(1)$ and $r(10)$ were 50 and 10, respectively.

TABLE 1
Comparison of numerical methods, test problem 1.

Method	Total iterations	$\max(i)\lvert(A(x^N)x^N-b)_i\rvert$	$\max(i)\lvert(x_i^{N+1}-x_i^N)\rvert$	$\frac{1}{N_g}\sum_{i=1}^{N_g} x_i^{N+1}$	$\max(i)\lvert x_i^{N+1}-x_i\rvert$
Newton-line SOR (1)	200	$.60\times10^{4}$	$.45\times10^{-1}$	−75	
Newton-line SOR (5)	200	$.39\times10^{4}$	1.1	−85	217.
Newton-SIP (1)	200	$.34\times10^{3}$	.55	−150	
Newton-SIP (5)	200	$.66\times10^{3}$	2.3	−151	59.
Picard-line SOR (1)	200	$.60\times10^{4}$	$.45\times10^{-1}$	−75	
Picard-line SOR (5)	200	$.38\times10^{4}$	1.1	−85	217.
Picard-SIP (1)	200	$.34\times10^{3}$	.55	−150	
Picard-SIP (5)	200	$.66\times10^{3}$	2.3	−151	59.
Picard-ICCG (0)	120	17.	$.13\times10^{-2}$	−176	.11
Picard-ICCG (1)	93	25.	.11	−176	$.31\times10^{-3}$
Picard-ICCG (2)	91	24.	.11	−176	$.67\times10^{-3}$
Picard-RFCG	396	12.	$.19\times10^{-2}$	−176	$.48\times10^{-2}$
Picard-Pt. J. CG	303	12.	$.20\times10^{-2}$	−176	$.12\times10^{-1}$
Picard-Blk. J. CG	244	12.	$.20\times10^{-2}$	−176	.12
Picard-SFPCG	105	20.	$.53\times10^{-3}$	−176	.11
Picard-SIPCG	98	20.	$.26\times10^{-3}$	−176	$.79\times10^{-4}$
Gen. line-SOR	200	$.59\times10^{4}$	$.45\times10^{-1}$	−75	

A solution was assumed when the maximum residual component, $\max(i)\lvert(A(x^n)x^n-b)_i\rvert$, became less than 70. This number is much less than the size of the wells b, and is also much less than the size of the flows between most of the grid elements.

Let us, for example, look in some detail at the solution of test problem 1 using the Picard-ICCG (0) method. Table 1 shows that a total of 120 iterations were needed to arrive at solution. The iteration index n in (7) had the values $1, 2, \cdots, N$, where $N=8$. The corresponding values for the number of inner iterations $M(n)$, not shown in Table 1, were 31, 27, 26, 16, 12, 3, 2, and 3, for a total of 120 iterations. From Table 1, the value of $\max(i)\lvert(A(x^N)x^N-b)_i\rvert$was 17, and the value of $\max(i)\lvert(x_i^{N+1}-x_i^N)\rvert$ was $.13\times10^{-2}$. As with all the test problems and methods, x_i^n was given the value 0.0, for $n=1$. The values obtained for $1/N_g\sum_{i=1}^{N_g} x_i^{N+1}$ and $\max(i)\lvert x_i^{N+1}-x_i\rvert$ were −176 and .11, respectively. Here, the x_i denote exact values, which were approximated by a high accuracy numerical solution with $\max(i)\lvert(A(x^N)x^N-b)_i\rvert<2$.

Note from Table 1, that the 8 Newton or Picard-line SOR or SIP methods, as well as the generalized line SOR method, did not arrive at solution within the allowed 200 maximum number of total iterations. Comparing Newton-line SOR or SIP with Picard-line SOR or SIP shows that they performed in an almost identical manner.

7.2. Other test problems. The results for test problem 2 and the test problems mentioned previously as being modifications of test problem 1 were similar to those given in Table 1.

The results for the two-dimensional test problem 3 were somewhat different. All of the methods except those involving line SOR were able to arrive at solution. The total number of iterations required by the 17 methods was 200, 200, 32, 48, 200, 200, 37, 61, 69, 47, 47, 189, 180, 134, 62, 59, and 200. Note the good showing of the Newton or Picard-SIP methods.

The results for the linear test problems 4 and 5 were similar to each other. For linear problems, the 17 nonlinear solution methods degenerate to line SOR, SIP, ICCG (0), ICCG (1), ICCG (2), RFCG, pt. J. CG, blk, J. CG, SFPCG, and SIPCG. For both test problems, line SOR and SIP did not arrive at solution. For test problem 4, with $1/N_g \sum_{i=1}^{N_g} x_i$ approximately equal to -162, $1/N_g \sum_{i=1}^{N_g} x_i^{N+1}$ had the values -68 and -138 for line SOR and SIP, respectively, after a total of 200 iterations. The total number of iterations taken by the 10 linear methods was 200, 200, 30, 21, 20, 120, 83, 68, 26, and 22. For test problem 5, the number of iterations required was 2 to 3 times larger.

The number of computer storage locations per grid element used to implement the line SOR, SIP, ICCG (0), ICCG (1), ICCG (2), RFCG, pt. J. CG, blk. J. CG, SFPCG, and SIPCG methods was 8, 10, 9, 10, 10, 8, 9, 9, 12, and 12, respectively. Storage requirements remained the same when these linear methods were used in one of the 17 nonlinear methods. For the 10 linear methods, the relative amounts of CPU time required per iteration per grid element, were 2.21, 1.0, .741, 1.08, 1.10, .493, .502, .604, .714, and .709, respectively. The relatively long time for line SOR resulted because the method, as used, performed operations in each of the i, j, and k directions for each iteration.

8. Conclusions. On the basis of the amount of computational effort required for a solution, the relative performance of the various methods was approximately the same for each nonlinear test problem. The Picard-preconditioned conjugate gradient methods did the best, followed by Newton or Picard-SIP, followed by Newton or Picard-line SOR and generalized line SOR. The only exception to this rule was the two-dimensional problem, for which Newton or Picard SIP did as well as Picard-preconditioned conjugate gradient. There was very little difference between using Newton versus Picard in connection with line SOR, or SIP. For the linear test problems, the conjugate gradient methods did the best followed by SIP followed by line SOR.

REFERENCES

[1] A. BRACHA-BARAK AND P. E. SAYLOR, *A symmetric factorization procedure for the solution of elliptic boundary value problems*, SIAM J. Numer. Anal., 10 (1973), pp. 190–206.

[2] P. CONCUS, *Numerical solution of the nonlinear magnetostatic field equation in two dimensions*, J. Comp. Phys., 1 (1967), pp. 330–342.

[3] G. GAMBOLATI, *Fast solution to finite element flow equations by Newton iteration and modified conjugate gradient method*, Internat. J. Numer. Methods Engrg., 15 (1980), pp. 661–675.

[4] L. A. HAGEMAN AND D. M. YOUNG, *Applied Iterative Methods*, Academic Press, New York, 1981.

[5] 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.

[6] L. K. KUIPER, *A comparison of the incomplete Cholesky-conjugate gradient method with the strongly implicit method as applied to the solution of two-dimensional groundwater flow equations*, Water Resour. Res., 17 (1981), pp. 1082–1086.

[7] J. M. ORTEGA AND W. C. RHEINBOLDT, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.

[8] J. M. ORTEGA AND M. L. ROCKOFF, *Nonlinear difference equations and Gauss-Seidel type iterative methods*, SIAM J. Numer. Anal., 3 (1966), pp. 28–41.

[9] I. REMSON, G. M. HORNBERGER AND F. J. MOLZ, *Numerical Methods in Subsurface Hydrology*, Interscience, New York, 1971.

[10] P. C. TRESCOTT AND S. P. LARSON, *Comparison of iterative methods of solving two-dimensional groundwater flow equations*, Water Resour. Res., 13 (1977), pp. 125–136.

[11] H. A. VAN DER VORST, *A vectorizable varient of some* ICCG *methods*, this Journal, 3 (1982), pp. 350–356.

[12] H. L. STONE, *Iterative solutions of implicit approximations of multi-dimensional partial differential equations*, SIAM J. Numer. Anal., 5 (1968), pp. 530–538.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 4, July 1987
© 1987 Society for Industrial and Applied Mathematics
005

AN ADAPTIVE FINITE ELEMENT METHOD FOR STEADY AND TRANSIENT PROBLEMS*

R. E. BENNER, JR.†‡, H. T. DAVIS† AND L. E. SCRIVEN†

Abstract. Distributing integral error uniformly over variable subdomains, or finite elements, is an attractive criterion by which to subdivide a domain for the Galerkin/finite element method when localized steep gradients and high curvatures are to be resolved. Examples are fluid interfaces, shock fronts and other internal layers, as well as fluid mechanical and other boundary layers, e.g. thin-film states at solid walls.

The uniform distribution criterion is developed here into an adaptive technique for one-dimensional problems. Nodal positions can be updated simultaneously with nodal values during Newton iteration, but it is usually better to adopt nearly optimal nodal positions during Newton iteration upon nodal values. Three illustrative problems are solved: steady convection with diffusion, gradient theory of fluid wetting on a solid surface and Buckley–Leverett theory of two-phase Darcy flow in porous media. The new adaptive technique resists entanglement of the nodes of the nodal mesh without requiring the special restrictions upon which the earlier moving finite element method relies.

Key words. adaptive elements, variable subdomains, error equidistribution, Newton's method, continuation, convection-diffusion, gradient theory, Buckley–Leverett

AMS(MOS) subject classifications. 65M50, 65M60, 65N30, 65N50

1. Introduction.

Finite element analysis, conceived by the mathematician Courant [16] but invented independently by structural engineers (cf. [3]) and later given a strong mathematical foundation [47], is now a powerful method of analysis for differential, integral (cf. [28], [51]) and integrodifferential (cf. [7], [25]) equations that arise in many areas of science and engineering. In the finite element method, the domain is partitioned or tessellated into subdomains and the solution $y(x, t)$ of an equation

$$L(y, t; \boldsymbol{\alpha}) = 0, \tag{1}$$

with appropriate boundary conditions, is expanded in a set of finite element basis functions, $\{\phi_i\}$:

$$y(x, t) = \sum_{j=1}^{M} q_j(t)\phi_j(x). \tag{2}$$

Here $\boldsymbol{\alpha}$ is the vector of parameters $\alpha_1, \cdots, \alpha_n$ that appear in (1) (and perhaps boundary conditions) and $\{q_j\}$ is the set of unknown coefficients of the basis functions. In this paper functions $y(x, t)$ are computed which vary in one spatial dimension x and which may or may not vary in time t. In [6] the methods presented here are extended to functions which vary in two spatial dimensions.

The basis functions chosen are usually polynomials of low order, e.g., the piecewise linear basis functions shown in Fig. 1. Each of the basis functions is nonzero only in the subdomain of so-called finite elements that surround the jth node of the tessellation

* Received by the editors August 7, 1984; accepted for publication (in revised form) July 18, 1986. This research was supported in part by the Department of Energy, the National Aeronautics and Space Administration Fund for Independent Research, and the University of Minnesota Computer Center, Minneapolis, Minnesota 55455.

† Department of Chemical Engineering and Materials Science and the Supercomputer Institute, University of Minnesota, Minneapolis, Minnesota 55455.

‡ Present address, Sandia National Laboratories, Division 1413, Albuquerque, New Mexico 87185.

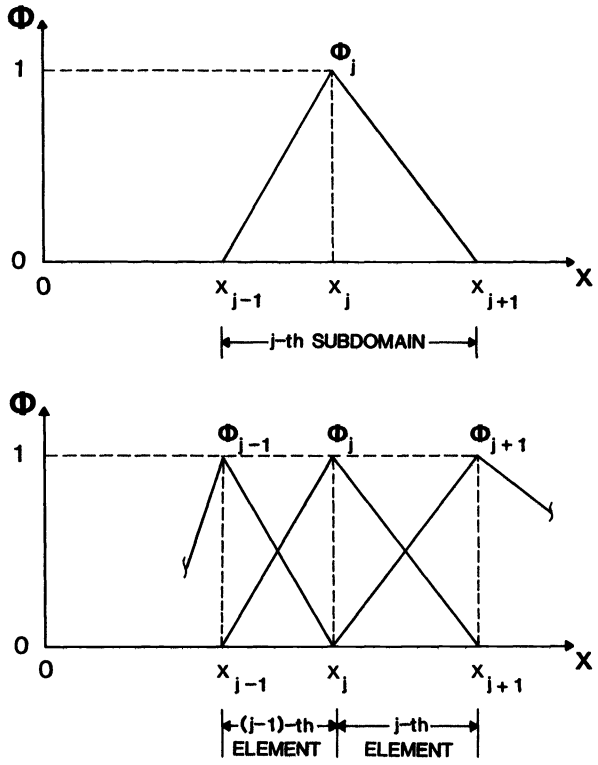


FIG. 1. *The subdomain, nodes and elements associated with a linear finite element basis function.*

of the domain. Elements are the regions in which adjacent subdomains overlap, and nodes are the points at which basis functions (or derivatives of certain Hermite polynomial basis functions) are unity; cf. Fig. 1.

In traditional finite element analysis, the domain tessellation remains fixed as a parameter is incremented or a time step is taken. Thus the nodes, basis functions, subdomains and elements remain fixed during the computations. In contrast, an adaptive method moves or deforms the finite element tessellation at each time-step or set of parameter values, so that localized steep gradients and high curvatures can be resolved as they develop. Relevant computer-aided analysis on a fixed tessellation is outlined in § 1.1. Adaptive finite element methods known to the authors are reviewed in § 1.2. The main content of this paper is outlined in § 1.3.

1.1. Galerkin/finite element analysis on a fixed tessellation. Galerkin weighted residuals of (1) with respect to the basis functions are

$$R_i(\mathbf{q}, t; \boldsymbol{\alpha}) = \int_\Omega L(y, t; \boldsymbol{\alpha})\phi_j(x)\, dx = 0, \qquad i = 1, \cdots, M. \tag{3}$$

$\mathbf{q} = [q_1, \cdots, q_M]^T$ is the vector of unknown coefficients found by solving (3) and Ω is the computational domain.

The discretized equations (3) are, in general, nonlinear and must be solved by iteration. Newton's is the method of choice (for several reasons):

$$\sum_{j=1}^{M} \frac{\partial R_i}{\partial q_j}(\mathbf{q}^{(k)})\Delta q_j^{(k)} = -R_i(\mathbf{q}^{(k)}), \qquad i = 1, \cdots, M. \tag{4}$$

Here $\mathbf{q}^{(k)}$ is the kth iterate and $\Delta\mathbf{q}^{(k)} = \mathbf{q}^{(k+1)} - \mathbf{q}^{(k)}$ the kth update of $\mathbf{q}$ at fixed α or t, or both. The Newton process (4) approaches quadratic convergence near a solution so long as the Jacobian $[\partial R_i/\partial q_j]$ is nonsingular, i.e., its determinant is nonzero. The integrals in (3) and (4) are evaluated by the isoparametric transformation of the subdomain of ϕ_i onto a subdomain of unit size (cf. [35], [47]), followed by Gaussian quadrature (cf. [47]).

Initial estimates $\mathbf{q}^{(0)}$ of new solutions $\mathbf{q}(\alpha_1, \cdots, \alpha_l + \Delta\alpha_l, \cdots)$ to steady problems can be predicted by first-order continuation [32], [44] in the increment $\Delta\alpha_l$,

$$\mathbf{q}^{(0)}(\alpha_1, \cdots, \alpha_l + \Delta\alpha_l, \cdots) = \mathbf{q}^{(k)}(\boldsymbol{\alpha}) + \frac{\partial \mathbf{q}}{\partial \alpha_l}\Delta\alpha_l,$$

(5)

$$\sum_{j=1}^{M} \frac{\partial R_i}{\partial q_j}(\mathbf{q}^{(k)})\frac{\partial q_j}{\partial \alpha_l} = -\frac{\partial R_i}{\partial \alpha_l}(\mathbf{q}^{(k)}).$$

Solution of (5) requires the converged Jacobian of the Newton process (4) in the calculation of sensitivity coefficients, $\partial q_j/\partial\alpha_l$.

Singularity of the Jacobian can correspond to a turning point or bifurcation point. At a bifurcation point two or more branches of solutions meet. As a solution branch is followed by continuation in increasing (decreasing) value of α_l, a turning point is encountered when the branch turns back towards decreasing (increasing) values of α_l. An efficient computer-aided method for continuing past a turning point [1] is used in § 4 and is described in greater detail elsewhere [6]. Computer-aided methods for converging onto a new solution branch at a bifurcation are also discussed elsewhere [49].

1.2. Adaptive finite element methods. The goal of an adaptive finite element method is an optimal or nearly optimal set of nodal positions $\{x_i\}$ at each time-step or set of parameter values, or both. Optimal means that solution error is, in some sense, minimized. Minimal error is also sought by other numerical methods, e.g., grid-adaptive finite difference methods (cf. [46]), knot-adaptive spline methods (cf. [30]), adaptive boundary elements [52] and isotherm migration methods [17].

Many methods of optimal node placement have been proposed. Most are designed solely for transient problems, as discussed below, but great value attaches to optimal solutions of steady state problems that feature boundary or internal layers. In addition, both steady and transient problems with one or more free boundaries require a deforming tessellation.

Deformation or movement of at least some subdomains is unavoidable when the Navier–Stokes system is solved for free surface flows such as that of liquid in a slot coater [38] as a parameter is incremented or time is advanced. The positions of nodes in the flow domain beneath the free surface must be changed if accuracy of the solution is to be maintained. The simplest strategy is to allow nodes to move in only one dimension, as for example by making each remain on a spine that intersects the free surface. Spines are fixed curves which spring from a backbone and form the skeleton of the tessellation [35]. Interior nodes move along the spines in preset proportions to the relocation of the free surface. This strategy has been applied to both steady [37], [38] and transient [10] free surface flow problems. A more flexible strategy is to allow the net of nodes to stretch as an elastic medium [36].

Movement of nodes along spines in proportion to free surface relocation is adequate for many problems, an example being finite element analysis of liquid drop oscillations [7]. However, more powerful adaptive techniques are needed for hyperbolic

or parabolic transient problems; the need has long been recognized. Adaptive techniques for transient problems include front-tracking with selective refinement near the front [26]; so-called rolling-travelling element methods [39] which move nodes along paths of constant strain or other physical property; and space-time finite element methods [10]. The space-time method, in which both spatial and temporal dimensions are represented in the finite element basis, is a framework for adaptive criteria such as shock-tracking [14], [50] and equidistribution of solution error [21]. Special discontinuous finite elements are being used in the space-time element method [50] and elsewhere.

The moving finite element method [24] was developed by a "variational approach," the optimal nodal set giving an extremum in what is sometimes called "energy," i.e., some functional to be minimized [47] with respect to nodal movement. For self-adjoint problems the variational formulation is equivalent to a Galerkin formulation with nodal movement weighting functions that are derived from the finite element basis. Of course, problems that are not self-adjoint must be analyzed by the Galerkin approach. Also, nonlinear problems may have many extrema or none. Bifurcation phenomena that arise from multiplicity of optimal nodal sets include nodal entanglement [24], i.e., $x_{i+1} < x_i$ and negative nodal spacing $\Delta x_i \equiv x_{i+1} - x_i$, and loss of the asymptotic, quadratic rate of convergence of Newton's method. Special restrictions upon nodal movement are required by the moving finite element method to prevent nodal entanglement. These restrictions become more cumbersome the higher the dimension of the problem [2].

A class of adaptive methods for steady-state problems in two dimensions uses refinement, i.e., selected elements are each replaced by two or more elements (additional nodes are introduced) and the problem is solved again on the new tessellation. Such adaptive methods [4], [13], [29], [43] require refinement until the estimate of local error on each element is less than a preset tolerance. This strategy has merit for multidimensional problems, but the software required to solve the problem on increasingly complex tessellations is a serious drawback.

A promising developent for analysis of certain one-dimensional problems is the hybrid asymptotic-finite element method [15]. This employs a composite coordinate system, viz., a function $y(x)$ is represented in part of the domain by a set of finite element basis functions on a tessellation in x and in the rest of the domain by asymptotic basis functions on a tessellation in y. The asymptotic basis functions are constructed from an asymptotic solution of the equation for the boundary layer or other feature of the solution that is difficult to resolve accurately with a polynomial finite element basis.

The most direct path to reduction of error in a solution computed by an adaptive method is provided by methods, such as adaptive refinement methods, which use an error estimate to decide where basis functions are most needed.

1.3. The new method. Here an error bound is used to adapt a finite element basis as a parameter or time is varied. The same error bound was used by Davis and Flaherty [21]. However, their independently developed method is solely for transient equations, uses finite element representations in space and in time and solves for the ratio $\Delta x_{i+1}/\Delta x_i$ of nodal spacings, rather than solving for $\{x_i\}$ directly, as is done here. The error bound and two ways of distributing error uniformly among the elements are discussed in § 2. In both a tessellation or node placement is sought that distributes the integral interpolation error in the first derivative of a solution uniformly or nearly uniformly among the elements. The piecewise linear basis functions of Fig. 1 are used exclusively here

because of their simplicity. But the error bound of § 2 is readily applied to basis functions of higher order, as are needed if a differential equation of order higher than second must be solved.

To illustrate the new method three problems in one spatial dimension are solved. (1) Steady convection with diffusion (§ 3) features an exponential boundary layer. (2) Gradient theory of a Peng-Robinson fluid between parallel walls (§ 4) features thin-film states, fluid interfaces, and layered structures that require highly nonuniform discretization for accurate determination of stability and bifurcations. A discussion of nodal bifurcations follows in § 5. (3) Buckley-Leverett theory of two-phase Darcy flow in porous media (§ 6) features the development of a shock front in time. The examples show that both versions of the method presented here require no recomputation of solutions and resist nodal entanglement. Section 7 is a summary of directions for future research, including adaptive solution of integral equations. Adaptive finite element analysis in two spatial dimensions is discussed elsewhere [9].

2. Theory of adaptive finite element analysis. The adaptive techniques developed in this section are based on an error bound that is independent of the equation (1) to be solved. Strang and Fix [47] present an error bound for the finite element representation (2) of $y(x)$ with polynomials $\phi_i(x)$ of order $m-1$:

$$e_i^{(k)} \leqq e\Delta x_i^{m-k} \int_{x_i}^{x_{i+1}} \left| \frac{d^m y}{dx^m} \right| dx, \qquad i = 1, \cdots, M-1. \tag{6}$$

Local error $e_i^{(k)}$, defined by

$$e_i^{(k)} \equiv \int_{x_i}^{x_{i+1}} \left| \frac{d^k y^*}{dx^k} - \sum_{j=1}^{M} q_j \frac{d^k \phi_j}{dx^k} \right| dx, \tag{7}$$

is the interpolation error in the kth derivative of $y(x)$. Here $y^*(x)$ denotes the exact solution of (1), i.e., the solution obtained as the largest of the nodal spacings Δx_i approaches zero. In (6) and (7) x_i is the position of the ith node (Fig. 1) and e is a constant for a given domain and order m of basis functions. The norm $|\cdot|$ chosen here is the norm of absolute value, the L_1 norm [40], because the integral in (6) is then easily evaluated; this point is discussed below. Equation (6) is the error bound of a polynomial interpolant, i.e., $y(x_i)$ is exact. In practice $y(x_i)$ is an approximation to $y^*(x_i)$ in finite element analysis [47]. That (6) is a valid approximation when (2) represents the solution of a differential equation is demonstrated in § 3. A bound on local error is suspect when (1) is an integral equation [18]; an investigation is needed into the local and global nature of the discretization error in analysis of different classes of integral equations.

Local error can be reduced by equidistribution of error among the elements [40]. Equidistribution of solution error, the case of $k=0$ in (6), is used in spline analysis [40] and in the space-time finite element method [21]. Equidistribution of error in the first derivative of the solution, i.e., $k=1$ in (6), is recommended by Strang and Fix [47] in order to minimize error in an "energy" norm, i.e., for self-adjoint problems. Here Strang and Fix's criterion is used, although it is applied to a more general class of problems than they treat. As is seen in (6), the error bounds when $k=0$ and $k=1$ differ only by a factor of the nodal spacing Δx_i.

In this paper a basis of piecewise linear functions (Fig. 1) is used, for which $m=2$ and

$$e_i^{(1)} \leqq e\Delta x_i \int_{x_i}^{x_{i+1}} \left| \frac{d^2 y}{dx^2} \right| dx, \qquad i = 1, \cdots, M-1. \tag{8}$$

A set of equidistribution residuals $\mathbf{R}^*$ of (8) is defined:

$$R_i^* = e_0 - \Delta x_i \int_{x_i}^{x_{i+1}} \left| \frac{d^2y}{dx^2} \right| dx = 0, \qquad i = 1, \cdots, M-1. \tag{9}$$

Equations (9) determine the position of interior nodes $x_2, \cdots, x_{M-1}$ and the constant e_0. The equidistribution residuals (9) are sensitive to oscillatory discretization error, i.e., change of sign of d^2y/dx^2 from element to element. Consequently, they may provide an ineffective adaptive criteria if such errors are allowed to become significant (as they can in time integration of the Buckley-Leverett equation by the trapezoid rule; see § 6), because the algorithm developed below attempts to resolve numerical oscillations as well as physical phenomena like shocks or interior layers and boundary layers.

With the L_1 norm and ϕ_i piecewise linear, the integral in (9) is that of the product of a unit square wave on the ith element and delta functions at the nodes. The delta functions have weights equal to the discontinuity in slope of $y(x)$ at the nodes; compare the discussion of "δ-mollified functions" by Gelinas et al. [24]. The integral of this product gives

$$R_i^* = e_0 - \Delta x_i \{ |m_{i+1} - m_i| + |m_i - m_{i-1}| \} = 0, \qquad i = 1, \cdots, M-1. \tag{10}$$

Here $m_i \equiv \Delta q_i / \Delta x_i$ is the slope of $y(x)$ on the ith element, and a factor of $\frac{1}{2}$ in the second term has been absorbed by e_0.

Evaluation of (10) on the elements $i = 1$ and $i = M-1$ adjacent to the boundaries requires slopes m_0 and m_M outside the domain. When available, boundary conditions of the second or third kind provide these slopes, e.g., $m_0 = c$ for a Neumann (second kind) condition $dy/dx(x_1) = c$ and $m_0 = cy(x_1) + d$ for a Robin (third kind) condition, where c and d are constants. Where a Dirichlet (first kind) condition is imposed, (10) is evaluated with only the slope discontinuity at the interior node:

$$\begin{aligned} R_1^* &= e_0 - 2\Delta x_1 |m_2 - m_1| = 0, \\ R_{M-1}^* &= e_0 - 2\Delta x_{M-1} |m_{M-1} - m_{M-2}| = 0. \end{aligned} \tag{11}$$

Adaptive finite element residuals (10) are independent of the physical problem of interest, except when m_0 or m_M are set by Neumann or Robin boundary conditions. Two strategies, here called Newton-adaptive and continuation-adaptive, for solving (3) and (10) are now described.

The Newton-adaptive strategy solves (3) and (10) simultaneously for coefficients $\mathbf{q}$, optimal node locations $\mathbf{x}$ and error constant e_0 during Newton iteration. This strategy also predicts estimates of $\mathbf{q}$, $\mathbf{x}$ and e_0 simultaneously in parameter continuation or in the predictor step of time integration. Both Newton's method

$$\begin{bmatrix} \dfrac{\partial R_i}{\partial q_j} & \dfrac{\partial R_i}{\partial x_j} & 0 \\ \dfrac{\partial R_i^*}{\partial q_j} & \dfrac{\partial R_i^*}{\partial x_j} & 1 \end{bmatrix} \begin{bmatrix} \Delta q_j \\ \Delta x_j \\ e_0 \end{bmatrix} = \begin{bmatrix} R_i \\ R_j^* \end{bmatrix} \tag{12}$$

and continuation require the Jacobian entries for (10):

$$\frac{\partial R_i^*}{\partial q_{i-1}} = \operatorname{sgn}(\Delta m_{i-1}) \frac{\Delta x_i}{\Delta x_{i-1}}, \qquad \frac{\partial R_i^*}{\partial q_{i+2}} = \operatorname{sgn}(\Delta m_i) \frac{\Delta x_i}{\Delta x_{i+1}},$$

$$\frac{\partial R_i^*}{\partial q_i} = \operatorname{sgn}(\Delta m_i) - \operatorname{sgn}(\Delta m_{i-1}) \left(1 + \frac{\Delta x_i}{\Delta x_{i+1}} \right),$$

$$
\begin{aligned}
&\frac{\partial R_i^*}{\partial q_{i+1}} = \operatorname{sgn}(\Delta m_{i-1}) - \operatorname{sgn}(\Delta m_i)\left(1+\frac{\Delta x_i}{\Delta x_{i-1}}\right),\\
(13)\qquad &\frac{\partial R_i^*}{\partial x_{i-1}} = -\operatorname{sgn}(\Delta m_{i-1})\frac{\Delta x_i}{\Delta x_{i-1}} m_{i-1}, \qquad \frac{\partial R_i^*}{\partial x_{i+2}} = -\operatorname{sgn}(\Delta m_i)\frac{\Delta x_i}{\Delta x_{i+1}} m_{i+1},\\
&\frac{\partial R_i^*}{\partial x_i} = |\Delta m_{i-1}| + |\Delta m_i| + \operatorname{sgn}(\Delta m_{i-1})\left(\frac{\Delta x_i}{\Delta x_{i-1}} m_{i-1} + m_i\right) - \operatorname{sgn}(\Delta m_i) m_i,\\
&\frac{\partial R_i^*}{\partial x_{i+1}} = -|\Delta m_{i-1}| - |\Delta m_i| + \operatorname{sgn}(\Delta m_i)\left(\frac{\Delta x_i}{\Delta x_{i+1}} m_{i+1} + m_i\right) - \operatorname{sgn}(\Delta m_{i-1}) m_i,\\
&\frac{\partial R_i^*}{\partial e_0} = 1.
\end{aligned}
$$

From these the Jacobian entries follow easily for (11). Here $\Delta m_i \equiv m_{i+1} - m_i$, the slope discontinuity at the ith node. The Newton-adaptive strategy (12) results in a system of linear equations of larger bandwidth (eleven) than the tridiagonal systems solved when the tessellation is fixed. Equation (12) also requires Jacobian entries $\partial R_i/\partial x_j$ which are not needed when the tessellation is fixed.

The continuation-adaptive strategy proves more efficient. It computes nodal positions $\mathbf{x}$ which approximately satisfy (10); these are computed after each continuation step or time step to $\mathbf{q}^{(0)}$, but before Newton iteration in the nodal values $\mathbf{q}^{(k)}$. This strategy preserves tridiagonal systems in the solution of (3) and eliminates nonlinear phenomena that arise from nodal movement (cf. § 5), thereby preserving the asymptotic, quadratic rate of convergence of Newton iteration. Moreover it eliminates the need for a good initial estimate of the optimal nodal set $\mathbf{x}$ in order to start Newton iteration at the first parameter value in a continuation sequence.

The continuation-adaptive strategy begins with a continuation or predictor step in which the nodes are fixed. Local error $e_i^{(1)}$ is then estimated from (9), viz.

$$
(14)\qquad e_i^{(1)} = \Delta x_i\{|m_{i+1} - m_i| + |m_i - m_{i-1}|\}, \qquad i = 1, \cdots, M-1.
$$

Here a factor of $2/e$ is again absorbed into $e_i^{(1)}$. The nodes are positioned so that the linear interpolant of $(e_i^{(1)})^{1/2}$ is distributed uniformly among the elements. The square root weighting of $e_i^{(1)}$ provides smoothing: for example, if $e_k^{(1)}$ is ten times greater than all other $e_i^{(1)}$, element k is replaced by about three elements in the new tessellation rather than by ten. The maximum size of a continuation-adaptive element is held to $0.1(x_M - x_1)$ to prevent oscillations in nodal positions from one continuation or transient prediction step to the next. Other smoothness conditions have been proposed by Kautsky and Nichols [31]: either the global ratio of largest to smallest Δx_i or the local ratio $\Delta x_{i+1}/\Delta x_i$ should be bounded. A local bound on $\Delta x_{i+1}/\Delta x_i$ appears to be adequate for accurate solution of differential equations [31] and is being tested by the authors. A global bound on $\Delta x_k/\Delta x_i$ may be critical in the selection of a net of nodes, or tessellation into elements, for solution of an integral equation, if the discretization error according to analysis of integral equations proves to be global in character which is a matter under investigation.

Once the new tessellation is selected, nodal values are interpolated from the approximation to the solution that was predicted on the old tessellation. The new tessellation remains fixed while Newton iteration proceeds to convergence.

3. Example: Steady convection-diffusion. Convection-diffusion equations are a standard test of innovative methods of analysis such as "upwinded" finite element

methods [5], [13]. Steady convection-diffusion in one-dimension with essential boundary conditions is described by

$$\begin{aligned} &\frac{d^2y}{dx^2} = N_{\mathrm{Pe}}\frac{dy}{dx}, \qquad 0<x<1, \\ &y(0)=0, \qquad\qquad y(1)=1. \end{aligned} \tag{15}$$

Here x is distance in units of the separation of the upstream and downstream boundaries (porous walls), y is concentration in units of outflow concentration at $y=1$, and N_{Pe} is the Peclet number, a measure of the relative importance of convective flux to diffusive flux. As the Peclet number increases, the relative importance of diffusion falls and an outflow boundary layer sharpens. This is evident in the analytic solution of (15),

$$y(x) = \frac{e^{N_{\mathrm{Pe}}x}-1}{e^{N_{\mathrm{Pe}}}-1}, \tag{16}$$

which provides an instructive comparison of the accuracy of fixed, uniform element, Newton-adaptive and continuation-adaptive finite element solutions.

The equations for the vanishing of the Galerkin-weighted residuals of (15) are

$$R_i = \sum_{j=1}^{M} q_j \int_0^1 \frac{d\phi_j}{dx}\left\{\frac{d\phi_i}{dx} + N_{\mathrm{Pe}}\phi_i\right\} dx = 0, \qquad i=2,\cdots,M-1. \tag{17}$$

Integration by parts reduces the second-order term in (15) to first order and the boundary conditions replace $R_1=0$ and $R_M=0$ by $q_1=0$ and $q_M=1$. Equations (17) are linear and are solved directly for $\mathbf{q}$ if either fixed or continuation-adaptive methods are used. However, the continuation-adaptive method uses the parameter sensitivities that are provided by the continuation computation. The Newton-adaptive method consists of Newton iteration and continuation on $\mathbf{q}$ and $\mathbf{x}$. The Jacobian entries and residual sensitivities from (17) required in (4) and (5) are

$$\begin{aligned} &\frac{\partial R_i}{\partial q_j} = \int_0^1 \frac{\partial\phi_j}{\partial x}\left\{\frac{\partial\phi_i}{\partial x} + N_{\mathrm{Pe}}\phi_i\right\} dx, \\ &\frac{\partial R_i}{\partial x_j} = \sum_{k=1}^{M} q_k \int_0^1 \left\{\frac{\partial^2\phi_k}{\partial x\,\partial x_j}\left[\frac{\partial\phi_i}{\partial x} + N_{\mathrm{Pe}}\phi_i\right] + \frac{\partial\phi_k}{\partial x}\left[\frac{\partial^2\phi_i}{\partial x\,\partial x_j} + N_{\mathrm{Pe}}\frac{\partial\phi_i}{\partial x_j}\right]\right\} dx, \\ &\frac{\partial R_i}{\partial N_{\mathrm{Pe}}} = \sum_{k=1}^{M} q_k \int_0^1 \frac{\partial\phi_k}{\partial x}\phi_i\, dx, \qquad i=2,\cdots,M-1. \end{aligned} \tag{18}$$

The first solution was obtained at $N_{\mathrm{Pe}}=1$. Successive solutions were estimated by continuation with $\Delta N_{\mathrm{Pe}}=1$ when either adaptive method was used. Methods of optimizing the increment ΔN_{Pe} are available (cf. [33]) but were not employed. The convergence criterion used with (4) was

$$\left(\sum_{i=1}^{M} \Delta q_i^{(k)2}\right)^{1/2} \leqq 10^{-8}, \tag{19}$$

which guaranteed that further Newton iteration would not alter the converged solution. The criterion (19) guarantees that Newton iteration ends in the region of quadratic convergence and that the solution $\mathbf{q}^{(k)}$ is accurate to within machine round-off on a 14-digit machine such as the Cray 1 or Cyber 205. Figure 2 shows how the Newton iteration process converged when the Newton-adaptive method was used. Four iterations of underrelaxation were required at $N_{\mathrm{Pe}}=1$ before the updated nodal set $\mathbf{x}^{(k)}$

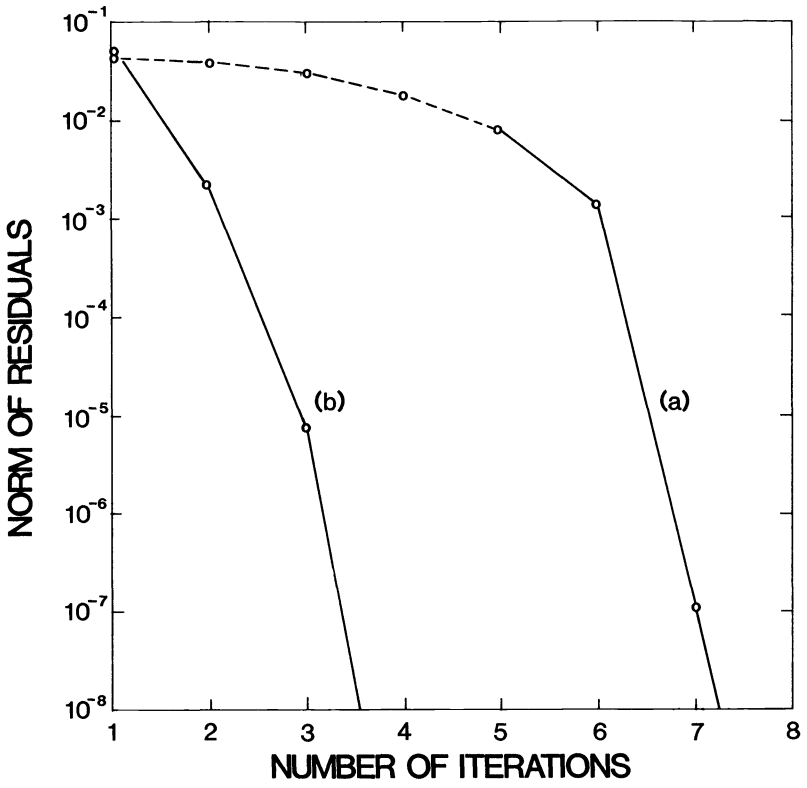


FIG. 2. *Convergence of Newton iteration during Newton-adaptive finite element analysis of convection-diffusion.* (a) $N_{Pe} = 1$ *(first solution).* (b) $N_{Pe} = 2$. (——) *full Newton,* (---) *underrelaxation at first solution. The norm of the residuals is the square root of the sum of the squares.*

fell within the domain of convergence of Newton's method to the optimal nodal set. Thereafter the Newton iteration converged virtually quadratically. Underrelaxation (tracking along the residual curve) was used only as a last resort when there was great difficulty in constructing an initial estimate that fell within the domain of convergence of Newton's method.

Figure 3 shows a sequence of the solutions. As the Peclet number increases, nodes automatically concentrate in the region where the concentration profile is steep and highly curved near the outflow boundary $x = 1$. Local errors (7) in the solution and its first derivative when $N_{Pe} = 10$ are shown in Fig. 4 for uniform fixed and Newton-adaptive elements. The Newton-adaptive method distributed error uniformly among the elements to within 1%, except for the first and last elements, where (11) was imposed. This happened even though the finite element solution was not an exact polynomial interpolant as assumed in (6).

The solution with 10 Newton-adaptive elements proved slightly more accurate than with 20 uniform fixed elements, as shown in Table 1. The total error is the difference between the finite element solution (2) and the analytic solution (16) integrated across the domain $0 < x < 1$. However, the solution with 20 fixed elements required far less computational effort than the Newton-adaptive method with 10 elements. The solutions with continuation-adaptive elements proved nearly as accurate as those with Newton-adaptive elements. The continuation-adaptive method required 50% more central processor (CP) time than the fixed-element method, because the

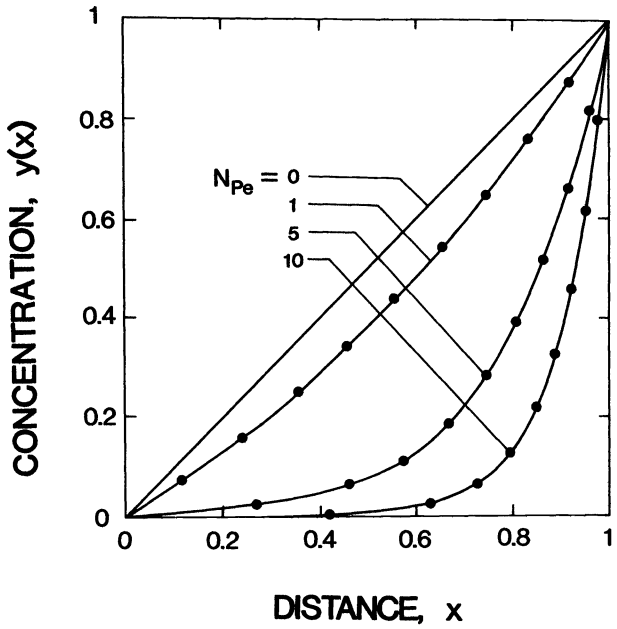


FIG. 3. *Boundary layer sharpens as Peclet number increases.* (●) *node locations given by Newton-adaptive finite element analysis. Ten adaptive elements were used.*

continuation prediction (5) is not needed for linear equations, e.g., (17) with the nodes **x** held fixed. In the continuation-adaptive finite element analysis of nonlinear equations which follows in §§ 4 and 6, the adaptive technique increases computation time about 5% above that required when nodes **x** are held fixed.

4. Example: Gradient theory of wetting. The gradient theory of fluid structure, proposed a century ago [42] and modernized in recent years (cf. [20]), has turned out to be a most revealing tool for theoretical analysis of inhomogeneous equilibrium fluid. Estimations of the surface tension of mixtures [11] and analyses of wetting transitions at a solid surface [48] and of spherical microdrops [22] demonstrated the reliability and practicality of the theory, which has now been applied to various other regimes with the aid of adaptive finite elements. An instructive example is the analysis of the structure of confined fluid multilayers, which is described here.

In dimensionless form the equation of equilibrium and the boundary conditions for one-component inhomogeneous fluid trapped between infinite plane-parallel walls are [8]:

$$\mu_0(n(x)) + u_e(x) - \mu - \frac{d^2n}{dx^2} = 0, \tag{20}$$

$$n(0) = 0, \tag{21}$$

$$\frac{dn}{dx}(x_s) = 0. \tag{22}$$

$n(x)$ is fluid density at distance x from the near wall and x_s is half the distance of the wall separation. Density n is measured in units of b^{-1}, length x in units of molecular length $(c/a)^{1/2}$, temperature T in units of ak/b, and potentials in units of a/b. The quantities a and b are the parameters of the Peng-Robinson equation of state, c is

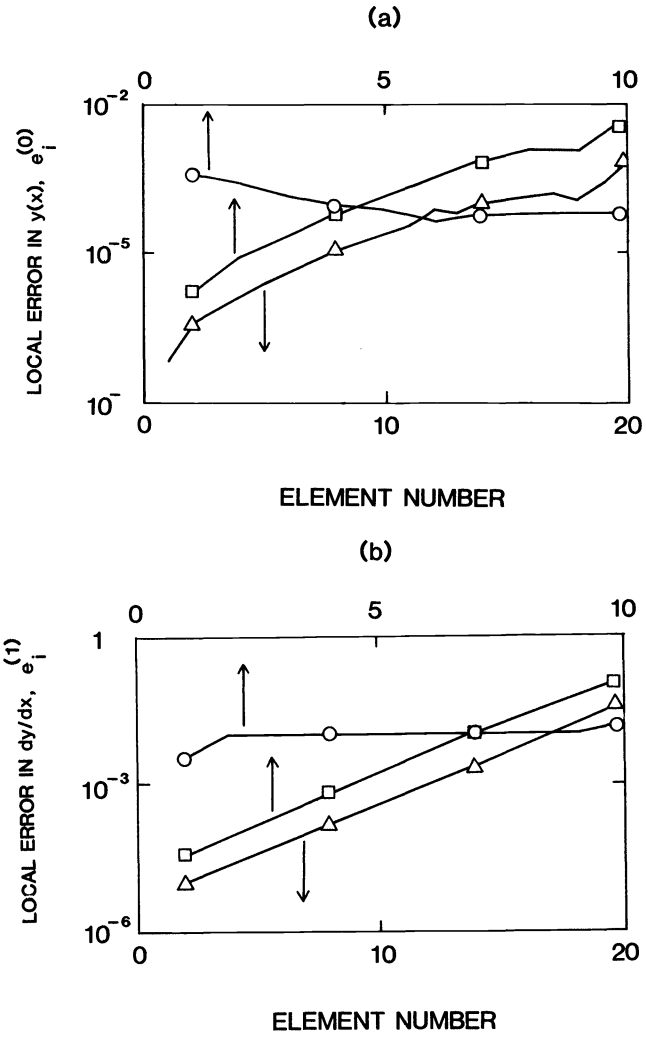


FIG. 4. *Local error as a function of element number i at* $N_{Pe} = 10$. (a) *Error* $e_i^{(0)}$ *in the solution* $y(x)$; (b) *Error* $e_i^{(1)}$ *in the first derivative of the solution,* dy/dx. (□) *ten uniform, fixed elements,* (△) 20 *uniform, fixed elements,* (○) *ten Newton-adaptive elements* (*tessellation shown in Fig.* 3). *Local errors* $e_i^{(k)}$ *estimated by means of equation* (9).

TABLE 1
Comparison of methods: convection-diffusion, $N_{Pe} = 10$.

Nodal strategy	Number of elements	Total error	Cray 1 CP sec
Fixed uniform	10	7.32×10^{-3}	2.0×10^{-2}
Fixed uniform	20	1.72×10^{-3}	2.8×10^{-2}
Continuation-adaptive	10	1.57×10^{-3}	3.1×10^{-2}
Newton-adaptive	10	1.34×10^{-3}	34.0×10^{-2}

the influence parameter of inhomogeneous fluid and k is Boltzmann's constant. μ is the chemical potential and $\mu_0(n(x))$ is the chemical potential of homogeneous fluid of density n, modeled here as a Peng-Robinson fluid. $u_e(x)$ is the potential of fluid-solid interactions, modeled here by a 6-12 Lennard-Jones potential integrated over the infinite solid walls [8].

The Galerkin/weighted residuals of (20) and (22) are required to vanish

$$R_i \equiv \int_{x_1}^{x_s} \left\{ \frac{dn}{dx}\frac{d\phi_i}{dx} + [\mu_0(n) + u_e(x) - \mu]\phi_i \right\} dx = 0, \qquad i = 2, \cdots, M. \tag{23}$$

It is advantageous to replace boundary condition (21) by a residual derived from an asymptotic solution for fluid density near an impenetrable solid [8],

$$R_1 \equiv q_1 - \exp\left\{ \frac{\mu - u_e(x_1)}{T} \right\} = 0 \tag{24}$$

where x_1 is no more than 0.7 molecular lengths from the wall. Equations (23) and (24) are solved under the constraint that N, the number of fluid molecules per unit area of the gap, be fixed:

$$R_{M+1} \equiv N - \int_{x_1}^{x_s} n(x)\, dx = 0. \tag{25}$$

In practice, the continuation parameter is selected adaptively from among $q_1, \cdots, q_M, \mu, N$ and the remaining variables are then solved for. Discussed elsewhere [9] are adaptive selection of continuation parameter, which makes possible efficient tracking of a solution branch in the neighborhood of limit (turning) points [33] and linear analysis of the stability of solutions of (23) and (25) to perturbations at fixed μ or N.

Below the thermodynamic critical temperature T_c, which in the case of the Peng-Robinson fluid equals 0.17 in units of ak/b, three classes of stable (or metastable) fluid states are known [48], namely liquid or vapor in equilibrium with an adsorbed layer, and thick films in equilibrium with vapor. As shown in Fig. 5, each class was resolved accurately with a basis of 40 adaptive elements. The optimal nodal sets are quite different in each case, because the regions of high slope and curvature in the density profiles are different. At $T = 0.14$, the stable states lie on the continuous branch of solutions shown in Fig. 6, portions of which represent unstable intermediate states. At lower temperatures there is still a single branch of solutions, e.g., at $T = 0.1$ (Fig. 7). A progression of unstable, multilayered fluid states was found between the thick film and liquid states. For example, the five-layered states shown in Fig. 8 are periodic far from the confining walls. Unconfined fluid structures that are spatially periodic and unstable are predicted by gradient theory [19]; so their existence in a confined fluid is no surprise. Other features of layered fluid structures are discussed elsewhere [9].

Eighty adaptive elements were used to represent four- and five-layered structures; otherwise forty elements were sufficient. Both Newton- and continuation-adaptive methods were tested. As shown in Table 2, the continuation-adaptive method with 40 elements computed 200 solutions per second on the Cray 1. Execution time for the continuation-adaptive method increased more slowly than the size of the basis because the computation time required to assemble the equations was much greater than the time required to solve the tridiagonal system of equations. Assembly of the equations benefitted fully from vectorization on the Cray 1.

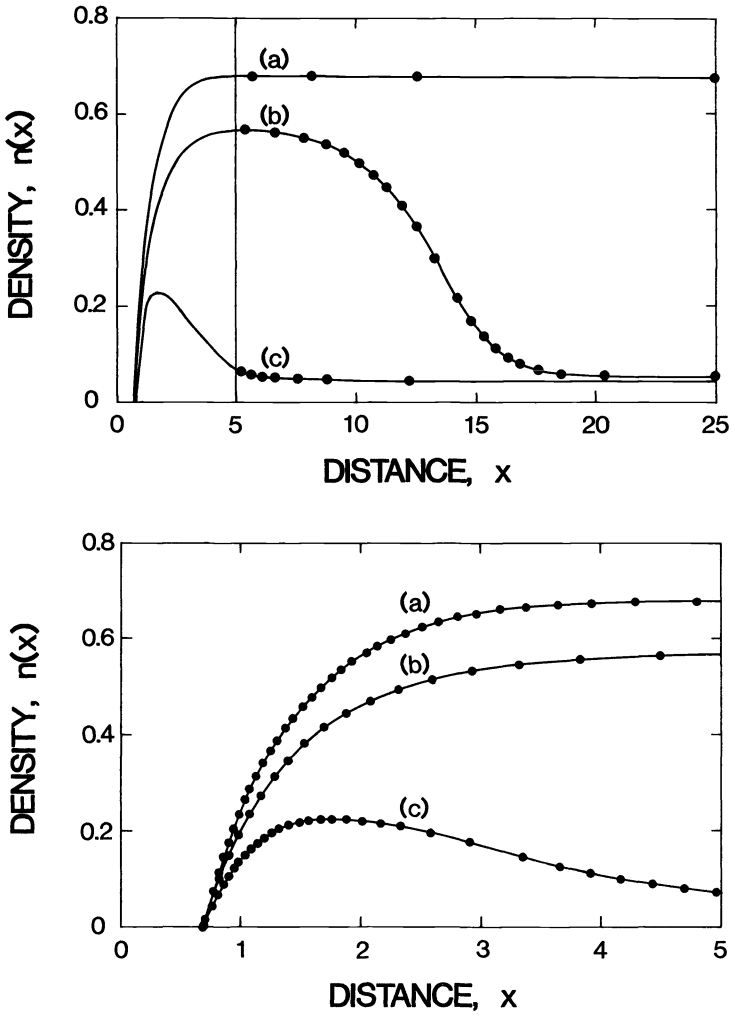


FIG. 5. *Newton-adaptive finite element analysis predicts stable fluid structures at* $T = 0.14$: (a) *liquid*; (b) *vapor in equilibrium with thick film*; (c) *vapor in equilibrium with absorbed layer. Forty continuation-adaptive elements were used.* (●) *node locations.*

The "transition region" in Fig. 7 is where the number of fluid layers changes from odd (five) to even (four) and the curvature of $n(x)$ at $x = x_s$ changes sign. As shown in Fig. 9, although solutions along the transition region can easily be calculated with 160 or fewer elements, they are not as accurately determined in the transition region as elsewhere. At a slightly lower temperature, there is a bifurcation in the transition region: a nearby branch of six-layered fluid structures links to the main solution branch. The three- and four-layered solutions were followed by continuation in increasing temperature [9], and they were discovered to break away from the main branch in the transition region as temperature was increased.

The key to resolving the transition region is simply to raise the number of basis functions. This can be done automatically by adding more basis functions whenever e_0 in (10) exceeds a preset tolerance, as in adaptive refinement methods, or by computing solutions with a small basis and then examining certain ones in greater detail, i.e., by employing a larger basis set, as in multilevel iteration.

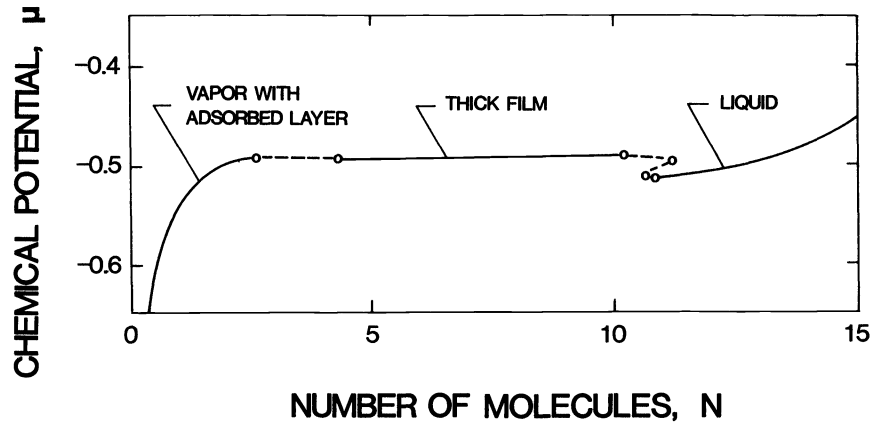


FIG. 6. *Stable and unstable fluid states form a single solution branch at* $T = 0.14$. *Forty continuation-adaptive finite elements were used.* (——) *stable,* (---) unstable, (○) *turning points with respect to* μ *or* N.

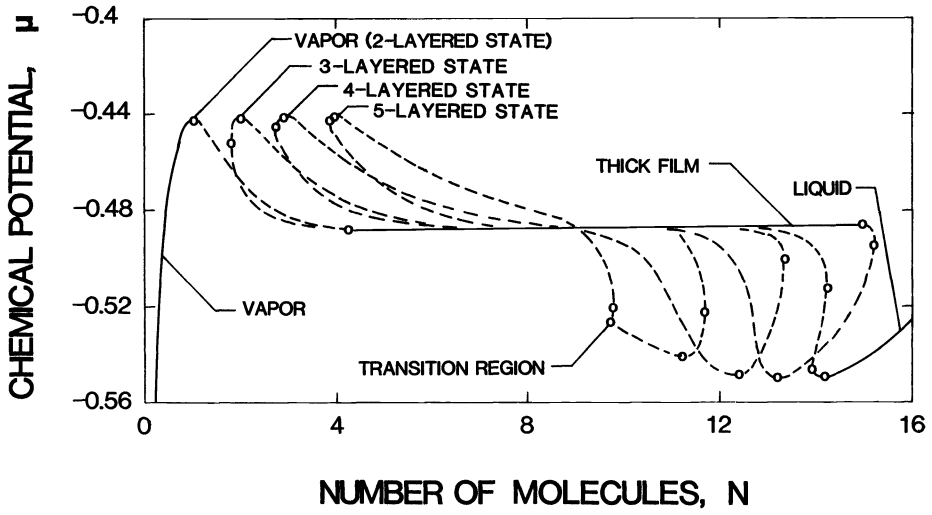


FIG. 7. *Stable and unstable fluid states form a single solution branch at* $T = 0.1$ *that includes unstable layered stuctures.* (——) *stable,* (---) *unstable,* (○) *turning points with respect to* μ *or* N. *Eighty continuation-adaptive elements were used when four- or five-layered structures were sought; otherwise forty adaptive elements were used.*

5. Nodal bifurcation phenomena. Branches of solutions of the equations of vanishing Newton-adaptive residuals (10) also meet at bifurcation points [32]. The solutions of the steady convection-diffusion equations in § 3 have no inflections and no bifurcations were found: Newton's method always converged virtually quadratically. In contrast, the Newton-adaptive finite element analysis of the gradient theory of wetting described in § 4 encountered what are here called nodal bifurcations. It was found that a density profile at fixed N can have several optimal tessellations, or node deployments, on which the inflection points associated with layering are contained in different elements. Nodal bifurcations take place when inflections pass from one element to the next. There are many nodal bifurcations on a solution branch; so the asymptotic, quadratic convergence of Newton's method is often lost. The nodal bifurcations do not lead to entanglement of the nodal set, i.e. $x_{i+1} < x_i$ and $\Delta x_i < 0$, because

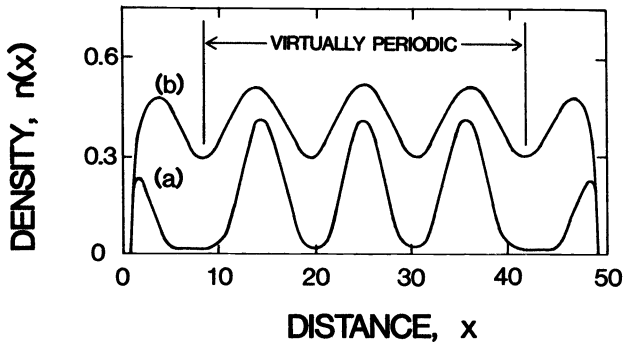


FIG. 8. *Density profiles of five-layered fluid states between walls.* (a) $\mu = 0.433$, $N = 3.84$; (b) $\mu = -0.52$, $N = 11.65$. *Eighty continuation-adaptive elements were used.*

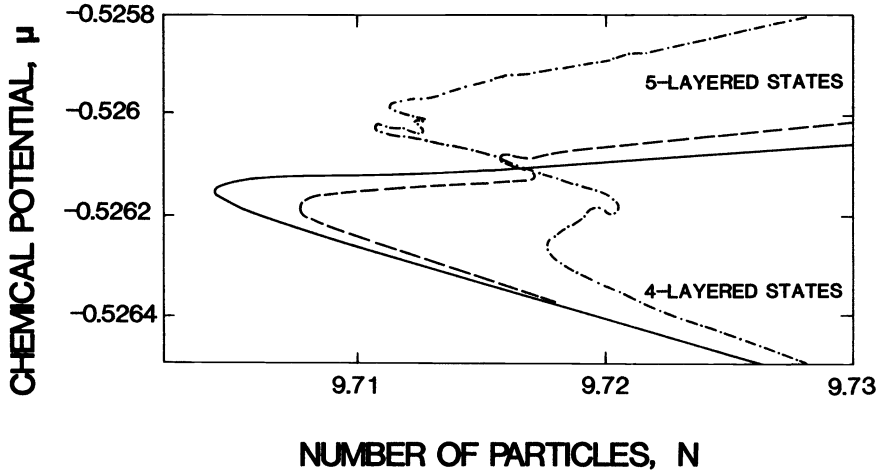


FIG. 9. *Effect of the size of the basis set on accuracy of computed density profiles near a point of bifurcation.* (—·—) 80 *elements,* (---) 160 *elements,* (——), 320 *elements.*

the Δx_i^{m-k} weighting in (6), (9), etc., inherently resists the possibility of negative values of Δx_i.

When new layers develop in a density profile, new inflection points appear. When this happens the Newton-adaptive method encounters a serious problem: imperfect bifurcation. As shown in Fig. 10, the Newton-adaptive method moves nodes and, therefore, basis functions from the flattening region where a new layer is about to appear, and the remaining functions are insufficient to represent the new structure in

TABLE 2
Comparison of methods: gradient theory.

Nodal strategy	Number of elements	Cray 1 CP sec.
Newton-adaptive	40	3.0×10^{-2}
Continuation-adaptive	40	0.5×10^{-2}
Continuation-adaptive	80	0.8×10^{-2}
Continuation-adaptive	160	1.5×10^{-2}
Continuation-adaptive	320	2.5×10^{-2}

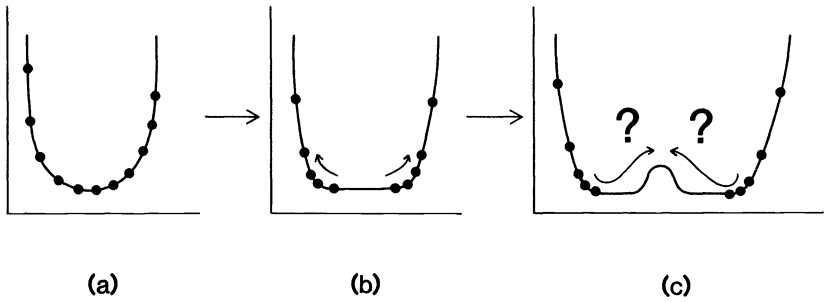


FIG. 10. *Adaptive placement of nodes as new regions of curvature develop in a solution profile*: (a) *Nodal placement when the profile is concave*; (b) *Nodes thinly spread in a region where new structure will develop*; (c) *Can nodes return to represent the new, convex structure in the solution profile*?

the profile. The result is an imperfect bifurcation, i.e., a regime in which there is no optimal nodal set that satisfies (10) and (11), although there are physical solutions that satisfy (23).

The continuation-adaptive method eliminates nodal bifurcations, because (10) is not satisfied exactly; cf. § 2. The local smoothness conditions [31] discussed in § 2 also inhibit nodal evacuation of a region where new structure in a solution profile is developing, because the maximum nodal spacing is restricted.

6. Buckley–Leverett theory. The Buckley–Leverett theory of two-phase Darcy flow in porous media features development of a shock front. Solving the equations of the theory is a challenge to any adaptive method [24] although those equations can be solved roughly by graphical construction [41] and sometimes even by conventional mathematics [23].

In this section the Buckley–Leverett system is solved with a numerical dispersion coefficient D in place of physical dispersion (the modeling of physical dispersion that arises from capillary pressure is an active research topic). For the purpose of comparison the initial and boundary conditions and fractional flow model $f(u)$ used here are those chosen earlier in a study of the moving finite element method [24]; thus the system is

$$\frac{\partial u}{\partial t}+\frac{Q}{P}\frac{\partial f}{\partial x}=D\frac{\partial^2 u}{\partial x^2}, \qquad 0<x<1, \quad t>0,$$

$$u(0,t)=1, \qquad \frac{\partial u}{\partial x}(1,t)=0, \tag{26}$$

$$u(x,0)=\frac{0.1}{0.1+x}, \qquad f(u)=\frac{u^2}{u^2+\alpha(1-u)^2}.$$

$u(x, t)$ is the saturation of the wetting fluid, α the ratio of fluid viscosities, Q the total flow rate and P the porosity of the medium.

The equations of vanishing Galerkin/weighted residuals of (26) are

$$R_i \equiv \int_0^1 \left\{ \left[\sum_{j=1}^{M} \dot{q}_j\phi_j + \sum_{j=2}^{M-1} \dot{x}_j \frac{\partial \phi_j}{\partial x_j} \right] \phi_i + \frac{2\alpha Q u(1-u)}{P(u^2+\alpha(1-u)^2)^2} \frac{du}{dx}\phi_i + D\frac{du}{dx}\frac{\partial \phi_i}{\partial x} \right\} dx = 0, \qquad i=2,\cdots,M-1. \tag{27}$$

Here $u=\sum_{j=1}^{M} q_j\phi_j$ and $\dot{q}_j$ and $\dot{x}_j$ are convective time derivatives of q_j and x_j taken at fixed isoparametric coordinates [27], [34], i.e., with the moving nodal tessellation (cf.

[6]). The adaptive strategies employ first-order backward-differences for $\dot{q}_j$ and $\dot{x}_j$, a time step of 0.005, and a first-order forward-difference prediction. When nodes are held fixed a second-order trapezoid rule for $\dot{q}_j$ is used with an adaptive time step to maintain local time truncation error ε less than 10^{-3}; a second-order Adams–Bashforth predictor and four backward-difference time steps for startup [27], [34] are also used. Oscillations of order ε are inherent in the trapezoid rule [27]; such oscillations interfere with an adaptive method that is sensitive to the curvature of the solution profile. A promising alternative is adaptive finite element analysis with a higher-order backward-difference method that relies on adaptive time steps, e.g., Gear's method.

The results of analysis with 39 Newton-adaptive elements are shown in Fig. 11(a); the parameter values are $Q/P = 1$, $\alpha = 0.5$ and $D = 10^{-3}$. The poor resolution of the saturation front with 39 fixed uniform elements, shown in Fig. 11(b), makes plain the superiority of the adaptive method. The saturation profiles of Fig. 11(a) are indistinguishable from those of moving finite element analysis [24] at the same parameter values. Fifteen adaptive elements eventually rode and resolved the advancing front. The shock was not followed through the outflow boundary with a fixed timestep and the Newton-adaptive method, because the drastic nodal movement required was far outside the domain of convergence of Newton's method. Moreover, outflow boundary condition (28) was violated when the shock left the domain.

The continuation-adaptive method gave saturation profiles that are virtually identical to those of the Newton-adaptive method in Fig. 11(a). The continuation-adaptive strategy is to be preferred, because convergence was difficult with the Newton-adaptive method when the pair of inflections first appeared in the saturation profile at $t \approx 0.065$. Computation time was also reduced: 0.41 and 2.32 Cray 1 central processor seconds were used in 85 time steps with the continuation- and Newton-adaptive methods, respectively.

7. Discussion and summary. The adaptive finite element method based on uniform distribution of integral error is a powerful means of solving the differential equations of one-dimensional equilibria and steady states and holds promise for the partial differential equations of evolution of one-dimensional systems. The continuation-adaptive method requires no recomputation of solutions, unlike adaptive refinement methods; it resists mesh entanglement, unlike the moving finite element method *sans* restrictions on nodal movement [24]; and it requires only a little more computational time (5% in the examples) than computations with a fixed finite element basis. Though inferior the Newton-adaptive method has led to an understanding of bifurcations in nodal positions that are inherent in adaptive finite element analysis of nonlinear equations.

The Newton- and continuation-adaptive methods are readily extended to polynomial basis functions of higher order and to systems of equations, as demonstrated by Davis and Flaherty [21] with the adaptive space-time finite element method based on error equidistribution. The continuation-adaptive strategy has been extended [6] to a simple two-dimensional case: bilinear basis functions on a rectangular tessellation. The nodel coordinates (x_i, y_j) are invariant along the edges of the respective rows and columns of elements that are defined by the tessellation. Error is summed along rows and along columns of elements and x- and y-nodal coordinates are chosen by equidistribution of error among those rows and columns, respectively. The continuation-adaptive method improved the accuracy (relative to results with a fixed tessellation) of the solution of the test problem. It appears likely that adaptive methods will be more effective in two dimensions when they are applied to more flexible and general

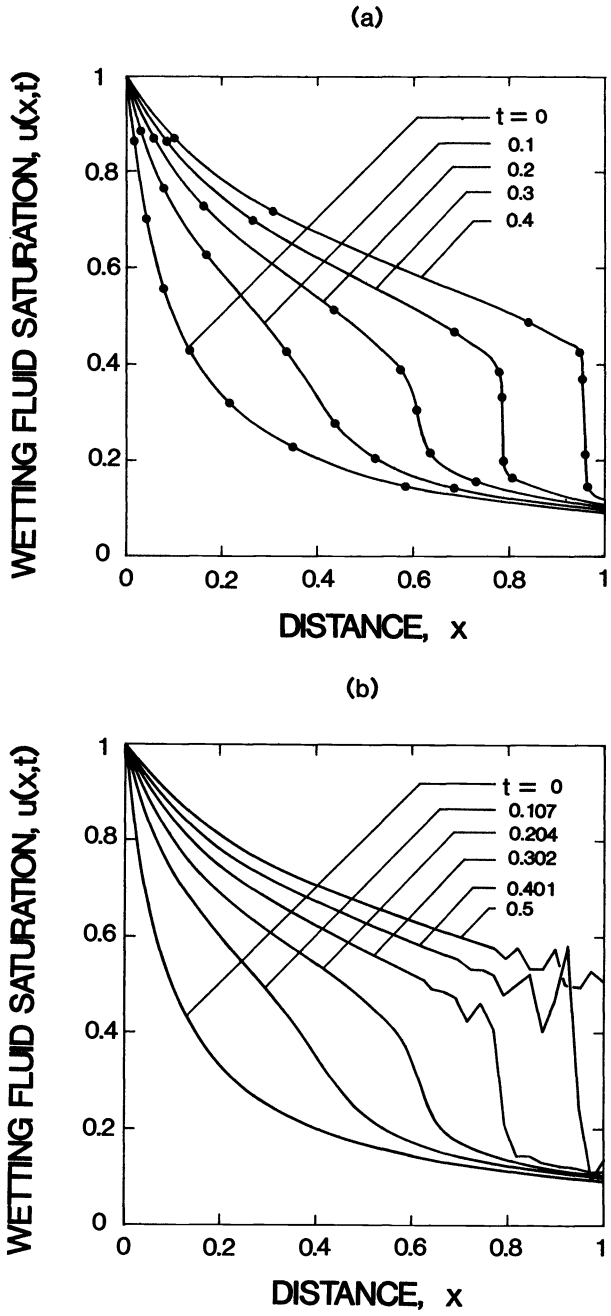


FIG. 11. *Saturation profiles predicted by Buckley–Leverett theory.* (a) *Analysis with* 39 *Newton-adaptive finite elements,* (●) *every fifth node in the adaptive tessellation*; (b) *Analysis with* 39 *uniform, fixed finite elements.*

discretizations; e.g., tessellations with both quadrilateral and triangular elements, which is an issue currently under investigation.

Adaptive finite element methods are needed for integral equations such as the general integral equations of the second kind encountered in molecular theories like van der Waals' and its modern variants. Our experience with van der Waals' theory

(cf. [6]) suggests tht an error bound based on local, polynomial interpolation is inadequate, because the error of discretized solutions of integral equations is far more global in character than that of differential equations.

Basis functions defined on composite coordinate systems [45] or truly local coordinate systems [35] provide flexible representation of solutions to problems that are posed on highly distorted or rapidly deforming domains. Combining adaptive node-placement and adaptive composite coordinates can lead to effective new versions of finite element-based analysis. An example is the analysis of oscillations of a liquid drop [7] so violent that the drop fissions.

Acknowledgments. The authors thank O. A. Basaran, M. Luskin, N. E. Bixler and T. Dupont for valuable discussions.

REFERENCES

[1] J. P. ABBOTT, *An efficient algorithm for the determination of certain bifurcation points*, J. Comput. Appl. Math., 4 (1978), pp. 19-27.

[2] R. ALEXANDER, P. MANSELLI AND K. MILLER, *Moving finite elements for the Stefan problem in two dimensions*, Rend. Accad. Naz. Lincei (Rome), serie VIII, vol. LXVII, fasc. 1-2 (1979), pp. 57-61.

[3] J. H. ARGYRIS, *Energy Theorems and Stuctural Analysis*, Plenum Press, New York, 1968.

[4] I. BABUSKA AND W. C. RHEINBOLDT, *Error estimates for adaptive finite element computations*, SIAM J. Numer. Anal., 15 (1978), pp. 734-754.

[5] I. BABUSKA AND W. G. SZYMCZAK, *An error analysis for the finite element method applied to convection-diffusion problems*, Comput. Methods Appl. Mech. Engrg., 31 (1982), pp. 19-42.

[6] R. E. BENNER, JR., *Equilibria, stability and bifurcations in the physics of fluid interfaces*, Ph.D. thesis, University of Minnesota, Minneapolis, MN, 1983.

[7] R. E. BENNER, JR., T. W. PATZEK AND L. E. SCRIVEN, *Nonblinear oscillations of translationally symmetric, inviscid rotating drops*, Proc. Roy. Soc. London. A, to be submitted, 1987.

[8] R. E. BENNER, JR., L. E. SCRIVEN AND H. T. DAVIS, *Structure and stress in the gas-liquid-solid contact region*, in Structure of the Interfacial Region, M. Lal, ed., Faraday Symposium No. 16, The Royal Society of Chemistry, London, 1982, pp. 169-190.

[9] R. E. BENNER, JR., T. K. VANDERLICK, L. E. SCRIVEN AND H. T. DAVIS, *Molecular theory of fluid multilayers at a wall*, J. Colloid Interf. Sci., to be submitted, 1987.

[10] R. BONNEROT AND P. JAMET, *Numerical computation of the free boundary for the two-dimensional Stefan problem by space-time finite elements*, J. Comput. Phys., 25 (1977), pp. 163-181.

[11] B. S. CAREY, L. E. SCRIVEN AND H. T. DAVIS, *Semiempirical theory of surface tension of binary systems*, AIChE J., 26 (1980), pp. 705-711.

[12] G. F. CAREY AND D. L. HUMPHREY, *Mesh refinement and iterative solution methods for finite element computations*, Internat. J. Numer. Methods Engrg., 17 (1981), pp. 1717-1734.

[13] G. F. CAREY AND T. PLOVER, *Variable upwinding and adaptive mesh refinement in convection-diffusion*, Internat. J. Numer. Methods Engrg., 19 (1983), pp. 341-353.

[14] A. CELLA, M. LUCCHESI AND G. PASQUINELLI, *Space-time elements for the shock wave propagation problem*, Internat. J. Numer. Methods Engrg., 15 (1980), pp. 1475-1488.

[15] R. C. Y. CHIN AND R. KRASNY, *A hybrid asymptotic-finite element method for stiff two point boundary value problems*, this Journal, 4 (1983), pp. 229-243.

[16] R. COURANT, *Variational methods for the solution of problems of equilibrium and vibrations*, Bull. Amer. Math. Soc., 49 (1943), pp. 1-23.

[17] J. CRANK AND R. S. GUPTA, *The isotherm migration method in two dimensions*, Internat. J. Heat Mass Transfer, 18 (1975), pp. 1101-1107.

[18] M. J. P. CULLEN, *Analysis of some low-order finite element schemes for the Navier-Stokes equations*, J. Comput. Phys., 51 (1983), pp. 273-290.

[19] H. T. DAVIS AND L. E. SCRIVEN, *Gradient theory of fluid microstructures*, J. Statist. Phys., 24 (1981), pp. 243-268.

[20] ———, *Stress and structure in fluid interfaces*, in Advances in Chemical Physics, Vol. 49, I. Prigogine and S. A. Rice, eds., John Wiley, New York, 1982, pp. 357-454.

[21] S. F. DAVIS AND J. E. FLAHERTY, *An adaptive finite element method for initial-boundary value problems for partial differential equations*, this Journal, 3 (1982), pp. 6-27.

[22] A. H. FALLS, L. E. SCRIVEN AND H. T. DAVIS, *Structure and stress in spherical microstructures*, J. Chem. Phys., 75 (1981), pp. 3986–4002.

[23] A. S. FOKAS AND Y. C. YORTSOS, *On the exactly solvable equation* $S_t = [(\beta S + \gamma)^{-2} S_x]_x + \alpha(\beta S + \gamma)^{-2} S_x$ *occurring in two-phase flow in porous media*, SIAM J. Appl. Math., 42 (1982), pp. 318–332.

[24] R. J. GELINAS, S. K. DOSS AND K. MILLER, *The moving finite element method: applications to general partial differential equations with multiple large gradients*, J. Comput. Phys., 40 (1981), pp. 202–249.

[25] W. A. GIFFORD, *The shapes and stability of charged, rotating liquid drops*, Ph.D. thesis, University of Minnesota, Minneapolis, MN, 1979.

[26] J. GLIMM, D. MARCHESIN AND O. MCBRYAN, *A numerical method for two phase flow with an unstable interface*, J. Comput. Phys., 39 (1981), pp. 179–200.

[27] P. M. GRESHO, R. L. LEE AND R. L. SANI, *On the time-dependent solution of the incompressible Navier-Stokes equations in two and three dimensions*, in Recent Advances in Numerical Methods in Fluids, Vol. 1, C. Taylor and K. Morgan, eds., Pineridge Press, Swansea, U.K., 1979, pp. 27–81.

[28] R. J. HANSON AND J. L. PHILLIPS, *An adaptive numerical method for solving linear Fredholm integral equations of the first kind*, Numer. Math., 24 (1975), pp. 291–307.

[29] R. E. HOFFMAN, F. M. GUERRA AND D. L. HUMPHREY, *Practical applications of adaptive mesh refinement (rezoning)*, Comput. & Structures, 12 (1980), pp. 639–655.

[30] D. L. B. JUPP, *Approximation to data by splines with free knots*, SIAM J. Numer. Anal., 15 (1978), pp. 328–343.

[31] J. KAUTSKY AND N. K. NICHOLS, *Equidistributing meshes with constraints*, this Journal, 1 (1980), pp. 499–511.

[32] H. B. KELLER, *Numerical solution of bifurcation and nonlinear eigenvalue problems*, in Applications of Bifurcation Theory, P. Rabinowitz, ed., Academic Press, New York, 1977, pp. 359–384.

[33] H. S. KHESHGI, O. A. BASARAN, R. E. BENNER, JR., S. F. KISTLER AND L. E. SCRIVEN, *Continuation in a parameter: experience with viscous and free surface flows*, in Numerical Methods in Laminar and Turbulent Flow, C. Taylor, J. A. Johnson and W. R. Smith, eds., Pineridge Press, Swansea, U.K., 1983, pp. 401–410.

[34] H. S. KHESHGI AND L. E. SCRIVEN, *Penalty finite element analysis of unsteady free surface flows*, in Finite Elements in Fluids, Vol. 5, J. T. Oden et al., eds., John Wiley, New York, 1983.

[35] S. F. KISTLER AND L. E. SCRIVEN, *Coating flow computations*, in Computers in Polymer Processing, J. R. A. Pearson and S. M. Richardson, eds., Applied Science Publishers, Ltd., London, 1983, Chap. 7.

[36] D. R. LYNCH AND K. O'NEILL, *Continuously deforming finite elements for the solution of parabolic problems, with and without phase change*, Internat. J. Numer. Methods Engrg., 17 (1981), pp. 81–96.

[37] R. E. NICKELL, R. I. TANNER AND B. CASWELL, *Solution of viscous incompressible jet and free surface flows using finite element methods*, J. Fluid Mech., 65 (1974), pp. 189–206.

[38] F. M. ORR, JR., *Numerical simulation of viscous flow with a free surface*, Ph.D. thesis, University of Minnesota, Minneapolis, MN, 1976.

[39] J. PADOVAN, *Transient finite element analysis of conduction problems subject to variable-speed moving fields*, Numer. Heat Trans., 4 (1981), pp. 147–158.

[40] V. PEREYRA AND E. G. SEWELL, *Mesh selection for discrete solution of boundary problems in ordinary differential equations*, Numer. Math., 23 (1975), pp. 261–268.

[41] W. PROSKUROWSKI, *A note on solving the Buckley–Leverett equation in the presence of gravity*, J. Comput. Phys., 41 (1981), pp. 136–141.

[42] LORD RAYLEIGH, *On the theory of surface forces. II. Compressible fluids*, Philos. Mag., 33 (1892), pp. 209–220.

[43] W. C. RHEINBOLDT, *On a theory of mesh-refinement processes*, SIAM J. Numer. Anal., 17 (1980), pp. 766–778.

[44] E. RIKS, *Application of Newton's method to the problem of elastic stability*, J. Appl. Mech., 39 (1972), pp. 1060–1065.

[45] H. SAITO AND L. E. SCRIVEN, *Study of coating flow by the finite element method*, J. Comput. Phys., 42 (1981), pp. 53–76.

[46] W. SCHÖNAUER, K. RAITH AND G. GOLTZ, *The principle of the difference of difference quotients as a key to the self-adaptive solution of nonlinear partial differential equations*, Comput. Methods Appl. Mech. Engrg., 28 (1981), pp. 327–359.

[47] G. STRANG AND G. J. FIX, *An Analysis of the Finite Element Method*, Prentice-Hall, Englewood Cliffs, NJ, 1973.

[48] G. F. TELETZKE, L. E. SCRIVEN AND H. T. DAVIS, *Gradient theory of wetting transitions*, J. Colloid Interf. Sci., 87 (1982), pp. 550–571.

[49] L. H. UNGAR AND R. A. BROWN, *The dependence of the shape and stability of captive rotating drops on multiple parameters*, Philos. Trans. Roy. Soc. London Ser. A, 306 (1982), pp. 347–370.

[50] E. VAROGLU AND W. D. L. FINN, *Space-time finite elements incorporating characteristics for the Burger's equation*, Internat. J. Numer. Meth. Engrg., 16 (1980), pp. 171–184.

[51] R. WAIT, *Finite-element type solution of integral equations*, in International Computing Symposium (1973), A. Günther et al., eds., North-Holland, Amsterdam, 1974.

[52] N. J. ZABUSKY AND E. A. OVERMAN, *Regularization of contour dynamical algorithms*, J. Comput. Phys., 52 (1983), pp. 351–373.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 4, July 1987

© 1987 Society for Industrial and Applied Mathematics
006

A LOCAL RELAXATION METHOD FOR SOLVING ELLIPTIC PDEs ON MESH-CONNECTED ARRAYS*

C.-C. JAY KUO†, BERNARD C. LEVY† AND BRUCE R. MUSICUS‡

Abstract. A local relaxation method for solving linear elliptic PDEs with $O(N)$ processors and $O(\sqrt{N})$ computation time is proposed. We first examine the implementation of traditional relaxation algorithms for solving elliptic PDEs on mesh-connected processor arrays, which require $O(N)$ processors and $O(N)$ computation time. The disadvantage of these implementations is that the determination of the acceleration factors requires some global communication at each iteration. The high communication cost increases the computation time per iteration significantly. Therefore, a local relaxation scheme is proposed to achieve the acceleration effect with very little global communication in the loading stage. We use a Fourier analysis approach to analyze the local relaxation method and also show its convergence. The convergence rate of the local relaxation method is studied by computer simulation.

Key words. mesh-connected processor arrays, elliptic partial differential equations, successive over-relaxation, local relaxation, Fourier analysis, parallel computation

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

1. Introduction.

Consider a 2-D linear elliptic PDE on a unit square discretized by a finite-difference method with a uniform grid. There is a finite difference equation associated with each grid point, so that a system of linear equations is obtained by this discretization procedure. We may assign one processor to each grid point and connect every processor to its four nearest neighbors. This kind of computer architecture, known as a mesh-connected processor array, suggests a natural parallel computation scheme to solve the above system of equations, i.e., parallel computation in the space domain. Jacobi and Gauss-Seidel relaxation methods seem particularly suitable for mesh-connected processors, since each processor uses only the most recent values computed by its neighbors to update its own value. Unfortunately, the convergence rate of these algorithms is slow. The convergence rate can be improved by various acceleration schemes such as successive over-relaxation (SOR) and Chebyshev semi-iterative relaxation (CSI) [17]. However, to obtain the acceleration effect requires that the acceleration factors should be estimated adaptively [8]. This procedure requires global communication on a mesh-connected processor array and increases the computation time per iteration enormously. Any time savings due to acceleration may be cancelled out by the increased communication time. In order to improve the convergence rate as well as to avoid global communication, a recently developed approach known as the ad hoc SOR [5], [6], or local relaxation [3] method seems to be useful.

The local relaxation scheme was found empirically by Ehrlich [5], [6] and Botta and Veldman [3]. They applied this method to a very broad class of problems and found its efficiency by studying many numerical examples. In this paper, we approach the same problem from an analytical point of view, clearly prove the convergence of

* Received by the editors October 21, 1985; accepted for publication (in revised form) April 18, 1986. This work was supported in part by National Aeronautics and Space Administration grant NAGW-448, Army Research Office grant DAAG29-84-K-0005, the Advanced Research Projects Agency monitored by the Office of Naval Research under contract N00014-81-K-0742, and Air Force Office of Scientific Research contract F49620-84-C-0004.

† Laboratory for Information and Decision Systems and Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.

‡ Research Laboratory of Electronics and Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.

this method for the case of symmetric positive-definite matrices and provide an analytical explanation for the good performance of the method.

The conventional way to analyze the SOR method is to use matrix analysis [17]. This approach depends heavily on the ordering of the grid points and on the properties of the resulting sparse matrix. An alternative technique, which was employed in [12], [14] and [15] to analyze relaxation algorithms, is to use Fourier analysis. Strictly speaking, Fourier analysis applies only to linear constant coefficient PDEs on an infinite domain, or with periodic boundary conditions. Nevertheless, at a heuristic level this approach provides a useful tool for the analysis of more general PDE problems, and it has been used by Brandt [4] to study the error smoothing effect of relaxation algorithms and to develop multigrid methods. Since then, the Fourier analysis approach has received a large amount of attention in the study of multigrid methods [16]. Following the same idea, we shall apply the Fourier analysis approach to the SOR method. For the Poisson Problem defined on the unit square with Dirichlet boundary conditions, we obtain the same result as Young's SOR method. However, our derivation is simpler. For space-varying PDEs, the local relaxation scheme uses space adaptive relaxation parameters. This is different from Young's SOR method which uses time adaptive relaxation parameters [8].

The paper is organized as follows. Section 2 discusses the implementation of the Jacobi, Gauss–Seidel, adaptive SOR and local relaxation methods on mesh-connected processor arrays. Section 3 proves the convergence of the local relaxation method. The Fourier analysis approach is used to analyze the local relaxation method for 5-point and 9-point stencils, respectively, in §§ 4 and 5. It turns out that the 9-point stencil analysis requires a slight modification of the basic convergence result of § 3. Section 6 shows the results of a computer simulation on a test problem which indicates that the convergence rate of the local relaxation method is superior to that of the adaptive SOR method. Some further extensions and conclusions are mentioned in § 7.

2. Implementing numerical PDE algorithms on mesh-connected processor arrays. The computation time for an iterative algorithm equals the product of the number of iterations and time per iteration. In a sequential machine, time per iteration is determined by operation counts, especially by the number of floating point operations required. In a multiprocessor machine, however, time per iteration depends heavily on the algorithm chosen and on the communication scheme. Consider mesh-connected processor arrays in which processors are organized in a geometrically regular two-dimensional square "tiling" pattern and connected only to their nearest neighbors. Algorithms using only local communication will take $O(1)$ communication time, while those using global communication require $O(\sqrt{N})$ communication time per iteration since for an array with N processors, communications between processors located on opposite sides of the array will take $O(\sqrt{N})$ time. We thus seek algorithms with fast convergence rate, short computation time, and primarily local communication.

Let us use an example to illustrate how the above considerations affect the implementation of various iterative algorithms. Consider a self-adjoint second-order linear PDE defined on a closed unit square $\Omega = [0, 1] \times [0, 1]$,

$$-\frac{\partial}{\partial x_1}\left\{p(x_1, x_2)\frac{\partial u}{\partial x_1}\right\} - \frac{\partial}{\partial x_2}\left\{q(x_1, x_2)\frac{\partial u}{\partial x_2}\right\} + \sigma(x_1, x_2)u = f(x_1, x_2), \tag{2.1a}$$

$$(x_1, x_2) \in \Omega,$$

with the following boundary condition on Γ, the boundary of Ω,

$$\alpha(x_1, x_2)u + \beta(x_1, x_2)\frac{\partial u}{\partial n} = \gamma(x_1, x_2), \qquad (x_1, x_2) \in \Gamma, \tag{2.1b}$$

where $\partial u/\partial n$ denotes the outward derivative normal to Γ. The coefficient functions are assumed to be smooth and to satisfy

$$p(x_1, x_2) > 0, \quad q(x_1, x_2) > 0, \quad \sigma(x_1, x_2) \geqq 0, \quad (x_1, x_2) \in \Omega,$$

$$\alpha(x_1, x_2) \geqq 0, \quad \beta(x_1, x_2) \geqq 0, \quad \alpha + \beta > 0, \quad (x_1, x_2) \in \Gamma.$$

If we discretize (2.1) on a $\sqrt{N} \times \sqrt{N}$ uniform grid [17], we get a 5-point stencil; and the finite difference equation at an interior point (i, j) can be written as

$$d_{i,j}u_{i,j} - r_{i,j}u_{i+1,j} - l_{i,j}u_{i-1,j} - t_{i,j}u_{i,j+1} - b_{i,j}u_{i,j-1} = s_{i,j}, \tag{2.2}$$

with

$$l_{i,j} = p_{i-1/2,j}, \quad r_{i,j} = p_{i+1/2,j}, \quad b_{i,j} = q_{i,j-1/2}, \quad t_{i,j} = q_{i,j+1/2}, \tag{2.3a}$$

$$d_{i,j} = p_{i-1/2,j} + p_{i+1/2,j} + q_{i,j-1/2} + q_{i,j+1/2} + \sigma_{i,j}h^2, \qquad s_{i,j} = f_{i,j}h^2 \tag{2.3b}$$

where h is the grid spacing and $p_{i,j}$ is defined as $p(ih, jh)$. Similar discretized equations can be obtained for the boundary points where $u_{i,j}$ is unknown. Let us choose a particular order for those equations, and construct vectors u and s from the variables $u_{i,j}$ and $s_{i,j}$ arranged in the selected order; then the interior and boundary equations can be arranged in matrix form

$$Au = s, \tag{2.4}$$

where A contains the coefficients $d_{i,j}$, $l_{i,j}$, $r_{i,j}$, $t_{i,j}$ and $b_{i,j}$. The matrix A is symmetric, since $l_{i+1,j} = r_{i,j}$ and $b_{i,j+1} = t_{i,j}$. In addition, A is positive definite, since it is irreducibly diagonal dominant [17, p. 23].

Starting from equation (2.2), we can discuss the details of implementing different iterative algorithms with a mesh-connected processor array. Assume that we build a $\sqrt{N} \times \sqrt{N}$ grid of processors, and assign the processor at coordinate (i, j) the responsibility of calculating the value of $u_{i,j}$. Direct communication is allowed only between neighboring processors. At iteration $n+1$, each processor may combine the estimated value of $u^{(n)}$ in neighboring processors, together with its own estimate of $u_{i,j}^{(n)}$, in order to develop a new estimate $u_{i,j}^{(n+1)}$. A particularly simple iteration, for example, is the Jacobi method, in which we iteratively calculate:

$$u_{i,j}^{(n+1)} = d_{i,j}^{-1}(l_{i,j}u_{i-1,j}^{(n)} + r_{i,j}u_{i+1,j}^{(n)} + b_{i,j}u_{i,j-1}^{(n)} + t_{i,j}u_{i,j+1}^{(n)} + s_{i,j}).$$

According to the above iterative equation, each processor uses the values of $u^{(n)}$ obtained by its nearest neighbors to update its value at the current iteration. Time per iteration is constant, because both the communication time and computation time are constant.

If the grid point (i, j) is called a red point when $i+j$ is even, and a black point when $i+j$ is odd, the Jacobi method can be viewed in space and time as consisting of two interleaved, and totally independent, *computational waves* alternating between red and black points. This phenomenon is illustrated in Fig. 1, where the one-dimensional grid with red/black partitioning is shown in the horizontal direction while the evolution from one iteration to the next is indicated in the vertical direction. The solid and dotted lines represent two value-updating processes evolving with time, or two computational waves. In fact, these two waves result in unnecessary redundancy. We need only one wave to get the answer, since both waves converge to the same final values. If we delete one computational wave, the rate of utilization of the processors becomes one half, i.e., every processor works only half of the time. Therefore, we may

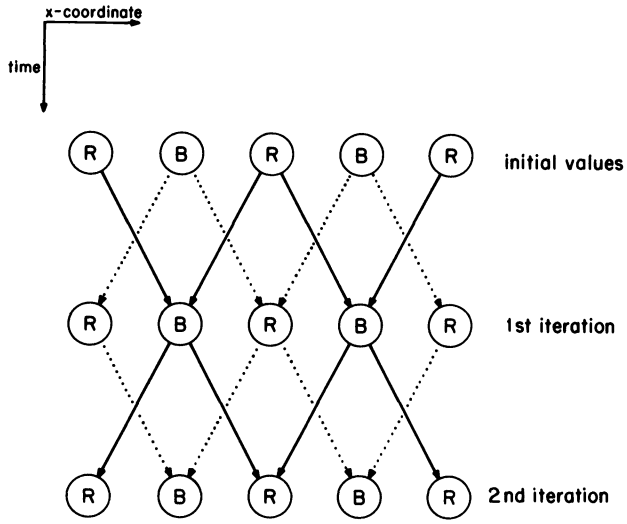


FIG. 1. 1-D *Jacobi relaxation with red/black partitioning.*

group one red and one black point together and assign them to a single processor. This saves half of the hardware cost without loss of computational efficiency (see Fig. 2).

To derive a Gauss-Seidel algorithm for this problem, let us consider the same red/black point partitioning and write the local equation as follows:

red points $(i+j$ *is even*):

$$u_{i,j}^{(n+1)} = d_{i,j}^{-1}(l_{i,j}u_{i-1,j}^{(n)} + r_{i,j}u_{i+1,j}^{(n)} + b_{i,j}u_{i,j-1}^{(n)} + t_{i,j}u_{i,j+1}^{(n)} + s_{i,j});$$

black points $(i+j$ *is odd*):

$$u_{i,j}^{(n+1)} = d_{i,j}^{-1}(l_{i,j}u_{i-1,j}^{(n+1)} + r_{i,j}u_{i+1,j}^{(n+1)} + b_{i,j}u_{i,j-1}^{(n+1)} + t_{i,j}u_{i,j+1}^{(n+1)} + s_{i,j}).$$

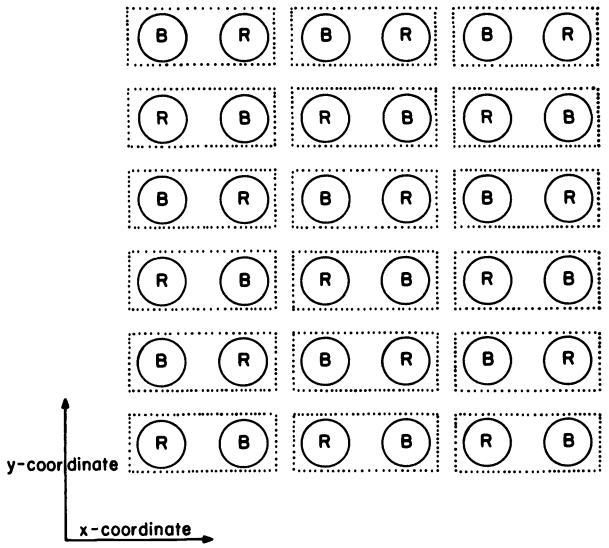


FIG. 2. 2-D *red/black partitioning and grouping.*

Other partitionings will lead to different Gauss-Seidel Schemes; however, the red/black partitioning approach is preferred for parallel implementation on mesh-connected arrays, because of its efficiency and simplicity. Note that the difference between the Jacobi and the Gauss-Seidel relaxation methods is that the Jacobi method updates the values of all nodes at one iteration while the Gauss-Seidel method updates the values of half of these nodes during a first step and updates the values of the other half during a second step based on the previously updated information, and these two steps form a complete iteration. For the case of a one-dimensional grid, we find that the Gauss-Seidel iteration is equivalent to the computational wave of the Jacobi iteration shown by the dotted line in Fig. 1. Therefore, we can save one half of the computational work by using Gauss-Seidel iteration on either a single processor or a mesh-connected array.

The chief shortcoming of the Jacobi or Gauss-Seidel iterative methods lies in their slow convergence rate. It usually happens that the spectral radius of the relaxation matrix is very close to 1, which causes the convergence rate to be extremely slow. The number of iterations needed is proportional to $O(N)$ [17]. Since time per iteration is constant, the total running time is also proportional to $O(N)$.

By applying different acceleration schemes to the Jacobi and Gauss-Seidel techniques, we can derive a variety of accelerated relaxation algorithms. Two typical examples are the Chebyshev semi-iterative (CSI) method and the successive over-relaxation (SOR) method. These acceleration schemes use carefully chosen relaxation parameters to reduce the spectral radii of the iterative matrices so that the iterative algorithms converge faster. To determine the relaxation parameters, CSI acceleration uses knowledge of the largest and smallest eigenvalues of the basic relaxation matrix and SOR acceleration uses knowledge of the spectral radius of the basic relaxation matrix [17]. For a given mesh-connected processor array, if we know these quantities a priori and broadcast them to all processors in the loading stage, each processor can compute the acceleration parameters on its own without additional communication cost. In this case, although the accelerated schemes require a little more computation and memory than the basic Jacobi and Gauss-Seidel relaxation schemes, they present some significant advantages. The reason is that the number of iterations needed is reduced tremendously, becoming $O(\sqrt{N})$ for both acceleration schemes [8]. However, in general we do not know the eigenvalues of the basic relaxation matrix in advance and have to estimate them by some adaptive procedure. To our knowledge, all the estimation procedures developed require the computation of the norms of some global vectors. Therefore, global communication cannot be avoided. This means that the communication cost for a single iteration in a mesh-connected array becomes $O(\sqrt{N})$. As a consequence, time per iteration is $O(\sqrt{N})$ and the total running time becomes $O(N)$ again.

Comparing this result with the result obtained for the basic relaxation methods, it seems that we do not benefit from acceleration schemes when we seek to implement iterative algorithms in parallel on mesh-connected arrays. This can be easily explained by noting that for a single processor, there is no distinction between local and global communications, since all data are fetched from the same memory, while for a mesh-connected processor array, long range communication costs much more than short range communication.

In addition to the above relaxation algorithms, another important class of algorithms for solving systems of linear equations can be derived from an optimization principle. The conjugate gradient (CG) algorithm is an example [8]. Without considering rounding errors, a theoretical analysis indicates that the CG algorithm is able to

solve the discretized PDE exactly in $O(N)$ steps, using only $O(N^2)$ computation and $O(N)$ storage on a single processor. In practice, experience shows that the CG method, when applied to the PDE problem, usually converges in $O(\sqrt{N})$ steps even with rounding errors. Unfortunately, on a mesh-connected array, this algorithm is slowed by the need to compute several inner products of $O(N)$ length vector. Computing the inner product of two vectors whose entries are distributed over a mesh-connected array requires global communication. We therefore encounter the same difficulties as for the accelerated relaxation methods.

The local relaxation method proposed by Ehrlich [5], [6] and Botta and Veldman [3] is a computational algorithm suitable for parallel implementation on mesh-connected processor arrays, because it has the same acceleration effect as SOR and uses only local communication. A local relaxation procedure for equation (2.2) can be written as follows:

red points $(i+j$ *is even*$)$:

$$u_{i,j}^{(n+1)} = (1-\omega_{i,j})u_{i,j}^{(n)} + \omega_{i,j}d_{i,j}^{-1}(l_{i,j}u_{i-1,j}^{(n)} + r_{i,j}u_{i+1,j}^{(n)} + b_{i,j}u_{i,j-1}^{(n)} + t_{i,j}u_{i,j+1}^{(n)} + s_{i,j}). \tag{2.5a}$$

black points $(i+j$ *is odd*$)$:

$$u_{i,j}^{(n+1)} = (1-\omega_{i,j})u_{i,j}^{(n)} + \omega_{i,j}d_{i,j}^{-1}(l_{i,j}u_{i-1,j}^{(n+1)} + r_{i,j}u_{i+1,j}^{(n+1)} + b_{i,j}u_{i,j-1}^{(n+1)} + t_{i,j}u_{i,j+1}^{(n+1)} + s_{i,j}), \tag{2.5b}$$

where $\omega_{i,j}$ is called *the local relaxation parameter.*

Assuming Dirichlet boundary conditions and $M_1 \times M_2 = N$ unknowns within the unit square, it was suggested in [5], [6] that a good choice of local relaxation parameters $\omega_{i,j}$ is given by

$$\omega_{i,j} = \frac{2}{1+\sqrt{1-\rho_{i,j}^2}}, \tag{2.6}$$

where

$$\rho_{i,j} = \frac{2}{d_{i,j}}\left(\sqrt{l_{i,j}r_{i,j}}\cos\frac{\pi}{M_1+1} + \sqrt{t_{i,j}b_{i,j}}\cos\frac{\pi}{M_2+1}\right). \tag{2.7}$$

Since we consider only the case of symmetric discretized matrices, the parameter $\rho_{i,j}$ is always real. This gives us the *ad hoc SOR method* or the *local relaxation method* for a symmetric matrix. However, the local relaxation method can also be applied to more general matrices such that $\rho_{i,j}$ is purely imaginary or complex. The formula to determine the relaxation parameters for these cases can be found in [3], [5] and [6]. In this paper, we will focus on the local relaxation method for a system of equations $Au = s$ where A is symmetric positive definite. The more general case will be considered in a subsequent paper.

The implementation of the local relaxation method is straightforward. It is easy to see that as long as we know the size of the grid, i.e., M_1 and M_2, we can broadcast this information to all processors in the loading stage. Each processor has to compute its own relaxation parameters once according to equations (2.6) and (2.7); then the local iterations specified by (2.5) can be performed in parallel for all processors with only local communication. Since the local relaxation method uses local communication, the computation time per iteration is $O(1)$. We will show that the number of iterations for typical test problems is proportional to $O(\sqrt{N})$ in § 6. Therefore, the total running time becomes $O(\sqrt{N})$. For a $\sqrt{N}\times\sqrt{N}$ processor array, the constraint that each processor should contain a minimum amount of global information implies that the lower bound for the computation time for any algorithm is $O(\sqrt{N})$, since it takes

$O(\sqrt{N})$ time for the data at one edge of the array to move to the opposite edge. It turns out that the local relaxation method achieves this lower bound.

Although the local relaxation method was empirically shown to be powerful, there are several questions which were left unanswered by the papers of Ehrlich, Botta and Veldman. First, they did not prove that the local relaxation method converges. Furthermore, there was no explanation of why the local relaxation method converges very fast. In the following sections, we will explore these two issues.

3. Convergence of the local relaxation method. In this section, we give a sufficient condition for the convergence of a local relaxation procedure. Then, we show that the local relaxation method given by equations (2.5)-(2.7) indeed converges.

In order to obtain a convergence result which covers the most general type of local relaxation procedure, we use a matrix formulation, since such a formulation includes not only the 5-point stencil corresponding to the discretized equation (2.2), but also other kinds of stencils. Given a linear system of equations, $Au = s$, where A is an $N \times N$ real symmetric positive definite matrix with positive diagonal elements, we may rewrite A as

$$A = D - E - F = D(I - L - U) \quad \text{and} \quad E^T = F$$

where I, D, E and F represent identity, diagonal, lower and upper triangular matrices, and $L = D^{-1}E$ and $U = D^{-1}F$. Let W be the diagonal matrix formed by the local relaxation parameters, i.e., $W = \text{diag}\,(\omega_1, \omega_2, \cdots, \omega_N)$. Then, a local relaxation procedure can be written in matrix iterative form as

$$u^{(n+1)} = (I - WL)^{-1}[(I - W) + WU]u^{(n)} + (I - WL)^{-1}WD^{-1}s. \tag{3.1}$$

Let $\bar{u}$ be the solution of the above iterative equation, so that

$$\bar{u} = (I - WL)^{-1}[(I - W) + WU]\bar{u} + (I - WL)^{-1}WD^{-1}s.$$

Define the error vector at nth iteration as $e^{(n)} = u^{(n)} - \bar{u}$. Then the matrix iterative equation in the error space becomes

$$e^{(n+1)} = (I - WL)^{-1}[(I - W) + WU]e^{(n)}. \tag{3.2}$$

The iteration matrix of the local relaxation procedure (3.1) is therefore given by $G(W) = (I - WL)^{-1}[(I - W) + WU]$. The iteration procedure will converge for all initial estimates $\bar{u}^{(0)}$ if and only if all eigenvalues of $G(W)$ are less than one in magnitude, i.e., if the spectral radius $\rho[G(W)]$ of the iteration matrix $G(W)$ is less than 1. A simple sufficient condition for convergence is given by the following theorem.

THEOREM 1 (Sufficient condition for the convergence of a local relaxation procedure). *Suppose A is an $N \times N$ real symmetric positive definite matrix. For the local relaxation procedure given by* (3.1), *if $0 < \omega_i < 2$ for $1 \leqq i \leqq N$, then $\rho[G(W)] < 1$ and the iterative algorithm converges.*

Proof. Let λ and p be an arbitrary eigenvalue, eigenvector pair of $G(W)$. Then $G(W)p = \lambda p$, or equivalently,

$$[(I - W) + WU]p = \lambda(I - WL)p. \tag{3.3}$$

Premultiplying by $p^H DW^{-1}$ on both sides, we obtain

$$p^H DW^{-1}p - p^H Dp + p^H DUp = \lambda p^H DW^{-1}p - \lambda p^H DLp.$$

Since $E^T = F$, $E = DL$, and $F = DU$, it is easy to check that

$$p^H DUp = (Lp)^H Dp = \overline{p^H DLp}.$$

Defining $z = p^H DLp/p^H Dp$ and $1/\omega = p^H DW^{-1}p/p^H Dp$, (3.3) can be simplified as

$$\frac{1}{\omega} - 1 + \bar{z} = \frac{\lambda}{\omega} - \lambda z,$$

or equivalently,

$$\lambda = \frac{1-\omega+\omega\bar{z}}{1-\omega z}.$$

Let $z = re^{j\theta}$, then

$$|\lambda|^2 = \lambda\bar{\lambda} = 1 - \frac{\omega(2-\omega)(1-2r\cos\theta)}{(1-\omega r\cos\theta)^2 + \omega^2 r^2 \sin^2\theta}. \tag{3.4}$$

We know that $|\lambda|^2$ is always positive. If we can show that the second term in the above expression is also positive, then we can conclude that $|\lambda|$ is less than 1. The denominator of the second term of equation (3.4) is positive, so that we only have to consider the numerator. We have

$$2r\cos\theta = 2\,\mathrm{Re}\,(z) = \bar{z} + z = \frac{p^H DLp}{p^H Dp} + \frac{p^H DUp}{p^H Dp} = 1 - \frac{p^H Ap}{p^H Dp} < 1,$$

where the inequality is due to the fact that A and D are both positive definite. Note that since A is positive definite, the matrix D formed with the diagonal elements of A is also positive definite. Therefore, we know that $1 - 2r\cos\theta > 0$. Now, consider the range of the parameter ω. Since $W = \mathrm{diag}\,(\omega_1, \omega_2, \cdots, \omega_N)$, $W^{-1} = \mathrm{diag}\,(\omega_1^{-1}, \omega_2^{-1}, \cdots, \omega_N^{-1})$. Assuming that all relaxation factors are positive, we have

$$\frac{1}{\omega_{\max}} < \frac{p^H DW^{-1}p}{p^H Dp} = \frac{\sum_{i=1}^N |p_i|^2 d_i \omega_i^{-1}}{\sum_{i=1}^N |p_i|^2 d_i} < \frac{1}{\omega_{\min}},$$

where $\omega_{\max}$ and $\omega_{\min}$ are the largest and smallest eigenvalues of the matrix W and p_i is the ith element of the vector p. If we set $0 < \omega_{\min} \leqq \omega_{\max} < 2$, then

$$0 < \omega_{\min} \leqq \omega \leqq \omega_{\max} < 2.$$

Under this condition, the second term in equation (3.4) is always positive, so that the eigenvalues of the matrix $G(W)$ are all less than 1 and the local relaxation procedure (3.1) converges. Q.E.D.

The above theorem gives the range of the local relaxation parameters which guarantees that a local relaxation procedure converges; however, it does not tell us how to choose the relaxation parameters to make a local relaxation procedure converge faster. The local relaxation method mentioned in the last section is a special case of a local relaxation procedure, where the local relaxation parameters are specified for a 5-point stencil discretization. To show its convergence, we only have to show that all relaxation parameters chosen by the rule (2.6), (2.7) are between 0 and 2.

COROLLARY (Convergence of the local relaxation method for a 5-point stencil discretization). *The local relaxation method for a 5-point stencil given by* (2.5)-(2.7) *converges.*

Proof. From the discussion in the previous section, we know that the matrix A obtained by discretizing (2.1) is symmetric positive definite.

Since $p(x_1, x_2)$ and $q(x_1, x_2)$ are positive functions and $\sigma(x_1, x_2)$ is a nonnegative function, we know from (2.3) that $l_{i,j}$, $r_{i,j}$, $b_{i,j}$, $t_{i,j}$ and $d_{i,j}$ are all positive. In addition,

$0 < \cos \pi/(M+1) < 1$ for $1 < M < \infty$. Therefore $\rho_{i,j}$ given by (2.7) is also positive. Using the inequalities

$$2\sqrt{l_{i,j}r_{i,j}} \leqq l_{i,j} + r_{i,j}, \qquad 2\sqrt{t_{i,j}b_{i,j}} \leqq t_{i,j} + b_{i,j},$$

we have

$$\rho_{i,j} = \frac{2}{d_{i,j}}\left(\sqrt{l_{i,j}r_{i,j}} \cos\frac{\pi}{M_1+1} + \sqrt{t_{i,j}b_{i,j}} \cos\frac{\pi}{M_2+1}\right)$$

$$\leqq \frac{l_{i,j}+r_{i,j}}{d_{i,j}} \cos\frac{\pi}{M_1+1} + \frac{t_{i,j}+b_{i,j}}{d_{i,j}} \cos\frac{\pi}{M_2+1} < \frac{l_{i,j}+r_{i,j}+t_{i,j}+b_{i,j}}{d_{i,j}} \leqq 1$$

where the last inequality is obtained by noting that $\sigma_{i,j} \geqq 0$ in (2.3b). It is easy to see that

$$0 < \omega_{i,j} = \frac{2}{1+\sqrt{1-\rho_{i,j}^2}} < 2$$

for $0 < \rho_{i,j} < 1$. The local relaxation parameters chosen by the local relaxation method satisfy the sufficient condition given in Theorem 1, so that the relaxation method converges. Q.E.D.

4. Fourier analysis of the local relaxation method—5-point stencil. The convergence rate of a local relaxation procedure depends on how we choose the local relaxation parameters. The conventional SOR method chooses a spatially invariant relaxation parameter $\omega_{i,j} = \omega$ to minimize the asymptotic convergence rate, or, equivalently, minimize the spectral radius of $G(W)$. Young [18] showed that the optimal choice for ω in the accelerated Gauss-Seidel iteration is

$$\omega = \frac{2}{1+\sqrt{1-\rho^2}}$$

where ρ is the spectral radius of $D^{-1}(E+F)$. For this relaxation parameter, all eigenvalues of $G(\omega I)$ can be shown to have magnitude $\omega - 1$. In practice, it is quite difficult to calculate ρ exactly, and thus adaptive procedures are required to estimate ρ as the computation proceeds. In this section, we will use a Fourier analysis approach to derive a simple formula for a spatially varying relaxation parameter. Our formula is identical to that suggested by Ehrlich [5]. Our approach demonstrates that this formula will indeed achieve an excellent convergence rate. This study also gives some new insight into Young's SOR method.

For a linear constant coefficient PDE with Dirichlet or periodic boundary conditions, the eigenfunctions of $D^{-1}(E+F)$ are sinusoidal functions. Therefore, the spectral radius of this iterative matrix can be obtained by using Fourier analysis. However, for a space-varying coefficient PDE with general boundary conditions, the sinusoidal functions are not eigenfunctions. As a consequence, Fourier analysis cannot be applied rigorously. Notwithstanding this disadvantage, Fourier analysis is still a convenient tool for understanding the convergence properties of relaxation methods [16]. A more rigorous treatment to make Fourier analysis applicable to space-varying coefficient PDEs with general boundary conditions is needed and is currently under study. Roughly speaking, the reason why Fourier analysis often works in spatially varying PDE problems is that the eigenfunctions can be regarded as sinusoidal functions plus some perturbations. As long as the perturbation is comparatively small, the sinusoidal function is a good approximation of the original eigenfunction. Therefore, Fourier analysis is still a good analytical tool. A detailed formulation of Fourier analysis in this general context will be presented elsewhere.

In § 4.1, we will show how to find the lowest Fourier component for given boundary conditions. Then, we use Fourier analysis to analyze the Jacobi relaxation method in § 4.2. This approach is sometimes called *the local Fourier analysis* [16]. Finally, we justify the efficiency of the local relaxation method. The derivation can be viewed as a generalization of Brandt's local Fourier analysis to the Successive Over-Relaxation case.

4.1. Admissible error function space and its lowest Fourier component. Let Γ_i, $1 \leqq i \leqq 4$ denote the four boundaries of the unit square. Consider a set of linear first-order boundary conditions such as (2.1b) on the boundaries of the unit square,

$$B_i u = g_i \quad \text{on } \Gamma_i, \quad 1 \leqq i \leqq 4 \tag{4.1}$$

where B_i represents the boundary condition operator on the ith boundary. It is more convenient to analyze the relaxation in the error space rather than in the solution space, because the error equations are homogeneous. The error formulation for the boundary conditions can be obtained as follows. Let $\bar{u}$ be the actual solution so that

$$B_i \bar{u} = g_i \quad \text{on } \Gamma_i, \quad 1 \leqq i \leqq 4. \tag{4.2}$$

Subtracting (4.2) from (4.1), we obtain the homogeneous PDE in the error,

$$B_i e = 0 \quad \text{on } \Gamma_i, \quad 1 \leqq i \leqq 4. \tag{4.3}$$

The functions defined on the unit square and satisfying the homogeneous boundary conditions (4.3) are called the *admissible error functions*, since any error function allowed in the relaxation process should always satisfy the given boundary conditions. All admissible error functions form the *admissible error function space.* The sinusoidal functions in the admissible error function space can be chosen as a basis of this space because of their completeness. As far as the convergence rate is concerned, we will see that only the lowest frequency component is relevant. Thus, we will find that only the lowest frequency of this basis needs to be determined.

We assume that all B_i's are constant-coefficient operators. Under this assumption, B_1 and B_3 are independent of the x_2-direction, B_2 and B_4 are independent of the x_1-direction, and since the problem domain is square, the admissible Fourier components can be written in separable form as $s_1(x_1)s_2(x_2)$, where $s_1(\cdot)$ and $s_2(\cdot)$ are two 1-D sinusoidal functions. The boundary condition on Γ_1 becomes

$$B_1 s_1(x_1) s_2(x_2) = s_2(x_2) B_1 s_1(x_1) = 0,$$

i.e.,

$$B_1 s_1(x_1) = 0.$$

Similarly, we simplify the boundary conditions on Γ_2, Γ_3 and Γ_4, and decompose the 2-D problem into two independent 1-D problems.

$$\text{(I)} \quad B_1 s_1(x_1) = 0 \text{ when } x_1 = 0, \qquad B_3 s_1(x_1) = 0 \text{ when } x_1 = 1, \tag{4.4a}$$

$$\text{(II)} \quad B_2 s_2(x_2) = 0 \text{ when } x_2 = 0, \qquad B_4 s_2(x_2) = 0 \text{ when } x_2 = 1. \tag{4.4b}$$

From (4.4a) and (4.4b), we can determine the lowest frequencies $\hat{k}_1$ and $\hat{k}_2$ separately. We only show how to get $\hat{k}_1$ from (I); then $\hat{k}_2$ can be obtained from (II) in the same way.

Consider the mixed type boundary operators,

$$B_1 = b_1 + b_2 \frac{d}{dx_1} \quad \text{for } x_1 = 0, \tag{4.5a}$$

$$B_3 = b_3 + b_4 \frac{d}{dx_1} \quad \text{for } x_1 = 1. \tag{4.5b}$$

The Fourier component $s(k_1, x_1)$ of $s(x_1)$ at the frequency k_1 can be written as a linear combination of two complex sinusoids $e^{ik_1x_1}$ and $e^{-ik_1x_1}$, i.e.,

$$s(k_1, x_1) = c(k_1)\, e^{ik_1x_1} + c(-k_1)\, e^{-ik_1x_1}. \tag{4.6}$$

Substituting (4.6) into (4.5), we obtain

$$(b_1 + ib_2k_1)c(k_1) + (b_1 - ib_2k_1)c(-k_1) = 0,$$

$$(b_3 + ib_4k_1)\, e^{ik_1x_1}c(k_1) + (b_3 - ib_4k_1)\, e^{-ik_1x_1}c(-k_1) = 0.$$

In order to get nonzero values for $c(k_1)$ and $c(-k_1)$, the determinant of the 2×2 coefficient matrix should equal zero, or equivalently,

$$e^{i2k_1} = \frac{(b_1 + ib_2k_1)(b_3 - ib_4k_1)}{(b_1 - ib_2k_1)(b_3 + ib_4k_1)}. \tag{4.7}$$

Therefore, we conclude that the frequency k_1 of any admissible 1-D sinusoidal function with respect to the boundary conditions (4.5) must satisfy equation (4.7).

Let us look at two examples. If the boundary conditions on both Γ_1 and Γ_3 are Dirichlet type boundary conditions, which means b_2 and b_4 are zeros, then (4.7) becomes

$$e^{i2k_1} = 1 \quad \text{or} \quad \cos 2k_1 + i \sin 2k_1 = 1.$$

The solutions are $k_1 = n\pi$, $n = 0, \pm 1, \pm 2, \cdots$. However, it is easy to see that the zero frequency cannot be allowed. Thus, the lowest Fourier frequency $\hat{k}_1$ in the admissible error space is π. If we change the boundary condition on Γ_3 to be of Neumann type, i.e., $b_3 = 0$ but $b_4 \neq 0$, then (4.7) becomes

$$e^{i2k_1} = -1 \quad \text{or} \quad \cos 2k_1 + i \sin 2k_1 = -1.$$

The solutions are $k_1 = \frac{1}{2}n\pi$, where n is odd and the lowest frequency $\hat{k}_1$ is $\pi/2$.

The same procedure applies to other boundary conditions. Notice that the determination of the lowest Fourier components of a given PDE requires only the knowledge of the boundary conditions and of the geometry of the problem domain. The above procedure does not require any information about the PDE operator.

4.2. Local Jacobi relaxation operator and its properties. In this section, we use a Fourier analysis approach to analyze the local Jacobi operator and to determine its largest eigenvalue, or spectral radius, for given boundary conditions as previously discussed. The spectral radius of a local Jacobi operator will be used to determine the optimal local relaxation parameter of the local relaxation scheme in § 4.3.

Define the x_1-*direction* (x_2-*direction*) *forward-shift* and *backward-shift operators*, E_1 and E_1^{-1} (E_2 and E_2^{-1}), as

$$E_1 u_{i,j} = u_{i+1,j}, \qquad E_1^{-1} u_{i,j} = u_{i-1,j},$$

$$E_2 u_{i,j} = u_{i,j+1}, \qquad E_2^{-1} u_{i,j} = u_{i,j-1}.$$

Then, the 5-point discretization formula for an interior grid point can be written as

$$L_{i,j} u_{i,j} = s_{i,j}$$

where $L_{i,j} \equiv d_{i,j} - (r_{i,j}E_1 + l_{i,j}E_1^{-1} + t_{i,j}E_2 + b_{i,j}E_2^{-1})$ is the *local discretized differential operator at node* (i, j). The Jacobi relaxation at a local node can be written as

$$u_{i,j}^{(n+1)} = J_{i,j}u_{i,j}^{(n)} + s_{i,j}, \qquad n \geqq 0,$$

where $J_{i,j} \equiv d_{i,j}^{-1}\ (r_{i,j}E_1 + l_{i,j}E_1^{-1} + t_{i,j}E_2 + b_{i,j}E_2^{-1})$ is the *local Jacobi relaxation operator.* From the error point of view, we get

$$e_{i,j}^{(n+1)} = J_{i,j}e_{i,j}^{(n)}, \qquad n \geqq 0.$$

If the input error function $e_{i,j}^{(n)}$ is the complex sinusoid $e^{i(k_1x_1+k_2x_2)}$, we have

$$J_{i,j}\, e^{i(k_1x_1+k_2x_2)} = \mu_{i,j}(k_1, k_2)\, e^{i(k_1x_1+k_2x_2)}$$

where $\mu_{i,j}(k_1, k_2) = d_{i,j}^{-1}\ (r_{i,j}\, e^{ik_1h} + l_{i,j}\, e^{-ik_1h} + t_{i,j}\, e^{ik_2h} + b_{i,j}\, e^{-ik_2h})$. Therefore, we may view $e^{i(k_1x_1+k_2x_2)}$ as an eigenfunction of $J_{i,j}$ with eigenvalue $\mu_{i,j}(k_1, k_2)$. The magnitude of $\mu_{i,j}(k_1, k_2)$ provides some information on how the errors of different frequency components are smoothed out by the Jacobi relaxation process. This quantity can be computed as

$$|\mu_{i,j}(k_1, k_2)| = \frac{([(r_{i,j} + l_{i,j}) \cos k_1h + (t_{i,j} + b_{i,j}) \cos k_2h]^2 + [(r_{i,j} - l_{i,j}) \sin k_1h + (t_{i,j} - b_{i,j}) \sin k_2h]^2)^{1/2}}{d_{i,j}}. \tag{4.8}$$

By assuming that the coefficient functions are smooth so that

$$r_{i,j} - l_{i,j} = l_{i+1,j} - l_{i,j} = O(h) \quad \text{and} \quad t_{i,j} - b_{i,j} = b_{i+1,j} - b_{i,j} = O(h),$$

then the two cosine terms in $|\mu_{i,j}|$ are the dominant terms.

The eigenvalue function $\mu_{i,j}(k_1, k_2)$ is usually called the *frequency response* in signal processing [11] and the Jacobi relaxation operator can be viewed as a filtering process in the frequency domain. The frequency response function with the magnitude shown in (4.8), in fact, represents a 2-D notch filter instead of a lowpass filter. However, if the discretization space h is small enough and the waveforms are band-limited, this is not a significant problem. The reason is best explained from the Taylor's series approximation of a function $f(x)$, i.e.,

$$f(x) = f(x_0) + (x - x_0)f'(x_0) + \frac{(x - x_0)^2}{2!}f''(x_0) + \cdots.$$

Supposing $f(x) = e^{ikx}$ and $x = x_0 + h$, the high order terms are negligible only if the product kh is reasonably small, say, less than 1. That means that as long as the magnitude of wavevector k is bounded, we can always find a discretization spacing h which is fine enough so that the dimensionless frequencies $\theta_1 = k_1h$ and $\theta_2 = k_2h$ are always inside the unit circle in the (θ_1, θ_2) plane. In this region, the notch filter behaves like a lowpass filter. The lowpass filtering property makes the error at higher frequencies converge to zero faster than that at lower frequencies.

The eigenvalue with the largest magnitude is the dominant factor in the *asymptotic convergence rate analysis,* so that we will focus our attention on this quantity. Following the above discussion, we define the *spectral radius* $\rho_{i,j}$ of $J_{i,j}$ as the largest magnitude of $\mu_{i,j}(k_1, k_2)$, i.e.,

$$\rho_{i,j} \equiv \max_{k_1,k_2} |\mu_{i,j}(k_1, k_2)|.$$

For the symmetric positive definite matrix case, the magnitude of $\mu_{i,j}(k_1, k_2)$ is the largest at the lowest frequency $(\hat{k}_1, \hat{k}_2)$, since such a choice makes the dominant cosine terms of (4.8) as large as possible. Therefore, we obtain

$$\rho_{i,j} = |\mu_{i,j}(\hat{k}_1, \hat{k}_2)|.$$

The above procedure, known as local Fourier analysis of $J_{i,j}$, has two implicit assumptions. First, $J_{i,j}$ is space-invariant. Secondly, the problem domain should be either extended to infinity or be rectangular but with periodic boundary conditions. In general, these two assumptions do not hold. As a consequence, $\rho_{i,j}$ is a spatially varying function and is not equal to the spectral radius ρ of the original Jacobi relaxation matrix $J = D^{-1}(E+F)$. An important question to be answered is whether the knowledge of $\rho_{i,j}$ can provide us some information about ρ. Two observations may be of help. First, the *same* lowest frequency gives the spectral radii of all local Jacobi relaxation operators, so this frequency should play a role in determining the eigenfunction giving the spectral radius of the Jacobi relaxation matrix J. Furthermore, for a given low frequency $(\hat{k}_1, \hat{k}_2)$, $\rho_{i,j}$ is a very smooth function in space. It is neither sensitive to variations of the coefficient functions nor sensitive to changes in the boundary conditions. For example, the values of $\rho_{i,j}$ given by equations (2.7) and (4.8), computed for Dirichlet and periodic boundary conditions separately, are only slightly different under the assumption that the coefficient functions are smooth. Let $\bar{\rho} = \max_{i,j} \rho_{i,j}$ and $\underline{\rho} = \min_{i,j} \rho_{i,j}$. Then, ρ should be a quantity somewhere between $\underline{\rho}$ and $\bar{\rho}$. Usually, the difference between $\bar{\rho}$ and $\underline{\rho}$ is so small that *any* $\rho_{i,j}$ can give us an estimate of ρ.

Notice that in order to determine the spectral radius of a local relaxation operator, we only have to know the lowest admissible Fourier component corresponding to the given boundary conditions, discussed in § 4.1, and then to compute $\rho_{i,j}$ according to (4.8).

4.3. Applying Fourier analysis to the local relaxation method. Let us reconsider the local relaxation method, i.e., equation (2.5). We divide the problem domain into red and black points and update one color at each time step. Suppose we start with the relaxation of the red points, then with the black points. The local equations for the error can be written as

$$e_{i,j}^{(n+1)} = (1-\omega_{i,j})e_{i,j}^{(n)} + \omega_{i,j}J_{i,j}e_{i,j}^{(n)} \quad \text{for } i+j \text{ even}, \tag{4.9}$$

$$e_{i,j}^{(n+1)} = (1-\omega_{i,j})e_{i,j}^{(n)} + \omega_{i,j}J_{i,j}e_{i,j}^{(n+1)} \quad \text{for } i+j \text{ odd}. \tag{4.10}$$

If all $J_{i,j}$'s are approximately the same within that small region, then we can combine (4.9) with (4.10) and rewrite (4.10) as

$$e_{i,j}^{(n+1)} = (1-\omega_{i,j})e_{i,j}^{(n)} + \omega_{i,j}(1-\omega_{i,j})J_{i,j}e_{i,j}^{(n)} + \omega_{i,j}^2 J_{i,j}^2 e_{i,j}^{(n)} \quad \text{for } i+j \text{ odd}. \tag{4.11}$$

Let e_R and e_B represent respectively the errors at the red and black points around node (i,j). Notice that equations (4.9) and (4.11) describe, in fact, the relation of two waves—the *red* and *black* waves in the local region around node (i,j). Rearranging and simplifying (4.9) and (4.11), we obtain the following relation between two successive iterations,

$$\begin{pmatrix} e_R^{(n+1)} \\ e_B^{(n+1)} \end{pmatrix} = \begin{bmatrix} 1-\omega_{i,j} & \omega_{i,j}J_{i,j} \\ \omega_{i,j}(1-\omega_{i,j})J_{i,j} & 1-\omega_{i,j}+\omega_{i,j}^2 J_{i,j}^2 \end{bmatrix} \begin{pmatrix} e_R^{(n)} \\ e_B^{(n)} \end{pmatrix}, \qquad n \geqq 0,$$

where the 2×2 matrix operator $G_{i,j}(\omega_{i,j}, J_{i,j})$ appearing in the above equation is called *the local relaxation operator with relaxation factor* $\omega_{i,j}$ *at node* (i,j).

By assuming that an eigenfunction of the local relaxation operator $G_{i,j}$ has the form $(c_1 e^{i(k_1x_1+k_2x_2)}, c_2 e^{i(k_1x_1+k_2x_2)})^T$ and that the corresponding eigenvalue is $\lambda_{i,j}$, we may write

$$G_{i,j}(\omega_{i,j}, J_{i,j}) \begin{pmatrix} c_1 e^{i(k_1x_1+k_2x_2)} \\ c_2 e^{i(k_1x_1+k_2x_2)} \end{pmatrix} = \lambda_{i,j} \begin{pmatrix} c_1 e^{i(k_1x_1+k_2x_2)} \\ c_2 e^{i(k_1x_1+k_2x_2)} \end{pmatrix}. \tag{4.12}$$

Equation (4.12) can be further simplified because of the assumption that the local operator $J_{i,j}$ is approximately constant in the region around the node (i, j) and has the eigenvalue $\mu_{i,j}$ for this complex sinusoid $e^{i(k_1x_1+k_2x_2)}$. We obtain

$$G_{i,j}(\omega_{i,j}, \mu_{i,j})\begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \lambda_{i,j}\begin{pmatrix} c_1 \\ c_2 \end{pmatrix}. \tag{4.13}$$

Note that the eigenvalues of the operator matrix $G_{i,j}(\omega_{i,j}, J_{i,j})$ are the same as those of the matrix $G_{i,j}(\omega_{i,j}, \mu_{i,j})$.

Furthermore, from (4.13), we know that $\mu_{i,j}(k_1, k_2)$ and $\lambda_{i,j}(k_1, k_2)$ are related via

$$|G_{i,j}(\omega_{i,j}, \mu_{i,j}) - \lambda_{i,j}I| = 0,$$

or, equivalently,

$$\lambda_{i,j}^2 - (2 - 2\omega_{i,j} + \omega_{i,j}^2\mu_{i,j}^2)\lambda_{i,j} + (1-\omega_{i,j})^2 = 0.$$

Therefore, we get

$$\lambda_{i,j} = 1 - \omega_{i,j} + \frac{\omega_{i,j}^2\mu_{i,j}^2}{2} \pm \frac{\sqrt{\Delta}}{2}, \tag{4.14a}$$

where

$$\Delta = 4(1-\omega_{i,j})\omega_{i,j}^2\mu_{i,j}^2 + \omega_{i,j}^4\mu_{i,j}^4. \tag{4.14b}$$

Let us consider the special case, $\omega_{i,j} = 1$, which corresponds to the Gauss-Seidel relaxation method. The eigenvectors of the 2×2 matrix $G_{i,j}(\omega_{i,j}, \mu_{i,j})$ are $(1, 0)^T$ and $(1, \mu_{i,j})^T$ and the corresponding eigenvalues are 0 and $\mu_{i,j}^2$. This means that if we start with two sinusoidal waveforms at the same frequency but with different amplitudes, one of them, the red wave represented by the vector (1, 0), disappears in one step. The other wave remains and alternates between the red and black points thereafter, as mentioned in § 2. The ratio between the updated wave and the old wave is equal to the constant $\mu_{i,j}$, so that the amplitude is reduced by a factor of $\mu_{i,j}^2$ per cycle.

The purpose of introducing the relaxation parameter $\omega_{i,j}$ is to make the eigenvalue $\lambda_{i,j}(k_1, k_2, \omega_{i,j})$ of the new operator $G_{i,j}$ smaller than the eigenvalue $\mu_{i,j}(k_1, k_2)$ of the old operator $J_{i,j}$. For a fixed real $\mu_{i,j}(k_1, k_2)$, the relationship between $\lambda_{i,j}$ and $\omega_{i,j}$ can be described by the root locus technique depicted in Fig. 3.

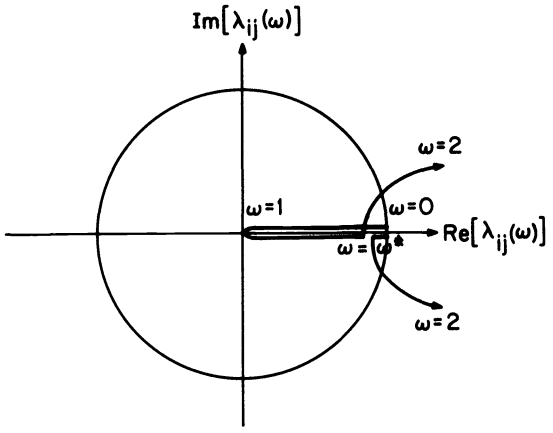


FIG. 3. *Root loci of* $\lambda_{i,j}(\omega)$ *with fixed* $\mu_{i,j}$.

When $0<\omega_{i,j}<2$, the magnitude of $\lambda_{i,j}$ is less than one. By Theorem 1, we know that if $0<\omega_{i,j}<2$ for all i,j then $\rho[G(W)]<1$ and the local relaxation algorithm converges. When $\Delta=0$, the two eigenvalues $\lambda_{i,j,1}$ and $\lambda_{i,j,2}$ coincide, and the largest possible magnitude of these two eigenvalues, $\lambda_{i,j,m}\equiv\max(|\lambda_{i,j,1}|,|\lambda_{i,j,2}|)$, is minimized. The value of $\omega_{i,j}$ which sets $\Delta=0$ is called the *optimal relaxation factor with respect to a specific* $\mu_{i,j}$ and is denoted by $\omega_{i,j,opt}(\mu_{i,j})$. By solving

$$\Delta=4(1-\omega_{i,j})\omega_{i,j}^2\mu_{i,j}^2+\omega_{i,j}^4\mu_{i,j}^4=0$$

and requiring

$$0<\omega_{i,j}<2,$$

we find that

$$\omega_{i,j,opt}(\mu_{i,j})=\frac{2}{1+\sqrt{1-\mu_{i,j}^2}}. \tag{4.15}$$

The general relation between $\lambda_{i,j,m}$ and $\omega_{i,j}$ can be derived in a straightforward way from equation (4.14) and is given by

$$\lambda_{i,j,m}=\omega_{i,j}-1, \qquad \omega_{i,j,opt}(\mu_{i,j})\leqq\omega_{i,j}<2, \tag{4.16a}$$

$$\lambda_{i,j,m}=\left[\frac{\omega_{i,j}|\mu_{i,j}|+\sqrt{\omega_{i,j}^2\mu_{i,j}^2+4(1-\omega_{i,j})}}{2}\right]^2, \qquad 0<\omega_{i,j}\leqq\omega_{i,j,opt}(\mu_{i,j}). \tag{4.16b}$$

The minimum value of all possible $\lambda_{i,j,m}$'s is, therefore, $\omega_{i,j,opt}(\mu_{i,j})-1$.

Since $\mu_{i,j}$ is a function of frequency, (4.15) implies that different frequencies require different optimal relaxation factors. However, we are allowed to choose only one $\omega_{i,j}$, so we have to consider the overall performance, i.e., $\omega_{i,j}$ has to be selected so that the spectral radius of the local relaxation operator $G_{i,j}(\omega_{i,j},J_{i,j})$ is minimized over all frequencies. Let $\rho_{i,j}$ be the spectral radius of the local operator $J_{i,j}$ and $\mu_{i,j}$ be an arbitrary eigenvalue of $J_{i,j}$. By definition, $|\mu_{i,j}|\leqq\rho_{i,j}$, so we know that $\omega_{i,j,opt}(\mu_{i,j})\leqq\omega_{i,j,opt}(\rho_{i,j})$ from (4.15). Using the relation in (4.16), we reason as follows. If we choose $\omega_{i,j}=\omega_{i,j,opt}(\mu_{i,j})$, $\lambda_{i,j,m}(\mu_{i,j})$ achieves its minimum value of $\omega_{i,j,opt}(\mu_{i,j})-1$ but $\lambda_{i,j,m}(\rho_{i,j})$ is greater than $\omega_{i,j,opt}(\rho_{i,j})-1$. On the other hand, if $\omega_{i,j,opt}(\rho_{i,j})$ is chosen as the relaxation factor, both $\lambda_{i,j,m}(\mu_{i,j})$ and $\lambda_{i,j,m}(\rho_{i,j})$ are equal to $\omega_{i,j,opt}(\rho_{i,j})-1$. Comparing these two cases, the latter choice is the best scheme to minimize the spectral radius of $G_{i,j}(\omega_{i,j},J_{i,j})$. This optimal value of $\omega_{i,j}$ is denoted as $\omega_{i,j}^*$, and is given by

$$\omega_{i,j}^*=\omega_{i,j,opt}(\rho_{i,j})=\frac{2}{1+\sqrt{1-\rho_{i,j}^2}}.$$

This is exactly the same formula as suggested by Ehrlich. The reason that this is a good choice is due to the fact that the eigenfunction with the largest eigenvalue of the Jacobi relaxation operator is the one corresponding to the lowest frequency component, and to the observation that the space varying relaxation parameter $\omega_{i,j}^*$ optimizes the convergence of this lowest frequency mode.

5. The local relaxation method for a 9-point stencil. The above derivation applies to a 5-point stencil, which appears when we discretize a linear second-order elliptic PDE without the crossover term $\partial^2/\partial x_1\,\partial x_2$. If there is a crossover term, a finite difference discretization gives a 9-point stencil. In this section, we will propose a local relaxation scheme for a 9-point stencil, give a sufficient condition for its convergence, and use a Fourier analysis approach to explain the rule for selecting good local relaxation parameters. The approach is similar to that used in §§ 3 and 4.

In order to make the presentation clear, we use a simple problem mentioned in [1] as an illustrative example. Consider the linear partial differential operator defined on $\Omega = [0, 1] \times [0, 1]$,

$$L = \frac{\partial^2}{\partial x_1^2} + a(x_1, x_2) \frac{\partial^2}{\partial x_1 \partial x_2} + \frac{\partial^2}{\partial x_2^2} \tag{5.1}$$

where $|a(x_1, x_2)| < 2$, with appropriate boundary conditions. The condition $|a(x_1, x_2)| < 2$ is required to guarantee that L is an elliptic operator. We also assume that $a(x_1, x_2)$ is sufficiently smooth so that it can be viewed as being approximately constant locally. The following discretization scheme is used

$$\frac{\partial^2}{\partial x_1^2} \leftrightarrow \frac{E_1 - 2 + E_1^{-1}}{h^2}, \qquad \frac{\partial^2}{\partial x_2^2} \leftrightarrow \frac{E_2 - 2 + E_2^{-1}}{h^2},$$

$$\frac{\partial^2}{\partial x_1 \partial x_2} \leftrightarrow \frac{E_1 E_2 + E_1^{-1} E_2^{-1} - E_1^{-1} E_2 - E_1 E_2^{-1}}{4h^2}.$$

The local Jacobi relaxation operator $J_{i,j}$ can be decomposed into two parts $J_{i,j,1}$ and $J_{i,j,2}$, i.e.,

$$J_{i,j} = J_{i,j,1} + J_{i,j,2},$$

where

$$J_{i,j,1} = \tfrac{1}{4}(E_1 + E_1^{-1} + E_2 + E_2^{-1}),$$

$$J_{i,j,2} = \tfrac{1}{16}(a_{i+1/2,j+1/2} E_1 E_2 + a_{i-1/2,j-1/2} E_1^{-1} E_2^{-1} - a_{i-1/2,j+1/2} E_1^{-1} E_2 - a_{i+1/2,j-1/2} E_1 E_2^{-1}),$$

and where

$$a_{i,j} = a(ih, jh).$$

Suppose we use the red/black partitioning; then the $J_{i,j,1}$ operator couples nodes of different colors while the $J_{i,j,2}$ operator couples nodes of the same color. A four-color scheme which leads to a four-color SOR method has been proposed for this problem [1]. Here, we propose a different local relaxation scheme which uses only red/black partitioning, so that the iteration equations for the error in the local region can be written as

$$e_R^{(n+1)} = (1 - \omega_{i,j}) e_R^{(n)} + \omega_{i,j} [J_{i,j,1} e_B^{(n)} + J_{i,j,2} e_R^{(n)}], \tag{5.2a}$$

$$e_B^{(n+1)} = (1 - \omega_{i,j}) e_B^{(n)} + \omega_{i,j} [J_{i,j,1} e_R^{(n+1)} + J_{i,j,2} e_B^{(n)}], \tag{5.2b}$$

or, equivalently,

$$\begin{pmatrix} e_R^{(n+1)} \\ e_B^{(n+1)} \end{pmatrix} = G_{i,j}(\omega_{i,j}, J_{i,j,1}, J_{i,j,2}) \begin{pmatrix} e_R^{(n)} \\ e_B^{(n)} \end{pmatrix}$$

where

$$G_{i,j}(\omega_{i,j}, J_{i,j,1}, J_{i,j,2}) = \begin{bmatrix} 1 - \omega_{i,j} + \omega_{i,j} J_{i,j,2} & \omega_{i,j} J_{i,j,1} \\ \omega_{i,j}(1 - \omega_{i,j}) J_{i,j,1} + \omega_{i,j}^2 J_{i,j,1} J_{i,j,2} & 1 - \omega_{i,j} + \omega_{i,j} J_{i,j,2} + \omega_{i,j}^2 J_{i,j,1}^2 \end{bmatrix}. \tag{5.3}$$

In general, the linear system of equations, $Au = s$ for a 9-point stencil which is obtained by discretizing an elliptic PDE with a crossover term can be decomposed as

$$Au = (D - E - F - C)u = D(I - L - U - V)u = s, \tag{5.4}$$

where A is an $N \times N$ real symmetric positive definite matrix and D, E, F and C are diagonal, lower and upper triangular, and block diagonal matrices respectively. In (5.4), we view the 9-point stencil as the superposition of a standard 5-point stencil and of a 4-point stencil formed by the nodes at the four corners. The standard 5-point stencil is accounted for by $D-E-F$, while the remaining 4-point stencil due to the crossover term is represented by the matrix $C = DV$. It is not hard to see that

$$E^T = F \quad \text{and} \quad C^T = C.$$

According to the local relaxation method specified by (5.2), the matrix iterative equation in the error space becomes

$$e^{(n+1)} = (I - WL)^{-1}[(I - W) + WU + WV]e^{(n)}, \tag{5.5}$$

where W is a diagonal matrix formed by local relaxation parameters. The iteration matrix is therefore given by

$$G(W) = (I - WL)^{-1}[(I - W) + WU + WV].$$

A simple sufficient condition for the convergence of (5.5), or (5.2), can be obtained by generalizing Theorem 1. Following the same steps as in the proof of Theorem 1, we find that

$$\lambda = \frac{1 - \omega(1-\alpha) + \omega\bar{z}}{1 - \omega z},$$

where λ, p is an arbitrary eigenvalue/eigenvector pair of $G(W)$, $z = p^H DLp / p^H Dp$, $1/\omega = p^H DW^{-1}p / p^H Dp$ and $\alpha = p^H Cp / p^H Dp$. Since C is symmetric, α is a real number. Let $z = r e^{j\theta}$; then

$$|\lambda|^2 = 1 - \frac{\omega[2 - \omega(1-\alpha)](1 - \alpha - 2r\cos\theta)}{(1 - \omega r\cos\theta)^2 + \omega^2 r^2 \sin^2\theta}, \tag{5.6}$$

which is similar to (3.4). Now, consider

$$\begin{aligned} \alpha + 2r\cos\theta &= \alpha + 2\,\mathrm{Re}\,(z) = \alpha + \bar{z} + z \\ &= \frac{p^H DVp}{p^H Dp} + \frac{p^H DLp}{p^H Dp} + \frac{p^H DUp}{p^H Dp} = 1 - \frac{p^H Ap}{p^H Dp} < 1, \end{aligned}$$

where the inequality is due to the fact that A and D are both positive definite. Furthermore, let us assume that $\alpha < 1$. In order to guarantee that $|\lambda| < 1$ for all possible eigenvalues, the sufficient condition becomes

$$0 < \omega_{\min} \leqq \omega_{i,j} \leqq \omega_{\max} < \frac{2}{1 - \alpha_{\min}},$$

where

$$\alpha_{\min} = \min_p \frac{p^H Cp}{p^H Dp}, \tag{5.7}$$

and where the minimization is over all eigenvectors p of the matrix $G(W)$. Therefore, we have the following theorem.

THEOREM 2 (Sufficient condition for the convergence of a local relaxation procedure for a 9-point stencil discretization). *Suppose that A is an $N \times N$ real symmetric positive definite matrix. For the local relaxation procedure given by* (5.4) *and the constant $\alpha_{\min}$ defined by* (5.7), *if $0 < \omega_i < 2/(1 - \alpha_{\min})$ for $1 \leqq i \leqq N$, then $\rho[G(W)] < 1$ and the iterative algorithm converges.*

Note that if there is no crossover term, the matrix C is zero and α is also zero. In this case, the 9-point stencil reduces to a 5-point stencil and Theorem 2 reduces to Theorem 1. Therefore, our proposed local relaxation scheme for a 9-point stencil, (5.5), is a natural generalization of the conventional SOR method for a 5-point stencil, specified by (3.2).

In the above derivation, we have used the assumption that $\alpha = p^H Cp / p^H Dp$ is less than 1 for any eigenvector p of the matrix $G(W)$. Now, let us estimate the value of α by examining the example given by (5.1). For simplicity, we consider the special case where $a(x_1, x_2) = a$ is constant and assume that the boundary conditions are *periodic*, i.e., $u(0, x_2) = u(1, x_2)$ for $0 \leqq x_2 \leqq 1$, $u(x_1, 0) = u(x_1, 1)$ for $0 \leqq x_1 \leqq 1$. In this case, the eigenvectors p of $G(W)$ can be found in closed form and are given by one of the following two-dimensional arrays, $\sin(k_1 ih + k_2 jh)$, $\cos(k_1 ih + k_2 jh)$, $\sin(k_1 ih - k_2 jh)$ and $\cos(k_1 ih - k_2 jh)$, where i and j range from 1 to $\sqrt{N}$, $h = 1/\sqrt{N}$, and k_1, k_2 are multiples of 2π. Then, after some computations, we find that

$$\alpha(p) = \frac{p^H Cp}{p^H Dp} = \pm \frac{a}{4} \sin k_1 h \sin k_2 h \quad \text{and} \quad |\alpha(p)| < \frac{|a|}{4} \quad \text{for } \textit{all } p.$$

Therefore, if we choose ω_i between 0 and $8/(4+|a|)$, the local relaxation algorithm for this particular problem will converge. However, this choice is too conservative to give a good convergence rate when $|a|$ is close to 2.

Generally speaking, two types of errors arise in the numerical solution of elliptic PDEs by iterative methods. The first of these is caused by the error between the initial guess and the true solution. The other is the numerical rounding error due to the finite precision arithmetic. The first error is usually concentrated in the low frequency region, whereas the second can exist at all possible frequencies. The numerical rounding error is usually so small that it can be ignored, provided it does not grow with the number of iterations. Thus, the error smoothing primarily aims at reducing errors in the low frequency region where the initial guess errors are substantial.

Let us temporarily ignore the numerical rounding error and focus on the initial guess error only. In order to guarantee the convergence of all components in the low frequency region, we need only to select

$$\alpha_{\min} := \alpha^L_{\min} \equiv -\frac{|a|}{4} \sin(\tilde{k}_1 h) \sin(\tilde{k}_2 h)$$

where $\tilde{k}_1$ and $\tilde{k}_2$ are the largest frequencies of interest. Usually the mesh is so fine that $\alpha^L_{\min}$ is of order $O(h^2)$. Although this conclusion is obtained from a simple example, it seems reasonable to believe that $\alpha^L_{\min}$ is also of order $O(h^2)$ for more general second-order elliptic PDEs with space-varying coefficients and other boundary conditions.

The remaining problem is to select a set of local relaxation parameters such that the iterative algorithm converges as quickly as possible in the low frequency region. We can use the Fourier analysis approach introduced in § 4 to analyze the local 2×2 matrix operator $G_{i,j}(\omega_{i,j}, J_{i,j,1}, J_{i,j,2})$ given by (5.3). Let $\mu_{i,j,1}(k_1, k_2)$, $\mu_{i,j,2}(k_1, k_2)$ and $\mu_{i,j}(k_1, k_2)$ be eigenvalues of $J_{i,j,1}$, $J_{i,j,2}$ and $J_{i,j}$ respectively. Following the procedure used before, we find that the optimal local relaxation factor for (5.1) is

$$\omega^*_{i,j} = \frac{2}{1 - \varepsilon_{i,j} + \sqrt{(1-\varepsilon_{i,j})^2 - \rho^2_{i,j}}} \tag{5.8}$$

where $\varepsilon_{i,j} = \mu_{i,j,2}(\hat{k}_1, \hat{k}_2)$, $\rho_{i,j} = \mu_{i,j,1}(\hat{k}_1, \hat{k}_2)$, and $(\hat{k}_1, \hat{k}_2)$ is the mode which maximizes

$\mu_{i,j} = \mu_{i,j,1} + \mu_{i,j,2}$. The spectral radius of the local relaxation operator $G_{i,j}$ is

$$\lambda_{i,j}(G_{i,j}(\omega^*_{i,j}, J_{i,j,1}, J_{i,j,2})) = \omega^*_{i,j}(1+\varepsilon_{i,j}) - 1. \tag{5.9}$$

Typical values for $\omega^*_{i,j}$ and $\lambda_{i,j}$ can be obtained by considering example (5.1) with Dirichlet boundary conditions, which is of more interest compared to the periodic boundary conditions in practice. Since $a(x_1, x_2)$ is smooth, we can approximate $J_{i,j,2}$ by

$$J_{i,j,2} \approx \frac{a_{i,j}}{16}(E_1E_2 + E_1^{-1}E_2^{-1} - E_1^{-1}E_2 - E_1E_2^{-1}).$$

The values of $\mu_{i,j,1}(k_1, k_2)$, $\mu_{i,j,2}(k_1, k_2)$ and $\mu_{i,j}(k_1, k_2)$ are

$$\mu_{i,j,1}(k_1, k_2) = \frac{1}{2}(\cos k_1h + \cos k_2h),$$

$$\mu_{i,j,2}(k_1, k_2) = \pm\frac{a_{i,j}}{8}[\cos(k_1+k_2)h - \cos(k_1-k_2)h] = \pm\frac{a_{i,j}}{4}\sin k_1h \sin k_2h$$

and

$$\mu_{i,j}(k_1, k_2) = \frac{1}{2}(\cos k_1h + \cos k_2h) \pm \frac{a_{i,j}}{4}\sin k_1h \sin k_2h.$$

In the low frequency region, $J_{i,j,1}$ is a lowpass filter, while $J_{i,j,2}$ is a highpass filter. $J_{i,j}$ is similar to $J_{i,j,1}$ in this region, because $\mu_{i,j,2}$ is almost zero for very low frequencies. Therefore, we can view $J_{i,j,2}$ as a *perturbation*. Thus, $J_{i,j}$ is a perturbed lowpass filter. Its spectral radius $\rho_{i,j}(J_{i,j})$ is determined by the lowest admissible frequency. For this particular problem, the lowest admissible frequency mode turns out to be $(\hat{k}_1, \hat{k}_2) = (\pi, \pi)$. Therefore,

$$\varepsilon_{i,j} = \frac{|a_{i,j}|}{4}\sin^2 \pi h, \qquad \rho_{i,j} = \cos \pi h.$$

Substituting these values back into (5.8) and (5.9), we find

$$\omega^*_{i,j} \approx \frac{2}{1+(\sqrt{2+|a_{i,j}|}/\sqrt{2})\pi h} \approx 2\left(1 - \frac{\sqrt{2+|a_{i,j}|}}{\sqrt{2}}\pi h\right), \tag{5.10a}$$

$$\lambda_{i,j} \approx 1 - \sqrt{4+2|a_{i,j}|}\,\pi h. \tag{5.10b}$$

In general, $\alpha^L_{\min}$ is of $O(h^2)$ so that for sufficiently small values of h, the above optimal relaxation parameters satisfy the sufficient condition of Theorem 2. Hence, the convergence of the local relaxation method in the low frequency region is guaranteed and the convergence rate can be estimated by examining the spectral radius of the local relaxation operator given by (5.10b).

Now let us go back to the effect of numerical rounding errors. The optimal relaxation parameters $\omega^*_{i,j}$ given by (5.10a) may be outside the convergence range defined by 0 and $2/(1-\alpha(p))$ for some eigenvectors p corresponding to high frequency error components. Therefore, we expect that the error in the high frequency region will grow. The error growth rate is problem dependent and can be analyzed by Fourier analysis. If the error growth rate is so slow that it does not effect the answer much, we can stick to a single set of optimal local relaxation parameters. On the other hand, if the error growth rate is relatively large, we may use two sets of local relaxation parameters. One set aims at reducing the low frequency error quickly and the other

set, formed by smaller values of $\omega_{i,j}$, is used to smooth the high frequency error once in a while so that the rounding errors do not accumulate. This mixed scheme should perform much better than a scheme using a single set of conservative local relaxation parameters. However, the optimal scheduling of these two sets of local relaxation parameters is still unknown. We believe that it depends on the problem to be solved. Some numerical experiments will be needed to gain a better understanding of the issue.

6. Performance analysis of the local relaxation algorithm.

6.1. Convergence rate analysis—linear constant coefficient PDEs. For a linear constant coefficient PDE defined on a unit square with Dirichlet boundary conditions, the spectral radii of all local Jacobi operators are the same, and thus all local relaxation parameters and the spectral radii of all local relaxation operators are the same, i.e., for all i, j

$$\rho_{i,j} = \rho, \qquad \omega_{i,j} = \omega, \qquad \lambda_{i,j} = \lambda. \tag{6.1}$$

In this case, the local relaxation method is the same as SOR.

The *asymptotic convergence rate* of an arbitrary global iterative operator P, denoted by $R_\infty(P)$, is defined as [17]

$$R_\infty(P) = -\ln \rho(P).$$

Under the conditions (6.1), the asymptotic convergence rate is also given by

$$R_\infty(P) = -\ln \rho(P_{i,j})$$

where $P_{i,j}$ is the local relaxation operator of P.

For a Poisson equation on the unit square with Dirichlet boundary conditions, the local Jacobi operator for this particular problem is

$$J_{i,j} = \tfrac{1}{4}(E_1 + E_1^{-1} + E_2 + E_2^{-1}).$$

Applying Fourier analysis to $J_{i,j}$, we find that the spectral radius of $J_{i,j}$ is $\cos \pi h$, where h is the grid spacing. The global asymptotic convergence rate of the Jacobi method, the Gauss-Seidel method and the local relaxation method can be computed as

$$R_\infty(\text{Jacobi}) = -\ln \rho(J_{i,j}) = -\ln \cos \pi h \approx \frac{1}{2}\pi^2 h^2,$$

$$R_\infty(\text{Gauss-Seidel}) = -\ln \lambda[G_{i,j}(1, J_{i,j})] = -\ln \cos^2 \pi h \approx \pi^2 h^2,$$

$$R_\infty(\text{local relaxation}) = -\ln \lambda[(G_{i,j}(\omega^*, J_{i,j})] = -\ln\left(\frac{1-\sin \pi h}{1+\sin \pi h}\right) \approx 2\pi h.$$

Therefore, the number of iterations is proportional to $O(1/h)$, i.e., $O(\sqrt{N})$, for the local relaxation method.

6.2. Computer simulation. For general space-varying coefficient PDEs, it is difficult to analyze the convergence rate as shown in § 6.1. So, a simple numerical example is used to illustrate the convergence rate of the local relaxation method for solving space-varying coefficient PDEs. The convergence rates of the SOR and CG methods are also shown for the purpose of comparison. For the SOR and CG methods, the Ellpack software package [13] was used.

The example chosen is

$$
\begin{aligned}
& e^{xy}\frac{\partial^2 u}{\partial x^2}+e^{-xy}\frac{\partial^2 u}{\partial y^2}+e^{xy}y\frac{\partial u}{\partial x}-e^{-xy}x\frac{\partial u}{\partial y}+\frac{1}{1+x+y}u \\
& \quad = e^{2xy}\sin \pi y[(2y^2-\pi^2)\sin \pi x+3\pi y\cos \pi x] \\
& \qquad +\pi \sin \pi x(x\cos \pi y-\pi \sin \pi y)+\frac{e^{xy}\sin \pi x\sin \pi y}{1+x+y}
\end{aligned} \tag{6.2}
$$

on the unit square, with the boundary conditions

$$
u(x, y)=0 \quad \text{for } x=0,\ x=1,\ y=0 \text{ and } y=1, \tag{6.3}
$$

and its solution is $e^{xy}\sin \pi x \sin \pi y$. Although equation (6.2) does not have the same form as (2.1), it is easy to verify that the discretized matrix is still symmetric positive definite so that Theorem 1 applies here. The lowest frequency mode for this problem is $(\hat{k}_1, \hat{k}_2)=(\pi, \pi)$ because of the Dirichlet boundary conditions.

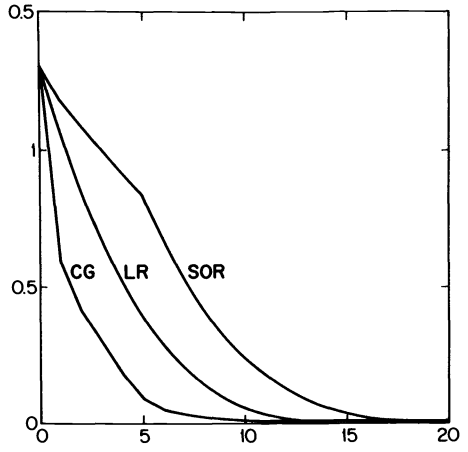
For this test problem, three 5-point discretization schemes are used with grid spacings 1/10, 1/30 and 1/50. Starting from the initial guess $u^{(0)}(x, y)=0$ for all grid points, the maximum errors at each iteration are plotted in Fig. 4. The results indicate that on a single processor the convergence rate of the local relaxation method is better than that of the SOR method and worse than that of the CG method. However, as mentioned in § 2, on a mesh-connected array the local relaxation takes constant time for each iteration while the CG and SOR methods take $O(\sqrt{N})$ time per iteration, so that the local relaxation method is much faster. We also note that the number of iterations required for the local relaxation method is proportional to $\sqrt{N}$. This is consistent with the analysis of the previous section.

7. Extensions and conclusions. The local relaxation method includes two important steps. The first is to determine the admissible lowest frequencies using boundary condition information. The second is to approximate the PDE operator locally by a linear finite difference operator, divide the nodes into red and black points and form a locally accelerated successive over-relaxation (local relaxation) operator. In previous discussions, some ideal assumptions were made so that the analysis and design of the local relaxation algorithm become very simple. However, we may encounter several difficulties in applying the local relaxation method directly to real world problems.

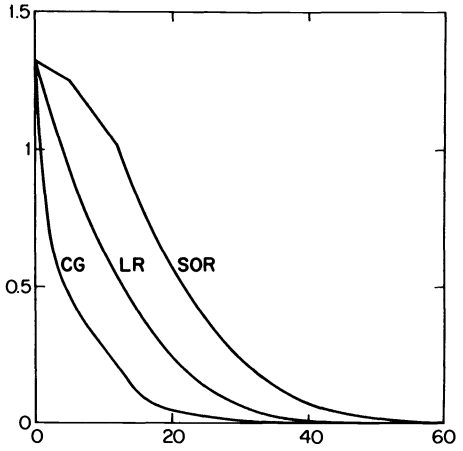
Under the assumption that the problem domain is a unit square and that the boundary condition operator is constant along each edge, the procedure for determining the lowest admissible frequencies is straightforward. These assumptions make the basis functions separable and easy to analyze. However, in practice, the above assumptions may not hold. The problem domain is usually of irregular shape and the boundary condition operators may have space-varying coefficients. As a consequence, it is considerably more difficult to find the lowest frequency error component than for the case we have considered in this paper.

The second difficulty is related to the construction of the local relaxation operator. If the coefficients of a PDE operator have some discontinuities in some region, the Jacobi relaxation operator is not smooth over the region with discontinuous coefficients. In this case, the determination of the optimal local relaxation factors for such abruptly changing operators is still an open question.

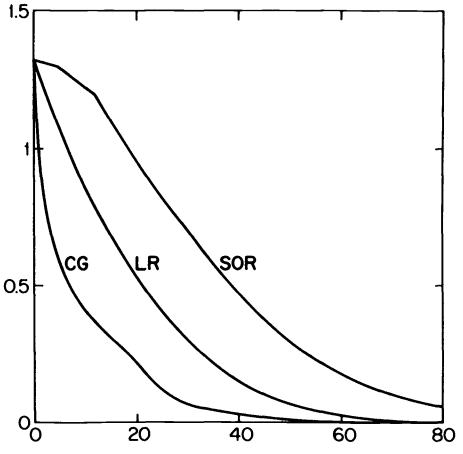
To map the irregular domain problem into a regular processor domain is important in practice. In addition, we need some schemes to partition the grid points evenly between all processors, when the number of grid points is larger than that of processors.



(a)



(b)



(c)

FIG. 4. *Computer simulation results for problem* (6.2)-(6.3) *with* (a) 11×11, (b) 31×31 *and* (c) 51×51 *grids.*

The mapping and partitioning problems have been studied recently [2], [7], but not too many results are known.

Other open questions include how to terminate the local relaxation method. A centralized termination scheme can work as follows. The residues of the local processors are pipelined to a certain processor, say, the central one. Then this processor determines the termination time of the relaxation algorithm and broadcasts a "stop" signal to all processors. The above procedure can be performed in parallel with the local relaxation procedures. Note however that because we avoid the use of global communications in the termination procedure by pipelining the local residues which are sent to the central processor, this processor uses the error residues of the local processors at *different* iterations in determining whether the algorithm should be stopped. A distributed termination scheme would also be useful.

It also seems interesting and challenging to see whether the local relaxation concept can be applied to elliptic PDEs which do not satisfy the assumptions made in § 2, such as the Helmholtz equation, or to other types of PDEs such as hyperbolic PDEs or nonlinear PDEs, and to other discretization schemes such as the finite-element method.

We may note that *distributed* computational PDE algorithms (local relaxation) are different from traditional *central* computational methods (SOR) in several ways. First, distributed computation provides a natural way to achieve a high degree of parallelism. Secondly, distributed algorithms suggest a space-adaptive acceleration scheme, which is not as convenient in centralized computation. Thirdly, although global information is required in determining the local optimal acceleration factors, it seems that only very little global information is relevant, namely the size of the domain. Finally, we benefit a lot in designing the local acceleration algorithm from the *simple* structure of the local operator and from the fact that the global information required is minimized, while the determination of the optimal uniform SOR acceleration factor is complicated and time-consuming.

These nice properties are closely related to the special structure of PDEs. Partial differential equations are formulated to describe local interactions in the physical world, where any interaction between two far space points is the result of a chain of local interactions between near space points. The locality property is very similar to the local communication constraint imposed by VLSI computation [10]. Therefore, although this constraint is a bottleneck in other types of problems, it is not as restrictive for numerical PDE problems.

Our paper has also demonstrated the use of the Fourier analysis approach, or frequency domain approach, to analyze the local relaxation algorithm. This methodology forms a bridge between numerical analysis for solving PDEs and digital signal processing [9]. This new method seems more informative than traditional matrix iterative methods, which usually hide information in a huge matrix. In addition, the local Fourier analysis approach provides a way to analyze distributed numerical algorithms while matrix iterative methods only can be applied to central numerical algorithms. A closer relationship between numerical analysis techniques and Fourier analysis is expected in the future.

Acknowledgments. The authors are grateful to Doctors Louis W. Ehrlich and Youcef Saad, and Professors Lloyd N. Trefethen and John Tsitsiklis for helpful discussions. Some comments from the reviewers were highly appreciated. The authors also wish to thank Professor William T. Thompkins, Jr. for his encouragement and support.

REFERENCES

[1] L. ADAMS AND J. M. ORTEGA, *A Multi-Color SOR Method for Parallel Computation*, ICASE report, 82-9, (1982).

[2] S. H. BOKHARI, *On the mapping problem*, IEEE Trans. Comput., C-30, 3 (1981), pp. 207-214.

[3] E. F. BOTTA AND A. E. P. VELDMAN, *On local relaxation methods and their application to convection-diffusion equations*, J. Comput. Phys., 48 (1981), pp. 127-149.

[4] A. BRANDT, *Multi-level adaptive solutions to boundary-value problems*, Math. Comput., 31 (1977), pp. 333-390.

[5] L. W. EHRLICH, *An Ad Hoc* SOR *Method*, J. Comput. Phys., 44 (1981), pp. 31-45.

[6] ———, *The ad-hoc* SOR *method: a local relaxation scheme*, in Elliptic Problem Solvers II, Academic Press, New York, 1984.

[7] D. GANNON, *On mapping non-uniform PDE structures and algorithms onto uniform array architectures*, in Proc. International Conference on Parallel Processing, 1981.

[8] L. A. HAGEMAN AND D. M. YOUNG, *Applied Iterative Methods*, Academic Press, New York, 1981.

[9] C.-C. KUO, *Parallel algorithms and architectures for solving elliptic partial differential equations*, MIT Technical Report (LIDS-TH-1432), Cambridge, MA, January 1985.

[10] C. A. MEAD AND L. A. CONWAY, *Introduction to* VLSI *Systems*, Addison-Wesley, Reading, MA, 1980.

[11] A. V. OPPENHEIM AND R. W. SCHAFER, *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1975.

[12] D. W. PEACEMAN, *Fundamentals of Numerical Reservoir Simulation*, Elsevier North-Holland, New York, 1977.

[13] J. R. RICE, ELLPACK: *Progress and Plans*, in Elliptic Problem Solvers, M. H. Schultz, ed., Academic Press, New York, 1981.

[14] P. J. ROACHE, *Computational Fluid Dynamics*, Hermosa Publishers, Albuquerque, NM, 1976.

[15] H. L. STONE, *Iterative solution of implicit approximations of multidimensional partial differential equations*, SIAM J. Numer. Anal., 5 (1958), pp. 530-558.

[16] K. STUBEN AND U. TROTTENBERG, *Multigrid methods: Fundamental algorithms, model problem analysis and applications*, in Multigrid Methods, U. Trottenberg, ed., Springer-Verlag, New York, 1982.

[17] R. S. VARGA, *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1962.

[18] D. M. YOUNG, *Iterative methods for solving partial differential equations of elliptic type*, Ph.D. thesis, Harvard University, Cambridge, MA, 1950.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 4, July 1987
© 1987 Society for Industrial and Applied Mathematics
007

GENERAL SPARSE ELIMINATION REQUIRES NO PERMANENT INTEGER STORAGE*

RANDOLPH E. BANK† AND R. KENT SMITH‡

Abstract. General sparse elimination is designed to take maximum advantage of the sparsity of an $N \times N$ matrix **A**. Only the nonzeros of **A** are stored, along with some extra integer overhead to identify the nonzero matrix elements. This extra integer storage may be avoided for the triangular factors generated by an **LDU** decomposition, generally without increasing the order of complexity. In addition to permanent storage for the nonzero elements of the factors, our procedure requires at most $5N$ temporary integer storage.

Key words. sparse Gaussian elimination, sparse matrix

AMS(MOS) subject classifications. 65F05, 65N20

1. Introduction.

We consider the solution of

$$\mathbf{Ax} = \mathbf{b} \tag{1}$$

where **A** is large sparse $N \times N$ nonsingular matrix with no special structure other than a symmetric sparsity pattern. Such problems can be treated by general sparse Gaussian elimination methods [1]–[6]. In such a scheme, one finds a permutation matrix **P** and computes the decomposition

$$\mathbf{PAP}^t = \mathbf{LDU}$$

where **L** is unit lower triangular, **U** is unit upper triangular, and **D** is diagonal. If $\mathbf{A}^t = \mathbf{A}$ then $\mathbf{L}^t = \mathbf{U}$. The solution of the linear system is then computed via

$$\mathbf{Lw} = \mathbf{Pb},$$

$$\mathbf{Dy} = \mathbf{w},$$

$$\mathbf{Uz} = \mathbf{y},$$

$$\mathbf{x} = \mathbf{P}^t\mathbf{z}.$$

As is common in such investigations, we assume the factorization exists for any permutation matrix **P**. This will be true, for example, if the symmetric part of **A** is positive definite.

Traditionally sparse Gaussian elimination algorithms have been partitioned into four distinct phases:

1) Ordering (compute **P**);
2) Symbolic factorization;
3) Numerical factorization (compute **LDU**);
4) Forward and backward substitution (compute **x**).

Several efficient heuristic algorithms are available for ordering general sparse matrices, for example, the minimum degree algorithm [4], [5]. Since our procedures are independent of the choice of **P**, we assume for convenience that $\mathbf{P} = \mathbf{I}$, or equivalently, that **A** has been reordered to reflect the appropriate choice of **P**.

* Received by the editors May 5, 1986; accepted for publication (in revised form) September 22, 1986.

† Department of Mathematics, University of California, La Jolla, California 92093. The research of this author was partly supported by the Office of Naval Research under contract N00014-82-K-0197.

‡ AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

To take maximum advantage of the sparsity, the matrix **A** is represented computationally by a linear array A containing only nonzero elements. In order to access this data, various pointers and row/column indices are stored to establish the correspondence between array entries and matrix entries.

In sparse Gaussian elimination schemes, the matrices **LDU** are also stored in a linear array, U, containing only the nonzero elements. Generally, this array is much larger than A, due to the "fillin" which occurs during the elimination process. The purpose of symbolic factorization is to compute an auxiliary integer data structure for U, similar to that used for A, which allows the numeric factorization and backsolves to be carried out statically. For the simplest schemes, the amount of integer storage required is of the same order as the size of U, ($|U|$) although compression schemes can result in significant reductions [2]–[4].

This paper presents an algorithm where the numeric factorization and solution steps can be carried out without an additional permanent integer data structure. The relevant information is computed as needed, thus eliminating the symbolic factorization step. In addition to the storage required for A, U, x, b, etc., the numeric factorization algorithm uses 5 integer vectors of total length $5N$, and the solution phase requires 3 integer vectors of total length $3N$. Using the ideas of Rose, Tarjan and Lueker [6], the fillin is computed in an optimal $O(N+|U|)$ time, which is usually far less than the cost of the numerical factorization [4], [5]. The overall order of complexity of the numerical operations in this procedure is the same as for other general sparse schemes. While some aspects of the nonnumerical complexity remain unclear, it appears to be of the same order as the numerical complexity in the practical problems we have solved.

The remainder of the paper is organized as follows. The data structure used to store **A** is described in § 2. This structure is a variation of the data structure described in [1] and is based on the data structure used in the symmetric codes of the Yale Sparse Matrix Package [2], [3]. In § 3, we briefly review the relevant graph theoretic results upon which our procedure is based. This is not self-contained and assumes a basic background of the graph theoretic model of Gaussian elimination [4], [5]. We then present our procedures and analyze their correctness and complexity. The Appendix contains a prototype FORTRAN implementation of our procedures.

2. A sparse matrix data structure. Let **A** be an $N \times N$ matrix with elements $\mathbf{a}_{ij}$. We assume **A** is sparse with a symmetric sparsity structure; that is, both $\mathbf{a}_{ij}$ and $\mathbf{a}_{ji}$ are to be treated as nonzero elements (i.e., stored and processed) if $|a_{ij}|+|a_{ji}|>0$. We further assume that the diagonal entries $\mathbf{a}_{ii}$ are nonzero.

In our scheme the nonzero entries of **A** are stored in a linear array A, and accessed through an integer array JA. Let η_i be the number of nonzeros in the strict upper triangular part of column i, and set $\eta = \sum_{i=1}^{N} \eta_i$. The array JA is of length $N+1+\eta$ and the array A is of length $N+1+\eta$ if $\mathbf{A}^t = \mathbf{A}$. If $\mathbf{A}^t \neq \mathbf{A}$, then the array A is of length $N+1+2\eta$. The entries of $JA(i)$ $1 \leqq i \leqq N+1$ are pointers defined as follows:

$$JA(1) = N+2,$$

$$JA(i+1) = JA(i) + \eta_i, \qquad 1 \leqq i \leqq N.$$

The locations $JA(i)$ to $JA(i+1)-1$ contain the η_i row indices corresponding to column i in the strictly upper triangular matrix.

In a similar manner, the array A is defined as follows:

$$A(i) = \mathbf{a}_{ii}, \qquad 1 \leqq i \leqq N,$$

$$A(N+1) \quad \text{is arbitrary,}$$

$$A(k) = \mathbf{a}_{ji}, \quad j = JA(k), \quad JA(i) \leqq k \leqq JA(i+1) - 1.$$

If $\mathbf{A}^t \neq \mathbf{A}$, then

$$A(k+\eta) = \mathbf{a}_{ij}, \quad j = JA(k), \quad JA(i) \leqq k \leqq JA(i+1) - 1.$$

In words, the A array stores the diagonal first, followed by the strict upper triangle stored column-wise. If $\mathbf{A}^t \neq \mathbf{A}$, then this is followed by the strict lower triangle stored row-wise. Since $\mathbf{A}$ is structurally symmetric, the row indices for the upper triangle are identical to the column indices for the lower triangle, and hence need not be duplicated in storage.

As an example, let

$$A = \begin{bmatrix} \mathbf{a}_{11} & \mathbf{a}_{12} & \mathbf{a}_{13} & 0 & 0 \\ \mathbf{a}_{21} & \mathbf{a}_{22} & 0 & \mathbf{a}_{24} & 0 \\ \mathbf{a}_{31} & 0 & \mathbf{a}_{33} & \mathbf{a}_{34} & \mathbf{a}_{35} \\ 0 & \mathbf{a}_{42} & \mathbf{a}_{43} & \mathbf{a}_{44} & 0 \\ 0 & 0 & \mathbf{a}_{53} & 0 & \mathbf{a}_{55} \end{bmatrix}.$$

Then

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
JA	7	7	8	9	11	12	1	1	2	3	3					
A	a_{11}	a_{22}	a_{33}	a_{44}	a_{55}		a_{12}	a_{13}	a_{24}	a_{34}	a_{35}	a_{21}	a_{31}	a_{42}	a_{43}	a_{53}
	Diagonal						Upper Triangle					Lower Triangle				

This data structure is similar to the data structure described in [1], except that the roles of the rows and columns have been reversed. The data structure has a number of interesting features which we briefly summarize below.

1. For symmetric matrices this data structure saves N integer storage locations compared to the data structure used by the symmetric Yale Sparse Matrix codes [2], [3], which store all entries for a given row in consecutive locations in A. For nonsymmetric matrices, about half of the integer storage is saved.
2. For many matrix calculations a single indirect address computation in JA can be used to process both a lower and an upper triangular element in A. This loop unrolling technique can save overhead costs on vector machines such as the Cray. For example, the following procedure[1] computes $\mathbf{y} = \mathbf{Ax}$:

```
      procedure MULT(N, A, x, y)
1.        for i ← 1 to N
2.            y(i) ← A(i)x(i)
3.        lshift ← JA(N+1) − JA(1)
4.        ushift ← 0
5.        for i ← 1 to N
6.            for k ← JA(i) to JA(i+1) − 1
7.                j ← JA(k)
8.                y(i) ← y(i) + A(k + lshift)x(j)
9.                y(j) ← y(j) + A(k + ushift)x(i)
      end MULT
```

[1] In this paper we adopt the convention that zero trip loops are not executed.

For symmetric matrices, set *lshift* $\leftarrow 0$ in line 3. Also, $\mathbf{y} = \mathbf{A}^t\mathbf{x}$ may be readily computed by setting, *lshift* $\leftarrow 0$, *ushift* $\leftarrow JA(N+1) - JA(1)$.

3. When solving (1) by iterative methods like SSOR, independent access to the diagonal, lower triangular and upper triangular parts of **A** is required. These components are easily accessible in our data structure but may be somewhat more complicated and expensive with other sparse storage schemes.
4. It is often convenient to view **A** as a bordered matrix of the form

$$\mathbf{A} = \begin{bmatrix} \bar{\mathbf{A}} & \mathbf{c} \\ \mathbf{r}^t & \mathbf{d} \end{bmatrix}$$

where $\bar{\mathbf{A}}$ is an $(N-1)\times(N-1)$ matrix, **c** and **r** are $N-1$ vectors and **d** is a scalar. In some applications $\bar{\mathbf{A}}$ is "fixed" while **c**, **r** and **d** may change, requiring an update of the numeric factorization. Since the nonzeros of c and r are stored in consecutive locations in U, the factorization is easy to update.

Our data structure for storing the factored matrix U is analogous to the real array A. The diagonal elements of **D** are stored first, followed by the strictly upper triangular part of **U**, stored column-wise. For nonsymmetric problems, this is followed by the strictly lower triangular part of **L**, stored row-wise. The diagonal elements of **U** and **L** are unity and therefore not stored. As we will see in the next section, an analogue for the complete integer data structure *JA* is not required.

3. A sparse Gaussian elimination procedure. Let $G = (V, E)$ denote the ordered, undirected graph corresponding to the matrix **A**. V is the vertex set $\{v_i\}_{i=1}^N$, and E is the edge set with $e_{ij} (= e_{ji}) \in E$ if and only if $|\mathbf{a}_{ij}| + |\mathbf{a}_{ji}| \neq 0$, $i \neq j$. For each vertex $v_i \in V$, $adj_G(v_i) = \{k \mid e_{ik} \in E\}$ is the index set of vertices adjacent to v_i in G.

Gaussian elimination on **A** corresponds to a sequence of elimination graphs G_i, $0 \leqq i \leqq N-1$, defined inductively as follows (with $G_0 \equiv G$): to create G_i from G_{i-1}, remove v_i and all its incident edges from G_{i-1}, and add new edges as required to pairwise connect all vertices in $adj_{G_{i-1}}(v_i)$ (see [4], [5]).

Let F denote the set of edges added during the elimination process, and let $G' = (V, E \cup F)$. Gaussian elimination applied to G' produces no additional fillin edges. Thus the object of symbolic factorization in this context is to compute the set $E \cup F$.

Following Rose, Tarjan and Lueker [6], we define the function $m(i)$, by

$$m(i) = \min \{k > i \mid k \in adj_{G'}(v_i)\}, \qquad 1 \leqq i \leqq N-1. \tag{2}$$

Note that $m(i)$ can also be defined as $\min \{k \in adj_{G_{i-1}}(v_i)\}$. The following characterization of the set $E \cup F$, although not explicitly given in [6], is implicit in that work (see [6, Lemma 5] and also [3], [4]).

THEOREM 1. *Let $G = (V, E)$ be the ordered, undirected graph corresponding to* **A**, *and let F denote the set of fillin edges generated by Gaussian elimination. Then $e_{ij} \in E \cup F$, $i < j$, if and only if*

(i) $e_{ij} \in E$, *or*

(ii) *there exists a sequence $(k_1, k_2 \cdots k_p)$ such that*

(a) $k_1 = l$, $k_p = j$, $e_{lj} \in E$;

(b) $i = k_q$ *for some* $2 \leqq q \leqq p-1$;

(c) $k_q = m(k_{q-1})$ $2 \leqq q \leqq p$.

Theorem 1 and its variations form the basis of the optimal algorithms for computing $E \cup F$; see [3], [4], [6] for several such algorithms.

Another optimal order algorithm for computing $|U|$, consistent with the storage scheme described in § 2, is presented below. The algorithm uses 3 integer arrays—M

of length $N-1$, *LIST* of length N, and *JU* of length $N+1$. The array *JU* is not necessary for computing $|E \cup F|$, but plays an important role in the factorization and solution steps based upon Procedure FILLIN.

```
      procedure FILLIN(N, JA, JU, M, LIST)
1.        M(i) ← 0, 1 ≦ i ≦ N − 1
2.        LIST(i) ← 0, 1 ≦ i ≦ N
3.        JU(1) ← N + 1
4.        for i ← 1 to N
5.            length ← 0
6.            LIST(i) ← i
7.            for j ← JA(i) to JA(i + 1) − 1
8.                k ← JA(j)
9.                while LIST(k) = 0 do
10.                   LIST(k) ← LIST(i)
11.                   LIST(i) ← k
12.                   length ← length + 1
13.                   If M(k) = 0, then M(k) ← i
14.                   k ← M(k)
15.           JU(i + 1) ← JU(i) + length
16.           k ← i
17.           for j ← 1 to length + 1
18.               ksave ← k
19.               k ← LIST(k)
20.               LIST(ksave) ← 0
      end FILLIN
```

THEOREM 2. *Procedure* FILLIN *correctly computes* $|E \cup F|$ *in* $O(N + |E \cup F|)$ *time.*

Proof. We show by induction that FILLIN correctly computes the set $S_i = \{k \mid m(k) = i\}$; thus $m(k)$ is computed correctly, and the correctness of the overall procedure then follows from Theorem 1. The case $i = 1$ is trivial, we assume the sets S_k, $1 \leqq k \leqq i-1$ have been correctly computed.

In processing, column i, we start with the seed indices, $k \leftarrow JA(j)$, line 8, corresponding to edges in E, and simply begin to apply Theorem 1 to generate those edges in F for this column. During this process we may encounter indices for which $m(k)$ has been previously defined. If an undefined value of $m(k)$ is found, line 13, then we set $m(k) = i$; if $m(k) < i$, it would have been previously defined, and if $m(k) > i$, it would violate the definition of m. Thus S_i is correctly defined.

Each edge found is added to the linked list *LIST*, so edges are counted exactly once. By setting $LIST(i) \leftarrow i$ on line 6, no edges e_{ki} with $k > i$ will be found. Thus *LIST* contains the row (column) indices of the nonzero elements in column i of **U** (row i in **L**) on completion of the loop beginning at line 7. The "cleanup" loop (lines 16–20) merely reinitializes the *LIST* array. The array *JU* contains pointers to U which are analogous to the first $N+1$ entries in *JA*.

As for the complexity, the two inner loops (lines 7–14 and 17–20) require $O(1)$ work for each found edge, and each edge in $E \cup F$ is found exactly once. The remaining computations are obviously $O(N)$. □

4. Applications. Procedure FILLIN forms the basis for the numerical factorization and solution steps in Gaussian elimination. The need for a separate symbolic factorization routine is eliminated by embedding floating point matrix operations involving

U in the procedure. In this section, we consider several variations on the procedure FILLIN.

4.1. Solution step. First consider the solution phase from a given **LDU** factorization. The strategy is to construct the relevant integer arrays while solving $\mathbf{Ly} = \mathbf{b}$. These arrays may then be used in the solution of the triangular system, $\mathbf{Ux} = \mathbf{z}$. Since the order of the elements in the triangular systems is arbitrary, the elements in U are processed in the order in which they were found in procedure FILLIN. To solve $\mathbf{Ly} = \mathbf{b}$ for symmetric matrices, the cleanup loop (lines 16–20 in FILLIN) could be replaced by:

```
16.   k ← i
17.   sum ← 0
18.   for j ← JU(i) to JU(i+1) − 1
19.       ksave ← k
20.       k ← LIST(k)
21.       LIST(ksave) ← 0
22.       sum ← sum + U(j)y(k)
23.   y(i) ← b(i) − sum
24.   LIST(k) ← 0
```

The solution of $\mathbf{Ux} = \mathbf{z}$ follows the same general pattern except that the columns are processed in the reverse order. This presents no difficulty since the arrays M and JU have been previously computed. Finally, note that by adding the lshift and ushift parameters as in procedure MULT, the same code can treat symmetric problems, and nonsymmetric problems for $\mathbf{A}$ and $\mathbf{A}^t$.

4.2. Numeric factorization. We now consider the prototype elimination step in the numerical factorization procedure. Let $\bar{\mathbf{A}}$ denote the upper left $k \times k$ principal submatrix of $\mathbf{A}$ and $\bar{\mathbf{L}}\bar{\mathbf{D}}\bar{\mathbf{U}}$ denote its factorization. At the $(k+1)$st step we inductively factor the upper left $(k+1) \times (k+1)$ submatrix of $\mathbf{A}$ as

$$\begin{bmatrix} \bar{\mathbf{A}} & \mathbf{c} \\ \mathbf{r}^t & \mathbf{d} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{L}} & 0 \\ \bar{\mathbf{r}}^t & 1 \end{bmatrix} \begin{bmatrix} \bar{D} & 0 \\ 0 & \bar{d} \end{bmatrix} \begin{bmatrix} \bar{U} & \bar{c} \\ 0 & 1 \end{bmatrix}$$

where

$$\bar{\mathbf{L}}\bar{\mathbf{D}}\bar{\mathbf{c}} = \mathbf{c},$$

$$\bar{\mathbf{U}}^t\bar{\mathbf{D}}\bar{\mathbf{r}} = \mathbf{r},$$

$$\bar{\mathbf{d}} = \mathbf{d} - \bar{\mathbf{r}}^t\bar{\mathbf{D}}\bar{\mathbf{c}}.$$

Since $\bar{\mathbf{L}}\bar{\mathbf{D}}\bar{\mathbf{U}}$ have previously been computed, this step requires the solution of at most two lower triangular systems of equations. The vectors $\bar{\mathbf{c}}$ and $\bar{\mathbf{r}}$ are sparse, say with p nonzeros. Note that the nonzeros of $\mathbf{c}$ and $\mathbf{r}$ are a subset of those in $\bar{\mathbf{c}}$ and $\bar{\mathbf{r}}$. Thus the linear systems can be reduced to $p \times p$ lower triangular systems by deleting rows and columns corresponding to zero elements in $\bar{\mathbf{c}}$ and $\bar{\mathbf{r}}$. These $p \times p$ subsystems are solved by forward substitution, taking advantage of any structural zeros that remain. The inner product used to compute $\bar{\mathbf{d}}$ is similarly reduced to length p, and the multiplication by $\bar{\mathbf{D}}$ is easy to avoid using intermediate values generated in the computation of $\bar{\mathbf{c}}$ and $\bar{\mathbf{r}}$.

To take advantage of the sparsity of these systems, the column indices associated with $\bar{\mathbf{r}}$ and $\bar{\mathbf{c}}$ are first generated by FILLIN. The resulting list of indices, *ORDER*, must be ordered in increasing k for the solution of the triangular systems. The sorting

procedure is quite simple since the lists of indices corresponding to each seed index, $k \leftarrow JA(i)$, are already sorted. The problem is thus one of merging a few ordered lists. Furthermore, once an element common to both lists is encountered, the remainder of both lists is identical (by Theorem 1) and the merging process can be terminated. If *ORDER* contains p entries, the cost of this ordering is certainly bounded by $O(p \log p)$, the cost of sorting a set of p completely unordered indices. This cost is usually much less than the cost of the numerical computations. Since *ORDER* is unordered with respect to the nonzeros in U, an additional vector, *INDEX*, which points to the location in U of each element is required.

If the seed indices in *JA* are ordered by decreasing size, then the sorting at this stage can be avoided entirely. To do this, one should process the seed indices in *JA* in reverse order (i.e., by increasing size) and order the indices as they are generated. This will not in general order all the indices by increasing size, but it will implicitly produce a permutation under whose application the lower triangular matrix remains lower triangular, and the forward substitution can proceed in the usual fashion.

The second step in the factorization procedure is to construct a linked list, *LIST*, of indices for each $k < i$ on the list *ORDER*. In this process, only indices which lie in the intersection of *ORDER* are needed. At the moment, we have no simple way to avoid the computation of the entire linked list for each column k, and intersecting it with that for column i. The intersection procedure itself is very efficient and can be done in $O(1)$ work per entry of *LIST*. On typical finite element *pde* problems, between 50–60% of the row indices computed in this step are discarded. Since for this class of problems the percentage seems to remain bounded independent of N, the order of complexity of the numerical factorization is unimpaired. However, since this computation contributes to the highest order term, it has significant impact on the overall performance of the algorithm. At present, we have no meaningful theoretical bound on the nonnumerical complexity of this step, and this remains the weakest aspect of our procedure.

The numerical factorization requires 5 integer arrays—*M*, *LIST* and *JU* as in FILLIN, plus *ORDER* and *INDEX* as described above. Both these arrays are of length N, so that a total of $5N$ temporary integer storage is used.

4.3. Incomplete factorization. The procedure FILLIN can be adapted in a simple way to compute storage for incomplete **LDU** factorizations to be used in preconditioned iterative methods for solving (1). In particular, an incomplete **LDU** factorization can be computed simply by limiting the length of the sequences generated by the seed indices. For example, if the length is bounded by one, only the seed indices themselves are allowed, and the sparsity pattern is the same as for **A**. If the length is bounded by two, k and $m(k)$ are allowed for each seed index k. This corresponds to allowing one extra level of fillin in the incomplete factorization. In general, if the length is bounded by p, then a maximum of $p-1$ extra levels of fillin are allowed. If there is a constraint on total storage, this procedure could be used in an iterative fashion to determine the maximum fillin level, p, that could be accommodated. All the results of this section carry over transparently to the case of incomplete factorizations.

4.4. Symbolic factorization. The algorithms presented in the proceeding sections are designed to minimize the storage requirements of sparse Gaussian elimination methods. However, it may be desirable to trade space for time by allowing some permanent integer storage. One must be cautious in this approach, since the advantage of our scheme may be diluted if this storage is comparable to standard compressed storage schemes [3].

One possibility is to define an integer data structure JL, of length $N+1+\eta$ (the same length as JA) as follows:

$JL(i) = m(i), \qquad 1 \leqq i \leqq N-1,$

$JL(k-1) =$ starting index in U of seed $JA(k), \qquad N+2 \leqq k \leqq N+1+\eta,$

$JL(N+1+\eta) = 1 +$ ending index in U of seed $JA(N+1+\eta).$

Once generated, JL eliminates the need for the M, $LIST$ and JU arrays in subsequent calculations, and its creation can be regarded as a symbolic factorization. In many problems, there is little or no storage penalty except that now the storage is permanent rather than temporary. For five-point finite difference matrices, JL is about $3N$ in length, as is the combined storage of M, $LIST$ and JU. In the numerical factorization, only one additional array of length N, $INDEX$, is required. The use of the JL array is illustrated in the algorithm for solving $\mathbf{Ly} = \mathbf{b}$

```
      procedure LSOLVE(N, JA, JL, U, y, b)
1.        for i ← 1 to N
2.            sum ← 0
3.            for j ← JA(i) to JA(i+1) − 1
4.                k ← JA(j)
5.                for l ← JL(j−1) to JL(j) − 1
6.                    sum ← sum + U(l)y(k)
7.                    k ← JL(k)
8.            y(i) ← b(i) − sum
      end LSOLVE
```

The order of complexity of LSOLVE is obviously the same as in the previous example, but the nonnumerical overhead is reduced. At present, the compromise between storage and computational efficiency is still an open question and deserves further study.

Appendix: Sample FORTRAN program. This Appendix contains three sample FORTRAN routines for sparse Gaussian elimination. To illustrate the algorithms presented in this paper, we assume the arrays JA and A have been reordered to reflect an appropriate choice of the permutation matrix $\mathbf{P}$. Furthermore, the row indices for each column in JA are given in descending order, to avoid sorting in the numerical factorization step. Subroutine FILLIN computes the JL array as described in § 4.4. Subroutine FACTOR computes the numerical factorization. Subroutine SOLVE performs the forward and backward substitution.

Subroutine FILLIN.

```
      subroutine FILLIN (n, ja, jl, list)
         integer ja(*), jl(*), list(*), uptr
c
c     compute jl
c
c     input: n, ja
c     output: jl
c     workspace: list(n)
c
c     remarks:
c
c         1. for convenience, jl(n) is the size of the strict
c            upper triangular part of the factored matrix
c
c         2. the row indices for each column in ja are assumed
c            to be in decreasing order
c
c     initialize
```

```
c
      do 10 i = 1, n
         jl(i) = 0
10       list(i) = 0
c
c     the main loop
c
      jl(n+1) = n + 2
      do 50 i = 1, n
         if (ja(i) .ge. ja(i+1))  go to 50
c
c     loop over seed indices in decreasing order
c
         list(i) = i
         length = 1
         do 30 iseed = ja(i), ja(i+1)-1
            k = ja(iseed)
            uptr = jl(iseed-1)
20          list(k) = list(i)
            list(i) = k
            uptr = uptr + 1
            length = length + 1
            if (jl(k) .eq. 0) jl(k) = i
            k = jl(k)
            if (list(k) .eq. 0) go to 20
30          jl(iseed) = uptr
c
c     clean up loop for list array
c
         k = i
         do 40 j = 1, length
            ksave = k
            k = list(k)
40          list(ksave) = 0
50    continue
c
c     compute size of upper triangle
c
      lenjl = n + 1 + ja(n+1) - ja(1)
      jl(n) = jl(lenjl) - jl(n+1)
      return
      end
```

Subroutine FACTOR.

```
      subroutine FACTOR (n, isym, ja, a, jl, u, index, ierr)
         integer ja(*), jl(*), index(*), ashift, ushift
         real a(*), u(*), dsum, usum, lsum
c
c     compute u
c
c     input: n, isym, ja, a, jl
c     output: u, ierr
c     workspace: index(n)
c
c     remarks:
c
c          1. isym = 0 means nonsymmetric storage for a and u
c             isym = 1 means symmetric storage for a and u
c
c          2. ierr = 0 is a normal return
c             ierr = i means the i-th diagonal element of the
c                       factored matrix was zero
c
c          3. for symmetric matrices, the computation of lsum in the
c             do 60 loop, and the computation of u(indexk+ushift) in
c             the do 80 loop are redundant
c
      ierr = 0
c
c     compute offsets for a and u
c
      if (isym .eq. 1) then
         ashift = 0
         ushift = 0
         lenu = n + 1 + jl(n)
```

```
      else
         ashift = ja(n+1) - ja(1)
         ushift = jl(n)
         lenu = n + 1 + 2 * jl(n)
      end if
c
c     initialize
c
      do 10 i = 1, lenu
   10    u(i) = 0.0e0
      do 20 i = 1, n
   20    index(i) = 0
c
c     the main loop
c
      do 100 i = 1, n
         dsum = 0.0e0
         if (ja(i) .ge. ja(i+1))  go to 90
c
c     loop over seed indices for column i in increasing order
c
         do 60 iseed = ja(i+1)-1, ja(i), -1
            k = ja(iseed)
c
c     move off diagonal entries from a to u
c
            u(jl(iseed-1)) = a(iseed)
            u(jl(iseed-1)+ushift) = a(iseed+ashift)
c
            do 50 indexk = jl(iseed-1), jl(iseed)-1
               index(k) = indexk
               if (ja(k) .ge. ja(k+1))  go to 50
               lsum = u(indexk+ushift)
               usum = u(indexk)
c
c     loop over seed indices for column k in decreasing order
c
               do 40 kseed = ja(k), ja(k+1)-1
                  j = ja(kseed)
                  do 30 indexj = jl(kseed-1), jl(kseed)-1
c
c     test for intersection between columns i and k
c
                     if (index(j) .gt. 0)  then
                        lsum = lsum - u(indexj) * u(index(j)+ushift)
                        usum = usum - u(indexj+ushift) * u(index(j))
                     end if
c
   30                j = jl(j)
   40             continue
                  u(indexk+ushift) = lsum
                  u(indexk) = usum
   50             k = jl(k)
   60       continue
c
c      clean up loop for index array, compute diagonal
c
            do 80 iseed = ja(i), ja(i+1)-1
               k = ja(iseed)
               do 70 indexk = jl(iseed-1), jl(iseed)-1
c
                  usave = u(indexk+ushift)
                  u(indexk) = u(indexk) * u(k)
                  u(indexk+ushift) = usave * u(k)
                  dsum = dsum + u(indexk) * usave
                  index(k) = 0
c
   70             k = jl(k)
   80       continue
c
   90       u(i) = a(i) - dsum
            if (u(i) .eq. 0.0e0)  go to 200
            u(i) = 1.0e0 / u(i)
  100    continue
         return
c
c        error return
c
  200    ierr = i
         return
         end
```

Subroutine SOLVE.

```
      subroutine SOLVE (n, isym, ja, jl, u, x, b)
         integer ja(*), jl(*), ushift, lshift
         real u(*), x(*), b(*)
c
c     compute the solution x of a * x = b
c
c     input: n, isym, ja, jl, u, b
c     output: x
c
c     remark: isym = 1 means symmetric storage is used for u
c             isym = 0 means nonsymmetric storage is used for u
c             isym = -1 means nonsymmetric storage is used, and the
c                       problem is to be solved with the transpose of a
c
c     compute offsets for u
c
      if (isym .eq. 0) then
         lshift = jl(n)
      else
         lshift = 0
      end if
      if (isym .eq. -1) then
         ushift = jl(n)
      else
         ushift = 0
      end if
c
c     lower triangular system
c
      do 30 i = 1, n
         sum = 0.0e0
         if (ja(i) .ge. ja(i+1)) go to 30
         do 20 iseed = ja(i), ja(i+1)-1
            k = ja(iseed)
            do 10 j = jl(iseed-1), jl(iseed)-1
               sum = sum + u(j+lshift) * x(k)
10             k = jl(k)
20       continue
30       x(i) = b(i) - sum
c
c     diagonal system
c
      do 40 i = 1, n
40       x(i) = u(i) * x(i)
c
c     upper triangular system
c
      do 70 i = n, 1, -1
         if (ja(i) .ge. ja(i+1)) go to 70
         do 60 iseed = ja(i), ja(i+1)-1
            k = ja(iseed)
            do 50 j = jl(iseed-1), jl(iseed)-1
               x(k) = x(k) - u(j+ushift) * x(i)
50             k = jl(k)
60       continue
70    continue
      return
      end
```

REFERENCES

[1] R. E. BANK, W. M. COUGHRAN, JR., W. FICHTNER, D. J. ROSE AND R. K. SMITH, *Transient simulation of silicon devices and circuits*, IEEE Trans. Electron Devices, ED-32 (1985), pp. 1992-2007.

[2] S. C. EISENSTAT, M. C. GURSKY, M. H. SCHULTZ AND A. H. SHERMAN, *Yale sparse matrix package I: The symmetric codes*, Internat. J. Numer. Meth. Engrg., 18 (1982), pp. 1145-1151.

[3] S. C. EISENSTAT, M. H. SCHULTZ AND A. H. SHERMAN, *Algorithms and data structures for sparse symmetric Gaussian elimination*, this Journal, 2 (1982), pp. 225-237.

[4] A. GEORGE AND J. W. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[5] D. J. ROSE, *A graph theoretic study of the numeric solution of sparse positive definite systems*, in Graph Theory and Computing, R. C. Read, ed., Academic Press, New York, 1972.

[6] D. J. ROSE, R. E. TARJAN AND G. S. LUEKER, *Algorithmic aspects of vertex elimination on graphs*, SIAM J. Comput., 5 (1976), pp. 226-283.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 4, July 1987
© 1987 Society for Industrial and Applied Mathematics
008

AN ADAPTIVE GENERAL SPARSE OUT-OF-CORE CHOLESKY FACTORIZATION SCHEME*

JOSEPH W. H. LIU†

Abstract. A practical out-of-core Cholesky factorization scheme is introduced that is based on reorganizations of the matrix data structure during the factorization. It is applicable to the factorization of both dense and sparse matrices. The scheme can be regarded as a simple extension of the conventional in-core sparse factorization method. It is highly adaptive in the sense that it will run successfully in a range of storage sizes. Experimental results on some large sparse practical problems are provided; they show significant reduction in storage requirement for Cholesky factors with little increase (and sometimes decrease) in execution time.

Key words. sparse matrix, out-of-core, Cholesky factorization, elimination tree, postordering

AMS(MOS) subject classifications. 65F50, 65F25

1. Introduction. In the solution of very large linear systems, the use of auxiliary storage becomes a necessity when the storage requirement of the matrix factor exceeds the space available in main memory. Such a problem has been considered by many researchers [4], [14], [15], [22]-[24], [27]. The various out-of-core approaches can be classified according to the way the matrix structures are being exploited: band, frontal, partitioned (or hypermatrix) and general sparse methods.

Let A be a large $n \times n$ symmetric and positive definite matrix, with L as its Cholesky factor. In this paper, we consider a practical out-of-core scheme for the Cholesky factorization of A into LL^T using secondary storage. The method can be regarded as one of the general sparse out-of-core methods, that is, all logical zeros in the factor matrix L are exploited. It is based on a simple observation on data access for the Cholesky factorization algorithm. The central idea of the scheme is to reorganize the matrix data structure throughout the factorization process. That is, garbage collection is performed to remove matrix entries of L that are no longer needed in main memory, and those required are reassigned to locations in the working storage vector for L to provide room for subsequent steps of factorization.

The resulting out-of-core algorithm will only use auxiliary storage for the input of matrix entries from A and the output of the Cholesky factor. No additional secondary storage is required. Though the algorithm may not be directly applicable to extremely huge matrices, it is appropriate for dense matrices of moderate size and for some practical classes of large sparse matrices.

The following is a list of key features of this scheme:

(a) The scheme can be regarded as an "upgrade" of the conventional implementation of sparse Cholesky factorization (e.g., Harwell code [6], Waterloo SPARSPAK [13], Yale YSMP [9]). Each factorization step is the same as before, so that a lot of the code already developed can be shared.

(b) The data structure used is also the same as in these in-core implementations. This means the symbolic factorization and storage setup routines remain unchanged.

* Received by the editors August 13, 1985; accepted for publication (in revised form) April 17, 1986. This research was supported in part by the Canadian Natural Sciences and Engineering Research Council under grant A5509, by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy under contract DE-AC05-84OR21400 with Martin Marietta Energy Systems Inc., and by the U.S. Air Force Office of Scientific Research under contract AFOSR-ISSA-84-00056.

† Department of Computer Science, York University, Downsview, Ontario, Canada M3J 1P3.

(c) The ordering routine for fill reduction can also be used. For example, the minimum degree and the nested dissection orderings will be appropriate.

(d) Though the scheme is mainly targeted for large sparse matrices, it is also applicable for dense systems.

(e) For a given matrix, the algorithm will run successfully within a range of storage sizes. Generally speaking, the more storage available, the less time is required by the algorithm. In other words, the scheme is highly and smoothly adaptive.

(f) The scheme is also very efficient in terms of storage reduction, and has low computational overhead.

In the proposed out-of-core scheme, we assume that there is a working storage vector to store entries in the Cholesky factor L. Throughout the paper, we use "primary storage vector" to refer to this working vector. Note that when the size of this working vector is more than the number of nonzeros in L, the factorization can be performed entirely in core.

An outline of the paper is as follows: In § 2, the scheme is described for the factorization of a dense matrix. It can be extended in a straightforward manner to the sparse case. Section 3 contains an analysis on the primary storage requirement for this out-of-core method on sparse matrices. A discussion on finding equivalent orderings appropriate for the scheme is also included in this section. It is based on the *elimination tree* structure, and is closely related to the analysis result by the author [18] on the working storage requirement for the out-of-core multifrontal method. The usage of the method as part of an overall sparse factorization scheme is also described. In this section, the readers are assumed to be familiar with tree terminologies: subtree, parent/child nodes, root, postordering.

In § 4, experimental results are presented for an implementation of the algorithm. For the 180×180 regular grid problem, it is demonstrated that primary storage for L can be reduced by over 85% with no or little increase in execution time for factorization. Indeed, sometimes the factorization time is actually decreased since reorganizations of storage vector help to reduce the number of page faults drastically in a virtual memory paging environment. The performance of the algorithm on some selected practical problems is given in § 5. Significant reduction in primary storage for L is also achieved. A comparison of the out-of-core and in-core schemes on the total storage requirement (including primary and overhead storage) is also included. Most of the problems tested show more than 60% reduction in total factorization storage.

Section 6 contains the concluding remarks. Some comments are made on a comparison of this scheme with the recent out-of-core TREESOLVE package by Reid [24], and the related out-of-core multifrontal method. Some possible future extensions of the work in this paper are also described.

2. Adaptive out-of-core scheme: dense case. The proposed out-of-core scheme for Cholesky factorization can be most easily described and understood in the case when the matrix is dense. Let A be an $n \times n$ dense symmetric positive-definite matrix. We assume that the lower triangular part of the matrix is stored column by column.

Following [12], we consider the *column-Cholesky* method of determining the Cholesky factor L of A. It is well known that the computation of the jth column of L requires column modifications of the jth column of A by the first $j-1$ columns of L. Furthermore, entries in the first $j-1$ rows of L are no longer needed (see, for example, [11]). In other words, only segments of these columns will be needed in the modification. This observation leads to the following out-of-core scheme, which requires in-core storage reorganization during the course of factorization.

Let ρ be the amount of primary storage available for matrix factorization, and we assume that

$$\rho \leqq n(n+1)/2.$$

Throughout the paper, we shall use $\eta(L_{i*})$ and $\eta(L_{*j})$ to represent the number of nonzeros in the ith row and jth column of L, respectively. Columns of the lower triangle of the matrix A are assumed to be either read in from an external file or generated as required.

ALGORITHM 2.1. Out-of-core Cholesky Factorization.

```
begin
  Open an external file for columns of L;
  Initially, no column is in core;
  For j := 1 to n do
  begin
    If primary storage left < η(L_{*j}) then
      for k := 1 to j-1 do
        repack column L_{*k} by deleting those entries l_{ik}
          with 1 ≦ i ≦ j-1 and compressing the storage vector;
    Append column j of the lower triangle of A to the storage vector;
    Perform column modifications to column j of A
      to form column j of L;
    Output column j of L to the external file;
  end;
  Close the external file;
end.
```

Since the matrix A is assumed to be dense, we have $\eta(L_{*j}) = n-j+1$. Obviously, Algorithm 2.1 assumes that in computing the jth column of L, all the column segments of L required for its modification are in core. This implies that for each $j \leqq n$, we must have $\rho \geqq j(n-j+1)$ so that

$$\rho \geqq (n - \lfloor n/2 \rfloor)(\lfloor n/2 \rfloor + 1) \approx n(n+2)/4.$$

This will ensure that auxiliary storage is used only for input of the original matrix A and output of the factor matrix L. No temporary auxiliary file is required.

It is clear from the algorithm that if $\rho \geqq n(n+1)/2$, the factorization will be performed with no storage reorganization. However, when the value of ρ is close to the minimum of about $n^2/4$, the working storage vector for L needs to be reorganized throughout the factorization.

Let $\rho(r)$ be the minimum amount of primary storage so that no more than r storage reorganizations will occur in Algorithm 2.1. It is easy to verify that

$$\rho(0) = n(n+1)/2 \approx 0.5n^2,$$

$$\rho(1) \approx 3n^2/8 = 0.375n^2,$$

$$\rho(2) \approx (4\sqrt{2}-5)n^2/2 = 0.329n^2,$$

$$\rho(3) \approx 39n^2/128 = 0.305n^2.$$

Therefore, only three storage reorganization steps are required in order to be able to reduce the amount of primary storage for the matrix factor by 40%. It should be noted that there will be a slight increase in overhead for indexing the matrix values in the reorganized storage vector.

$$A = \begin{pmatrix} d & x & x & x & x \\ x & d & x & x & x \\ x & x & d & x & x \\ x & x & x & d & x \\ x & x & x & x & d \end{pmatrix}$$

$$\begin{pmatrix} d & & & & \\ x & d & & & \\ x & x & - & & \\ x & x & - & - & \\ x & x & - & - & - \end{pmatrix} \quad \begin{pmatrix} + & & & & \\ + & + & & & \\ x & x & d & & \\ x & x & x & - & \\ x & x & x & - & - \end{pmatrix} \quad \begin{pmatrix} + & & & & \\ + & + & & & \\ + & + & + & & \\ x & x & x & d & \\ x & x & x & x & d \end{pmatrix}$$

FIG. 2.1. *Storage reorganization for* $n = 5$ *and* $\rho = 9$.

TABLE 2.1
Storage reorganization for $n = 1000$.

r	$\rho(r)$	% Storage reduced
0	500500	0.00
1	375250	25.02
2	328690	34.33
3	304875	39.09
4	290872	41.88
5	281722	43.71
10	262844	47.48
20	254472	49.16
40	251479	49.75
60	250936	49.86
79	250500	49.95

As an example, consider $n = 5$. Figure 2.1 contains the content of the storage vector with size $\rho = 9$ for the steps in the execution of Algorithm 2.1. It is clear that two reorganizations are required. In the figure, the symbol "−" is used for matrix entries that are yet to be processed, and "+" for matrix entries that are out of core and are no longer needed in the factorization. For simplicity, we have used "d" and "x" to stand for diagonal and off-diagonal entries in both A and L.

To illustrate the possible reduction in storage by reorganization, we have tabulated in Table 2.1 the values of $\rho(r)$ for different values of r when the size of the full matrix is 1000. Note that 250500 represents the minimum amount of in-core matrix storage for Algorithm 2.1 to execute successfully; and in such case, the primary storage has to be reorganized 79 times. From Table 2.1, we observe that 254472 serves as a practical lower bound on the storage size, in the sense that further small reduction in core will result in significant increase in reorganizational overhead. The next section will explore the use of Algorithm 2.1 in the out-of-core factorization of sparse matrices.

3. Adaptive out-of-core scheme: general sparse case.

3.1. Primary storage analysis. In most implementations of sparse Cholesky factorization, the lower triangle of the factor matrix L is stored column by column. Moreover, the sparse columns of L are stored in a compact form with all logical zeros removed from the data structure.

It is clear that Algorithm 2.1 in § 2 can be adapted quite easily as a sparse out-of-core scheme for this representation of the Cholesky factor. Indeed, the only difference is that the number of nonzeros $\eta(L_{*j})$ in the jth column of the factor L is no longer $n-j+1$. Henceforth, we shall refer to Algorithm 2.1 as the algorithm for both dense and sparse out-of-core Cholesky factorization.

For a fixed ordering of the matrix A, the analysis to determine the minimum amount of primary storage for the successful execution of Algorithm 2.1 is quite simple. Let m_j be the minimum amount of primary storage required for the processing of column j in Algorithm 2.1. Then we have

$$m_1 = \eta(L_{*1})$$

and, for $j > 1$,

$$m_j = m_{j-1} - \eta(L_{j-1,*}) + \eta(L_{*j}).$$

Therefore, for the given ordering, the required minimum amount of primary storage for Algorithm 2.1 is given by

$$\rho_{\min} = \max \{m_j \mid 1 \leqq j \leqq n\}.$$

Moreover, given a value of ρ not less than $\rho_{\min}$, it is easy to determine the required number of storage reorganizations using only the values of $\eta(L_{j*})$ and $\eta(L_{*j})$. The following algorithm will return r, the number of required storage reorganizations for a given value of ρ.

ALGORITHM 3.1. Storage Reorganization Count.

```
begin
  r := 0;
  NZCOUNT := 0;
  ROWSTART := 1;
  For j := 1 to n do
    begin
      if ρ < NZCOUNT + η(L*j) then
      begin
        r := r + 1;
        for k := ROWSTART to j - 1 do
          NZCOUNT := NZCOUNT - η(Lk*);
        ROWSTART := j;
      end;
      NZCOUNT := NZCOUNT + η(L*j);
    end;
end.
```

3.2. Elimination tree reordering. It is well known for a given ordering or elimination sequence of the sparse matrix, there is a class of equivalent orderings. They are equivalent to the original one in terms of the number of fills and arithmetic operations. Moreover, for a given sparse matrix with a fixed ordering, a tree structure called the *elimination tree* [3], [20], [25] can be associated with its Cholesky factorization process. It is also known that this tree defines a class of equivalent orderings, each having the same *set* of filled edges as the original ordering.

Let A be a given $n \times n$ symmetric matrix and L be its Cholesky factor. Following [20], we define the elimination tree of A as the tree with n nodes $\{1, 2, \cdots, n\}$, and node i is the parent of node j if and only if

$$i = \min \{k \mid l_{kj} \neq 0\},$$

that is, i is the index of the first off-diagonal nonzero in column L_{*j}. Without loss of generality, we assume that the matrix A is irreducible so that the structure is indeed a tree. It should be clear that node n is the root of this tree. Figure 3.1 contains an 8×8 matrix structure and the diagonal entries are labeled by their corresponding equation/variable numbers. Its corresponding elimination tree is displayed in Fig. 3.2. Note that this matrix has the same structure as its filled matrix. This example will be referred to later in the paper.

Consider any ordering P which numbers the children nodes before their parent in the elimination tree. Such an ordering is referred to in the literature as a *topological ordering* of the tree [1]. It is known that the two matrices A and PAP^T have the same set of filled edges when they are being factored [20]. This implies that they are equivalent as far as sparse Cholesky factorization is concerned. From this class of equivalent orderings, we explore ways to find one so as to reduce the amount of primary storage for the successful execution of the out-of-core scheme of Algorithm 2.1.

Naturally, we want to find the best possible topological ordering for Algorithm 2.1. The consideration of this general graph optimization problem will carry us too far afield here. Its solution is quite complicated and will be presented elsewhere in [19]. However, it is interesting to point out that this problem can be formulated as a generalized version of the (black) *pebble game* for trees, which was originally motivated as a model for register allocation for straight-line programs [2], [16], [21].

3.3. Optimal elimination tree postordering. An algorithm to determine the best topological ordering in terms of primary storage requirement for the out-of-core

$$A = \begin{pmatrix} 1 & x & & & & & & x \\ x & 2 & x & & & & & x \\ & x & 3 & & & & & x \\ & & & 4 & x & & x & x \\ & & & x & 5 & x & x & x \\ & & & & x & 6 & x & x \\ & & & x & x & x & 7 & x \\ x & x & x & x & x & x & x & 8 \end{pmatrix}$$

FIG. 3.1. *A matrix example.*

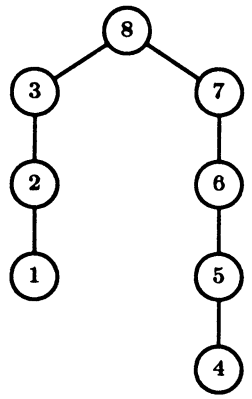


FIG. 3.2. *The elimination tree of matrix in Fig.* 3.1.

factorization scheme is described by the author in [19]. For all practical purposes, we prefer to restrict our search for an optimal ordering from the class of *postorderings* [1] of the elimination tree (that is, orderings obtained by a *depth first search* of the tree). For postorderings, nodes within each subtree are always numbered consecutively. They form quite a significant subclass of topological orderings on the given tree structure. The best postordering will generally give close to an optimal topological ordering for our application. Furthermore, as we shall see in this section, the algorithm to find such a postordering is relatively simple.

The approach to determine the best postordering on the elimination tree for Algorithm 2.1 is similar to the one employed in [18] for the out-of-core multifrontal method. We first express the minimum amount of primary storage $\rho_{\min}$ established in § 3.1 in terms of the structure of the tree. We need a few definitions for the formulation.

Assume that the ordering is already a postordering of the elimination tree. Consider any node j in the elimination tree. We shall use $T[j]$ to denote the subtree rooted at the node j. Define the function

$$\tau[j] = \sum_{k \in T[j]} \{\eta(L_{*k}) - \eta(L_{k*})\}.$$

The value $\tau[j]$ represents the total number of nonzeros in columns of L associated with $T[j]$ that are required in subsequent steps of column Cholesky. In other words, after computing column j of L, we need at least $\tau[j]$ locations to store nonzeros in columns associated with the subtree $T[j]$.

For node j, we let $\mu[j]$ to be the minimum amount of storage required to execute Algorithm 2.1 restricted only to the nodes in the subtree $T[j]$ (in the same relative order). Since n is the root of the elimination tree, the subtree $T[n]$ is the entire elimination tree so that the value $\mu[n]$ will be the same as the desired value of $\rho_{\min}$. Consider any node j. If j is a leaf node in the elimination tree, then

$$\mu[j] = \eta(L_{*j}).$$

On the other hand, let j be an interior node with children nodes $s_1, s_2, \cdots, s_t$ (ordered in $s_1 < s_2 < \cdots < s_t$) as shown in Fig. 3.3.

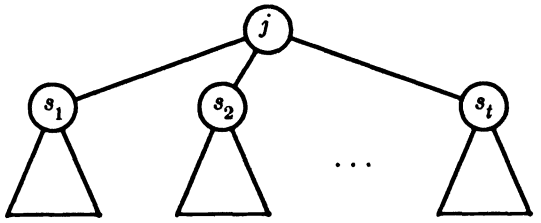


FIG. 3.3. *Subtree $T[j]$ rooted at j.*

In the processing of columns associated with the subtree rooted at the child node s_k, the storage requirement is given by

$$\mu[s_k] + \tau[s_1] + \cdots + \tau[s_{k-1}]$$

since $\mu[s_k]$ is the minimum storage required to process the subtree $T[s_k]$, and $\tau[s_1] + \cdots + \tau[s_{k-1}]$ is the number of nonzeros from the first $k-1$ subtrees that need

to be kept. Therefore, we have

$$\begin{aligned}\mu[j] = \max\{&\mu[s_1],\\ &\mu[s_2]+\tau[s_1],\\ &\cdots\\ &\mu[s_t]+\tau[s_1]+\cdots+\tau[s_{t-1}],\\ &\eta(L_{*j})+\tau[s_1]+\cdots+\tau[s_{t-1}]+\tau[s_t]\}.\end{aligned}$$

Note that the last expression in the formula is the exact number of nonzeros required from columns in $T[j]$ during the processing of column j. This recursive formula is based on the assumption that the ordering is already a postordering. The nodes in the subtree $T[s_1]$ are processed before those in $T[s_2]$, and so on until $T[s_t]$. On applying this formula recursively, we can determine the value $\mu[n]$ at the root.

It is instructive to apply the formula for $\mu[j]$ to the matrix example in Fig. 3.1. It is easy to verify that

$$\mu[3]=5, \qquad \tau[3]=3,$$
$$\mu[7]=8, \qquad \tau[7]=4.$$

Therefore, we have

$$\mu[8]=\max\{\mu[3],\ \mu[7]+\tau[3],\ \eta(L_{*8})+\tau[3]+\tau[7]\}=11.$$

It then follows that for this ordering, 11 primary storage locations for L are required for Algorithm 2.1.

The defining equation for $\mu[j]$ bears a close resemblance to the formula for the working storage requirement for the out-of-core multifrontal method established in [18]. Indeed, we can apply the results from [18] directly to this situation in order to reduce the amount of primary storage. Accordingly, the children nodes should be reordered in decreasing order of the values $\{\mu[s_j]-\tau[s_j]\}$. This will give us the *best possible postordering* on the elimination tree in minimizing the primary storage for the successful execution of Algorithm 2.1. We shall omit the proof here and readers are referred to [18] for the details.

On applying this result to the matrix example of Fig. 3.1, we note that

$$\mu[3]-\tau[3]=2,$$
$$\mu[7]-\tau[7]=4,$$

so that a rearrangement of the children nodes of node 8 should be performed. Indeed, this will correspond to a different postordering of the elimination tree, numbering the nodes 4, 5, 6, 7 before 1, 2, 3. In terms of the matrix, this implies a symmetric permutation moving the first three variables to become the 5th, 6th and 7th variables. In Fig. 3.4, PAP^T is this permuted matrix structure as illustrated by the diagonal labels. It is clear that its primary storage requirement is now reduced to 9.

This equivalent reordering based on the elimination tree can have significant impact on storage reduction for the out-of-core scheme in Algorithm 2.1. Indeed, the matrix pattern in Fig. 3.1 can be generalized to large $n \times n$ matrices to give examples where substantial reduction in primary storage can be demonstrated.

It is also interesting to consider the second reordered matrix QAQ^T in Fig. 3.4, where the reordering is again illustrated by the diagonal labels. This structure suffers one fill (illustrated by "f"), and yet its primary storage requirement is further reduced

$$PAP^T = \begin{pmatrix} 4 & x & & x & & & & x \\ x & 5 & x & x & & & & x \\ & x & 6 & x & & & & x \\ x & x & x & 7 & & & & x \\ & & & & 1 & x & & x \\ & & & & x & 2 & x & x \\ & & & & & x & 3 & x \\ x & x & x & x & x & x & x & 8 \end{pmatrix} \qquad QAQ^T = \begin{pmatrix} 4 & x & & x & x & & & \\ x & 5 & x & x & x & & & \\ & x & 6 & x & x & & & \\ x & x & x & 7 & x & & & \\ x & x & x & x & 8 & x & x & x \\ & & & & x & 1 & x & f \\ & & & & x & x & 2 & x \\ & & & & x & f & x & 3 \end{pmatrix}$$

FIG. 3.4. *Reordered matrix structures for example in Fig.* 3.1.

to 8. This does not contradict the result established in this section, since the reordering given by Q is not a topological ordering (hence not a postordering) of the elimination tree in Fig. 3.2. Indeed, QAQ^T has a completely different elimination tree structure.

3.4. The overall adaptive scheme. The adaptive nature of Algorithm 2.1 is one of its major advantages. When the amount of primary storage available is more than the number of nonzeros in the Cholesky factor, the scheme becomes the conventional in-core factorization method. Otherwise, the scheme will adapt itself to the amount available so long as the amount is greater than a certain minimum. Generally speaking, given more storage, the computation time required by the algorithm is reduced.

The following presents an overall scheme using the ideas in this paper. Since the scheme is suitable for general sparse problems, the reliable minimum degree ordering is used to generate a fill-reducing ordering.

Step 1 (Ordering). Determine the minimum degree ordering of the given matrix structure and generate the corresponding elimination tree.

Step 2 (Reordering). Reorder the elimination tree so that for each interior node, the children nodes are in descending sequence with respect to the values $\{\mu[s]-\tau[s]\}$. Generate the equivalent postordering. Form the composite ordering given by the minimum degree ordering determined in Step 1 followed by this postordering.

Step 3 (Symbolic factorization). Perform symbolic factorization on the reordered matrix to generate the structure of the Cholesky factor L.

Step 4 (Out-of-core preprocessing). Compute the minimum storage requirement $\rho_{\min}$ for the successful execution of the out-of-core scheme. Then determine the exact number ρ of storage locations available to store nonzeros in the factor matrix. If $\rho \geqq \rho_{\min}$, compute the number r of storage reorganizations using Algorithm 3.1.

Step 5 (Numerical factorization). If $r = 0$, perform the conventional in-core sparse Cholesky factorization. Otherwise, if $\rho \geqq \rho_{\min}$, perform the out-of-core factorization as given in Algorithm 2.1.

Implicitly assumed in this overall scheme is that the first four steps can be performed within the given amount of core storage. The author is currently developing a complete out-of-core package based on this scheme to go with the SPARSPAK [13] package. Initially, it will be appropriate for matrices where ordering and symbolic factorization can be performed in core.

4. Experimental results: $k \times k$ grid problem. The symmetric factorization routine from SPARSPAK [13] was modified to implement the out-of-core algorithm of § 2. The basic factorization step remains the same. An input parameter ρ is added to indicate the size of the working storage vector for nonzeros in L. The major change

in the routine is the addition of code for storage reorganization; this code is simply a sequence of real vector moves.

In Table 4.1, results for the $k \times k$ regular grid model problem are presented with $k = 180$. For this problem, the number of equations is 32,400 and the number of off-diagonal nonzeros is 128,522. Note that the number of equations is close to the maximum possible integer representable in a half-word (using 16 bits).

As before, we use the notation $\rho(r)$ for the minimum size of the storage vector required to execute successfully Algorithm 2.1 with r storage reorganizations. A minor variation from before is that the $\rho(r)$ reported does not include storage required by diagonal entries. This is to be consistent with the factorization routine from SPARSPAK, where the diagonal entries are stored in a separate vector.

The factorization time reported is time in seconds on a VAX 11/780 with a floating point accelerator. These experiments were performed in a *single-user mode*, so that both the CPU and elapsed time were obtained with no interference from other programs. To obtain a more accurate comparison, we have excluded the input/output time required for column input of A and column output of L. This is achieved by generating the column entries of A in memory and by having a dummy output routine for columns of L.

The matrix is permuted by the *minimum degree* ordering for reducing fills. The recent implementation of the version with multiple elimination in [17] is used. Note that for this matrix example with this ordering, the number of multiplicative operations for the symmetric factorization is 61,376,815. If we use $r^*\rho(r)$ as an upper bound on the number of assignments required for storage reorganization, for $r \leqq 30$, this value is relatively insignificant when compared with the number of multiplicative operations. The extra overhead incurred will be small relative to the entire factorization process.

With the extra reorganization overhead, we would expect that the factorization time would increase as the amount of primary storage decreases. However, the tabulated result in Table 4.1 shows that there is a drop in execution time initially before it increases. This anomaly can be explained by the amount of page faults incurred on the VAX (running VMS) for the different storage vector sizes. We have included in Table 4.1 the number of page faults for the particular run from which the data is

TABLE 4.1
Storage reorganization and factorization time for the 180×180 *regular grid problem.*

r	Min storage $\rho(r)$	% Reduction in L storage	Factorization time		Page faults
			CPU	Elapsed	
0	1180771	0.00	1102.01	1207.54	171961
1	656188	44.43	1088.44	1129.86	152750
2	462640	60.82	1072.38	1106.00	101609
3	375906	68.16	1069.27	1096.62	97999
4	323934	72.57	1057.54	1087.95	68869
5	285102	75.85	1064.57	1101.88	83452
10	209705	82.24	1068.85	1107.63	62692
15	181003	84.67	1062.09	1102.03	48821
20	168372	85.74	1069.61	1115.41	51078
40	155524	86.83	1106.54	1168.25	74737
60	154233	86.94	1149.58	1217.40	103538
80	153946	86.96	1193.56	1283.55	132364
85	153925	86.96	1203.78	1278.81	139151

obtained. Indeed, there is a reduction by more than three times in page faults when $r = 15$ or 20. The storage reorganization helps to reduce the number of page faults, which will offset the overhead for data movement for small values of r.

For completeness, the factorization elapsed time is also included in Table 4.1. For the case when $r = 4$, there is about 10% reduction in elapsed time. Note that these results were obtained when the program was executed in a single-user mode. The reduction in paging activities, and hence in elapsed time, will be more drastic when it is run in the usual multi-user environment.

The value $\rho(0)$ is the same as the number of off-diagonal nonzeros in the factor matrix L. We note that by allowing $r = 20$, the amount of primary storage for L can be reduced by over 85% relative to $\rho(0)$. Moreover, the overhead from data movement contributes rather insignificantly to the actual CPU time. Indeed, there is actually a slight decrease in factorization time due to a sizable decrease in page faults.

5. Experimental results: selected Harwell–Boeing matrix problems.

5.1. Storage reorganizations. To test the practical usage of the out-of-core scheme, it was applied to a number of selected sparse matrix problems from the Harwell-Boeing collection [5]. We have selected from the collection all the symmetric problems that have more than 3500 equations. Table 5.1 contains this list of problems, and we use the same keys as provided by the Harwell-Boeing tape to identify the different matrices. Here, "# Nonz" is used to indicate the number of off-diagonal nonzeros in the upper triangular part of the given symmetric matrix. Note that similar results were obtained on applying the scheme to (smaller) sparse problems in the collection that arise from different application areas.

TABLE 5.1
Selected Harwell-Boeing test matrices.

Key	n	# Nonz	Description
BCSPWR10	5300	8271	Representation of entire US power network
BCSSTK15	3948	56934	Module of an offshore platform
BCSSTK16	4884	142747	Corp. of engineers dam
BCSSTK17	10974	208838	Elevated pressure vessel
BCSSTK18	11948	68571	R.E. Ginna nuclear power station
BCSSTK21	3600	11500	Clamped square plate
BCSSTK24	3562	78174	Winter sports arena
BCSSTK25	15439	118401	76-story skyscraper

All the matrix examples are permuted by the minimum degree ordering, as implemented in [17]. In Table 5.2, we tabulate the $\rho(r)$ values for some selected values of r. The value r_m is the number of storage reorganizations so that $\rho(r_m) = \rho_{\min}$. The reductions in storage for the Cholesky factors in these matrix examples are also significant. For $r = 20$, the reductions range from 70% to over 90%. Factorization time is not included in Table 5.2; its behavior is similar to those in Table 4.1 for the $k \times k$ grid problem.

5.2. Compressed versus uncompressed structural representation. Most in-core sparse packages use the compressed column structural representation of L by Sherman [26]. However, in the context of out-of-core factorization, the uncompressed form of representation becomes a viable alternative. Indeed, the uncompressed subscripts of the columns can be stored in parallel with the column nonzeros of the factor matrix. In

TABLE 5.2
Storage reorganization for selected problems.

Problem	$\rho(0)$	$\rho(1)$	$\rho(5)$	$\rho(10)$	$\rho(20)$	r_m	$\rho(r_m)$
BCSPWR10	22764	12110	5004	3418	2607	50	2199
BCSSTK15	647274	404304	236705	202628	190518	43	188230
BCSSTK16	736294	392963	185129	136800	113934	54	105299
BCSSTK17	994885	530877	208030	136289	101232	45	91135
BCSSTK18	650777	399278	233284	202822	193070	47	190521
BCSSTK21	86854	43427	19116	13897	11043	49	10118
BCSSTK24	275360	159900	92065	79405	75529	33	74900
BCSSTK25	1401129	741209	311671	212421	159404	85	131832

this way, storage reorganization can be performed on both the subscript and the column nonzero arrays.

To compare the two approaches, we tabulate in Table 5.3 the overhead storage used for the subscript information. From Table 5.2, $r=20$ seems to be a reasonable compromise in terms of storage reduction and reorganization overhead. We shall assume that $\rho(20)$ is the amount of available storage for the factor matrix. Note that the overhead for the compressed storage is the number of integers for all the compressed subscripts together with an index vector of size $n+1$. For the uncompressed scheme, the overhead for subscripts is the same as the value $\rho(20)$. No additional index vector is required, since the one for the factor matrix can also be used to index the subscript storage vector.

The results in Table 5.3 do not offer clear indication as to which scheme for L should be used. Since the out-of-core scheme is treated as an extension to the conventional in-core method, it seems reasonable to retain the compressed column representation. This allows minimal changes to the existing in-core factorization software. Furthermore, it permits a smooth transition from in-core to out-of-core scheme, since both use the same structural representation.

5.3. Total factorization storage requirement. In the actual factorization of a sparse matrix, primary storage is required for nonzeros in the factor matrix L. Additional overhead storage is also required for indexing, structural representation and temporary working vectors. A fair comparison between the conventional in-core and the proposed out-of-core methods in terms of storage requirement should include both the primary and overhead storage.

TABLE 5.3
Compressed/uncompressed structural overhead for $\rho(20)$.

Problem	n	No. of compressed subscripts	Structural overhead for	
			Compressed	Uncompressed
BCSPWR10	5300	15526	20827	2607
BCSSTK15	3948	59860	63809	190518
BCSSTK16	4884	48647	53532	113934
BCSSTK17	10974	89141	100116	101232
BCSSTK18	11948	100912	112861	193070
BCSSTK21	3600	27756	31357	11043
BCSSTK24	3562	21331	24894	75529
BCSSTK25	15439	177650	193090	159404

For the out-of-core scheme, we shall again assume that the amount of primary storage available for the factor matrix is $\rho(20)$. Moreover, the compressed structural representation is used in the comparison. A simple count on the storage vectors used in the symmetric factorization routine in SPARSPAK gives the following overall storage requirement (real: LNZ, DIAG, TEMP; integer: XLNZ, XNZSUB, FIRST; short integer: NZSUB, LINK):

	Real	Integer	Short Integer
In-core:	$\rho(0)+2n$	$3n+2$	$\#\ subs+n$
Out-of-core:	$\rho(20)+2n$	$3n+2$	$\#\ subs+n.$

Here, "*# subs*" is the number of compressed subscripts in the structural representation of the factor matrix L. Table 5.4 tabulates the total factorization storage requirement for the two methods. The counts are obtained by simply adding the number of reals, integers and short integers required.

TABLE 5.4
Total factorization storage for in-core and out-of-core ($r=20$) *schemes.*

Problem	No. of L off-diag Nonz	Total factor storage		% Storage reduced
		In-core	Out-of-core	
BCSPWR10	22764	70092	49935	28.76
BCSSTK15	647274	730824	274068	62.25
BCSSTK16	736294	814247	191887	76.43
BCSSTK17	994885	1149872	256219	77.72
BCSSTK18	650777	823379	365672	55.59
BCSSTK21	86854	136212	60401	55.66
BCSSTK24	275360	318065	118234	62.83
BCSSTK25	1401129	1671415	429690	74.43

The percentage reduction in total factorization storage is quite remarkable. It ranges from 55% to 77% except for the first problem. For this problem "BCSPWR10," note that the number of nonzeros in L is 22764, while the total storage requirement is 70092. This is the only example where the overhead storage exceeds the primary storage, and it explains the small reduction in overall storage. Note that the counts in Table 5.4 assume that reals, integers and short integers all take the same number of bytes. The savings in storage will even be more drastic if nonzeros are complex numbers and/or if short integers are exploited.

6. Concluding remarks. We have presented a simple and yet effective out-of-core extension to the current sparse matrix factorization software. This extension allows very much larger sparse matrices to be factored in the given primary memory space. This method will have significant advantages over conventional out-of-core schemes based on band or frontal approaches, since it is well known that band or frontal methods can suffer many more fills and require significantly more arithmetic operations.

Recently, Reid [24] has developed the TREESOLVE out-of-core package, which uses the *multifrontal* approach [8]. No direct comparison of the performance of TREESOLVE with the algorithm in this paper has been obtained yet. However, some preliminary comparisons will be made based on the experience that the author has on the in-core multifrontal code of MA27 [7]. For very sparse problems, the minimum amount of working in-core storage using the out-of-core multifrontal approach is

usually less than the minimum $\rho_{\min}$ of Algorithm 2.1. The difference will be quite small relative to the number of nonzeros in L. However, for some less sparse problems, sometimes the working storage requirement for the out-of-core multifrontal method exceeds the number of nonzeros in the factor matrix L (for example, see the experiments reported in [18]). In such cases, $\rho_{\min}$ will be much smaller. Indeed, the idea in this paper is applicable even to the case when the matrix is dense.

In terms of performance, we expect that more overhead will be required in the out-of-core multifrontal method in managing a stack of intermediate factor matrices and in matrix assembly. Most of the overhead incurred cannot be removed even if more storage is made available. On the other hand, the only nonnumeric overhead in Algorithm 2.1 is a number of vector moves to reorganize the matrix data structure. Furthermore, less reorganization is required when more memory is available.

TREESOLVE, an adaptation of the out-of-core multifrontal approach for finite element problems, has been designed to tackle huge problems. It has provisions to manage part of the stack and the frontal matrix using secondary storage. Comparatively speaking, Algorithm 2.1, as described, will only handle problems of moderate size. If the amount of primary storage available is less than that of $\rho_{\min}$, the algorithm needs to be modified. In such case, additional secondary storage input/output is required. One possibility is to use auxiliary storage for column segments required for subsequent column modifications. Another alternative is to read in the column segments from the (partially formed) Cholesky factor external file. These are possible extensions of Algorithm 2.1 to handle even larger problems.

As a final note, we relate the work here with the minimal storage sparse elimination algorithm of Eisenstat, Schultz and Sherman [10]. Their objective is to solve a large sparse problem with limited core and their approach discards the columns of the factor matrix no longer required in the factorization. The idea of matrix data reorganization in this paper can be applied to their scheme to provide a different but practical implementation.

Acknowledgment. The author would like to thank the referees for their constructive suggestions, which have greatly improved the overall presentation of this paper.

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *Data Structures and Algorithms*, Addison-Wesley, Reading, MA, 1983.

[2] S. COOK AND R. SETHI, *Storage requirements for deterministic polynomial time recognizable languages*, J. Comput. System Sci., 13 (1976), pp. 25–37.

[3] I. S. DUFF, *Full matrix techniques in sparse Gaussian elimination*, in Proc. Dundee Conference on Numerical Analysis, Lecture Notes in Math. (912), G. A. Watson, ed., Springer-Verlag, Berlin, New York, Heidelberg, 1982, pp. 71–84.

[4] ———, *Design features of a frontal code for solving sparse unsymmetric linear systems out-of-core*, this Journal, 5 (1984), pp. 270–280.

[5] I. S. DUFF, R. GRIMES, J. LEWIS AND W. POOLE, *Sparse matrix test problems*, SIGNUM Newsletter (June 1982), p. 22.

[6] I. S. DUFF AND J. K. REID, *Some design features of a sparse matrix code*, ACM Trans. Math. Software, 5 (1979), pp. 18–35.

[7] ———, MA27—*a set of Fortran subroutines for solving sparse symmetric sets of linear equations*, AERE R-10533, Computer Science and Systems Division, AERE Harwell, Oxfordshire, 1982.

[8] ———, *The multifrontal solution of indefinite sparse symmetric linear systems*, ACM Trans. Math. Software, 9 (1983), pp. 302–325.

[9] S. C. EISENSTAT, M. C. GURSKY, M. H. SCHULTZ AND A. H. SHERMAN, *The Yale sparse matrix package*, I. *The symmetric code*, Research Report #112, Department of Computer Science, Yale University, 1977.

[10] S. C. EISENSTAT, M. H. SCHULTZ AND A. H. SHERMAN, *Software for sparse Gaussian elimination with limited core storage*, in Sparse Matrix Proceedings, I. S. Duff and G. W. Stewart, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1979, pp. 135–153.

[11] A. M. ERISMAN, *Sparse matrix approach to the frequency domain analysis of linear passive electrical networks*, in Sparse Matrices and Their Applications, D. J. Rose and R. A. Willoughby, eds., Plenum Press, New York, 1972, pp. 31–40.

[12] A. GEORGE, M. T. HEATH AND J. W. H. LIU, *Parallel Cholesky factorization on a shared-memory multiprocessor*, Linear Algebra Appl., 77 (1986), pp. 165–187.

[13] A. GEORGE, J. W. H. LIU AND E. G. Y. NG, *User guide for* SPARSPAK, *Waterloo Sparse Linear Equations Package*, Technical Report CS 78-30 (revised 1980), Department of Computer Science, University of Waterloo, 1980.

[14] B. M. IRONS, *A frontal solution program for finite elements*, Internat. J. Numer. Methods Engrg., 2 (1970), pp. 5–32.

[15] H. A. KAMEL AND M. W. MCCABE, *Direct numerical solution of large sets of simultaneous equations*, Comput. Structures, 9 (1978), pp. 113–123.

[16] T. LENGAUER AND R. E. TARJAN, *The space complexity of pebble games on trees*, Inform. Process. Lett., 10 (1980), pp. 184–188.

[17] J. W. H. LIU, *Modification of the minimum degree algorithm by multiple elimination*, ACM Trans. Math. Software, 11 (1985), pp. 141–153.

[18] ———, *On the storage requirement in the out-of-core multi-frontal method for sparse factorization*, ACM Trans. Math. Software, 1986, to appear.

[19] ———, *An application of generalized tree pebbling to sparse matrix factorization*, Technical Report CS-86-02, Department of Computer Science, York University, 1986.

[20] ———, *A compact row storage scheme for sparse Cholesky factors using elimination trees*, ACM Trans. Math. Software, 1986, to appear.

[21] M. C. LOUI, *The space complexity of two pebble games on trees*, Technical Report MIT/LCS/TM-133, Laboratory for Computer Science, Massachusetts Inst. of Technology, Cambridge, MA, 1979.

[22] J. R. PERRY, *Secondary storage methods for solving symmetric, positive definite, banded linear systems*, Research Report #201, Department of Computer Science, Yale University, 1981.

[23] H. S. RASHWAN, *Auxiliary storage methods for sparse positive definite linear systems*, Ph.D. thesis, Department of Computer Science, University of Waterloo, 1983.

[24] J. K. REID, TREESOLVE, *a Fortran package for solving large sets of linear finite element equations*, CSS 155, Computer Sciences and Systems Division, AERE Harwell, Oxfordshire, 1984.

[25] R. SCHREIBER, *A new implementation of sparse Gaussian elimination*, ACM Trans. Math. Software, 8 (1982), pp. 256–276.

[26] A. H. SHERMAN, *On the efficient solution of sparse systems of linear and nonlinear equations*, Ph.D. thesis, Yale University, 1975.

[27] E. L. WILSON, K. J. BATHE AND W. P. DOHERTY, *Direct solution of large systems of linear equations*, Comput. Structures, 4 (1974), pp. 363–372.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 4, July 1987
© 1987 Society for Industrial and Applied Mathematics

EVALUATION OF ORDERINGS FOR UNSYMMETRIC SPARSE MATRICES*

A. M. ERISMAN†, R. G. GRIMES†, J. G. LEWIS†, W. G. POOLE, JR.†‡ AND H. D. SIMON†

Abstract. The computational complexity of obtaining optimal reorderings for performing sparse Gaussian elimination justifies the heuristic nature of all practical reordering algorithms. The concomitant lack of analytic results on heuristic orderings requires that any general comparisons between orderings must be empirical. We report our findings from an extensive investigation of the Markowitz, P^4 and P^5 ordering heuristics applied to a spectrum of sparse matrices. The ordering heuristics differ in overall effectiveness in reducing fill, particularly on certain problem classes, and in their ability to provide stable as well as low fill factorizations. Statistical analyses indicate that certain easily observed, general, structural parameters of the matrices are predictors for the amount of fill observed. Various estimates of the stability of the factorizations were compared for reliability and accuracy. We close with recommendations for use and for further investigation.

Key words. sparse matrices, Gaussian elimination, Markowitz ordering, P^4 algorithm

AMS (MOS) subject classification. 65F05

1. Introduction. It is widely believed that the effectiveness of ordering algorithms for sparse matrices depends in large part on the origins of the problems. Yet, the heuristic nature of these algorithms often does not permit analytic results. Realistic evaluation of heuristic orderings demands the use of a representative set of test problems. This report contains the results of a comparative study on the effectiveness of general purpose algorithms for reordering unsymmetric sparse matrices. The study is empirical and is based upon an expanded version of the Harwell collection of test matrices [7], [9].

It is common folk knowledge that sparse matrices which occur in practice have special features that reflect, in unclear fashion, the scientific origins of the problems. The following quote from [7] indicates this wisdom: "Telling an expert that a matrix has arisen from circuit theory or structural mechanics immediately gives him a considerable feel for its features." While the statement certainly contains a good deal of truth, it is, like most results on sparse orderings, hardly a theorem. It is also irrelevant to the inexpert consumer of sparse matrix software, whose desire is to use a general purpose ordering code to solve the actual problem that has (possibly) arisen in some discipline unknown to the software designer.

The goals of this study were two-fold:

—To study the effectiveness of *orderings* across test problems spanning a wide spectrum of scientific origins;

—To investigate whether any simple measures of problem difficulty can be obtained at less cost than actually ordering the problem.

The first goal differs from those of previous studies, particularly that of Duff and Reid [7], in evaluating orderings rather than codes. The selection of data structures has an undeniable impact on the actual computation and on coding practices, but that was explicitly not of interest in this study. Our purpose was to investigate the validity

* Received by the editors August 25, 1983; accepted for publication (in revised form) May 27, 1986. This paper is a result of research supported by the Air Force Office of Scientific Research (AFSC), under contract F49620-81-C-0072.

† Engineering Technology Applications Division, Boeing Computer Services Company, Seattle, Washington 98124.

‡ Pacific Analysis and Computing Corporation, Bellevue, Washington 98004.

of the folk wisdom, especially in evaluating how different orderings would perform when presented with problems not anticipated by the designers of the heuristics.

The data that are the basis of this study, the algorithms and matrices, are discussed in this introductory section, as are the limitations of this study and the conclusions that can be drawn from it. The second section of this report is a discussion of the symbolic, i.e., nonnumeric, measures and properties of the data. The third section discusses some important numerical properties and behavior. In the concluding sections we summarize the results and conclusions, and indicate further directions suggested by this study.

1.1. Algorithms. In contrast to the rich collection of ordering heuristics available for reordering symmetric matrices, the literature contains few orderings designed for any class of unsymmetric matrices, and much less to serve in a general purpose way. Only one algorithm, the Markowitz reordering [23], is commonly accepted as being general in capability. The P^3 and P^4 algorithms by Hellerman and Rarick [17], [18] are frequently used in linear programming codes, also the origin of the Markowitz ordering. In related work [14] the authors developed a modification of the second of the Hellerman-Rarick orderings, which we have denoted as P^5. Another ordering was developed by Lin and Mah [20]-[22] to solve chemical engineering problems. All of these are based on no special properties other than sparsity, so they could reasonably be applied to quite general problems. Unfortunately, the only implementation of the Lin and Mah algorithm known to the authors is part of a proprietary chemical engineering code and was unavailable to us. Development of a new implementation was beyond the scope of this study, so this ordering was omitted and so were symmetric orderings. These orderings, applied to the connectivity structure of $A+A^T$, are usually viewed by their designers as being applicable only to problems that are nearly symmetric in structure. It has been considered to be ineffective to apply them to problems in which the structure of $A+A^T$ is not close to that of A. They were, therefore, omitted from this study as being not general in purpose.

In applying these orderings to sparse matrices, we precede the ordering with a preordering that places the sparse matrix in a lower block triangular form in which each diagonal block is irreducible. The factorization of a matrix in this form can be performed by independently factoring each diagonal block and, in addition, without causing fill in the blocks in the strict lower triangle [4], [14]. Consequently, the ordering heuristics and the factorization work can be confined to a series of subproblems of smaller order. Fast algorithms have been developed for this task [2], [5], [6], [29]. In general the block reduction results in smaller time and arithmetic requirements, dramatically so if the block form results in diagonal matrices of small order only. There are counterexamples in which ordering the entire matrix is more effective, but generally block triangular reduction is an accepted practice, questioned only for problems expected to be (essentially) irreducible. Indeed, the essential distinction between the P^3 and P^4 algorithms is that the P^4 algorithm is the P^3 algorithm preceded by a reduction to irreducible block triangular form. (Consequently the P^3 algorithm, per se, was not used in this study.) Section 2 contains further discussion of this preordering and of its effectiveness in reducing the complexity of the reordering work for the problems in our test set.

The heuristic due to Markowitz has the simplest statement of the three orderings used in this study, although it is the most complex to implement. It yields a local minimization of the amount of computational work. At each stage in the factorization row and column interchanges are performed so that the leading diagonal element of

the reduced matrix, the next pivot entry, is any element that minimizes the number of operations required at this step of elimination. The reordering can be also considered as a series of local decisions to reduce the amount of fill that occurs, because the minimal operation choice also minimizes a bound on the number of new fill entries. The computation, however, requires that the sparsity structure of the current reduced matrix be known; the ordering routine must perform a partial symbolic factorization as it proceeds. Dynamic data structures capable of representing the structure of the factored matrix, as opposed to the original matrix, are required.

Arbitrary tie-breaking is a common occurrence in heuristic reorderings. All of the orderings of interest to this study contain decision steps in which essentially arbitrary choices are made, as, for example, which to choose among several entries that equally reduce the bound on fill. In some sense then, our study is a study of particular members of classes of orderings.

For a Markowitz type of ordering we used the MA28 code developed by Duff at Harwell [3], [4]. This code has some features that distinguish it from the original ordering by Markowitz, but it is widely accepted as a general purpose ordering and as a Markowitz ordering. A particularly valuable feature of the MA28 code is its attention to numerical stability. The data requirements of the Markowitz ordering are such that it is relatively easy to perform the numerical factorization directly in conjunction with the ordering. This makes the numerical values as well as the structure of the reduced matrix available to the ordering. The numerical values are used by MA28 to control the stability of the factorization by rejecting the natural Markowitz choice if it would cause too much immediate growth in the elements of the factors of the matrix. In such a case an alternative pivot is chosen that minimizes the bound on fill among all the entries that are acceptable on growth grounds. How much growth is acceptable is specified by the user through a tolerance setting which controls the compromise being made between preserving the sparsity of the problem and maintaining the stability of the factorization.

In the study MA28 was used in a mode as near to pure Markowitz as possible by setting the pivot tolerance to zero. The Markowitz ordering may still be perturbed for numerical reasons, but only when the natural Markowitz choice would be accidentally an exactly zero entry. To evaluate the general effectiveness of MA28 as a numerically stable algorithm, corresponding additional tests were made using a particular nonzero value, one tenth, for the growth tolerance.

The P^4 ordering has a significantly more complicated description. The clearest explanation is found in [14]. This heuristic differs from the Markowitz heuristic in two particularly important ways. First, the heuristic is based on obtaining a particular form for the matrix, as opposed to locally reducing fill. The desired form is essentially a lower triangular matrix, in which a few occasional columns known as spikes stick up above the main diagonal. In such a form fill occurs only in the spike columns. This may be used to create less complicated data structures for the factors than are used in MA28. It is also advantageous for updating factorizations when isolated columns of the matrix are changed, as occurs in linear programming problems [26].

A second significant distinction between the P^4 heuristic and the Markowitz heuristic are the data structure requirements for the ordering. Where the Markowitz ordering requires access to the filled reduced matrix, the P^4 ordering proceeds with a static representation of the original matrix. The computational complexity of the ordering is apparently much less than that of Markowitz; an efficient implementation will be much faster and require less storage than a Markowitz ordering. On the other hand, the simple data structures do not allow knowledge of the numerical values that

will arise in the factored matrix. Accidental zero pivots may occur, as may small pivots, which cause unacceptable growth. Indeed, the heuristic may result in nonaccidental structurally zero pivots on the diagonal of the reordered matrix.

The natural way of avoiding such failures of this algorithm requires interchanging the column in which the unacceptable pivot occurs with some particular spike column, which is ordered later in the matrix. This does not affect in any way the ordering of any other columns, so it is usually incorporated in the numerical factorization, which occurs after the ordering. While this maintains the simplicity of the ordering heuristic, it does not allow the heuristic to interact with the changes that are made in the resulting matrix. The result is that fill may increase significantly. Gill et al. [15] and Saunders [26], [27] indicate that this is often the result for even limited control on element growth, and their results do not include measures to demonstrate that the resulting factorizations were acceptable.

The problem of nonaccidental zero pivots in a P^4 ordering can also be avoided by making minor changes in the heuristic. One particular way in which this can be done is the so-called P^5 algorithm, which results from earlier work of the authors. This algorithm is very similar to the P^4 heuristic in its complexity, but the reordered matrix has more structure than that resulting from either P^4 or Markowitz. Advantage of this structure can be taken to reduce dramatically the storage requirements (effective fill) for the factors of the matrix. (The explicit fill can be shown to be never less than for the P^4 ordering). Details are given in [14] and in § 2. Unfortunately, avoiding structurally zero pivots does not guarantee numerical stability. The requirements for stability appear to conflict with the special structure of the factored matrix. Low fill and numerical stability appear to be in direct conflict with this ordering, as is discussed further in § 3.

Two implementations of a P^4 ordering were available to us, one due to Bisschop, Levy and Meeraus at the World Bank [1] and the other by Murtagh and Saunders at Stanford [24]. These differ in slight respects of tie-breaking, which we judged after some experimentation to be insignificant. The World Bank code was more easily modified to create a P^5 code, and so the results herein are for

—Markowitz—the Harwell MA28 code;

—P^4—the World Bank P^4 code;

—P^5—the BCS P^5 code based on the World Bank P^4 code.

1.2. Evaluation program. The goals of this study depend on measuring effectiveness of orderings independent of details of implementation. The variability of implementation was removed by using the same code for the symbolic and numerical factorization of the sparse matrices. The primary measures of effectiveness of orderings were judged to be fill and numerical stability. A program was written to extract these statistics from comparable factorizations based on the Markowitz, P^4 and P^5 orderings. In addition, the program computed a number of statistics on the distribution of nonzeroes in the original matrix and on the effect of block triangularization. These included a series of measures of regularity and symmetry in the original matrix, as discussed further in § 2.

To isolate ordering properties from details of factorization implementation, the orderings were extracted from the respective codes as permutation vectors. The actual factorizations were performed then by a common code, the so-called refactor code, MA28B, from the Harwell package. In the case of the Markowitz orderings, with and without pivoting, we were able to use the package directly. We first factored the matrices with MA28A, and then refactored identically the same matrices with MA28B. For the other two orderings the permutation vectors were input to a symbolic factorization

subroutine extracted from the MA28A code. The fill is, of course, independent of the factorization implementation. Storage requirements and numerical details do depend on implementation; ours are at least comparable because the implementation is the same for all orderings.

The common factorization module allowed us to compare the effects of the various orderings in general, but it also imposed some limitations on the study. The most important is that no provision was made to investigate the effect of pivoting on the P^4 heuristic. Part of the design of the study was to investigate the effectiveness of P^4 and P^5 as pure preordering schemes. The program design sufficed for this investigation, but makes further investigation of P^4 more difficult. In addition, whenever a structurally zero pivot resulted from the P^4 heuristic, no symbolic factorization could be performed. Although the frequency of this failure was one statistic to be explored in this study, it occurred so frequently as to reduce the overall value of the statistics on the other orderings. The design also affected the evaluation of the P^5 heuristic, since it provided an explicit factorization, which did not use the highly structured matrix form. No meaningful operation counts on the corresponding implicit factorization were obtained, although fill statistics were. The common factorization routine also prohibited using dense code, perhaps with pivoting, for the dense diagonal blocks produced by P^5. This lack of pivoting may have contributed to some of the numerical failings of the P^5 ordering.

1.3. Matrices. Fifty-eight unsymmetric matrices of various orders were used to test the orderings. Many of these were collected by Duff and Reid [7]. The others were collected by either Duff or the authors, to supplement the coverage of the earlier collection. For purposes of description and of analysis in this study, the fifty-eight problems were classified into six groups. These were

—Chemical Engineering Plant models (sixteen matrices, of order from 59 to 2021; eleven from Prof. A. Westerberg at the University of Pittsburgh; five matrices from Imperial College);

—Linear Programming Bases (sixteen matrices, matrices 12 to 27 in [7], supplied by Michael Saunders);

—Partial Differential Equation models on grids (seven problems, of order 80 to 3564; three from Dr. P. Saylor at the University of Illinois; three 3-D steam models of petroleum reservoirs; one from J. P. Whelan at Philips, Ltd.);

—Simulation matrices with poor numerical behaviors (eleven matrices of order 115 to 1107; four are from atmospheric pollution studies, matrices 33-36 of [7]; seven are computer science simulations from Francois Cachard, Grenoble);

—Miscellaneous (six matrices of order 32 to 199; matrices 4 to 8 and 11 from [7]);

—Optimal power flow problems (two matrices of order 4929 from Rob Burchett, General Electric Company).

Further details on these matrices, including order and initial nonzero density figures, can be found in Table 1.1.

It is clear that these problems cannot stand, in any democratic or statistical sense, as representatives of the universe of unsymmetric sparse matrices. Readers who find this collection unrepresentative of their interests are urged to help correct the situation by contributing to the ongoing effort by Duff and the authors to provide better bases for such studies as this one [9]. This collection does, however, span a wide enough spectrum of problem types to illustrate rather wide differences in behavior among the various orderings across the various classes of problems. The limitations of this test set for testing the dependence of problem complexity on its origin are illustrated by

TABLE 1.1
Test matrices.

Origin	Order	No. of nonzeros
1) Chemical Engineering Plant Models		
from Prof. A. Westerberg, University of Pittsburgh		
Cavett problem	67	294
Rigorous flash unit	132	414
Evaporator/condenser/compressor	156	371
Redlich-Krong Equations of State	167	507
Multiply-feed column	381	2157
Staged column with 8 stages	479	1910
Flash unit with recycling	497	1727
Staged column with 16 stages	655	2854
Staged column with 7 stages	989	3537
Staged column with 11 stages	1105	5445
Staged column with 15 stages	2021	7353
from Imperial College		
Cavett's process	59	312
Ethylene plant	137	411
Heat exchange network	207	572
Hydrocarbon separation	225	1308
Nitric acid plant	425	1339
2) Linear Programming Bases		
from Michael Saunders, Stanford University		
Stair (after 0 iterations in simplex method)	663	1687
(after 200 iterations)	663	1726
(after 400 iterations)	663	1712
Shell (after 0 iterations)	363	2454
(after 200 iterations)	363	3068
(after 400 iterations)	363	3157
(after 600 iterations)	363	3279
BP (after 0 iterations)	822	3276
(after 200 iterations)	822	3807
(after 400 iterations)	822	4028
(after 600 iterations)	822	4172
(after 800 iterations)	822	4534
(after 1000 iterations)	822	4661
(after 1200 iterations)	822	4726
(after 1400 iterations)	822	4790
(after 1600 iterations)	822	4841
3) Simulation Matrices (with poor numerical behavior)		
from Francois Cachard, Grenoble		
Simulation studies in computer science.	115	421
Two of the matrices have identical structure, but	185	1005
differ in their numerical values.	216	876
	216	876
	343	1435
	512	2192
	1107	5664
from A. R. Curtis, Harwell		
Matrices of identical structure and different	541	4285
conditioning, from atmospheric pollution studies	541	4285
involving chemical kinetics and 2D transport,	541	4285
from the FACSIMILE stiff ODE package.	541	4285

TABLE 1.1 (*continued*)

Origin	Order	No. of nonzeros
4) Partial Differential Equations on Grids		
from Paul Saylor, Univ. of Illinois		
14 by 17 2D grid	238	1128
10 by 10 by 10 3D grid	1000	3750
33 by 6 by 18 3D grid	3564	22316
collected by Roger G. Grimes, from thermodynamic simulations of oil reservoirs		
4 by 4 by 5 grid, 3 degrees of freedom	240	3762
5 by 5 by 6 grid, 4 degrees of freedom	600	13760
20 point 1D grid, 4 degrees of freedom per element	80	928
J. P. Whelan, Philips Ltd.	991	6027
5) Miscellaneous		
Matrix advertising 1971 IBM sparse matrix conference	32	126
Biochemical ODE Jacobian (A. R. Curtis)	54	291
Jacobian from emitter-follower circuit (R. Willoughby)	57	281
Statistical application (M. Gentleman)	113	655
Jacobian from laser ODE system (A. R. Curtis)	130	1282
Stress analysis matrix (R. Willoughby)	199	701
6) Optimal Power Flow		
from R. Burchett, General Electric		
Western utility systembasis 0	4929	33185
Western utility systembasis 1	4929	33111

the fact that we originally broke the "chemical engineering" matrices into two classes, because of their different effects on two of the ordering algorithms. The behavior on the third heuristic was quite similar in both groups, an argument that they should not be regarded separately. Similarly the miscellaneous group includes three small Jacobians from ordinary differential equations, whose numerical properties differ greatly from the larger set of simulation problems. The optimal power flow problems were obtained on recommendation of Michael Saunders only after the formal part of the study had been completed. The very distinct behavior of one of the ordering heuristics on these matrices warranted their belated inclusion in this paper. Thus they are listed as a separate group and are not included in our statistical analysis in § 2.

As a test set these matrices do provide a range of problems to test the capabilities of the orderings. As will be shown in the following sections, they suffice to show that the "general purpose" sparse matrix ordering heuristics do differ in their robustness. Our goal of providing relative evaluation of the orderings can be met with these problems; they provide only a direction towards our goal of providing means to anticipate the expert's analysis of problems.

1.4. Limitations. In many ways any study of this sort must be incomplete and limited, and this study is no exception. The size of the test set is inadequate to draw deep conclusions. For example, we have only two sources of chemical engineering problems, and a very small number of problems in other significant disciplines. This is particularly unfortunate in limiting the value of statistical analyses of certain density properties of the matrices. These analyses indicate differences among the problem classes that correspond to differences in ordering behavior. But they cannot be deemed to be significant; instead, they are only interesting and provocative. They are discussed in more detail in the following sections.

Our selection of algorithms is also limited. We have omitted the Lin and Mah algorithm because of the difficulty in incorporating it. There may be other algorithms in other disciplines, which are effective, but unknown to the authors. Several reasons led us to omit a post-processing phase in the factorization to stabilize numerically the P^4 ordering. One was our desire to use a common factorization module for all orderings to ensure that the results were comparable. Our initial testing indicated that the P^4 ordering was less effective in general than Markowitz; since the required pivoting can be expected to increase the fill we felt the results would not change any conclusions. This is reinforced by the results from [15], [26], [27], which indicate that the fill in P^4 orderings is very sensitive to the number of column interchanges. However, these results are not general; the arguments that fill will increase are substantial, but not proofs. As the results stand, our P^4 results are at least parallel to those from P^5, which we do not yet know how to stabilize efficiently.

A final limitation on this work is its emphasis on generality. As such it represents a bias against the importance of nearly structurally symmetric problems. Interesting problems not addressed in this study because of this bias are discussed in § 4.

2. Symbolic properties. Among the purposes of this study were the investigations of several purely symbolic aspects of general unsymmetric sparse matrix reorderings. These included

—Determining if any or all of the ordering algorithms are generally successful in obtaining low fill and low work factorizations;

—Determining if there are simple statistical measures of matrix properties that can predict how well specific matrices will be reordered.

Each of these leads to a series of further questions to be answered, which are addressed in this section. The discussion of storage and work reduction is addressed first, followed by the suggestive work on predicting which problems will be difficult.

2.1. Symbolic (preordering) effectiveness of ordering heuristics. A sparse factorization, one with low fill and low operation counts, is a paramount goal of these reordering heuristics. Several factors in addition to the heuristics contribute to achieving this goal and to our being able to measure the results. One of these is the effectiveness of the block triangularization preordering. Failures of the evaluation code are another. The evaluation program was not designed to extend a symbolic factorization beyond a structurally zero diagonal element, if such occurred with the P^4 ordering. Further, the MA28 underpinnings of the evaluation code are not capable of adjusting space among the three required and separate workspaces when the space in just one of the workspaces was exhausted. Several of the largest (and most expensive) evaluation runs failed when one of the MA28 workspaces was inadequate. Unfortunately, our resources were also inadequate to allow repeated runs to obtain a proper distribution of space. For these reasons, analyses of all of the matrices were not possible.

Block triangularization. All of the ordering heuristics in this study were preceded by a preordering phase in which a permutation to the finest block lower triangular form of the matrix was found. An implementation of Tarjan's algorithm in the World Bank P^4 code was used for this reduction in conjunction with the P^4 and P^5 heuristics. The separate option in MA28, also Tarjan's algorithm, was used when the Markowitz heuristic was applied. The uniqueness of the reduction up to permutations within the blocks (see [2]) guarantees that the results of the two different implementations are comparable. In both cases, contiguous diagonal blocks of order one were coalesced into larger, triangular, blocks.

Tables 2.1 and 2.2 demonstrate the effect of block triangularization on our test matrices. The first table shows the average number of equations, average number of

TABLE 2.1
Block triangularization statistics for the classes of test matrices.

Problem class	No. of matrices	Average order	Ave. no. diagonal blocks	Ave. no. nontrivial diag. blks.	Ave. percent of columns in nontrivial diag. blks.	Ave. size of nontrivial blocks
Chemical engineering	16	506.4	6.9	3.3	62.1	122.5
Linear programming	16	677.4	21.8	10.50	29.4	19.0
Simulations	11	441.6	1.4	1.00	99.9	441.3
PDE grids	7	959.0	1.7	1.29	93.4	694.8
Miscellaneous	6	97.5	2.2	1.33	96.3	69.4
Optimal power flow	2	4929.0	2.5	1.00	92.6	4565.5
Overall	58	706.1	8.7	4.33	68.6	235.8

TABLE 2.2
Significance of block triangularization for the classes of test matrices.

Problem class	Matrices with no reducibility	Matrices with insignificant reducibility	Matrices with significant reducibility
Chemical engineering	0	2	14
Linear programming	0	0	16*
Simulations	7	4	0
PDE grids	5	0	2
Miscellaneous	3	2	1
Optimal power flow	0	2	0
Overall	15	10	33

* Includes 5 lower triangular matrices.

diagonal and nontriangular diagonal blocks, the average size of the nontriangular blocks, and the average percentage of columns in nontriangular diagonal blocks. These numbers are presented by problem class as well as the overall numbers, which highlights a clear, and understandable difference between the classes of problems. The partial differential equation matrices arise from grid discretizations, which are irreducible unless there are inactive cells or internal boundary conditions. The simulation and optimal power flow problems are representative of models of closed loop systems, but not of other classes of simulations. The problems from the other disciplines show large reductions in the order of the matrices that actually must be factored. Nothing in our results argues against the conventional wisdom that this preordering should be performed, unless there are reasons to believe that the problem is irreducible. Summary figures, by origin, are given in Table 2.2, which shows the distribution of matrices, for which the block triangularization significantly reduced the effective order of the problem.

Fill characteristics. Statistics on the factors of the matrices were obtained from the common symbolic factorization routine adapted from MA28A. The statistic denoted by "explicit fill" is the increase in the number of nonzeros stored to represent the factors L and U over the number of nonzeros in the original matrix A. Since all of the matrices were partitioned into a block lower triangular form, the explicit fill is exactly the collective fill in the nontriangular diagonal blocks.

Within the block lower triangular form each diagonal block acquires a structure from the ordering heuristic. The Markowitz ordering is generally unstructured, with

some increases in density towards the lower right corner of the factors. The P^4 heuristic confines the fill to the spike columns, which can be used to simplify the factorization or auxiliary computations in some contexts. Use of this may change the data structures, but does not change the fill statistics. The diagonal blocks resulting from the P^5 ordering have a very explicit structure; they are block bordered lower triangular matrices. This structure allows us to use implicit block factorization techniques, in which the only fill occurs in the final diagonal subblock of each (outer) diagonal block. This is discussed in detail in [14]. The actual factorization implemented in our evaluation program used the standard explicit approach, which gave us explicit fill statistics for this algorithm. The special character of the implicit factorization and statistics extracted on the partitioning obtained by the P^5 code allowed us to compute the fill that would have been required by an implicit factorization. We refer to this statistic as "implicit fill." Unfortunately, storage requirements and operation counts for this alternate factorization were not available.

The Markowitz ordering was used in two modes, one as pure a Markowitz ordering as can be obtained from MA28, the other as a hybrid ordering combining the Markowitz heuristic with limited pivoting for numerical stability. Although the discussion of numerical properties is the subject of the next section, the fill statistics for the stabilized version of the Markowitz heuristic are presented here for comparison.

Table 2.3 gives the average of actual fill of the various orderings with respect to the various problem classes. Table 2.4 gives the average percentage of fill relative to the original number of nonzero entries for the same problems.

TABLE 2.3
Average of actual fill for the classes of test matrices.

Problem class	No. of matrices	Explicit fill MA28 (.0)	Explicit fill MA28 (.1)	Explicit fill P^4	Explicit fill P^5	Implicit fill P^5
Chemical engineering	10	126.2	142.4	244.1	262.7	63.5
Linear programming	6	17.0	17.7	23.2	23.8	3.2
Simulations	4	2227.5	2276.5	2099.0	3140.8	485.3
PDE grids	4	12866.5	9935.0	10356.0	10818.5	1440.5
Miscellaneous	6	170.0	186.2	328.0	416.0	53.3
Optimal power flow	2	18518.0	18561.0	—	218466.0	29495.5
Overall	30	2092.0	1716.4	1812.3	2036.8	289.2

TABLE 2.4
Average percentage of relative fill for the classes of test matrices.

Problem class	No. of matrices	Explicit fill MA28 (.0)	Explicit fill MA28 (.1)	Explicit fill P^4	Explicit fill P^5	Implicit fill P^5
Chemical engineering	10	22.1	24.5	35.9	41.5	11.4
Linear programming	6	.5	.5	.6	.6	.1
Simulations	4	265.6	268.3	259.7	380.6	59.3
PDE grids	4	182.5	157.0	176.0	184.9	23.7
Miscellaneous	6	37.3	41.0	68.0	88.0	14.8
Optimal power flow	2	55.9	56.0	—	659.1	89.0
Overall	30	74.6	73.2	83.8	106.9	17.9

These statistics were restricted to the subset of problems where both the ordering and the symbolic factorization were computed. Five of the test problems were excluded because the storage demands for the Markowitz code were excessive, six failed for the same reason for the P^4 heuristic, and the same failure occurred seven times for the P^5 code. The most frequent case of failure in the symbolic factorization was due to the P^4 ordering, which produced structurally zero diagonal entries for twenty-two of the problems. This is itself a surprising result. Table 2.5 contains the distribution of these P^4 failures by problem origin. Overall these failures reduced the set of test matrices for which comparable results were obtained from the total of fifty-eight to thirty matrices.

TABLE 2.5
Distribution of P^4 failures for the classes of test matrices.

Problem class	No. of storage aborts	No. of symbolic factorization aborts	No. of numeric aborts	No. of numerically unstable factors*	No. of complete successes
Chemical engineering	0	6	4	1	5
Linear programming	0	10	0	0	6
Simulations	5	2	0	2	2
PDE grids	1	2	3	1	0
Miscellaneous	0	0	0	2	4
Optimal power flow	0	2	0	0	0
Overall	6	22	7	6	17

* As defined in § 3 as failures.

Because of their unusual fill statistics the results for the optimal power flow problems are listed separately in Tables 2.4 and 2.5. Since P^4 failed for these problems, the results for these problems are not included in the overall figures.

The following general observations can be made on the data in the above tables on fill statistics. They are

—The cost in increased fill of incorporating numerical safeguards in the Markowitz heuristic is slight. The fill for MA28 with a nonzero pivot tolerance is never much more than the fill for the pure Markowitz heuristic; on average it is actually less.

—The explicit fill for P^4, even for the problem classes where P^4 is commonly used, is almost always greater than Markowitz with pivoting.

—The explicit fill for P^5 is always worst.

—The implicit fill for P^5 is always less, sometimes significantly so, than the explicit fill of all the other orderings, except for the optimal power flow problems.

—The problem classes with the largest amount of fill are the PDE problems, the simulation problems, and the optimal power flow problems. These are also the classes for which the block triangularization was least effective.

Overall, both the Markowitz and the implicit P^5 heuristics perform well across a spectrum of problems. The P^4 heuristic is not much worse in its performance when it works, but it frequently fails as a strict preordering heuristic because of structural zero pivots. The implicit fill results indicate that further research on the P^5 algorithm may prove fruitful, although the large amount of implicit fill for the optimal power flow problems indicates that P^5 may be far less a general purpose reordering than originally

expected. However, as is discussed in § 3, a numerically robust version of the P^5 algorithm does not yet exist. By contrast, the Markowitz heuristic, constrained for stability, functions well in general.

The last observation is that the PDE, simulation, and optimal power flow problems are the hardest of these test problems. It was expected that the PDE problems would have large fill as they arise from well connected grid discretizations, which yield irreducible matrices. Similarly the classes of simulation problems and of optimal power flow problems show insignificant irreducibility (see Table 2.2) and the comparatively large amount of fill came as no surprise. What was surprising was the fact that both problem classes also exhibited poor stability (see Table 3.2). That these two factors were correlated was unexpected. Although this is too small a sample to draw conclusions, this correlation should be investigated further.

There are other important statistics on the performance of the ordering heuristics, in particular, factorization storage requirements and operation counts. These statistics were not available for the implicit factorization for the P^5 heuristic. They are correlated with the fill statistics for the explicit factorization, since we used the same data structures and factorization routines throughout. For these reasons, we confined our attention to performance in terms of fill.

2.2. Statistical predictors for fill. Sparse matrix ordering heuristics are not universally effective. The relative performance between heuristics usually depends on the type of problem being reordered. Certain heuristics are known to be best for certain problem classes, and not best for others. The algorithms and test cases in this study show this behavior only to a limited degree; the more extensive set of orderings for symmetric matrices illustrates this point more. One of our goals, however, was to investigate the extent to which problem origin affected the outcome of our unsymmetric ordering heuristics and to search for predictors for the fill that occurs. Although our ordering heuristics fail to show spectacular relative differences, our classes of test matrices do. The PDE matrices and the simulation matrices are clearly more difficult to reorder well than are the other classes. Can this be predicted from observable properties of the original matrix?

We measured a number of variables that may indicate properties of the test matrices. For simple density considerations we computed and saved the number of nonzeros in the original matrix, and also the number in the lower and upper triangles of the original matrix, respectively. We attempted to measure the regularity of the distribution of nonzeros by computing the minimum, the maximum and the standard deviation of the nonzeros by row and by column respectively. Several measures of matrix structural symmetry were developed that indicate how close a matrix is to structural symmetry. The measure used in the analyses in the section is the percentage of entries in the matrix for which the corresponding entry in the transpose of the matrix is also nonzero. Statistics were kept on the number, size and average size of nontrivial diagonal blocks found by the block triangularization.

These measures of density, regularity and symmetry were compared by class with the performance of the ordering heuristics. The two classes with the highest relative fill contrasted with the other classes, according to these measures. The class of PDE matrices, for example, has the highest average symmetry ratings, has high initial density, and is more regular than the other classes. The simulation matrices on the other end are much less regular. Their density, as measured by the average of nonzeros by row (column), is typical for the test set, but the nonzeros are arranged irregularly. They have high maximum column counts and a high standard deviation of column counts.

By contrast, the linear programming matrices have high maximum and standard deviation of row counts, with similar density.

Table 2.6 contains the measures for density, regularity and symmetry. On the basis of these tables, the PDE problems stand out for the reasons suggested above. The LP and simulation problems stand out for their irregularity, and there are few other obvious differences. The major difference between the LP and the simulation problems would appear to be the degree to which they split into smaller irreducible problems. We might have left this topic with the conclusion that this more costly measure represents the most significant difference between the problem classes. Instead, we proceeded with further statistical analyses.

TABLE 2.6
Average density, symmetry and regularity measures for the classes of test matrices.

Problem class	Average nonzeros per row/column	Nonzeros per row maximum	Nonzeros per row standard deviation	Nonzeros per column maximum	Nonzeros per column standard deviation	Overall symmetry rating
Chemical engineering	3.84	11.81	2.41	22.88	3.51	0.05
Linear programming	5.49	256.38	11.83	20.50	4.64	0.01
Simulations	5.68	7.73	1.28	200.36	8.94	0.46
PDE grids	10.14	13.71	1.96	13.71	1.89	0.99
Miscellaneous	5.57	30.17	4.30	32.33	5.17	0.54
Optimal power flow	6.70	35.50	3.00	28.00	3.40	0.20
Overall	5.69	81.45	4.95	55.93	5.57	0.27

We performed a series of regression analyses to determine how well the measures discussed above predict fill and relative fill. We used the widely available SPSS statistical package [25] to compute step-wise regressions of the fill statistics for each ordering against the symmetry rating, the standard deviations of column and of row counts (measures of regularity), the number of diagonal blocks produced in the block triangularization, and the original number of nonzeros. The relative fill statistics were regressed against the order of the matrix, the symmetry, regularity and reducibility measures, and the average number of original entries per row (column).

Table 2.7 contains a summary of these analyses. Only the variables that were significant in the regression equations are listed. It comes as no surprise that the original number of nonzeros is significant in estimating the (additional) fill. It is surprising that the symmetry ratings and the regularity measures are often significant, particularly in predicting relative fill. The results of these regressions indicate that the ordering heuristics are more effective for "general" problems, which are far from symmetric and which are irregular, than they are on problems with more structure. This conclusion is not new, but the fact that these structural measures have predictive power for the conclusion is.

It would be inappropriate, if not foolhardy, to claim statistical significance for these results. The usual normality assumptions would be rather difficult to justify for our small test collection. In addition, a linear model, as is assumed by these regressions, is unlikely. Only very special sparse matrices are known to have only a linear dependence on such parameters as order. Most matrices show a faster, nonlinear, growth. On the other hand, these results do suggest that further analysis may confirm what is now anecdotal: that general orderings for unsymmetric matrices perform poorly on symmetric matrices and matrices with a regular structure.

TABLE 2.7
Regressions for fill and relative fill.

	Variable	Fill Coefficient	Significance	Variable	Relative fill Coefficient	Significance
Markowitz	Nonzeros	3.2	.00	Symmetry	.016	.01
	Symmetry	48.4	.01	S.d. rows	−.104	.01
	S.d. rows	−32.0	.01			
	No. of blocks	−129.0	.01			
Goodness of fit		0.91			0.46	
Markowitz (with stability tolerance 0.1)	Nonzeros	2.5	.00	Symmetry	.015	.01
	S.d. rows	−311.2	.02	S.d. rows	−.095	.01
	Symmetry	55.2	.00			
	No. of blocks	−115.8	.01			
Goodness of fit		0.89			0.43	
P^4	Nonzeros	2.6	.00	S.d. rows	−.112	.04
	Order	−9.2	.00			
	S.d. columns	−260.0	.00			
	No. of blocks	−51.1	.05			
Goodness of fit		0.98			0.46	
P^5 (explicit factorization)	Order	88.0	.00	Order	.005	.00
	S.d. rows	−2713.1	.03	S.d. rows	−.254	.00
				Symmetry	.019	.02
Goodness of fit		0.83			0.78	
P^5 (implicit factorization)	Nonzeros	0.9	.00	Order	.003	.00
				S.d. rows	−.022	.04
				S.d. columns	−.016	.04
Goodness of fit		0.71			0.35	

Variables included in regression: order, symmetry rating, standard deviation of column counts, standard deviation of row counts, number of diagonal blocks in block triangular form, and either original number of nonzeros (for fill) or average number of nonzeros per row/column (for relative fill). Only variables which are significant at a 0.05 level are listed. Other variables were not significant at this level. Variables are listed in the order that they entered the stepwise regression.

Goodness of fit statistic is R square (norm of residual for fit/norm of residual for constant model).

3. Numerical properties. The basic ordering heuristics, Markowitz, P^4 and P^5, were all originally proposed as *preorderings*, that is, methods for reordering sparse matrices based only on their nonzero structures, but not on their numerical values. Unsymmetric sparse matrices cannot, in general, be factored accurately and stably without some pivoting based on the numerical entries. This study included an investigation of several questions that address the value of these heuristics as preorderings, and as bases for stable algorithms. Some of these questions were

—Do the problems differ in their numerical behavior because of their differing origins? Are there classes of problems for which preordering the matrices is sufficient?

—What are the costs of providing modified ordering heuristics that are stable?

—How reliable are a posteriori error bounds on the error in the factorization?

3.1. Numerical stability without pivoting. Three different outcomes may result from the numerical factorization of a preordered sparse matrix: a sufficiently accurate factorization may be found, a factorization that is inaccurate may be computed, or the factorization may break down because an exactly zero pivot may be encountered. The last case is usually thought of as unlikely in numerical analysis, but it is, in fact, encountered frequently in this particular area. We shall first investigate how frequently the factorization simply cannot be computed.

Exact zero pivots result from the interaction of small numbers of equations with closely related entries from approximate models. For example, one test matrix caused the P^5 factorization to fail because the heuristic had produced a two by two diagonal block, all of whose entries were the square root of two to machine precision. The numerical factorization of this matrix must conclude that this submatrix has rank one. The Markowitz strategy and the P^4 heuristic can fail in the same way. The P^4 heuristic, as discussed earlier, can also fail by producing a structurally zero diagonal element.

The comparison of the preordering stability properties is complicated by the fact that the MA28 implementation of the Markowitz strategy cannot break down in this fashion. Since the code computes the numerical factorization during the ordering process, it can detect an exactly zero diagonal element. When this occurs, the Markowitz ordering is perturbed; the pivot chosen is some truly nonzero entry with the best Markowitz count obtainable in these circumstances. In this sense, MA28 does not provide a pure Markowitz ordering. By contrast, the numerical factorization for P^4 and P^5 will break down if an exactly zero pivot is encountered.

Table 3.1 provides a partial answer to the value of these heuristics as preorderings. The number of successful symbolic and numerical factorizations obtained for each ordering are listed. The MA28 code cannot fail to provide some numerical factorization

TABLE 3.1
Success of symbolic and numerical factorization for the classes of test matrices.

Problem class	No. of matrices	MA28	P^4		P^5	
			Symbolic	Numeric	Symbolic	Numeric
Chemical engineering	16	16	10	6	15	8
Linear programming*	16	16	6	6	16	12
Simulations	11	9	4	4	6	6
PDE grids	7	4	4	1	6	1
Miscellaneous	6	6	6	6	6	6
Opt. power flow	2	2	0	0	2	0
Overall	58	53	30	23	51	33

* These numbers include 5 lower triangular matrices on which all algorithms were successful.

if it has space for the symbolic factorization, so only a single column is listed for this heuristic. These operations are listed separately for the P^4 and P^5 heuristics. As discussed in § 2, the symbolic factorization for P^4 can fail for two reasons, for lack of space or because of structurally zero diagonal elements. The difference between the failures for the symbolic and numerical factorizations of these two algorithms demonstrate how frequently exact zero diagonal elements are encountered. All told, the P^4 heuristic produced twenty-two structurally zero diagonal entries and seven "accidental," but exact, zero entries; the P^5 heuristic produced such accidental zero entries eighteen times.

It is certainly naive to assume that nonbreakdown of the numerical factorization is, in fact, success. The results in Table 3.1 simply indicate that these preorderings are not robust in the weakest sense. Table 3.1 does not measure numerical stability. For example, the norm of the error, $\|E\| = \|A - LU\|$, in the LU decomposition relative to A was of the order of 10^{30} for the only PDE problem "successfully" factored with the P^4 ordering. In order to obtain a better picture on the quality of performance of the algorithms, we used this relative measure of the error to rate the stability as follows:

$\|E\|/\|A\| \leqq 10^{-8}$	good stability,
$10^{-8} < \|E\|/\|A\| \leqq 10^{-4}$	fair stability,
$10^{-4} < \|E\|/\|A\| \leqq 1$	poor stability,
$\|E\|/\|A\| > 1$	failure.

Tables 3.2 through 3.4 summarize the stability results, using the qualitative descriptors above. The three tables provide a comparison across algorithms.

The numbers in these tables allow several conclusions about the relative performance of the algorithms with regards to stability. The tables suggest that the Markowitz

TABLE 3.2
Stability rating of MA28 *with* 0.0 *pivoting threshold for the classes of test matrices.*

Problem class	No. of matrices	No factors	Fail	Poor	Fair	Good
Chemical engineering	16	0	0	2	1	13
Linear programming	16	0	1	2	1	12
Simulations	11	2	8	0	1	0
PDE grids	7	3	3	0	0	1
Miscellaneous	6	0	0	1	1	4
Opt. power flow	2	1	1	0	0	0
Overall	58	6	13	5	4	30

TABLE 3.3
Stability rating of P^4 *algorithm for the classes of test matrices.*

Problem class	No. of matrices	No factors	Fail	Poor	Fair	Good
Chemical engineering	16	10	1	0	0	5
Linear programming	16	10	0	0	0	6
Simulations	11	7	2	0	1	1
PDE grids	7	6	1	0	0	0
Miscellaneous	6	0	1	1	0	4
Opt. power flow	2	2	0	0	0	0
Overall	58	35	5	1	1	16

TABLE 3.4
Stability rating of P^5 algorithm for the classes of test matrices.

Problem class	No. of matrices	No factors	Fail	Poor	Fair	Good
Chemical engineering	16	8	0	0	1	7
Linear programming	16	4	1	2	1	8
Simulations	11	5	2	0	3	1
PDE grids	7	6	1	0	0	0
Miscellaneous	6	0	2	0	1	3
Opt. power flow	2	2	0	0	0	0
Overall	58	25	6	2	6	19

ordering yields generally better results than either P^4 or P^5. In thirty-four cases the stability rating of MA28 was either good or fair, whereas the corresponding numbers are only 17 for P^4 and 25 for P^5. This is somewhat misleading, since the modification discussed above allows the MA28 Markowitz code to provide some factorization for almost all of the test problems. In fact, on comparable problems, the Markowitz ordering does often fare better, but this is not true for the simulation matrices. It is clearer that the P^5 ordering is an improvement over the P^4 ordering in terms of numerical stability. With the exception of one test matrix, the stability rating of P^5 is at least as good as that of P^4. For all of the orderings there are many matrices for which preordering alone is not satisfactory.

There are clear differences in numerical stability between the problem classes. The high rates of failure in the PDE problems, in the simulation problems and in the optimal power flow problems of all three algorithms confirm the observation made in the last section that these problems apparently are more difficult than the other matrices in the study. All three algorithms almost always failed for the PDE problems. The simulation problems are the only problems where P^4 and P^5 perform better than MA28, yet all three algorithms have a high rate of failure. The linear programming problems are not as difficult as the other classes, and yet the P^4 and P^5 heuristics succeed on only six and eight of these problems, respectively. Of these, five are triangular, for which the decomposition is trivially stable. This result is somewhat surprising, since the P^4 heuristic was designed for LP problems. On none of the classes do the P^4 and P^5 algorithms provide satisfactory results more than half the time. Hence there appears to be no particular problem class for which either P^4 or P^5 orderings are more appropriate than the Markowitz ordering for numerical stability. It is also clear from the overall failure rate of all three heuristics that any general use of these orderings requires modification for stability.

3.2. Costs and effects of stability modifications. The heuristic orderings must be modified to provide numerical stability in the factorizations. The cost of incorporating such modifications, and its effects on stability and fill are the topic of this section.

The discussion of the Markowitz ordering in § 1 concluded that, with little additional cost, the ordering heuristic could have knowledge of the numerical entries in each current reduced matrix. The MA28 code demonstrates that incorporating some control on element growth is possible with essentially the data structures already required for the Markowitz ordering. Duff's discussion of the development of the code [3], [4] does indicate that the development of a practical code was not trivial. However,

from the perspective of the user, the changes needed to compute a stable factorization with this code are trivial.

The Harwell MA28 Markowitz code controls the growth of elements in the factorization, and hence instability, by using a growth or pivoting threshold that can take values between zero and one. A growth tolerance of one corresponds to a pivot selection based essentially on numerical considerations; it is in fact Gaussian elimination with partial pivoting, with pivots for sparsity preservation chosen only from the largest elements in rows. The experience of the Harwell designers and others has been that this is too restrictive to provide practical sparse factorizations, and that a weaker control still suffices to control instability. We chose, with some assistance from the standard recommendations for the code, to use a growth threshold of one tenth (0.1). This results in the rejection of a Markowitz pivot only if there is an element in the same row at least ten times larger than the proposed pivot. It also potentially allows an order of magnitude growth in some of the elements of the factorization at each step.

Table 3.5 provides the basis for comparing the Markowitz ordering with limited stability control to the preorderings discussed in the previous section. Of the fifty-three matrices for which numerical factorizations were computed, in only five did the MA28 code fail to obtain a "good" factorization. It is a surprising fact that the rather substantial growth allowed at each step does not accumulate. It is also surprising, although it is consistent with previous published discussion of this algorithm, that the cost of obtaining a stable factorization is little more than the cost of the Markowitz preordered factorization. Tables 2.3 and 2.4 illustrate the small (if any) increase in fill associated with the restrictions placed on the ordering. There is a corresponding increase in the number of operations required to factor the matrix, and also in the time for ordering the matrix. The latter is beyond the scope of this study; some examples are given in [3], [4].

TABLE 3.5
Stability rating of MA28 *with* 0.1 *pivoting threshold for the classes of test matrices.*

Problem class	No. of matrices	No factors	Fail	Poor	Fair	Good
Chemical engineering	16	0	0	0	0	16
Linear programming	16	0	0	0	0	16
Simulations	11	2	0	0	0	9
PDE grids	7	3	0	0	0	4
Miscellaneous	6	0	0	0	0	6
Opt. power flow	2	0	0	0	0	2
Overall	58	5	0	0	0	53

Modifications to stabilize the ordering provided by the P^4 heuristic were discussed in the introduction. Growth and instability are controlled by a very similar threshold for required pivoting. This frequently used technique requires no modification to the heuristic itself, but does require increased complexity of the factorization data structures. In particular, the space requirements for the factorization will not be known until the factorization is completed. Although this increases the overall complexity of the process, it is, in fact, still less complex than the Markowitz ordering with control.

As was discussed earlier, limitations on this project led to the omission of a stabilized P^4 factorization from the evaluation code. The only results known to us that compare the increased cost in fill for stabilizing the P^4 heuristic are contained in [15],

[26], [27]. These results indicated that the amount of increased fill was more sensitive to the growth threshold than is the case for the Markowitz code. In particular, Saunders indicated that a threshold of 0.1 was substantially too large, that more growth would have to be allowed if sufficiently sparse factors were to be obtained. This could well result in numerically less satisfactory factorizations. There are arguments that explain why the modification to P^4 could result in rapid increases in fill. Since the fill for the stabilized Markowitz code with threshold of one tenth is often less than the fill for the unstabilized P^4 code there appear to be few reasons to study a stabilized P^4 code for great reduction of fill.

In contrast, the implicit fill statistics for the P^5 ordering are very small, except for the optimal power flow problems. They suggest that a stable version of this ordering and factorization would be quite valuable for certain problem classes if they have similar fill statistics. Unfortunately, we do not know how to obtain such a factorization scheme. We shall discuss why we believe this to be a difficult problem.

Let us note at the outset that it is not hard to find a stable factorization for the P^5 heuristic: the ideas used in conjunction with P^4 work also in this context. Note however that the P^5 ordering induces a special structure, a block bordered triangular form, to any irreducible matrix. Specifically, an irreducible matrix A, of order n, will be partitioned as follows:

$$A = \begin{bmatrix} D & B \\ C & S \end{bmatrix}, \tag{1}$$

where the $(n-p)$ by $(n-p)$ matrix D is lower block triangular:

$$\begin{bmatrix} D_{11} & & & \\ D_{21} & D_{22} & & 0 \\ \vdots & \vdots & \ddots & \\ D_{n1} & D_{n2} & \cdots & D_{nn} \end{bmatrix}. \tag{2}$$

We assume that the number of rows/columns p in the border blocks is small as compared to n, and that both B and C as well as the offdiagonal blocks in D are sparse.

It is possible to incorporate pivoting within the matrices D_{ii} and the matrix S without destroying the block partitioning. Such pivoting may be used to improve the stability of the factorization. It is, however, not enough to guarantee stability. Generally, stability requires that columns be interchanged between D and B (or that rows be interchanged between D and C). The ideas used for stabilizing the P^4 ordering interchange columns.

The crux of the matter is that the interchange of columns of D and columns of B may well destroy the block triangular form of D. If it does, we do not know how to perform the factorization implicitly. If we must form the factorization explicitly, the fill will be too great, even without the increases we expect from perturbing the ordering. The difference between the fill of implicit and explicit factorizations is evident from Table 2.3. We shall illustrate the problem of instability with a model problem.

Consider the following matrix of order n:

$$\begin{bmatrix} -4 & 1 & 0 & \cdot & \cdot & 0 \\ 1 & -4 & 1 & \cdot & \cdot & \cdot \\ 0 & 1 & -4 & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & 1 & -4 & 1 \\ 0 & \cdot & \cdot & 0 & 1 & -4 \end{bmatrix}.$$

That this matrix is symmetric is irrelevant to this discussion, as is the fact that it can be factored stably without pivoting as it is written. The matrix, after it has been reordered with the P^5 heuristic, is given below:

$$\begin{bmatrix} 1 & 0 & \cdot & \cdot & \cdot & -4 \\ -4 & 1 & 0 & \cdot & \cdot & 1 \\ 1 & -4 & 1 & 0 & \cdot & 0 \\ 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & -4 & 1 & 0 \\ 0 & \cdot & 0 & 1 & -4 & 0 \end{bmatrix}.$$

Note that the border has only a single column, so we will denote it as a vector b. The leading principal block D is unit lower triangular. The implicit fill is exactly the order of the block S, one, independent of the order, n, of the original matrix.

Gaussian elimination on the reordered matrix requires, either explicitly or implicitly, the solution of the triangular linear system $Dy = b$. It is easy to show that $y_1 = -4$, $y_2 = -15$, $y_3 = -56$ and that the elements of y thereafter grow by a factor of about $2+\sqrt{3}$ at each step. This is unacceptable growth, which leads to serious stability problems. Element growth similar to this was encountered in the factorizations of the PDE matrices in our test collection.

The model problem explains the nature of the serious stability problems associated with the P^5 ordering. We have investigated other approaches to avoid element growth while preserving the block bordered triangular form. Approaches based on the Sherman–Morrison–Woodbury updating form are possible, but they appear to be equivalent to the original in terms of growth, that is, to be only implicit reformulations that must solve a similar linear system at some point. Another attempt to circumvent the element growth in the formation of the Schur complement is to use an orthogonal factorization of the matrix instead of using block Gaussian elimination. Since the matrix is almost in lower triangular form, one is tempted to believe that a strategy based on the LQ factorization might be successful. This is unlikely, for the following reason. The matrix L in the LQ factorization is the Cholesky factor of the normal equations, i.e. the Cholesky factor of

$$\begin{bmatrix} DD^T + BB^T & DC^T + BS^T \\ CD^T + SB^T & CC^T + SS^T \end{bmatrix}.$$

There is no reason to suspect that the partitioning obtained by P^5 is a good reordering for this symmetric matrix. Indeed, there are reasons to suspect the opposite: the heuristic tends to put denser columns into the border, which will cause the leading principal block in the above to be substantially more full.

It should be mentioned that Heath [16] suggests a reordering scheme for the solution of particular sparse linear least squares problems by an orthogonal factorization, which is closely related to some ideas expressed here. When full rows are present, Heath proposes to reorder them last and use block Gaussian elimination to solve the complete problem. Even though the leading sparse part of the matrix is factored with stable orthogonal techniques, the use of block Gaussian elimination may lead to the same stability difficulties encountered with the P^5 algorithm.

3.3. Evaluation of measures of stability. Numerical stability is a concern, even with a factorization that provides some control over element growth. A side issue we addressed was the reliability of various estimates of the error in the actually computed factorization.

The error in the factorization is defined by $E = A - LU$. The scalar that best summarizes this error is the relative error norm, $\|E\|/\|A\|$. Since computing E, or $\|E\|$, is often unacceptably expensive, several heuristic algorithms have been suggested. These estimates, which have dramatically different costs, are

(1) Error in the solution $\|x_{\text{true}} - x\|_1$ where $x_{\text{true}} = (1, 1, \cdots, 1)^T$, $b = Ax_{\text{true}}$, and x is the computed solution;

(2) Erisman-Reid error bound [13]

$$\max_{i,j} |e_{ij}| \leqq 3.01 \varepsilon m (\max_{i,j} |a_{ij}| + \|L\|_1 \|U\|_1)$$

where ε = machine epsilon, and m is the maximum number of operations needed to compute one entry of L or U;

(3) Wilkinson error bound [28]

$$\max_{i,j} |e_{ij}| \leqq 3.01 \varepsilon m (\max_{i,j,k} |a_{ij}^{(k)}|).$$

The values obtained for these estimates and made relative to $\|A\|$ are summarized in Tables 3.6–3.8. These tables give the average values of the base 10 logarithms of the estimates for all of the obtained numerical factorizations. The average of the logarithms of the actual relative error norm are given for comparison, in Table 3.9.

The quantity (3) is a bound on the actual error; the quantity (2) is a bound on (3). We expect, and find, that the Erisman-Reid bound is larger, more pessimistic about the success of the factorization, than the Wilkinson bound. This is, in turn, larger than the actual error. On average, the Wilkinson bound is larger than the actual error

TABLE 3.6
Solution error for the classes of test matrices.

Problem class	MA28 (.0)	MA28 (.1)	P[4]	P[5]
Chemical engineering	−10.1	−13.9	−10.7	−10.4
Linear programming	−8.0	−13.1	−14.1	−9.5
Simulations	+19.3	−11.6	−0.2	+1.1
PDE grids	+11.6	−19.0	+18.3	+22.2
Miscellaneous	−8.9	−13.1	−6.0	−4.5
Opt. power flow	+14.8	−11.6	—	—
Overall	−2.1	−13.5	−7.2	−5.9

TABLE 3.7
Erisman-Reid error bound for the classes of test matrices.

Problem class	MA28 (.0)	MA28 (.1)	P[4]	P[5]
Chemical engineering	−3.4	−10.4	−6.3	−5.6
Linear programming	−3.6	−11.2	−11.2	−4.5
Simulations	+23.9	−8.9	+7.8	+8.4
PDE grids	+13.6	−8.9	+22.2	+22.2
Miscellaneous	−2.1	−10.4	−0.3	+4.4
Opt. power flow	+28.1	−7.8	—	—
Overall	+3.3	−10.2	−2.3	+0.0

TABLE 3.8
Wilkinson error bound for the classes of test matrices.

Problem class	MA28 (.0)	MA28 (.1)	P^4	P^5
Chemical engineering	−8.7	−12.4	−10.0	−9.9
Linear programming	−7.1	−12.7	−11.5	−7.5
Simulations	+18.2	−10.6	+0.8	+2.5
PDE grids	+13.6	−11.8	+22.2	+22.2
Miscellaneous	−7.5	−11.9	−4.8	−3.2
Opt. power flow	+16.6	−10.8	—	—
Overall	−1.2	−12.0	−3.7	−4.6

TABLE 3.9
Actual error for the classes of test matrices.

Problem class	MA28 (.0)	MA28 (.1)	P^4	P^5
Chemical engineering	−11.5	−15.3	−12.7	−13.0
Linear programming	−10.3	−14.4	−14.6	−10.1
Simulations	+17.1	−13.9	−0.8	−0.7
PDE grids	+13.3	−13.8	+22.2	+22.2
Miscellaneous	−9.4	−14.5	−6.7	−5.2
Opt. power flow	+13.8	−13.7	—	—
Overall	−3.5	−14.5	−8.0	−7.2

by three orders of magnitude. This is a large error in an absolute sense, but it is unlikely to cause a satisfactory factorization to be rejected. On the other hand, the Erisman-Reid bound is large enough so that some factorizations might be unduly rejected. The results for this bound on the LP bases as ordered by the pure Markowitz ordering are an example. However, none of the factorizations with the stabilized Markowitz ordering would be rejected. The Erisman-Reid bound appears to be a reasonable tool in conjunction with this particular code. (Note that the version of this bound for symmetric matrices is usually considerably more sharp.)

The error in the solution of a specific equation, (1), is only an estimate, not a bound on the factorization error. While it is usually a better estimate of the error than either of the bounds, it does have the potential to mislead. The PDE matrices ordered by the stable Markowitz ordering are a particular case where this estimate indicates that the factorizations are substantially more accurate than they really are.

The four potential measures or estimates of factorization error differ greatly in cost. The cost of computing the error in a specific equation is only a matrix multiply to generate a right-hand side and a matrix solve. The Erisman-Reid error bound requires a single scan of the numeric values of A and its factors as well as monitoring to determine m. Both of these measures can be performed without affecting the factorization modules and with little additional cost. The Wilkinson error bound requires monitoring of all values in each step of the elimination. Its order of complexity is the same as the factorization and must be performed simultaneously with the factorization. The computation of the actual error can be performed at cost similar to

the evaluation of the Wilkinson bound, without affecting the factorization. It also requires more storage.

In practice, only the first two estimates, finding the error in a specific solution or computing the Erisman-Reid bound, are likely to be low enough in cost. Since the MA28 code provides a built-in option to compute the product of the norms of L and U used in the Erisman-Reid bound, this bound is reasonable to use with this code. (However, in his complex version of MA28 Duff actually monitors the growth, as in (3); he argues that the overhead of complex arithmetic on some computers makes the monitoring inconsequential on these computers.) Estimating the error in a specific solution could be used with other codes, but it should be used with caution. The particular vector used here is not a good choice in general [19]. More reliability would be gained by using two or more vectors with random values.

4. Conclusions and recommendations for further study. We set out to determine whether any of the orderings in this study could be regarded as generally effective. The results clearly indicate that none of the orderings, considered strictly as preorderings, can be so regarded. However, we confirmed earlier findings that the stabilized version of the Markowitz heuristic, as represented by the MA28 code, is a generally useful and effective reordering.

Our study of orderings is incomplete in that it did not include stabilized versions of the P^4 and P^5 heuristics. As we indicated in the previous section, a stable and implicit P^5 factorization would be quite valuable, if it exists, although its range of applications would be restricted to certain problem classes. We are not sanguine about the likelihood of developing such a variant. A stable version of the P^4 heuristic is already known. The published experience with this algorithm is generally negative, but these results are also confined to a narrow class of problems, linear programming bases.

Our study is incomplete also in that fill and stability are not the only measures of effectiveness for codes; ordering and factorization times are also important. The P^4 heuristic is less complex than the Markowitz heuristic, so ordering times for a good implementation of P^4 should be much less than the ordering time for the Markowitz ordering. It is often the case that the ordering dominates the computation for MA28 factorizations. A stable P^4 ordering and factorization may have smaller total cost, if more fill, than its Markowitz counterpart when only a single system with a given structure is to be solved. Whether this is true will depend on how sensitive P^4 is to pivoting for stability, and whether the limited pivoting, "living dangerously," recommended in [26] is, in fact, sufficient to obtain stability. These questions should be addressed directly by further work on this algorithm.

Recently Michael Saunders [27] has developed a variation on the standard sparse Gaussian elimination that simplifies the modification of the factors of the sequence of basis matrices in linear programming problems. Saunders uses two by two stabilized elementary matrices to compute and update the factors, and obtains potentially quite different interchanges in computing stable factorizations. Stability is assured for the initial factorization, given P^4 or any other suggested ordering. The results of this study do not apply directly to his new algorithm, but it remains likely that P^4 would not perform well on structurally symmetric problems.

Our results support the conventional wisdom that there are classes of sparse matrices, which differ in difficulty. Some of the areas studied resulted in matrices that were difficult for all of the algorithms, and others resulted in matrices that were easy for all of the algorithms. Our statistical analyses provide some suggestive results, but the size and characteristics of our sample are too limited to do more than point the

way for further investigation. A larger sample would have statistical benefits; it would have a correspondingly large cost in computing. Care should be taken to provide a wide enough selection to control the effects of certain parameters. For example, structural symmetry appears to be an indicator of high fill, but it is clearly related in most of our sample to additional parameters such as regularity and relatively high density. This study presupposed the value of general purpose sparse matrix heuristics and codes. The existence of problems such as the relatively unstructured matrices in our test collection, the linear programming bases and the chemical engineering models, confirms this assumption. However, the highly structured and nearly symmetric problems, which proved the hardest for these general heuristics, point to the need for further study of the special cases where variations of symmetric orderings are appropriate. This becomes particularly true when stability can be incorporated into the factorization. This is not the case for most current orderings for nearly symmetric matrices, which assume that stability is not an issue. The recent work on extensions of frontal methods by Duff and Reid [8], [10]-[12] provides a possible basis for such methods. Such methods are quite different in character from the general sparse orderings, so careful analysis will be required in any comparison. Only with such comparison will we determine if the general sparse heuristics are still valuable for this special class of problems.

Note added in proof. An additional measure of estimating the error in Gaussian elimination without pivoting was reported by Chu and George after it could have been incorporated into this paper. We regret this omission and refer interested readers to the report [E. Chu and J. A. George, *An algorithm to estimate the error in Gaussian elimination without pivoting*, Faculty of Mathematics, University of Waterloo, Waterloo, Ontario, CS-84-21, 1984].

REFERENCES

[1] J. BISSCHOP, Y. LEVY AND A. MEERAUS, *Programs for structure analysis of sparse matrices*, Tech. Report, Development Research Ctr., World Bank, Washington, DC, 1979, submitted to ACM Trans. Math. Software.

[2] I. S. DUFF, *On permutations to block triangular form*, Bull. Inst. Math. Appl., 19 (1977), pp. 339-342.

[3] I. S. DUFF AND J. K. REID, *Some Design Features of a Sparse Matrix Code*, AERE-Harwell Report CSS 48, 1977.

[4] I. S. DUFF, MA28—*A set of* FORTRAN *subroutines for sparse unsymmetric linear equations*, AERE—R8730, Her Majesty's Stationery Office, London, 1977.

[5] I. S. DUFF AND J. K. REID, *An implementation of Tarjan's algorithm for the block triangularization of a matrix*, ACM Trans. Math. Software, 4 (1978), pp. 137-147.

[6] ———, *Algorithm* 529: *permutations to block triangular form*, ACM Trans. Math. Software, 4 (1978), pp. 189-192.

[7] ———, *Performance evaluation of codes for sparse matrix problems*, in Performance Evaluation of Numerical Software, L. Fosdick, ed., North-Holland, New York, 1979, pp. 121-135.

[8] I. S. DUFF, *Design features of a code for solving sparse unsymmetric linear systems out-of-core*, AERE-Harwell Report CSS 89, 1981.

[9] I. S. DUFF, R. G. GRIMES, J. G. LEWIS AND W. G. POOLE, JR., *Sparse Matrix Test Problems*, SIGNUM Newsletter, 17, 2 (1982), p. 22.

[10] I. S. DUFF AND J. K. REID, MA27—*A set of* FORTRAN *subroutines for solving sparse symmetric sets of linear equations*, AERE—R10533, Her Majesty's Stationery Office, London, 1982.

[11] ———, *The multifrontal solution of indefinite sparse symmetric linear systems*, AERE-Harwell Report CSS 122, 1982.

[12] ———, *The multifrontal solution of unsymmetric sets of linear equations*, AERE-Harwell Report CSS 133, 1983.

[13] A. M. ERISMAN AND J. K. REID, *Monitoring the stability of the triangular factorization of a sparse matrix*, Numer. Math., 22 (1974), pp. 183-186.

[14] A. M. ERISMAN, R. G. GRIMES, J. G. LEWIS AND W. G. POOLE, JR., *A structurally stable modification of Hellerman-Rarick's* P^4 *algorithm for reordering unsymmetric sparse matrices*, SIAM J. Numer. Anal., 22 (1985), pp. 369-385.

[15] P. E. GILL, W. MURRAY, M. H. WRIGHT AND M. A. SAUNDERS, *Sparse matrix methods in optimization*, Tech. Rep. SOL 82-17, Dept. of Operations Research, Stanford Univ., 1982.

[16] M. T. HEATH, *Some extensions of an algorithm for sparse linear least squares problems*, this Journal, 3 (1982), pp. 223-237.

[17] E. HELLERMAN AND D. C. RARICK, *Reinversion with the preassigned pivot procedure*, Math. Programming, 1 (1971), pp. 195-216.

[18] ———, *The Partitioned Preassigned Pivot Procedure* (P^4), in Sparse Matrices and Their Applications, D. J. Rose and R. A. Willoughby, eds., Plenum Press, New York, 1972, pp. 67-76.

[19] W. KAHAN, private communication.

[20] T. D. LIN AND R. S. H. MAH, *Hierarchical partition—a new optimal pivoting algorithm*, Math. Programming, 12 (1977), pp. 260-278.

[21] ———, *A sparse computation system for process design and simulation, part* I. *Data structures and processing techniques*, AIChE J., 24 (1978), pp. 830-839.

[22] R. S. H. MAH AND T. D. LIN, *A sparse computation system for process design and simulation, part* II. *A performance evaluation based on the simulation of a natural gas liquefaction process*, AIChE J., 24 (1978), pp. 839-848.

[23] H. M. MARKOWITZ, *The elimination form of the inverse and its application to linear programming*, Management Sci., 3 (1957), pp. 255-269.

[24] B. A. MURTAGH AND M. A. SAUNDERS, MINOS *User's Guide*, Technical Report 77-9, Systems Optimization Laboratory, Dept. of Operations Research, Stanford Univ., 1977.

[25] N. H. NIE, C. H. HULL, J. G. JENKINS, K. STEINBRENNER AND D. H. BENT, *Statistical Package for the Social Sciences* (*SPSS*), McGraw-Hill, New York, 1975.

[26] M. A. SAUNDERS, *A fast, stable implementation of the simplex method using Bartels-Golub updating*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1976, pp. 213-226.

[27] ———, private communication.

[28] J. K. REID, *A note on the stability of Gaussian elimination*, Bull. Inst. Math. Appl., 8 (1971), pp. 374-375.

[29] R. E. TARJAN, *Depth first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146-160.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 4, July 1987
© 1987 Society for Industrial and Applied Mathematics
010

GENERATION OF RANDOM ORTHOGONAL MATRICES*

T. W. ANDERSON†, I. OLKIN† AND L. G. UNDERHILL‡

Abstract. In order to generate a random orthogonal matrix distributed according to Haar measure over the orthogonal group it is natural to start with a matrix of normal random variables and then factor it by the singular value decomposition. A more efficient method is obtained by using Householder transformations. We propose another alternative based on the product of $n(n-1)/2$ orthogonal matrices, each of which represents an angle of rotation. Some numerical comparisons of alternative methods are made.

Key words. simulation, Monte Carlo, Haar measure, random normal numbers

AMS(MOS) subject classifications. 15A52, 65C20

1. Introduction. A number of approaches have been used in generating random orthogonal matrices distributed according to the Haar measure over the orthogonal group.

As a starting point let $X=(x_{ij})$ be a $p\times p$ matrix consisting of p^2 independent standard normal random variables x_{ij}, $i,j=1,\cdots,p$. An application of the Gram–Schmidt orthogonalization procedure amounts to finding an upper triangular matrix R with positive diagonal elements such that $X=QR$. This method is called the QR factorization of X, and the matrix Q is the desired orthogonal matrix.

From another point of view we can choose pn independent standard normal random variables x_{ij}, $i=1,\cdots,p$; $j=1,\cdots,n$ from which we form the $p\times n$ matrix $X=(x_{ij})$. An orthogonal matrix G is now obtained from the eigenvalue decomposition

$$XX'=GD_lG', \tag{1}$$

where $D_l=\operatorname{diag}(l_1,\cdots,l_p)$ is a diagonal matrix with elements $l_1,\cdots,l_p$ which are the characteristic roots of XX'. Alternatively, G can be obtained from the singular value decomposition

$$X=GD_l^{1/2}H, \tag{2}$$

where H is a $p\times n$ matrix satisfying $HH'=I_p$.

Another characterization of a random orthogonal matrix is obtained from a skew-symmetric matrix S:

$$G=D_\varepsilon(I+S)^{-1}(I-S), \tag{3}$$

where $D_\varepsilon=\operatorname{diag}(\varepsilon_1,\cdots,\varepsilon_n)$, $\varepsilon_i=\pm 1$. (The matrix D_ε is determined in order to guarantee that $\det(I+D_\varepsilon G)\neq 0$.) Although this approach could be used to generate random orthogonal matrices G, it requires first generating random skew-symmetric matrices S with joint density function proportional to

$$\det(I+S)^{-a}, \tag{4}$$

* Received by the editors January 17, 1986; accepted for publication July 17, 1986. This work was supported by National Science Foundation grants DMS-84-11411 and DMS-82-19748.

† Department of Statistics, Stanford University, Stanford, California 94305.

‡ University of Cape Town, Rondebosch, Republic of South Africa.

Olkin (1951). However, there does not seem to be a simple way to generate such random skew-symmetric matrices. Thus, it appears to be easier to generate random orthogonal matrices than random skew-symmetric matrices.

Another characterization is obtained directly from the definition: an $n \times n$ orthogonal matrix is an ordered set of n rows and of n-dimensional vectors, each of which is defined over the surface of a unit n-dimensional sphere, and each vector is orthogonal to each of the rows preceding it. This definition was the basis of an algorithm by Heiberger (1978). (See Tanner and Thisted (1982) for additional commentary.) The algorithm makes use of independent standard normal random variables.

A characterization based on Householder transformations was used by Stewart (1980). A symmetric orthogonal matrix is defined by

$$H = I - 2\frac{uu'}{\|u\|^2}, \tag{5}$$

where u is a nonzero vector and $\|\cdot\|$ is the Euclidean norm. For any x, there is a Householder transformation H_x such that

$$H_x x = r(1, 0, \cdots, 0)', \qquad r = \pm\|x\|. \tag{6}$$

Stewart's method is based on choosing independent $N(0, \sigma^2)$ random vectors $x_1 \in R^n$, $x_2 \in R^{n-1}, \cdots, x_{n-1} \in R^2$. For $j = 1, \cdots, n-1$, let $\bar{H}_{x_j}$ be the Householder transformation (6), and define the $n \times n$ matrices $H_1 = \bar{H}_{x_1}$ and

$$H_j = \operatorname{diag}(I_{j-1}, \bar{H}_{x_j}), \qquad j = 2, \cdots, n.$$

The product

$$Q = H_1 H_2 \cdots H_{n-1} D_\varepsilon \tag{7}$$

is a random orthogonal matrix, where $D_\varepsilon = \operatorname{diag}(\varepsilon_1, \cdots, \varepsilon_n)$, and $P\{\varepsilon_i = +1\} = P\{\varepsilon_i = -1\} = \frac{1}{2}$, $i = 1, \cdots, n$.

We propose yet another characterization of an orthogonal matrix in terms of $n(n-1)/2$ rotations plus reflections, which is formally equivalent to (7). The differences are centered on how to generate the random variables. More specifically, define the Givens matrix

$$G_{ij} \equiv G_{ij}(\theta_{ij}) = \begin{array}{c} \\ i \\ \\ j \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccc} \quad & i & \quad & j & \quad \end{array} \\ \begin{pmatrix} I & 0 & 0 & 0 & 0 \\ 0 & \cos\theta_{ij} & 0 & -\sin\theta_{ij} & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & \sin\theta_{ij} & 0 & \cos\theta_{ij} & 0 \\ 0 & 0 & 0 & 0 & I \end{pmatrix} \end{array}, \tag{8}$$

where $-\pi/2 < \theta_{ij} \leqq \pi/2$. Then every orthogonal matrix G can be written as

$$G = (G_{12} G_{13} \cdots G_{1n})(G_{23} \cdots G_{2n}) \cdots (G_{n-1,n}) D_\varepsilon, \tag{9}$$

where $D_\varepsilon = \operatorname{diag}(\varepsilon_1, \cdots, \varepsilon_n)$, $\varepsilon_i = \pm 1$, $i = 1, \cdots, n$. The θ_{ij} represent the angles of the rotation and the matrix D_ε represents reflections.

Although the representation (9) is generally well known, it has not been used very often in this form. Because of this we give a derivation in the Appendix. The simplicity of the derivation makes it a very accessible procedure. It also may provide an elementary proof of rather complicated results. For example, Tumura (1965) uses this representation to provide a derivation of the noncentral distribution of the characteristic roots

of a Wishart matrix, thereby providing an elementary development of zonal polynomials. (There is an error in this paper in the definition of G; we have not checked whether the subsequent methodology is correct.)

To generate a random orthogonal matrix choose D_ε at random, i.e.,

$$P\{\varepsilon_i = +1\} = P(\varepsilon_i = -1\} = \tfrac{1}{2}, \qquad i = 1, \cdots, n,$$

and let the $n(n-1)/2$ angles θ_{ij} be mutually independent with joint density function which is proportional to

$$\left(\prod_{j=2}^{n} \cos^{j-2}\theta_{1j}\right)\left(\prod_{j=3}^{n} \cos^{j-3}\theta_{2j}\right)\cdots\left(\prod_{j=n}^{n} \cos\theta_{n-1,j}^{j-n}\right). \tag{10}$$

Once the angles and ε's have been chosen, they are combined as in (8) and (9).

To carry out this program, we need to generate random angles according to the density (10). The density of an angle α is of the form

$$2\frac{\cos^{a-1}\alpha \sin^{b-1}\alpha}{B(a/2, b/2)}, \qquad 0 \leqq \alpha \leqq \pi/2, \tag{11}$$

from which the density of y defined by $\sqrt{y} = \cos\alpha (\sin\alpha = \pm\sqrt{1-y})$ is a beta density

$$\beta(y; a/2, b/2) = \frac{y^{(a-2)/2}(1-y)^{(b-2)/2}}{B(a/2, b/2)}, \qquad 0 \leqq x \leqq 1. \tag{12}$$

Thus, the procedure requires generation of random variables y with density function (12). But this is standard by virtue of the representation $y = u/(u+v)$, where u and v are independent, with u and v having chi-square distributions with a and b degrees of freedom respectively. Of course, chi-square random variables can be generated directly from random normal variables or from uniform random variables.

More specifically, from (12) for fixed $i = 1, \cdots, n-1$, we need to generate $n-i$ independent random variables having respective $\beta(x; \frac{1}{2}, \frac{1}{2})$, $\beta(x; \frac{2}{2}, \frac{1}{2}), \cdots, \beta(x; (n-i)/2, \frac{1}{2})$ densities. A critical simplification occurs from the fact that if x_1, x_2 and x_3, are independent chi-square random variables with a, b and c degrees of freedom respectively, then $x_1/(x_1+x_2)$ is independent of $(x_1+x_2)/(x_1+x_2+x_3)$. Consequently, we begin our process by generating $y_1 = x_1/(x_1+x_2)$ and then using the denominator to generate $y_2 = (x_1+x_2)/(x_1+x_2+x_3)$. The process then continues. This means that for each successive y_j we need only one additional chi-square variate. This method provides considerable simplification and rapidity in the generation of the required random angles. However, there may be other more efficient procedures for generating the random angles θ_{ij}.

2. Numerical results. To test this procedure we carried out a simulation using 1000 replications for each case, $n = 4(2)12$. Random orthogonal matrices were generated by the method of successive rotations (ROT). The polar method (see Knuth (1981, p. 117)) was used to generate the random normal numbers. The execution times in CPU seconds were

n	4	6	8	10	12
ROT	30.8	69.2	136.8	207.3	313.8

(13)

These times are approximately cubic mn. For each n we calculate the first four sample moments (over replications) of the entry in each of the n^2 positions, and compare these with the expectations $Ev_i = 0$, $Ev_i^2 = 1/n$, $Ev_i^3 = 0$, $Ev_i^4 = 3/n(n+2)$. The number of entry positions within k standard errors is listed in Table 1. Although the

TABLE 1
Number of moment entries in standard error intervals.

n \ k	Ev				Ev^2				Ev^3				Ev^4			
	0-1	1-2	2-3	>3	0-1	1-2	2-3	>3	0-1	1-2	2-3	>3	0-1	1-2	2-3	>3
4	11	3	2	0	9	7	0	0	10	4	2	0	10	6	0	0
6	25	9	2	0	24	7	4	1	26	7	3	0	26	6	3	1
8	43	18	3	0	45	17	2	0	41	20	3	0	46	17	1	0
10	71	26	2	1	60	39	1	0	74	19	6	1	70	27	3	0
12	93	45	5	1	93	40	10	1	98	38	8	0	95	41	7	1

n^2 elements of an orthogonal matrix are dependent, we do not take this fact into account in the tabulation.

Appendix.

A.1. Construction of an orthogonal matrix. Let $G=(g_{ij})$ be an arbitrary orthogonal matrix. We shall show how G determines uniquely the representation (9). Let $\Gamma_{ij}=G'_{ij}$. First find $\theta_{12}(-\pi/2<\theta_{12}\leqq\pi/2)$ so

$$(\Gamma_{12}G)_{21}=-g_{11}\sin\theta_{12}+g_{21}\cos\theta_{12}=0. \tag{14}$$

Then $\cos\theta_{12}\geqq 0$ and $\sin\theta_{12}$ has the sign of g_{21}/g_{11}. (If $g_{11}=0$, let $\theta_{12}=\pi/2$.) The matrix $\Gamma_{1j}, j=3,\cdots,n$, is constructed $(-\pi/2<\theta_{ij}\leqq\pi/2)$ so

$$\begin{aligned}(\Gamma_{1j}\Gamma_{1,j-1}\cdots\Gamma_{12}G)_{j1}&=-\sin\theta_{1j}(\Gamma_{1,j-1}\cdots\Gamma_{12}G)_{11}\\&\quad+\cos\theta_{1j}(\Gamma_{1,j-1}\cdots\Gamma_{12}G)_{j1}=0.\end{aligned} \tag{15}$$

Then $\cos\theta_{1j}\geqq 0$. The matrix $\Gamma_{1n}\cdots\Gamma_{12}G$ has 0's in the first column, except the first element which is ± 1 (since the matrix is orthogonal). That fact implies that the elements in the first row are 0 except the first element. We define the scalar ε_1 and the $(n-1)\times(n-1)$ matrix $G^{(2)}$ by

$$\Gamma_{1n}\cdots\Gamma_{12}G=\begin{bmatrix}\varepsilon_1 & 0\\ 0 & G^{(2)}\end{bmatrix}. \tag{16}$$

Then $G^{(2)}$ is orthogonal.

At the jth stage

$$(\Gamma_{j-1,n}\cdots\Gamma_{j-1,j})\cdots(\Gamma_{1n}\cdots\Gamma_{12})G=\begin{bmatrix}D^{(j-1)} & 0\\ 0 & G^{(j)}\end{bmatrix}, \tag{17}$$

where $D^{(j-1)}$ is a $(j-1)\times(j-1)$ diagonal matrix with $\varepsilon_1,\cdots,\varepsilon_{j-1}$ as diagonal elements and $G^{(j)}$ is an $(n-j+1)\times(n-j+1)$ orthogonal matrix. Note that $\Gamma_{j,j+1},\cdots,\Gamma_{n-1,n}$ are of the form

$$\begin{bmatrix}I_{j-1} & 0\\ 0 & \Gamma^{(j)}\end{bmatrix}, \tag{18}$$

where the orthogonal $\Gamma^{(j)}$ has order $n-j+1$. Then $\theta_{j,j+1},\cdots,\theta_{jn}$ are taken as at the first stage to make the first column of $\Gamma^{(j)}_{jn}\cdots\Gamma^{(j)}_{j,j+1}G^{(j)}$ consist of 0's. Thus

$$\Gamma^{(j)}_{jn}\cdots\Gamma^{(j)}_{j,j+1}G^{(j)}=\begin{bmatrix}\varepsilon_j & 0\\ 0 & G^{(j+1)}\end{bmatrix}. \tag{19}$$

Finally, $G^{(n)}=\varepsilon_n$. Thus

$$\Gamma_{n-1,n}\cdots\Gamma_{12}G=D_\varepsilon. \tag{20}$$

The representation (9) is obtained from (20) with $G_{ij}=\Gamma'_{ij}$.

A.2. Representation of a random orthogonal matrix. The Haar measure of G is characterized by the property that ΓG has the same measure as G for every orthogonal Γ, or equivalently that $G\Gamma$ has the same measure as G for every orthogonal Γ. It follows that any column or any row of G has the uniform distribution over the n-sphere. If Γ_2 is an orthogonal matrix of order $n-1$, then from (16) we see that

$$\Gamma_{1n}\cdots\Gamma_{12}G\begin{pmatrix}1 & 0\\ 0 & \Gamma_2\end{pmatrix}=\begin{bmatrix}\varepsilon_1 & 0\\ 0 & G^{(2)}\Gamma_2\end{bmatrix} \tag{21}$$

has the same measure as (16) because the first column of G (on which $\Gamma_{1n},\cdots,\Gamma_{12}$ depend) is the same as the first column of $G\left(\begin{smallmatrix}1 & 0\\ 0 & \Gamma_2\end{smallmatrix}\right)$. Hence $G^{(2)}\Gamma_2$ has the same measure as $G^{(2)}$ for every orthogonal Γ_2. Moreover, $\Gamma_{1n},\cdots,\Gamma_{12},\varepsilon_1$, and $G^{(2)}$ are independent.

The first column of $G_{1n}\cdots G_{12}$ is the transpose of

$$\left(\prod_{j=2}^{n}\cos\theta_{1j},\sin\theta_{12}\prod_{j=3}^{n}\cos\theta_{1j},\sin\theta_{13}\prod_{j=4}^{n}\cos\theta_{1j},\cdots,\sin\theta_{1,n-1}\cos\theta_{1n},\sin\theta_{1n}\right). \tag{22}$$

It consists of the polar coordinates of the first column of G. The density of $\theta_{12},\cdots,\theta_{1n}$ is a constant times

$$(\cos^{n-2}\theta_{1n})(\cos\theta_{1,n-1}^{n-3})\cdots(\cos\theta_{13})$$

(see e.g. Anderson (1984, problem 4, Chap. 7)). The scalar $\varepsilon_1=-1$ and $+1$ with probability $\frac{1}{2}$.

In turn the distributions of $\theta_{j,j+1},\cdots,\theta_{jn}$, and ε_j are found.

Acknowledgment. The authors are grateful to Hal Stern for carrying out the computations.

REFERENCES

T. W. ANDERSON (1984), *An Introduction to Multivariate Statistical Analysis*, second edition, John Wiley, New York.

R. M. HEIBERGER (1978), *Algorithm* AS 127, *Generation of random orthogonal matrices*, Appl. Statist., 27, pp. 199-206.

R. M. HEIBERGER, P. F. VELLEMAN AND M. A. YPELAAR (1983), *Generating test data for independently controllable features for multivariate general linear forms*, J. Amer. Statist. Assoc., 78, pp. 585-595.

D. KNUTH (1981), *Seminumerical Algorithms: The Art of Computer Programming*, second edition, Addison-Wesley, New York.

I. OLKIN (1951), *On distribution problems in multivariate analysis*, Doctoral dissertation, University of North Carolina, Chapel Hill, NC.

G. W. STEWART (1980), *The efficient generation of random orthogonal matrices with an application to condition estimators*, SIAM J. Numer. Anal., 17, pp. 403-409.

M. A. TANNER AND R. A. THISTED (1982), *A remark on* AS127. *Generation of random orthogonal matrices*, Appl. Statist., 31, pp. 190-192.

Y. TUMURA (1965), *The distributions of latent roots and vectors*, TRU Math., 1, pp. 1-16.

P. F. VELLEMAN AND M. A. YPELAAR (1980), *Constructing regressions with controlled features: a method for probing regression performance*, J. Amer. Statist. Assoc., 75, pp. 839-844.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 4, July 1987
© 1987 Society for Industrial and Applied Mathematics
011

ON GENERALIZED CROSS-VALIDATION FOR MULTIVARIATE SMOOTHING SPLINE FUNCTIONS*

FLORENCIO I. UTRERAS†

Abstract. Let there be given data $y_i = f(t_i) + \varepsilon_i$, $i = 1, \cdots, n$, where f is an element of $H^m(\Omega)$, Ω is a bounded subset of R^d with smooth boundary and satisfying a uniform cone condition, and the errors ε_i are i.i.d. random variables with zero mean. To estimate f we use the Thin Plate Smoothing Spline $S_{m,n,\lambda}$ defined as the unique solution to the minimization problem

$$\underset{u \in D^{-m}L^2(R^d)}{\text{Minimize}} \ \lambda |u|_m^2 + \frac{1}{n} \sum_{i=1}^{n} [u(t_i) - y_i]^2$$

where the smoothing parameter λ is obtained using the method of Generalized Cross-Validation. Under mild assumptions on the limiting behavior of the knots, we prove that this method is asymptotically optimal, and

$$\lim_{n \to \infty} E[|f - S_{m,n,\lambda_n^*}|_{0,\Omega}^2] = 0$$

where λ_n^* is the minimizer of the Generalized Cross-Validation function.

Key words. smoothing splines, cross-validation, thin-plate, multivariate

AMS(MOS) subject classification. 65 DO5

1. Introduction. Let f be an element of $H^m(\Omega)$, where Ω is an open bounded subset of R^d with Lipschitz boundary and satisfying a uniform cone condition (cf. [8]). Let there be given data

$$y_i = f(t_i) + \varepsilon_i, \qquad i = 1, \cdots, n \tag{1.1}$$

where ε_i, $i = 1, \cdots, n$ are independent identically distributed random variables with zero mean and variance σ^2.

In order to approximate f it has been proposed to use $S_{m,n,\lambda}$, the Smoothing "Thin Plate" or D^m-spline defined for each $\lambda > 0$ as the unique solution (cf. [8], [9]) of the following minimization problem

$$\underset{u \in D^{-m}L^2(R^d)}{\text{Minimize}} \ \lambda |u|_m^2 + \frac{1}{n} \sum_{i=1}^{n} [y_i - u(t_i)]^2 \tag{1.2}$$

where

$$|u|_k^2 = \int_{R^d} \sum_{i_1, i_2, \cdots, i_k = 1}^{d} \left[\frac{\partial^k u}{\partial x_{i_1} \partial x_{i_2} \cdots \partial x_{i_k}} \right]^2, \qquad k = 0, \cdots, m \tag{1.3}$$

and

$$D^{-m}L^2(R^d) = \{u \in D'(R^d) \mid D^\alpha u \in L^2(R^d) \text{ for } |\alpha| = m\}. \tag{1.4}$$

As in the one-dimensional case (cf. [7], [20]) the smoothing spline minimizes a functional resulting of a "trade-off" between approximation to the data as measured by $(1/n)\sum_{i=1}^{n} [y_i - u(t_i)]^2$ and smoothness of the function as measured by $|u|_m^2$. In the

* Received by the editors December 24, 1985; accepted for publication April 4, 1986.

† Center for Approximation Theory, Department of Mathematics, Texas A&M University, College Station, Texas 77843-3368. Present address, Departamento de Matemáticas y Ciencias de la Computación, Facultad de Ciencias Fısicas y Matemáticas, Universidad de Chile, Casilla 5272 Correo 3, Santiago, Chile.

limiting cases, as λ goes to ∞ $S_{m,n,\lambda}$ goes to the unique element in P_{m-1}, the set of polynomials of degree less than or equal to $m-1$, minimizing $(1/n)\sum_{i=1}^{n}[y_i-u(t_i)]^2$, i.e. the least squares polynomial fitting the data. On the other hand, as λ goes to zero $S_{m,n,\lambda}$ converges to the interpolating thin plate spline (cf. [12]). Both limiting cases show that the behavior of $S_{m,n,\lambda}$ for different values of λ can be very different and a method to choose the smoothing parameter becomes necessary. Practical and theoretical evidence (cf. [7], [11], [17], [18], [19], [20], [23], [25] and the references therein) has shown that the method of Generalized Cross-Validation is a very good alternative to do this choice in one dimension. This method amounts to choosing λ by minimizing the Generalized Cross-Validation function $V_n(\lambda)$ defined as

$$V_n(\lambda)=\frac{\frac{1}{n}\sum_{i=1}^{n}[y_i-S_{m,n,\lambda}(t_i)]^2}{[1-(1/n)\operatorname{tr}(A_n(\lambda))]^2} \tag{1.5}$$

where $A_n(\lambda)$ is the matrix associating $[S_{m,n,\lambda}(t_1), S_{m,n,\lambda}(t_2), \cdots, S_{m,n,\lambda}(t_n)]^T$ to y.

For the multivariate case there has been a large number of papers dealing with applications of this method to choose λ (cf. [19], [24], [25], [26], [27], [28], [31]). Nevertheless, very little is known at present on the theoretical behavior of Cross-Validation. The first attempt to obtain theoretical properties for this spline was done in [24] where some conjectures were established. Similar conjectures and partial results were given in [19] for the case of Thin Plate Splines on bounded domains, i.e., in the case where the integration over R^d is replaced by and integration over Ω (cf. [1]-[4], [6]). Later in [6] convergence rates were given under the assumption that the smoothing parameter behaves in such a way that $\lambda \to 0$ and $n\lambda^{2m/d} \to \infty$. The same type of results had been given in [14] for the univariate case and recently in [22] for the splines used in this paper. At present there is no result concerning the behavior of Cross-Validation and telling for example whether or not the parameter obtained by this method can be expected to satisfy the conditions imposed in [22] to give the convergence rates.

Our aim in this paper is to contribute to the study of Generalized Cross-Validation showing that it satisfies an asymptotic optimality condition and to prove that under the assumption that the knots have an asymptotic behavior defined by a cumulative distribution function (c.d.f.) with bounded density, we are in position to apply the convergence results given in [22], hence proving the consistency of Generalized Cross-Validation. More precisely, as in [6], we assume that H_1 is satisfied.

H_1: There exists a c.d.f. G defined in R^d such that

$$\lim_{n\to\infty}\sup_{t\in R^d}|G(t)-G_n(t)|=0 \tag{1.6}$$

where G_n is the c.d.f. of the probability measure that assigns mass n^{-1} to t_i for $i=1,\cdots,n$ (that is,

$$G_n(t)=\sum_{x\in T_k(t)}\frac{1}{n} \tag{1.7}$$

and $T_k(t)=\{t_i \mid t_i \leqq t\}$), and the limiting distribution G has a density $g\in C^{\infty}(\bar{\Omega})$ with respect to the Lebesgue measure in $\mathbb{R}^d$ such that for all $t\in\Omega$

$$0<\alpha_1 \leqq g(t) \leqq \alpha_2$$

for some positive constants α_1, α_2.

Also define

$$T_n(\lambda)=\frac{1}{n}\sum_{i=1}^{n}[S_{m,n,\lambda}(t_i)-f(t_i)]^2 \tag{1.8}$$

and denote by $E[\cdot]$ the mathematical expectation. The main result of this paper can then be stated.

THEOREM 1.1. *Let f be an element of $H^m(\Omega)$ where Ω is open and bounded with Lipschitz Boundary and satisfies a uniform cone condition. Let the knots $t_1, t_2, \cdots, t_n$ satisfy H_1 and let λ_n^* and $\hat{\lambda}_n$ be defined by*

$$E[T_n(\hat{\lambda}_n)] = \min_{\lambda>0} [ET_n(\lambda)] \tag{1.9}$$

and

$$E[V_n(\lambda_n^*)] = \min_{\lambda>0} E[V_n(\lambda)]. \tag{1.10}$$

Then

$$\lim_{n\to\infty} \frac{E[T_n(\lambda_n^*)]}{E[T_n(\hat{\lambda}_n)]} = 1. \tag{1.11}$$

To prove this theorem, we follow the ideas in [7], [11] and develop in the second paragraph some basic inequalities connecting $E[T_n(\lambda)]$ and $E[V_n(\lambda)]$. In the third paragraph we study the eigenvalues of $A_n(\lambda)$ and obtain the behavior of $\operatorname{tr}(A_n(\lambda))$, $\operatorname{tr}(A_n^2(\lambda))$ and prove Theorem 1.1. We then apply the results from [22] to obtain the consistency results on Cross-Validation. Finally, in the fourth paragraph we discuss the extension of these results to the case of thin splines over bounded domains as described in [6].

Remark. Assumption H_1 implies that the knots satisfy a quasi-uniform condition for sufficiently large n, i.e.

QU: There exists a constant M independent of n such that

$$\frac{h}{h_{\min}} \leqq M \tag{1.12}$$

where

$$h = \sup_{t\in\Omega} \min_{i=1,\cdots,n} |t - t_i| \tag{1.13}$$

and

$$h_{\min} = \min_{\substack{i=1,\cdots,n \\ j=i+1,\cdots,n}} |t_i - t_j|. \tag{1.14}$$

Nevertheless, the converse is not true. We feel that this assumption should be sufficient to prove the result just stated but have not been able to prove it.

2. Some basic inequalities. The results in this paragraph follow the ideas in [7], [11] and their proofs are essentially multivariate versions of the ones given there.

Let $\mathbf{f}$ and $\boldsymbol{\varepsilon}$ be defined as the vectors in R^n

$$\mathbf{f} = [f(t_1), f(t_2), \cdots, f(t_n)]^T \tag{2.1}$$

and

$$\boldsymbol{\varepsilon} = [\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_n]^T. \tag{2.2}$$

Then $T_n(\lambda)$ becomes

$$T_n(\lambda) = \frac{1}{n} \|A_n(\lambda)(\mathbf{f}+\boldsymbol{\varepsilon}) - \mathbf{f}\|^2$$

where $\|\cdot\|$ is the Euclidean norm in R^n. Using now the well-known linearity of E and the independence of the random variables ε_i, we obtain

$$E[T_n(\lambda)] = \sigma^2 \mu_{2,n}(\lambda) + b_n^2(\lambda) \tag{2.3}$$

where $\mu_{2,n}(\lambda)$ and $b_n^2(\lambda)$ are defined as in [7] by

$$\mu_{2,n}(\lambda) = \frac{1}{n} \operatorname{tr}(A_n^2(\lambda)), \tag{2.4}$$

$$b_n^2(\lambda) = \frac{1}{n} \|\mathbf{f} - A_n(\lambda)\mathbf{f}\|^2. \tag{2.5}$$

On the other hand, for $E[V_n(\lambda)]$ we get

$$E[V_n(\lambda)] = \frac{b_n^2(\lambda) + \sigma^2(1 - 2\mu_{1,n}(\lambda) + \mu_{2,n}(\lambda))}{(1 - \mu_{1,n}(\lambda))^2} \tag{2.6}$$

where

$$\mu_{1,n}(\lambda) = \frac{1}{n} \operatorname{tr}(A_n(\lambda)). \tag{2.7}$$

Let us now give the following important inequality.

LEMMA 2.1. *The following inequality is satisfied for any* $\lambda > 0$

$$\frac{|E[T_n(\lambda)] - (E[V_n(\lambda)] - \sigma^2)|}{E[T_n(\lambda)]} \leqq \Delta(\lambda) \tag{2.8}$$

where

$$\Delta(\lambda) = \left[\mu_{1,n}^2(\lambda) - 2\mu_{1,n}(\lambda) + \frac{\mu_{1,n}^2(\lambda)}{\mu_{2,n}(\lambda)} \right] \frac{1}{(1 - \mu_{1,n}(\lambda))^2}. \tag{2.9}$$

Proof. Using (2.3) and (2.6), we obtain

$$\begin{aligned}
&E[T_n(\lambda)] + \sigma^2 - E[V_n(\lambda)] \\
&\quad = b_n^2(\lambda) + \sigma^2 \mu_{2,n}(\lambda) + \sigma^2 - \frac{b_n^2(\lambda) + \sigma^2(1 - 2\mu_{1,n}(\lambda) + \mu_{2,n}(\lambda))}{(1 - \mu_{1,n}(\lambda))^2} \\
&\quad = b_n^2(\lambda) + \sigma^2 \mu_{2,n}(\lambda) + \frac{\sigma^2(1 - \mu_{1,n}(\lambda))^2 - b_n^2(\lambda) - \sigma^2(1 - 2\mu_{1,n}(\lambda) + \mu_{2,n}(\lambda))}{(1 - \mu_{1,n}(\lambda))^2} \\
&\quad = b_n^2(\lambda) + \sigma^2 \mu_{2,n}(\lambda) - \frac{b_n^2(\lambda) + \sigma^2(\mu_{2,n}(\lambda) - \mu_{1,n}^2(\lambda))}{(1 - \mu_{1,n}(\lambda))^2} \\
&\quad = (b_n^2(\lambda) + \sigma^2 \mu_{2,n}(\lambda)) \left(1 - \frac{1}{(1 - \mu_{1,n}(\lambda))^2} \right) + \frac{\sigma^2 \mu_{1,n}^2(\lambda)}{(1 - \mu_{1,n}(\lambda))^2} \\
&\quad = E[T_n(\lambda)] \left[\frac{\mu_{1,n}^2(\lambda) - 2\mu_{1,n}(\lambda)}{(1 - \mu_{1,n}(\lambda))^2} + \frac{\sigma^2 \mu_{2,n}(\lambda)}{b_n^2(\lambda) + \sigma^2 \mu_{2,n}(\lambda)} \frac{\mu_{1,n}^2(\lambda)}{\mu_{2,n}(\lambda)(1 - \mu_{1,n}(\lambda))^2} \right] \\
&\quad \leqq \frac{E[T_n(\lambda)]}{(1 - \mu_{1,n}(\lambda))^2} \left[\mu_{1,n}^2(\lambda) - 2\mu_{1,n}(\lambda) + \frac{\mu_{1,n}^2(\lambda)}{\mu_{2,n}(\lambda)} \right].
\end{aligned}$$

But $E[T_n(\lambda)]$ is always different from zero since $\operatorname{tr}(A_n^2(\lambda))$ is different from zero for any f, then we can divide the last inequality by $E[T_n(\lambda)]$ and obtain the desired result. □

Thus, in order to study the behavior of $E[T_n(\lambda)]$ and $E[V_n(\lambda)]$ we must know the behavior of $b_n^2(\lambda)$, $\mu_{1,n}(\lambda)$ and $\mu_{2,n}(\lambda)$. We first study $b_n^2(\lambda)$. The study of $\mu_{1,n}(\lambda)$ and $\mu_{2,n}(\lambda)$ is done in the next paragraph.

LEMMA 2.2. *There exists a constant K_m depending only on m and Ω such that for any $\lambda > 0$ we have*

$$b_n^2(\lambda) \leqq \lambda K_m |f|_{m,\Omega}^2 \tag{2.10}$$

where

$$|f|_{k,\Omega}^2 = \int_\Omega \sum_{i_1,i_2,\cdots,i_k=1}^{d} \left[\frac{\partial^k u}{\partial x_{i_1}\,\partial x_{i_2}\cdots\partial x_{i_k}}\right]^2, \qquad k=0,\cdots,m. \tag{2.11}$$

Proof. Let s be the unique solution to the minimization problem

$$\underset{u\in D^{-m}L^2(R^d)}{\text{Minimize}}\ \lambda|u|_m^2 + \frac{1}{n}\sum_{i=1}^{n}(u(t_i)-f(t_i))^2 \tag{2.12}$$

and let f^Ω as defined in [8] be the minimum semi-norm extension of f to R^d, i.e. the solution to

$$\underset{\substack{u\in D^{-m}L^2(R^d)\\ u|_\Omega = f}}{\text{Minimize}}\ |u|_m^2. \tag{2.13}$$

Then we have

$$\lambda|s|_m^2 + \frac{1}{n}\sum_{i=1}^{n}(s(t_i)-f(t_i))^2 \leqq \lambda|f^\Omega|_m^2$$

since $f = f^\Omega$ at every point t_i. Now we apply Lemma 3.1 in [8] which asserts that there exists a constant K_m depending only on m, Ω such that

$$|f^\Omega|_m^2 \leqq K_m |f|_{m,\Omega}^2 \tag{2.14}$$

and conclude that

$$b_n^2 = \frac{1}{n}\|\mathbf{f} - A_n(\lambda)\mathbf{f}\|^2 = \frac{1}{n}\sum_{i=1}^{n}(s(t_i)-f(t_i))^2 \leqq \lambda K_m |f|_{m,\Omega}^2.$$

This concludes the proof. □

We have thus proved that $b_n^2(\lambda)$ goes to zero as λ goes to zero. To prove the converse, we will need two results on the Tikhonov regularizer f_ε of f, defined as the unique (Lemma 2.3) solution of the following minimization problem

$$\underset{u\in D^{-m}L^2(R^d)}{\text{Minimize}}\ \varepsilon|u|_m^2 + \int_\Omega (f-u)^2\, dG. \tag{2.15}$$

LEMMA 2.3. *Let f be an element of $H^m(\Omega)$ and $\varepsilon > 0$; then there exists a unique function $f_\varepsilon \in D^{-m}L^2(R^d)$ solution of problem* (2.15).

Proof. From [8] we obtain that $D^{-m}L^2(R^d)/P_{m-1}$ is a Hilbert space of continuous functions ($m > (d/2)$) with norm induced by the semi-norm $|\cdot|_m$. On the other hand, $|\cdot|_{0,\Omega}$ is a norm on P_{m-1} and g, the density of G is bounded from below and from above; then

$$u \to ((u,u))^{1/2} = \left[\varepsilon|u|_m^2 + \int_\Omega (u)^2\, dG\right]^{1/2}$$

defines a norm on $D^{-m}L^2(R^d)$. Also the linear form $u \to \int_\Omega uf$ is continuous with this norm; then there exists a unique minimizer [5] of

$$\varepsilon((u, u)) - 2\int_\Omega fu\, dG + \int_\Omega f^2\, dG$$

which is exactly the functional minimized in (2.15). □

In order to study $b_n^2(\lambda)$ we will also need the following result on the behavior of f_ε.

LEMMA 2.4. *Let $f_{n,\varepsilon}$ be the unique solution to*

$$\underset{u \in D^{-m}L^2(R^d)}{\text{Minimize}}\ \varepsilon |u|_m^2 + \frac{1}{n}\sum_{i=1}^{n} (f(t_i) - u(t_i))^2 \tag{2.16}$$

and assume that $\{t_1, t_2, \cdots, t_n\}$ satisfy H_1; then

$$\lim_{n\to\infty} \int_\Omega (f_{n,\varepsilon} - f_\varepsilon)^2\, dG = 0.$$

Proof. Clearly $f_{n,\varepsilon}$ results from minimizing an approximation to the functional minimized in (2.15) since

$$\frac{1}{n}\sum_{i=1}^{n} (f(t_i) - u(t_i))^2 = \int_\Omega (f(t) - u(t))^2\, dG_n(t).$$

More precisely, in [6] it was shown that

$$\underset{\substack{u \in D^{-m}L^2(R^d) \\ \|u\|_{m,\Omega} = 1}}{\text{Sup}} \left[\int_\Omega u^2\, dG - \frac{1}{n}\sum_{i=1}^{n} u(t_i)^2 \right] = o(1)$$

where $o(1)$ tends to zero as $n \to \infty$. We can then use the results by Fix [10] to conclude the desired result. □

LEMMA 2.5. *Let f be different from a polynomial of degree $m-1$. Then a necessary condition to have $b_n^2(\lambda) \to 0$ is that $\lambda \to 0$.*

Proof. We will prove that λ bounded away from zero implies that $b_n^2(\lambda)$ is bounded away from zero. Thus let us assume that there exist $\varepsilon > 0$ independent of n such that $\lambda \geqq \varepsilon$.

It is well known that $A_n(\lambda) = (I + \lambda\Sigma)^{-1}$ where Σ (cf. [22]) is positive semi-definite and has exactly $N = \dim(P_{m-1})$ eigenvalues equal to zero. Let Q be the orthogonal matrix diagonalizing Σ and $d_{1,n} \leqq d_{2,n} \leqq \cdots \leqq d_{n,n}$ be the eigenvalues of Σ in ascending order. Then,

$$b_n^2(\lambda) = \frac{1}{n}\|\mathbf{f} - A_n(\lambda)\mathbf{f}\|^2$$

$$= \frac{1}{n}\sum_{i=1}^{n} \left[\hat{f}_i - \frac{1}{1 + n\lambda d_{i,n}} \hat{f}_i \right]^2$$

where $\hat{f} = Q^*\mathbf{f}$. Thus

$$b_n^2(\lambda) = \frac{1}{n}\sum_{i=1}^{n} \frac{n^2\lambda^2 d_{i,n}^2}{(1 + n\lambda d_{i,n})^2} \hat{f}_i^2. \tag{2.17}$$

But f is not a polynomial of degree $m-1$, then $\hat{f}$ is not zero and $b_n^2(\lambda)$ is a strictly increasing function of λ. Thus $b_n^2(\lambda) \geqq b_n^2(\varepsilon)$.

But in [22] we proved that there exist positive constants A_1, A_2 such that for any function $u \in H^m(\Omega)$ we have

$$\frac{1}{n}\sum_{i=1}^{n}[u(t_i)]^2 \geqq A_1|u|^2_{0,\Omega} - \frac{A_2}{n^{2m/d}}|u|^2_{m,\Omega}.$$

Taking $u = f - f_{n,\varepsilon}$ and using again the fact that G has a density g bounded from above by α_2, we conclude that

$$b_n^2(\varepsilon) \geqq \frac{A_1}{\alpha_2}\int_\Omega (f - f_{n,\varepsilon})^2\, dG - O(n^{-2m/d}). \tag{2.18}$$

The last term comes from the fact that there exists a constant A_3 independent of f such that

$$|f - f_{n,\varepsilon}|^2_{m,\Omega} \leqq A_3|f|^2_{m,\Omega},$$

as it has been proved in [22]. Thus, using now Lemma 2.4, we finally obtain

$$\lim_{n\to\infty} b_n^2(\varepsilon) \geqq \int_\Omega (f - f_\varepsilon)^2\, dG.$$

In order to prove the desired result, we only need to show that f_ε is different from f. To do this, we observe that f not in P_{m-1} implies that

$$f^{\#} = \frac{\int_\Omega f^2\, dG}{\varepsilon|f|^2_m + \int_\Omega f^2\, dG} f$$

is different from f. Moreover, a straightforward calculation proves that

$$\varepsilon|f^{\#}|^2_m + \int_\Omega (f^{\#} - f)^2\, dG < \varepsilon|f|^2_m.$$

Hence f cannot be equal to f_ε since it does not minimize (2.15) and the result is proved. □

3. The matrix $A_n(\lambda)$ and the asymptotic behavior of $E[T_n(\lambda)]$ and $E[V_n(\lambda)]$. As it was seen in the last section, the quantities $\mu_{1,n}(\lambda)$ and $\mu_{2,n}(\lambda)$ play an important role in the study of $E[T_n(\lambda)]$ and $E[V_n(\lambda)]$. To begin this section, we give the following result on the behavior of these quantities for large values of n.

LEMMA 3.1. *There exist constants $l_{m,i}$ $i = 1, 2, 3, 4$ independent of λ, n such that for any $\lambda > 0$ satisfying $\lambda \leqq 1$, $n\lambda^{d/2m} \geqq 2N$, we have*

$$\frac{1}{n\lambda^{d/2m}} l_{m,1} \leqq \mu_{1,n}(\lambda) \leqq \frac{1}{n\lambda^{d/2m}} l_{m,2}, \tag{3.1}$$

$$\frac{1}{n\lambda^{d/2m}} l_{m,3} \leqq \mu_{2,n}(\lambda) \leqq \frac{1}{n\lambda^{d/2m}} l_{m,4} \tag{3.2}$$

where

$$N = \binom{d+m-1}{m-1}.$$

Proof. As remarked in the preceding section we have

$$\operatorname{tr}(A_n(\lambda)) = \sum_{i=1}^{n} \frac{1}{1 + n\lambda d_{i,n}} \tag{3.3}$$

where $d_{1,n} \leqq d_{2,n} \leqq \cdots \leqq d_{n,n}$ are the eigenvalues of Σ. But, as it has been proved in [22], there exist positive constants β_1, β_2 such that for $i \geqq N$ we have

$$\beta_1 i^{2m/d} \leqq n d_{i,n} \leqq \beta_2 i^{2m/d}. \tag{3.4}$$

Combining now (3.3) and (3.4), we see that it is sufficient to study the behavior of

$$\frac{1}{n} \sum_{i=N+1}^{n} \frac{1}{1+\beta\lambda i^{2m/d}}$$

where $\beta = \beta_1, \beta_2$. But using the fact that $1/(1+\beta\lambda i^{2m/d})$ is a decreasing function of i, we get

$$\int_{N+1}^{n+1} \frac{1}{1+\beta\lambda x^{2m/d}}\, dx \leqq \sum_{i=N+1}^{n} \frac{1}{1+\beta\lambda i^{2m/d}} \leqq \int_{N}^{n} \frac{1}{1+\beta\lambda x^{2m/d}}\, dx. \tag{3.5}$$

Introducing the change of variables $t = (\beta\lambda)^{d/2m} x$ and using the fact that $\lambda \leqq 1$, $n\lambda^{d/2m} \geqq 2N$, we obtain

$$\frac{1}{n\lambda^{d/2m}} \int_{\theta_1}^{\theta_2} \frac{1}{1+x^{2m/d}}\, dx \leqq \frac{1}{n} \sum_{i=N+1}^{n} \frac{1}{1+\beta\lambda i^{2m/d}} \leqq \frac{1}{n\lambda^{d/2m}} \int_{0}^{\infty} \frac{1}{1+x^{2m/d}}\, dx$$

where $\theta_1 = \beta^{d/2m}$ and $\theta_2 = 2N\beta^{d/2m}$. This concludes the proof of (3.1). The proof of (3.2) is analogous and we omit it. □

Let λ_n be a sequence of smoothing parameters such that

$$\lim_{n\to\infty} \lambda_n = 0, \qquad \lim_{n\to\infty} n^{2m/d} \lambda_n = \infty;$$

from Lemmas 2.2 and 3.1 we obtain that

$$\lim_{n\to\infty} E[T_n(\lambda_n)] \leqq \lim_{n\to\infty} \left\{ \lambda K_m |f|^2_{m,\Omega} + l_{m,2} \frac{1}{n\lambda_n^{d/2m}} \right\} = 0.$$

Let $\hat{\lambda}_n$ be the minimizer of $E[T_n(\lambda)]$ as defined in (1.9). It is then clear that we must have

$$\lim_{n\to\infty} E[T_n(\hat{\lambda}_n)] = 0, \tag{3.6}$$

since $E[T_n(\hat{\lambda}_n)] \leqq E[T_n(\lambda_n)]$. But $E[T_n(\lambda)] = b_n^2(\lambda) + \sigma^2 \mu_{2,n}(\lambda)$ and both terms are positive; hence we must have

$$\lim_{n\to\infty} b_n^2(\hat{\lambda}_n) = 0 \tag{3.7}$$

and

$$\lim_{n\to\infty} \mu_{2,n}(\hat{\lambda}_n) = 0. \tag{3.8}$$

Hence from Lemma 2.5 we conclude that $\lim_{n\to\infty} \hat{\lambda}_n = 0$.

Also from (3.4) it is clear that

$$\frac{n+1-N}{n} \frac{1}{[1+\beta_2 \lambda n^{2m/d}]^2} \leqq \mu_{2,n}(\lambda);$$

hence (3.8) implies that $\lim_{n\to\infty} n^{2m/d} \hat{\lambda}_n = \infty$.

We have just shown the following.

LEMMA 3.2. *As* $n \to \infty$ *we have*

$$\lim_{n\to\infty} \hat{\lambda}_n = 0, \tag{3.9}$$

$$\lim_{n\to\infty} n^{2m/d}\hat{\lambda}_n = \infty, \tag{3.10}$$

$$\lim_{n\to\infty} E[T_n(\hat{\lambda}_n)] = 0. \tag{3.11}$$

We are now going to prove that the same holds for λ_n^* the sequence of minimizers of $E[V_n(\lambda)]$. From (2.8) we get for $\lambda = \hat{\lambda}_n$

$$E[V_n(\hat{\lambda}_n)] - \sigma^2 \leqq E[T_n(\hat{\lambda}_n)](1+\Delta(\hat{\lambda}_n)). \tag{3.12}$$

But the minimizer of $E[V_n(\lambda)]$ is λ_n^*; then

$$E[V_n(\lambda_n^*)] - \sigma^2 \leqq E[V_n(\hat{\lambda}_n)] - \sigma^2 \leqq E[T_n(\hat{\lambda}_n)](1+\Delta(\hat{\lambda}_n)). \tag{3.13}$$

We already know that $E[T_n(\hat{\lambda}_n)]$ goes to zero as $n \to \infty$. Let us study the term $\Delta(\hat{\lambda}_n)$. We state the result in the following lemma.

LEMMA 3.3. *Let* λ_n *be a sequence of smoothing parameters such that* $\lambda_n \to 0$, $n\lambda_n^{d/2m} \to \infty$ *as* $n \to \infty$. *Then*

$$\lim_{n\to\infty} \Delta(\lambda_n) = 0. \tag{3.14}$$

Proof. Using Lemma 3.1, we obtain for sufficiently large n

$$\Delta(\lambda_n) \leqq \left[l_{m,2}^2 \left[\frac{1}{n\lambda_n^{d/2m}} \right]^2 + 2l_{m,2} \frac{1}{n\lambda_n^{d/2m}} + \frac{l_{m,2}^2}{l_{m,3}} \frac{1}{n\lambda_n^{d/2m}} \right] \frac{1}{(1-o(1))^2}$$

where $o(1) \leqq l_{m,2}(1/n\lambda_n^{d/2m})$ tends to zero as $n \to \infty$. Hence the result follows. □

From (3.13) and (3.14) we obtain that $E[V_n(\lambda_n^*)] - \sigma^2$ goes to zero as $n \to \infty$. In the next lemma we prove that this implies that $\lambda_n^* \to 0$ and $n^{2m/d}\lambda_n^* \to \infty$.

LEMMA 3.4. *Let* λ_n^* *be the sequence of minimizers of* $E[V_n(\lambda)]$; *then*

$$\lim_{n\to\infty} \lambda_n^* = 0, \tag{3.15}$$

$$\lim_{n\to\infty} n^{2m/d}\lambda_n^* = \infty, \tag{3.16}$$

$$\lim_{n\to\infty} E[T_n(\lambda_n^*)] = 0. \tag{3.17}$$

Proof. From (2.6) we get

$$E[V_n(\lambda_n^*)] - \sigma^2 = \frac{b_n^2(\lambda_n^*) + \sigma^2(\mu_{2,n}(\lambda_n^*) - \mu_{1,n}^2(\lambda_n^*))}{(1-\mu_{1,n}(\lambda_n^*))^2}. \tag{3.18}$$

But a straightforward application of the Schwartz inequality gives

$$\mu_{1,n}^2(\lambda_n^*) = \frac{1}{n^2}\left[\sum_{i=1}^{n} \frac{1}{1+n\lambda d_{i,n}} \right]^2 \leqq \frac{1}{n} \sum_{i=1}^{n} \frac{1}{(1+n\lambda d_{i,n})^2} = \mu_{2,n}(\lambda_n^*);$$

thus the two terms in the numerator of (3.18) are positive and the denominator is not larger than 1. Therefore a necessary condition to have $\lim_{n\to\infty} E[V_n(\lambda_n^*)] - \sigma^2 = 0$ is that

$$\lim_{n\to\infty} b_n^2(\lambda_n^*) = 0 \tag{3.19}$$

and

$$\lim_{n\to\infty} \frac{\mu_{2,n}(\lambda_n^*) - \mu_{1,n}^2(\lambda_n^*)}{(1-\mu_{1,n}(\lambda_n^*))^2} = 0. \tag{3.20}$$

From (3.18) and Lemma 2.5 we immediately obtain (3.15). To obtain the second limit, we first obtain a more convenient expression for the numerator of (3.20). Using the fact that the variance of a group of numbers is equal to the average of the squares minus the square of the average, we get

$$\mu_{2,n}(\lambda_n^*) - \mu_{1,n}^2(\lambda_n^*) = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{1+n\lambda_n^* d_{i,n}} - \mu_{1,n}(\lambda_n^*)\right]^2. \tag{3.21}$$

We will prove the result by contradiction. Let us assume that there exists a subsequence of $\{\lambda_n^*\}$ that we shall denote $\{\lambda_n^*\}$ also, such that $\theta_n = \lambda_n^* n^{2m/d}$ remains bounded as $n\to\infty$. We will prove that in this case (3.20) cannot occur. We divide the proof into two parts.

Case I. The subsequence (or a subsequence of it) satisfies $\lim_{n\to\infty}\theta_n = 0$.

In this case we can write

$$\frac{1}{1+\lambda_n^* n d_{i,n}} = \frac{1}{1+\theta_n\kappa_{i,n}} = 1-\theta_n\kappa_{i,n} + O(\theta_n^2) \tag{3.22}$$

where

$$\kappa_{i,n} = \frac{n d_{i,n}}{n^{2m/d}} \leqq \beta_2\left(\frac{i}{n}\right)^{2m/d} \leqq \beta_2.$$

Using (3.22), we obtain for $\mu_{1,n}(\lambda_n^*)$ and $\mu_{2,n}(\lambda_n^*)$.

$$\mu_{1,n}(\lambda_n^*) = 1-\theta_n\frac{1}{n}\sum_{i=N+1}^{n}\kappa_{i,n} + O(\theta_n^2)$$

and

$$\mu_{2,n}(\lambda_n^*) = 1-2\theta_n\frac{1}{n}\sum_{i=N+1}^{n}\kappa_{i,n} + O(\theta_n^2).$$

Thus the numerator and denominator of (3.19) become

$$\begin{aligned}\mu_{2n}(\lambda_n^*) - \mu_{1,n}^2(\lambda_n^*) &= \frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{1+\theta_n\kappa_{i,n}} - \mu_{1,n}(\lambda_n^*)\right]^2 \\ &= \theta_n^2\frac{1}{n}\sum_{i=N+1}^{n}\left[\kappa_{i,n} - \frac{1}{n}\sum_{i=N+1}^{n}\kappa_{i,n}\right]^2 + O(\theta_n^3),\end{aligned} \tag{3.23}$$

$$(1-\mu_{1,n}(\lambda_n^*))^2 = \theta_n^2\left[\frac{1}{n}\sum_{i=N+1}^{n}\kappa_{i,n}\right]^2 + O(\theta_n^3). \tag{3.24}$$

Using now (3.4), we obtain $\beta_1(i/n)^{2m/d} \leqq \kappa_{i,n} \leqq \beta_2(i/n)^{2m/d}$. Thus

$$\beta_1\frac{1}{n}\sum_{i=N+1}^{n}\left(\frac{i}{n}\right)^{2m/d} \leqq \frac{1}{n}\sum_{i=N+1}^{n}\kappa_{i,n} \leqq \beta_2\frac{1}{n}\sum_{i=N+1}^{n}\left(\frac{i}{n}\right)^{2m/d},$$

$$\beta_1\int_{N/n}^{1} x^{2m/d}\,dx \leqq \frac{1}{n}\sum_{i=N+1}^{n}\kappa_{i,n} \leqq \beta_2\int_{0}^{1+(1/n)} x^{2m/d}\,dx,$$

$$\beta_1\frac{d}{2m+d}\left(1-O\left(\frac{1}{n}\right)\right) \leqq \frac{1}{n}\sum_{i=N+1}^{n}\kappa_{i,n} \leqq \beta_2\frac{d}{2m+d}\left(1+O\left(\frac{1}{n}\right)\right)$$

and for sufficiently large n, we finally obtain

$$\gamma_1 \leqq \frac{1}{n} \sum_{i=N+1}^{n} \kappa_{i,n} \leqq \gamma_2.$$

Let us now study (3.23)

$$\frac{1}{n} \sum_{i=N+1}^{n} \left[\kappa_{i,n} - \frac{1}{n} \sum_{i=N+1}^{n} \kappa_{i,n} \right]^2 \geqq \frac{1}{n} \sum_{i=N+1}^{K} (\kappa_{i,n} - \gamma_1)^2$$

$$\geqq \int_{(N+1)/n}^{\gamma_3} (x^{2m/d} - \gamma_1)^2 \, dx$$

where

$$K = \operatorname{Max} \left\{ i \,\middle|\, \left(1 + \beta_1 \left(\frac{i}{n} \right)^{2m/d} \right)^{-1} \leqq \gamma_1 \right\} \quad \text{and} \quad \gamma_3 = \left[\frac{1}{\beta_1} \left(\frac{1}{\gamma_1} - 1 \right) \right]^{d/2m}.$$

Clearly $K > N+1$ for sufficiently large n and $\gamma_3 > 0$ also for sufficiently large n. Thus

$$\frac{\mu_{2,n}(\lambda_n^*) - \mu_{1,n}^2(\lambda_n^*)}{(1 - \mu_{1,n}(\lambda^*))^2} \geqq \frac{\theta_n^2 \gamma_4 + O(\theta_n^3)}{\theta_n^2 \gamma_2 + O(\theta_n^3)},$$

which proves that this term cannot go to zero as $n \to \infty$.

Case II. $\{\theta_n\}$ or a subsequence of it is bounded and bounded away from zero, i.e. there exist positive constants c_1 and c_2 such that for any n we have $c_1 \leqq \theta_n \leqq c_2$.

In this case we have

$$\mu_{1,n}(\lambda_n^*) = \frac{1}{n} \sum_{i=N+1}^{n} \frac{1}{1 + \theta_n \kappa_{i,n}}$$

$$\geqq \frac{1}{n} \sum_{i=N+1}^{n} \frac{1}{1 + c_2 \kappa_{i,n}}$$

$$\geqq \frac{1}{n} \sum_{i=N+1}^{n} \frac{1}{1 + c_2 \beta_2 (i/n)^{2m/d}}$$

$$\geqq \int_{(N+1)/n}^{1} \frac{1}{1 + c_2 \beta_2 x^{2m/d}} \, dx$$

$$\geqq \int_{0}^{1} \frac{1}{1 + c_2 \beta_2 x^{2m/d}} \, dx \left[1 - O\left(\frac{1}{n} \right) \right].$$

Thus for sufficiently large n we have

$$\mu_{1,n}(\lambda_n^*) \geqq c_4 > 0.$$

Using this, (3.21) and the bounds for $\kappa_{i,n}$ we get

$$\mu_{2,n}(\lambda_n^*) - \mu_{1,n}^2(\lambda_n^*) = \frac{1}{n} \sum_{i=N+1}^{n} \left[\frac{1}{1 + \theta_n \kappa_{i,n}} - \mu_{1,n}(\lambda_n^*) \right]^2$$

$$\geqq \frac{1}{n} \sum_{i=N+1}^{K} \left[\frac{1}{1 + c_2 \beta_1 (i/n)^{2m/d}} - c_4 \right]^2$$

$$\geqq \int_{(N+1)/n}^{c_5} \left[\frac{1}{1 + c_2 \beta_1 x^{2m/d}} - c_4 \right]^2$$

where

$$K = \operatorname{Max} \{ i \,|\, (1 + \beta_1 (i/n)^{2m/d})^{-1} \leqq c_4 \} \quad \text{and} \quad c_5 = \operatorname{Max} \{ x \,|\, (1 + \beta_2 c_2 x^{2m/d})^{-1} \leqq c_4 \}.$$

Clearly $K > N+1$ for sufficiently large n, and $c_5 > 0$. Hence for sufficiently large n we conclude that

$$\mu_{2,n}(\lambda_n^*) - \mu_{1,n}^2(\lambda_n^*) \geqq \int_0^{c_5} \left[\frac{1}{1 + c_2\beta_1\beta_1 x^{2m/d}} - c_4 \right]^2 dx \left(1 - o\left(\frac{1}{n}\right)\right).$$

This inequality says that the numerator in (3.20) cannot tend to zero, which is a contradiction since, as already seen, the denominator in (3.20) is bounded.

We have thus proved that we must have

$$\lim_{n\to\infty} n^{2m/d}\lambda_n^* = +\infty$$

and using Lemma 3.1 we conclude that

$$\lim_{n\to\infty} \mu_{2,n}(\lambda_n^*) = 0.$$

This together with (3.19) proves that

$$\lim_{n\to\infty} E[T_n(\lambda_n^*)] = \lim_{n\to\infty} b_n^2(\lambda_n^*) + \sigma^2\mu_{2,n}(\lambda_n^*) = 0$$

and the proof is complete. □

We are now ready to prove Theorem 1.1.

Proof of Theorem 1.1. Using Lemma 2.1 with $\lambda = \lambda_n^*$, we get

$$E[T_n(\lambda_n^*)](1 - \Delta(\lambda_n^*)) \leqq E[V_n(\lambda_n^*)] - \sigma^2 \tag{3.25}$$

and using now (3.13) to bound the right-hand side, we get

$$E[T_n(\lambda_n^*)](1 - \Delta(\lambda_n^*)) \leqq E[T_n(\hat{\lambda}_n)](1 + \Delta(\hat{\lambda}_n)).$$

But from Lemma 3.3 we have $\lambda_n^* \to 0$ and $n^{2m/d}\lambda_n^* \to \infty$; then by Lemma 3.2 we get

$$\lim_{n\to\infty} \Delta(\lambda_n^*) = 0.$$

We can then divide both terms in (3.25) by $1 - \Delta(\lambda_n^*)$ to obtain

$$E[T_n(\lambda_n^*)] \leqq E[T_n(\hat{\lambda}_n)] \frac{1 + \Delta(\hat{\lambda}_n)}{1 - \Delta(\lambda_n^*)}.$$

Thus

$$1 \leqq \lim_{n\to\infty} \frac{E[T_n(\lambda_n^*)]}{E[T_n(\hat{\lambda}_n)]} \leqq \lim_{n\to\infty} \frac{1 + \Delta(\hat{\lambda}_n)}{1 - \Delta(\lambda_n^*)} = 1.$$

This concludes the proof. □

We are now in position to apply the main result in [22] to obtain the following consistency result for Generalized Cross-Validation.

THEOREM 3.5. *Let f be an element of $H^m(\Omega)$ where Ω is an open bounded subset of R^d with Lipschitz boundary and satisfying a uniform cone condition. Let the knots $t_1, t_2, \cdots, t_n$ satisfy H_1 and let λ_n^* be the minimizer of $E[V_n(\lambda)]$. Then*

$$\lim_{n\to\infty} E[|f - S_{m,n,\lambda_n^*}|_{0,\Omega}^2] = 0. \tag{3.26}$$

Proof. As it was proved in Lemma 3.3, we have $\lambda_n^* \to 0$ and $n^{2m/d}\lambda_n^* \to \infty$. This, together with the assumptions on Ω and the knots, allow us to apply Theorem 1.1 in [22] to conclude that

$$E[|f - S_{m,n,\lambda_n^*}|_{0,\Omega}^2] \leqq P\lambda_n^* |f|_{m,\Omega}^2 + Q\frac{\sigma^2}{n(\lambda_n^*)^{d/2m}}.$$

Thus as $n \to \infty$ we obtain the required result. □

This last result gives only the fact that the Integrated Mean Square Error converges to zero. From [22] it is apparent that better results can be obtained only if we are able to prove that the following behavior also holds

$$\lim_{n\to\infty} n^{2m/(2m+d)}\lambda_n^* = \infty.$$

The proof of this kind of result requires a better knowledge of $E[T_n(\lambda)]$, which we have not been able to obtain even in the univariate case.

4. Extensions. As in the preceding sections, let $\Omega \subseteq R^d$, $d < 2m$, be an open bounded domain with Lipschitz boundary and satisying a uniform cone condition. Let $S_{m,n,\lambda}^b$ be the "Thin Plate" spline over the bounded domain defined as the unique solution to

$$\underset{u\in H^m(\Omega)}{\text{Minimize}}\ \lambda|u|_{m,\Omega} + \frac{1}{n}\sum_{i=1}^{n}[y_i - u(t_i)]^2. \tag{4.1}$$

This kind of smoothing splines was defined in [1], [2], [3] and investigated later in [19]. More recently, in [6] Cox gave a very detailed description of the Integrated Mean Square Error between f and the spline $S_{m,n,\lambda}^b$. More precisely he proves that if $\lambda \to 0$ and $n^{2m/d}\lambda \to \infty$ as $n \to \infty$, there exist constants P, Q independent of n such that

$$E[|f - S_{m,n,\lambda}^b|_{k,\Omega}^2] \leqq P\lambda^{(m-k)/m}|f|_{m,\Omega}^2 + Q\frac{\sigma^2}{n\lambda^{(2k+d)/2m}}, \qquad k = 0, \cdots, m. \tag{4.2}$$

It is obvious that the consistency result also holds in this case since the preceding proofs rely essentially on the behavior of the eigenvalues of $A_n(\lambda)$ and it was proved in [22] that these eigenvalues behave in the same form for both cases. Once the behavior of λ_n^* has been obtained, the application of Cox's result gives the desired consistency.

Acknowledgments. The author wishes to thank the members of the Center for Approximation Theory at Texas A&M University, where this work was developed, for their support and encouragement during all his stay in College Station. Also, the author wishes to thank the Mathematics Research Center at Madison for providing the opportunity to visit the Center and discuss the development of this work with Professors Grace Wahba and Dennis Cox.

REFERENCES

[1] M. ATTEIA, *Generalisation de la definition et des proprietés des spline-fonctions*, C.R. Acad. Sci. Paris, 260 (1965), pp. 3550-3553.

[2] ———, *Spline-fonctions generalisées*, C.R. Acad. Sci. Paris, 261 (1965), pp. 2149-2152.

[3] ———, *Existence et Determination des Fonctions Spline à Plusieurs Variables*, C.R. Acad. Sci. Paris, 262 (1966), pp. 575-578.

[4] ———, *Fonctions "spline" et noyaux reproduissants d'Aronzajn-Bergman*, RAIRO Rech. Opér., 3 (1070), pp. 31-43.

[5] PHILIPPE G. CIARLET, *The Finite Element Method for Elliptic Problems*, North-Holland, Amsterdam, 1978.
[6] DENNIS D. COX, *Multivariate smoothing spline functions*, SIAM J. Numer. Anal., 21 (1984), pp. 789-813.
[7] P. CRAVEN AND G. WAHBA, *Smoothing noisy data with spline functions: estimating the correct degree of smoothing by the method of generalized cross-validation*, Numer. Math., 31 (1979), pp. 377-403.
[8] JEAN DUCHON, *Interpolation des fonctions de deux variables suivant le principle de la flexion des plaques minces*, RAIRO Anal. Numér., 10 (1976), pp. 5-12.
[9] ———, *Splines minimizing rotation invariant semi-norms in Sobolev spaces*, in Constructive Theory of Functions of Several Variables, Proceedings of a Conference in Oberwolfach, W. Schempp and K. Zeller, eds., Springer-Verlag, Berlin, 1977.
[10] G. FIX, *Effects of quadrature errors in finite element approximation of steady state, eigenvalue and parabolic problems*, in The Mathematical Foundations of the Finite Element Method with Applications to Partial Differential Equations, A. K. Aziz, ed., Academic Press, New York, 1972.
[11] G. GOLUB, M. HEATH AND G. WAHBA, *Generalized cross-validation as a method for choosing a good ridge parameter*, Technometrics, 21 (1979), pp. 215-223.
[12] J. L. JOLY, *Theorèmes de convergence des fonctions "spline" generales d'interpolation et d'adjustement*, C.R. Acad. Sci. Paris Ser. A, 264 (1967), pp. 126-128.
[13] L. PAIHUA, *Quelques méthodes numériques pour le calcul de fonctions spline à une et deux variables*, Ph.D. dissertation, U. de Grenoble, Grenoble, France, 1978.
[14] D. RAGOZIN, *Error bounds for derivative estimates based on spline smoothing of exact or noisy data*, J. Approx. Theory, 37 (1983), pp. 335-355.
[15] J. RICE AND M. ROSEMBLATT, *Integrated mean square error of a smoothing spline*, J. Approx. Theory, 33 (1981), pp. 353-369.
[16] ———, *Smoothing splines: regression, derivatives and deconvolution*, Ann. Statist., 11 (1983), pp. 141-156.
[17] B. W. SILBERMAN, *A fast and efficient cross-validation method for smoothing parameter choice in spline regression*, J. Amer. Statist. Assoc., 79 (1984), pp. 584-589.
[18] F. I. UTRERAS, *Quelques resultats d'optimalité pour la methode de validation croisée*, Séminaire d'Analyse Numérique No. 301, University of Grenoble, Grenoble, France, 1978.
[19] ———, *Cross-validation techniques for smoothing spline functions in one or two dimensions*, in Smoothing Techniques for Curve Estimation, Proceedings of a Conference in Heidelberg, M. Rosemblatt and Th. Gasser, eds., Lecture Notes in Mathematics 757, Springer-Verlag, Heidelberg, 1979.
[20] ———, *Optimal smoothing of noisy data using spline functions*, this Journal, 2 (1981), pp. 349-362.
[21] ———, *Natural spline functions: their associated eigenvalue problem*, Numer. Math., 42 (1983), pp. 107-117.
[22] ———, *Convergence rates for multivariate smoothing spline functions*, Technical Report No. 75, Center for Approximation Theory, Texas A&M University, College Station, TX, 1985.
[23] G. WAHBA, *Smoothing noisy data with spline functions*, Numerische Mathematik, 24 (1975), pp. 383-393.
[24] ———, *Convergence rates of 'thin plate' smoothing splines when the data are noisy*, in Smoothing Techniques in Curve Estimation, Proceedings of a Conference in Heidelberg, M. Rosemblatt and Th. Gasser, eds., Lecture Notes in Mathematics 757, Springer-Verlag, Heidelberg, 1979.
[25] ———, *Spline bases, regularization and generalized cross-validation for solving approximation problems with large quantities of nosy data*, in Approximation Theory III, Proceedings of a Conference in Austin, Texas, W. E. Cheney, ed., Academic Press, New York, 1980, pp. 905-912.
[26] G. WAHBA AND J. WENDELBERGER, *Some new mathematical methods for variational objective analysis using splines and cross-validation*, Monthly Weather Review, 108 (1980), pp. 36-57.
[27] G. WAHBA, *Spline interpolation and smoothing on the sphere*, this Journal, 2 (1981), pp. 5-16.
[28] ———, *Constrained regularization for ill-posed linear operator equations with applications in meteorology and medicine*, in Statistical Decision Theory and Related Topics, Vol. 2, S. S. Gupta and J. O. Berger, eds., Academic Press, New York, 1982.
[29] ———, *Cross-validated spine methods for the estimation of multivariate functions from data on functionals*, in Statistics: An Appraisal, Proceedings of a Conference in the Iowa Statistical Laboratory, H. A. David and H. T. David, eds., The Iowa State University Press, 1984, pp. 205-235.
[30] ———, *Variational methods for multidimensional inverse problems*, in Advances in Remote Sensing Retrieval Methods, Proceedings of a Conference in Williamsburg, Virginia, A. Deepak, ed., to appear.
[31] J. WENDELBERGER, *The computations of Laplacian smoothing splines with examples*, Technical Report 648, Dept. of Statistics, University of Wisconsin, Madison, 1981.

SIAM J. SCI. STAT. COMPUT
Vol. 8, No. 4, July 1987
© 1987 Society for Industrial and Applied Mathematics
012

FURTHER SYMMETRIES OF IN-PLACE FFTS*

WILLIAM L. BRIGGS†

Abstract. It has long been known that an in-place version of the Fast Fourier Transform (FFT) exists for real sequences of data. More recently, in-place FFTs have been devised for real sequences with even, odd, or quarter wave symmetries. All of these symmetric FFTs take the input sequence in scrambled (bit-reversed) order and produce the transform sequence in natural order. For many applications, this is the opposite of what is needed, i.e., one would like to provide the input sequence in natural order. In this paper, an in-place version of the FFT is presented which takes a real sequence in natural order and produces the transform in scrambled order. The algorithm requires half of the operations and storage of the complex algorithms. Analogous in-place algorithms are also given for naturally ordered even, odd, quarter wave even and quarter wave odd sequences.

Key words. Fast Fourier Transform (FFT), symmetric transforms

AMS(MOS) subject classifications. 65, 68

1. Introduction. In a recent paper, Swarztrauber [6] makes a valuable contribution to the understanding of the Fast Fourier Transform (FFT) by presenting the underlying symmetries which relate the various forms of the discrete Fourier transform. These symmetries lead to in-place algorithms for evaluating the real transform as well as the sine, cosine and quarter wave transforms. These in-place algorithms, which we shall call *compact*, are generally more economical in both computation costs and storage than previous algorithms which require pre- and post-processing of the data. This study proceeds in the spirit of Swarztrauber's paper by identifying an additional set of symmetries which leads to a new real version of the FFT. This version is an in-place algorithm which seems to have some practical advantages over previous versions.

Before beginning, it might be useful to give a brief qualitative discussion of the various FFT algorithms which presently exist. One general issue which distinguishes the various FFT algorithms is the need for auxiliary storage. The original FFT algorithms [3], [4] are in-place algorithms, i.e., an N-point sequence can be transformed within the N storage locations of the input sequence. The price which is paid for this convenience is that either the input sequence or the output sequence appears in scrambled (or bit-reversed) order. There are four canonical in-place FFT algorithms (summarized on p. 181 of [2]). The original Cooley-Tukey version (CT1) takes the input sequence in natural order, produces the transform in scrambled order and also uses powers of $w_N = e^{i2\pi/N}$ in scrambled order. Another version, which has also come to be called Cooley-Tukey (CT2), requires the input sequence in scrambled order, produces the transform in natural order and uses powers of w_N in natural order. This version is most easily derived by the splitting approach which will be used in the following section. These two versions are associated with the term "decimation in time." There are also two "decimation in frequency" algorithms. The original Gentleman-Sande algorithm (GS1) takes the input sequence in natural order, produces the transform in scrambled order and uses powers of w_N in natural order. Finally, a rearrangement of GS1 produces a fourth in-place version (GS2) which takes the input

* Received by the editors November 6, 1985; accepted for publication (in revised form) July 16, 1986. This work was supported by National Science Foundation grant DMS-8504350.

† Mathematics Department, University of Colorado at Denver, Denver, Colorado 80202.

sequence in scrambled order, produces the transform in natural order and requires powers of w_N in scrambled order. It may be shown that CT1 and GS2 are formal inverses of each other, as are CT2 and GS1. For the general complex FFT, versions CT2 and GS1 have developed a preference because the powers of w_N appear in natural order. These two versions also offer both choices for the ordering of the input and output sequences.

Some interesting issues arise in considering special forms of the FFT. The version CT2 has the property that the partial transforms which are produced at intermediate stages possess the same properties as the full transform. For example, a real input sequence has a conjugate symmetric transform and CT2 will generate partial transforms which are also conjugate symmetric. This property leads to a compact FFT for real sequences which requires half the operations and storage of the complex FFT. This real version is now known as Edson's algorithm [1]. Unfortunately, the other three versions of the FFT do not produce intermediate transforms with the properties of the full transform. Thus, compact, real versions of these three algorithms have never been developed.

In summary, only one of the four in-place FFTs yields a compact real form. This version, Edson's algorithm, takes its input in scrambled order and produces a naturally ordered transform. This paper results from the observation that while Edson's algorithm offers significant savings, it does "run in the wrong direction." Many applications, such as the solution of elliptic boundary value problems and many procedures in image processing, provide real data in natural order. Furthermore, in many of these same applications, the forward transform of the input data is ultimately followed by the inverse transform. This suggests a need for a compact FFT which takes a real sequence in *natural* order and produces a conjugate symmetric transform in *scrambled* order. The inverse transform would then take the scrambled transform sequence and produce a real output sequence in natural order. This procedure eliminates the scrambling of input and output sequences. It does require that all operations which are done in the transform space be done with respect to permuted indices. This requires $O(N)$ storage which is negligible in problems in two or more dimensions.

The above discussion has dealt entirely with in-place algorithms and the complications of scrambling input and output sequences. There are also two classes of algorithms which avoid the scrambling of either input or output sequences. The first class of algorithms, generally associated with the name Stockham [5], avoids the scrambling by using an additional array of storage which doubles the storage requirement of the algorithm. Thus, the Stockham algorithms are no longer in-place. The second class of algorithms which avoid scrambling is the class of prime factor algorithms [7]. These algorithms achieve the ideal combination of being in-place and avoiding scrambling of data. When applied to complex sequences of suitable length N, the prime factor algorithms show some improvement over the original in-place algorithms. At this time, however, the prime factor algorithms do not have real (or other symmetric) forms. Therefore, the question of a superior FFT for real data remains open.

We now turn to the question of finding a compact version of the FFT which takes a real, ordered sequence into its conjugate symmetric, scrambled transform. This will be done in two steps. Since the splitting approach produces an input sequence in scrambled order, we begin (§ 2) by deriving an FFT which takes a conjugate symmetric sequence and produces its real transform. In § 3, this algorithm is formally inverted to give the required real FFT. In § 4, some concluding remarks are made about the other (sine, cosine and quarter wave) symmetric transforms.

2. An FFT for conjugate symmetric sequences. In all that follows, it is assumed that N, the length of the input sequence, is a power of two. Given a complex sequence x_n of length N, the discrete Fourier transform (DFT) is given by

$$X_k = \sum_{n=0}^{N-1} x_n w_N^{-nk}, \qquad 0 \leqq k \leqq N-1 \tag{1}$$

where $w_N = e^{i2\pi/N}$. At this point, a brief presentation of the splitting approach to FFTs might be useful since it underlies everything that follows. The input sequence x_n may be split into two subsequences $y_n = x_{2n}$ and $z_n = x_{2n+1}$ where $0 \leqq n \leqq (N/2)-1$. Incorporating these subsequences into (1), we obtain

$$X_k = \sum_{n=0}^{(N/2)-1} y_n w_N^{-2nk} + w_N^{-k} \sum_{n=0}^{(N/2)-1} z_n w_N^{-2nk}.$$

When we use the fact that $w_N^{-2nk} = w_{(N/2)}^{-nk}$, these two sums may be identified as the DFTs of y_n and z_n, respectively, which we denote by Y_k and Z_k for $0 \leqq k \leqq (N/2)-1$. This gives

$$X_k = Y_k + w_N^{-k} Z_k \quad \text{for } 0 \leqq k \leqq (N/2)-1. \tag{2a}$$

To find X_k for $(N/2) \leqq k \leqq N-1$, note that Y_k and Z_k have period $N/2$ and that $w_N^{N/2} = -1$. This gives

$$X_{k+(N/2)} = Y_k - w_N^{-k} Z_k \quad \text{for } 0 \leqq k \leqq (N/2)-1. \tag{2b}$$

Expressions (2a, b) are the combine (or butterfly) formulas for the complex FFT. They express the DFT of an N-point sequence in terms of the DFTs of two $(N/2)$-point sequences. The procedure may be repeated to express Y_k and Z_k in terms of $(N/4)$-point transforms. After $\log_2 N$ steps, the problem reduces to finding the transform of N 1-point sequences. This splitting process results in the $O(N \log_2 N)$ complexity of the FFT. Note that each successive splitting of the input sequence x_n leads to the requirement that the input sequence (for this particular version of the FFT) must be in bit-reversed order. We now use this splitting strategy to find a compact FFT for conjugate symmetric sequences.

Recall that a sequence which satisfies the property $x_{N-n} = \bar{x}_n$ is called *conjugate symmetric* (CS). It can be shown that if x_n is CS, then its transform X_k is real. The goal of this section is to find an FFT algorithm for CS sequences. In particular, we will look for an algorithm which takes a CS sequence in scrambled order and produces a real transform sequence in natural order. Then by formally inverting this algorithm, we will produce a new FFT algorithm for real sequences in natural order.

Let x_n be a CS sequence of length $N = 2^m$. Define a splitting of x_n by $y_n = x_{2n}$, $z_n = x_{2n+1}$. It may be shown that y_n is also CS, while z_n satisfies the property $z_n = \bar{z}_{(N/2)-n-1}$ which we shall call *quarter wave conjugate symmetry* (QCS). A further splitting of y_n will produce a CS sequence and a QCS sequence. In order to study the symmetries produced when a QCS is split, let x_n be a QCS sequence of length N and again let $y_n = x_{2n}$ and $z_n = x_{2n+1}$. Individually, y_n and z_n possess no particular symmetry. However, noting that

$$y_n = x_{2n} = \bar{x}_{N-2n-1} = \bar{x}_{2((N/2)-n-1)+1} = \bar{z}_{(N/2)-n-1},$$

we see that y_n and z_n satisfy an intersequence symmetry. We shall call two such sequences *dual sequences* and denote them D, D^*. Finally, we must determine the effect of splitting two dual sequences. Let y_n and z_n be two dual sequences of length N. The splitting of y_n produces two sequences $y_n^{(1)} = y_{2n}$, $z_n^{(1)} = y_{2n+1}$, while the splitting

of z_n produces $y_n^{(2)} = z_{2n}$, $z_n^{(2)} = z_{2n+1}$. These four subsequences have the property that $y_n^{(1)}$ and $z_n^{(2)}$ are dual and $y_n^{(2)}$ and $z_n^{(1)}$ are dual. For example,

$$y_n^{(1)} = y_{2n} = \bar{z}_{N-2n-1} = \bar{z}_{2((N/2)-n-1)+1} = \bar{z}^{(2)}_{(N/2)-n-1}.$$

It now follows that subsequent splittings of the dual pairs $y_n^{(1)}$, $z_n^{(2)}$ or $y_n^{(2)}$, $z_n^{(1)}$ will produce additional dual pairs of half the length. The symmetries induced by the splitting algorithm when applied to a CS sequence of length $N = 16$ are illustrated in Fig. 1.

Having seen the symmetries which are produced when a CS sequence is split, it is now necessary to determine how each of these subsequences transforms. As mentioned, a CS sequence x_n is transformed by the DFT into a real sequence X_k. Now let x_n be a QCS sequence of length N. Then

$$\begin{aligned} X_k &= \sum_{n=0}^{N-1} x_n w_N^{-nk} = \sum_{n=0}^{N-1} x_{N-n-1} w_N^{-(N-n-1)k} \\ &= w_N^k \sum_{n=0}^{N-1} \bar{x}_n w^{nk} = w_N^k \bar{X}_k. \end{aligned} \tag{3}$$

The useful fact that emerges from this property is that the transform of a QCS sequence may be written as

$$X_k = w_N^{k/2} \tilde{X}_k \quad \text{where } \tilde{X}_k \in \mathbf{R}. \tag{4}$$

This property will allow for the elimination of complex arithmetic.

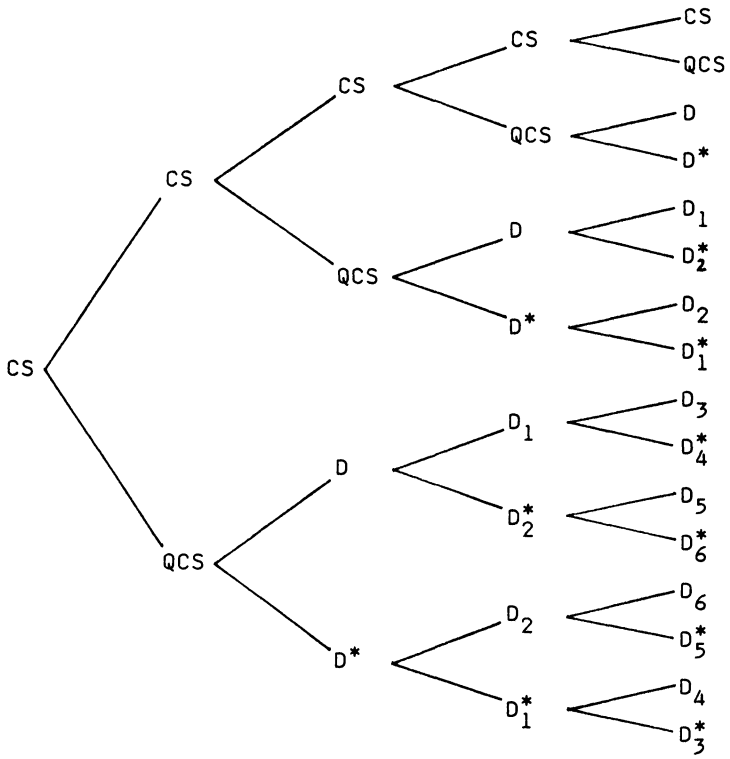


FIG. 1. *The symmetries induced when a conjugate symmetric sequence of length $N = 16$ is split.*

Finally, it must be determined how a pair of dual sequences transforms. Let y_n and z_n be dual sequences of length N. We see that

$$\begin{aligned} Y_k &= \sum_{n=0}^{N-1} y_n w_N^{-nk} = \sum_{n=0}^{N-1} \bar{z}_{N-n-1} w_N^{-nk} \\ &= w_N^k \sum_{n=0}^{N-1} \bar{z}_n w^{nk} = w_N^k \bar{Z}_k. \end{aligned} \tag{5}$$

It is now necessary to find the actual combine formulas for each stage of the algorithm. First, consider the combining of a CS and a QCS sequence. Let Y_k be the (real) transform of a CS sequence y_n and let Z_k be the transform of a QCS sequence z_n, all of length $N/2$. The resulting (real) transform X_k of the associated CS sequence x_n is given by the usual combine formula (2)

$$X_k = Y_k + w_N^{-k} Z_k = Y_k + \tilde{Z}_k, \tag{6a}$$

$$X_{k+(N/2)} = Y_k - w_N^{-k} Z_k = Y_k - \tilde{Z}_k, \qquad 0 \leqq k \leqq (N/2)-1 \tag{6b}$$

where (4) and the fact that $w_N^{-k} = w_{N/2}^{-k/2}$ have been used. Since Y_k and $\tilde{Z}_k$ are real sequences, these combine steps may be carried out in real arithmetic.

Now consider a step in which two dual sequences are combined to form a QCS sequence. Let Y_k and Z_k be the transforms of two dual sequences of length $N/2$. The resulting transform X_k of the associated QCS sequence x_n is given by the splitting formula (2)

$$\begin{aligned} X_k &= Y_k + w_N^{-k} Z_k = Y_k + w_N^k \bar{Y}_k, \\ X_{k+(N/2)} &= Y_k - w_N^{-k} Z_k = Y_k - w_N^k \bar{Y}_k, \end{aligned} \qquad 0 \leqq k \leqq (N/2)-1$$

where property (5) has been used. But since X_k is the transform of a QCS sequence, by (4) we have

$$\begin{aligned} \tilde{X}_k &= w_N^{-k/2} X_k \\ &= w_N^{-k/2} Y_k + w_N^{k/2} \bar{Y}_k = 2 \operatorname{Re}\{w_N^{-k/2} Y_k\}, \qquad 0 \leqq k \leqq (N/2)-1, \end{aligned} \tag{7a}$$

$$\begin{aligned} \tilde{X}_{k+(N/2)} &= -i w_N^{-k/2} X_{k+(N/2)} \\ &= -i(w_N^{-k/2} Y_k - w_N^{k/2} \bar{Y}_k) = 2 \operatorname{Im}\{w_N^{-k/2} Y_k\}, \qquad 0 \leqq k \leqq (N/2)-1. \end{aligned} \tag{7b}$$

This result suggests that the transform of only one dual sequence Y_k needs to be stored. Instead of storing the transforms Y_k and Z_k, a saving may be achieved by storing only $\operatorname{Re}\{Y_k\}$ and $\operatorname{Im}\{Y_k\}$, $0 \leqq k \leqq (N/2)-1$, as real variables in the same locations. The combine formulas (7) may then be written as

$$\tilde{X}_k = 2[\operatorname{Re}\{Y_k\} c'_k + \operatorname{Im}\{Y_k\} s'_k], \tag{8a}$$

$$\tilde{X}_{k+(N/2)} = 2[\operatorname{Im}\{Y_k\} c'_k - \operatorname{Re}\{Y_k\} s'_k], \qquad 0 \leqq k \leqq (N/2)-1 \tag{8b}$$

where $c'_k = \cos(\pi k/N)$ and $s'_k = \sin(\pi k/N)$.

Finally, the combine formulas which take two pairs of dual sequences into one pair of dual sequences must be determined. Let $y_n^{(1)}$ and $z_n^{(2)}$ be dual sequences of length $N/2$ with transforms $Y_k^{(1)}$ and $Z_k^{(2)}$. Let $y_n^{(2)}$ and $z_n^{(1)}$ be dual sequences of length $N/2$ with transforms $Y_k^{(2)}$ and $Z_k^{(1)}$. Using the notation introduced earlier, $y_n^{(1)}$, $y_n^{(2)}$, $z_n^{(1)}$ and $z_n^{(2)}$ arise from the splitting of the dual sequences y_n and z_n which have length

N and transforms Y_k and Z_k. Recall from splitting formula (7) that only the transform Y_k needs to be computed and stored. Applying the general splitting formula, we have

$$Y_k = Y_k^{(1)} + w_N^{-k} Z_k^{(1)}, \tag{9a}$$

$$Y_{k+(N/2)} = Y_k^{(1)} - w_N^{-k} Z_k^{(1)}, \tag{9b}$$

$$0 \leqq k \leqq (N/2)-1.$$

Once again we see that only one pair of transforms $(Y_k^{(1)}, Z_k^{(1)})$ needs to be available to compute the transform Y_k. In fact, the four storage locations used for $Y_k^{(1)}$, $Y_k^{(2)}$, $Z_k^{(1)}$ and $Z_k^{(2)}$ may be filled by Re and Im of $Y_k^{(1)}$ and $Z_k^{(1)}$. Thus, this combine step may also be done in-place and in real arithmetic. In terms of real quantities, expressions (9) appear as

$$\mathrm{Re}\{Y_k\} = \mathrm{Re}\{Y_k^{(1)}\} + \mathrm{Re}\{Z_k^{(1)}\}c_k + \mathrm{Im}\{Z_k^{(1)}\}s_k, \tag{10a}$$

$$\mathrm{Im}\{Y_k\} = \mathrm{Im}\{Y_k^{(1)}\} + \mathrm{Im}\{Z_k^{(1)}\}c_k - \mathrm{Re}\{Z_k^{(1)}\}s_k, \tag{10b}$$

$$\mathrm{Re}\{Y_{k+(N/2)}\} = \mathrm{Re}\{Y_k^{(1)}\} - \mathrm{Re}\{Z_k^{(1)}\}c_k - \mathrm{Im}\{Z_k^{(1)}\}s_k, \tag{10c}$$

$$\mathrm{Im}\{Y_{k+(N/2)}\} = \mathrm{Im}\{Y_k^{(1)}\} - \mathrm{Im}\{Z_k^{(1)}\}c_k + \mathrm{Re}\{Z_k^{(1)}\}s_k, \tag{10d}$$

for $0 \leqq k \leqq (N/2)-1$, where $c_k = \cos(2\pi k/N)$ and $s_k = \sin(2\pi k/N)$.

3. An FFT for real sequences. In summary, the three sets of splitting formulas (6), (8) and (10) when applied in accordance with the symmetry diagram of Fig. 1 give an in-place FFT algorithm which takes a bit-reversed CS sequence and produces a real ordered transform. Our primary interest is really the inverse of this process, i.e., given a real sequence in natural order, produce a CS transform in scrambled order. To obtain this new real FFT, the splitting formulas derived above must be formally inverted, as must the structure of the algorithm.

The inversion of relations (6) is done easily and gives the relations

$$Y_k = \tfrac{1}{2}(X_k + X_{k+(N/2)}), \qquad \tilde{Z}_k = \tfrac{1}{2}(X_k - X_{k+(N/2)}), \tag{11}$$

$$0 \leqq k \leqq (N/2)-1.$$

The inversion of (8) gives the relations

$$\mathrm{Re}\{w_N^{-k/2} Y_k\} = \tfrac{1}{2}(\tilde{X}_k c'_k - \tilde{X}_{k+(N/2)} s'_k), \qquad \mathrm{Im}\{w_N^{-k/2} Y_k\} = \tfrac{1}{2}(\tilde{X}_k s'_k + \tilde{X}_{k+(N/2)} c'_k), \tag{12}$$

$$0 \leqq k \leqq (N/2)-1$$

where $c'_k = \cos(\pi k/N)$ and $s'_k = \sin(\pi k/N)$.

Finally, the inversion of (10) eventually results in the expressions

$$\mathrm{Re}\{Y_k^{(1)}\} = \tfrac{1}{2}[\mathrm{Re}\{Y_k\} + \mathrm{Re}\{Y_{k+(N/2)}\}], \tag{13a}$$

$$\mathrm{Re}\{Z_k^{(1)}\} = \tfrac{1}{2}[(\mathrm{Re}\{Y_k\} - \mathrm{Re}\{Y_{k+(N/2)}\})c_k - (\mathrm{Im}\{Y_k\} - \mathrm{Im}\{Y_{k+(N/2)}\})s_k], \tag{13b}$$

$$\mathrm{Im}\{Y_k^{(1)}\} = \tfrac{1}{2}[\mathrm{Im}\{Y_k\} + \mathrm{Im}\{Y_{k+(N/2)}\}], \tag{13c}$$

$$\mathrm{Im}\{Z_k^{(1)}\} = \tfrac{1}{2}[(\mathrm{Re}\{Y_k\} - \mathrm{Re}\{Y_{k+(N/2)}\})s_k + (\mathrm{Im}\{Y_k\} - \mathrm{Im}\{Y_{k+(N/2)}\})c_k] \tag{13d}$$

for $0 \leqq k \leqq (N/2)-1$.

In all of the expressions (11), (12), and (13), the division by 2 may be omitted and the appropriate scaling may be done at the end of the computation. As with all FFT algorithms, efficiency is improved by computing c_k and s_k once in each stage and using them immediately for all combine steps that require them.

The operation count for these algorithms is rather involved. However, it may be shown that both the forward algorithm (CS → Real) and its inverse require $\frac{3}{2}Nm - \frac{5}{2}N + 2m + 2$ additions and $Nm - 3N + 4m$ multiplications, where $m = \log_2 N$. These counts are virtually identical to those for the original Edson's algorithm.

4. Other symmetric FFTs. Swarztrauber's paper [6] presents the symmetries in the FFT which are induced by odd, even, quarter wave even and quarter wave odd sequences. These symmetries are the basis for compact sine, cosine and quarter wave transforms. Since these algorithms are derived from the splitting approach, they also assume an input sequence in scrambled order. It remains to find analogous algorithms which take the input sequence in natural order.

This question is easily settled for the sine and cosine transforms. Recall that the discrete sine and cosine transforms are inverses of themselves (up to a multiplicative constant). Therefore, algorithms which take the input sequence in natural order may be found by formally inverting the corresponding algorithms which take the input in scrambled order.

We now consider the problem of finding in-place algorithms for the quarter wave transforms. A procedure analogous to that used in §§ 2 and 3 will be used to develop compact versions of the quarter wave even and quarter wave odd transforms.

A CS sequence x_n of length N which also satisfies the property $x_n = w_N^{-n}\bar{x}_n$ will be called *w-even* (wE). It is not difficult to show that the transform X_k of a wE sequence is real and *quarter wave even* (QE), i.e., $X_{N-k-1} = X_k$. The strategy is now to identify the symmetries and splitting formulas for a wE sequence. The formal inverse of this process will be the desired transform for a real QE sequence.

A wE sequence of length N may be split into two sequences $y_n = x_{2n}$ and $z_n = x_{2n+1}$ both of length $N/2$. The sequence y_n is also wE, while the sequence z_n satisfies

$$z_{(N/2)-n-1} = \bar{z}_n \quad \text{and} \quad z_n = w_N^{2n+1} z_n \quad \text{for } 0 \leqq n \leqq (N/2) - 1.$$

A sequence with these latter properties will be called *w-quarter wave even* (wQE).

In order to study the symmetries produced when a wQE sequence is produced, let x_n be a wQE sequence of length N and again let $y_n = x_{2n}$ and $z_n = x_{2n+1}$. Individually, y_n and z_n possess no particular symmetry. However, as with the splitting of a QCS sequence, y_n and z_n possess the intersequence symmetry $y_n = \bar{z}_{(N/2)-n-1}$ and will also be called *dual sequences.*

Finally, let y_n and z_n be dual sequences of length N. The splitting of y_n produces two sequences $y_n^{(1)}$ and $z_n^{(1)}$ of length $N/2$, while the splitting of z_n produces two sequences $y_n^{(2)}$ and $z_n^{(2)}$ of length $N/2$. These four sequences have the property that $y_n^{(1)}$ and $z_n^{(2)}$ are dual and $y_n^{(2)}$ and $z_n^{(1)}$ are dual. Subsequent splittings of dual pairs will continue to produce two dual pairs of half the length. This closes the set of symmetries induced by the splitting of a wE sequence.

It is now necessary to determine the properties of the transform of each of these symmetric sequences. As mentioned previously, a wE sequence has a transform which is real and quarter wave even (QE). Let X_k be the transform of a wQE sequence x_n of length N. Since x_n is also QCS by definition, it follows by (3) that

$$X_k = w_N^k \bar{X}_k \quad \text{and} \quad X_k = w_N^{k/2} \tilde{X}_k \quad \text{where } \tilde{X}_k \in \mathbf{R}. \tag{14}$$

In addition, by direct substitution, it can be shown that

$$X_{N-k-1} = w_N^{-k-1/2} X_k \quad \text{for } 0 \leqq k \leqq (N/2) - 1. \tag{15}$$

Combining (14) and (15) we obtain the important relationship

$$\tilde{X}_{N-k-1} = -\tilde{X}_k \quad \text{for } 0 \leqq k \leqq (N/2) - 1, \tag{16}$$

which will be used short to eliminate the computation and storage of half of the $\tilde{X}_k$ sequence.

Now consider the dual sequences y_n and z_n of length $N/2$ which are produced by the splitting of a wQE sequence. Denoting the transforms of y_n and z_n by Y_k and Z_k, we have

$$\begin{aligned} Y_k &= \sum_{n=0}^{(N/2)-1} y_n w_{N/2}^{-nk} = \sum_{n=0}^{(N/2)-1} \bar{z}_{(N/2)-n-1} w_{N/2}^{-nk} \\ &= \sum_{n=0}^{(N/2)-1} \bar{z}_n w_{N/2}^{-((N/2)-n-1)k} = w_{N/2}^{k} \bar{Z}_k \end{aligned} \tag{17}$$

where the relationship $y_n = \bar{z}_{(N/2)-n-1}$ has been used. This expression gives a useful relationship between the transforms of two dual sequences. One further symmetry will be needed which relates terms within the Y_k sequence.

Once again letting X_k be the transform of a wQE sequence of length N, we have by (14) and (2a) that

$$\tilde{X}_k = w_N^{-k/2} X_k = w_N^{-k/2}(Y_k + w_N^{-k} Z_k), \qquad 0 \leqq k \leqq (N/2)-1 \tag{18}$$

where Y_k and Z_k are the transforms of the associated dual sequences. By (16), (14) and (2b), we also have

$$\begin{aligned} \tilde{X}_k &= -\tilde{X}_{N-k-1} = -w_N^{-(N-k-1)/2} X_{N-k-1} \\ &= w_N^{(k/2)+1/2} X_{(N/2)+((N/2)-k-1)} \\ &= w_N^{(k/2)+1/2}[Y_{(N/2)-k-1} - w_N^{-((N/2)-k-1)} Z_{(N/2)-k-1}]. \end{aligned} \tag{19}$$

Matching (18) and (19) we obtain the relationships

$$Y_{(N/2)-k-1} = w_N^{-k-1/2} Y_k \tag{20a}$$

and

$$Z_{(N/2)-k-1} = w_N^{-3k-3/2} Z_k, \tag{20b}$$

the first of which will be valuable.

With the properties of the transforms of the various symmetric subsequences determined, we now turn to the actual combine formulas. First consider the combining of a wE and a wQE sequence. Let Y_k be the real QE transform of a wE sequence y_n and let Z_k be the transform of a wQE sequence z_n, all of length $N/2$. Note that the transform X_k of the associated wE sequence x_n is real and QE. Because of this symmetry ($X_k = X_{N-k-1}$), only half of the sequence X_k needs to be computed and stored. Using the combine formulas (2), we have

$$X_k = Y_k + w_N^{-k} Z_k = Y_k + \tilde{Z}_k, \qquad 0 \leqq k \leqq (N/4)-1 \tag{21a}$$

where $\tilde{Z}_k \in \mathbf{R}$ and use has been made of (14). To obtain components $N/4 \leqq k \leqq (N/2)-1$ of X_k, use the QE symmetry of X_k in the form $X_{(N/2)+k} = X_{N-((N/2)-k-1)-1} = X_{(N/2)-k-1}$ to write

$$X_{(N/2)-k-1} = Y_k - w_N^{-k} Z_k = Y_k - \tilde{Z}_k, \qquad 0 \leqq k \leqq (N/4)-1 \tag{21b}$$

where again $\tilde{Z}_k \in \mathbf{R}$ and (14) has been used. Since Y_k and $\tilde{Z}_k$ are real, the combine steps (21a, b) may be done in real arithmetic.

Now consider the combining of two dual sequences to form a wQE sequence. Note from (21a, b) that only half of the real sequence $\tilde{Z}_k$ needs to be computed. Let Y_k and Z_k be the transforms of two dual sequences of length $N/2$. The transform X_k of the associated wQE sequence is given by

$$\begin{aligned}\tilde{X}_k &= w_N^{-k/2} X_k = w_N^{-k/2}(Y_k + w_N^{-k} Z_k) \\ &= w_N^{-k/2} Y_k + w_N^{k/2}(w_N^{-2k} Z_k) \\ &= w_N^{-k/2} Y_k + w_N^{k/2} \bar{Y}_k \\ &= 2 \operatorname{Re}\{w_N^{-k/2} Y_k\}, \qquad 0 \leqq k \leqq (N/4) - 1\end{aligned} \tag{22a}$$

where the relationship (17) between the transforms of two dual sequences has been used. To obtain components $N/4 \leqq k \leqq (N/2) - 1$ of the real sequence $\tilde{X}_k$, note that by (16),

$$\tilde{X}_{k+(N/2)} = \tilde{X}_{N-((N/2)-k-1)-1} = -\tilde{X}_{(N/2)-k-1}.$$

However, by (14) and (2b),

$$\tilde{X}_{k+(N/2)} = w_N^{-(k+(N/2))/2} = -iw_N^{-k/2}(Y_k - w_N^{-k} Z_k).$$

Combining these two results, we obtain

$$\begin{aligned}\tilde{X}_{(N/2)-k-1} &= i(w_N^{-k/2} Y_k - w_N^{k/2}(w_N^{-2k} Z_k)) \\ &= i(w_N^{-k/2} Y_k - w_N^{k/2} \bar{Y}_k) \\ &= -2 \operatorname{Im}\{w_N^{-k/2} Y_k\}, \qquad 0 \leqq k \leqq (N/4) - 1\end{aligned} \tag{22b}$$

where again (17) has been used. Thus, the combine steps (22a, b) allow half of the sequence $\tilde{X}_k$ to be computed from the transform of just one dual sequence. A saving in storage is achieved by storing $\operatorname{Re}\{Y_k\}$ and $\operatorname{Im}\{Y_k\}$, $0 \leqq k \leqq (N/4) - 1$ rather than both complex sequences Y_k and Z_k.

It now remains to determine the combine formulas for a single dual sequence. Assume that in the dual pair y_n and z_n each of length N, y_n is split into the subsequences $y_n^{(1)}$ and $z_n^{(1)}$ which have transforms $Y_k^{(1)}$ and $Z_k^{(1)}$, respectively. Notice that only half of the resulting transform Y_k needs to be computed. Applying the general splitting formulas (2), we have

$$Y_k = Y_k^{(1)} + w_N^{-k} Z_k^{(1)}, \qquad 0 \leqq k \leqq (N/4) - 1. \tag{23a}$$

The remaining components, $(N/4) \leqq k \leqq (N/2) - 1$, of Y_k are obtained by

$$Y_{k+(N/2)} = Y_{N-((N/2)-k-1)-1} = w_{2N}^{-((N/2)-k-1)-1/2} Y_{(N/2)-k-1},$$

which uses property (20a). By the splitting formula (2b)

$$Y_{k+(N/2)} = Y_k^{(1)} - w_N^{-k} Z_k^{(1)}.$$

Combining these last two expressions, we obtain

$$Y_{(N/2)-k-1} = iw_N^{-(k/2)-1/4}(Y_k^{(1)} - w_N^{-k} Z_k^{(1)}), \qquad 0 \leqq k \leqq (N/4) - 1. \tag{23b}$$

The combine formulas (23a, b) may be further simplified in terms of real quantities. The important point to note is that to compute the real quantities $\operatorname{Re}\{Y_k\}$ and $\operatorname{Im}\{Y_k\}$ for $0 \leqq k \leqq (N/2) - 1$, the real quantities $\operatorname{Re}\{Y_k^{(1)}\}$, $\operatorname{Re}\{Z_k^{(1)}\}$, $\operatorname{Im}\{Y_k^{(1)}\}$ and $\operatorname{Im}\{Z_k^{(1)}\}$ for $0 \leqq k \leqq (N/4) - 1$ are needed. Thus, this combine step may be done in-place and in real arithmetic.

The expressions (21), (22) and (23) give the complete set of combine formulas which are needed to transform a wE sequence. The result of this process is a real QE sequence in natural order. Recall that the ultimate objective of this exercise is to find an algorithm which takes a naturally ordered QE sequence into its bit-reversed transform. This algorithm may be found by formally inverting the expressions (21), (22) and (23).

The final symmetry which needs to be examined is that exhibited by a *quarter wave odd* (QO) sequence. As before, we proceed in an inverse fashion and find an algorithm which produces a naturally ordered QO sequence. The formal inverse of this algorithm will be the desired transform.

We begin by defining a sequence x_n of length N to be *w-odd* (wO) if it is conjugate symmetric ($x_{N-n} = \bar{x}_n$) and also satisfies $x_n = -w_N^{-n}\bar{x}_n$. It then follows that the DFT X_k of x_n is real and *quarter wave odd* (QO), i.e., $X_k = -X_{N-k-1}$. As before, we examine the splitting of a wO sequence.

A wO sequence of length N may be split in the usual manner into a subsequence y_n which is also wO and a sequence z_n which satisfies

$$z_{(N/2)-n-1} = \bar{z}_n \quad \text{and} \quad z_n = -w_N^{2n+1} z_n \quad \text{for } 0 \leqq n \leqq (N/2)-1.$$

A sequence with these properties will be called *w-quarter wave odd* (wQO).

A further splitting of a wQO sequence x_n of length N produces a pair of *dual sequences* y_n and z_n which satisfies $y_n = \bar{z}_{(N/2)-n-1}$. As in the case of wQE sequences, subsequent splittings of a pair of dual sequences produce two pairs of dual sequences.

We now investigate the way in which these symmetric sequences transform. As noted earlier, a wO sequence has a sequence which is real and QO. Following arguments analogous to those used in the QE case, we can show that the transform x_k of a wQO sequence of length N satisfies

$$X_k = w_N^k \bar{X}_k \quad \text{and} \quad X_k = w_N^{k/2} \tilde{X}_k \quad \text{where } \tilde{X}_k \in \mathbf{R}. \tag{24}$$

In addition, we have

$$X_{N-k-1} = -w_N^{-k-1/2} X_k \quad \text{for } 0 \leqq k \leqq (N/2)-1. \tag{25}$$

Combining (24) and (25) we obtain the relationship

$$\tilde{X}_{N-k-1} = \tilde{X}_k \quad \text{for } 0 \leqq k \leqq (N/2)-1. \tag{26}$$

Again following arguments used in the previous case, it may be shown that the transforms Y_k and Z_k of two dual sequences of length $N/2$ satisfy

$$Y_k = w_{N/2}^k \bar{Z}_k. \tag{27}$$

In addition, the transform Y_k of a dual sequence has the important intrasequence symmetry

$$Y_{(N/2)-k-1} = -w_N^{-k-1/2} Y_k, \qquad 0 \leqq k \leqq (N/2)-1, \tag{28}$$

which may be derived in the same manner as (20).

It is now possible to determine the combine formulas for these splittings. A real QO transform Y_k and a wO transform, both of length $N/2$, may be combined to produce the associated QO transform X_k by

$$\begin{aligned} X_k &= Y_k + \tilde{Z}_k, \\ X_{(N/2)-k-1} &= Y_k - \tilde{Z}_k, \end{aligned} \qquad 0 \leqq k \leqq (N/4)-1 \tag{29}$$

where $\tilde{Z}_k = w_{N/2}^{-k/2} Z_k \in \mathbf{R}$. Notice that because of the QO symmetry of X_k, only half of the sequence needs to be computed.

The real sequence $\tilde{X}_k$ associated with the transform of a wQO sequence x_n of length N may be computed by the combine formulas

$$\begin{aligned} &\tilde{X}_k = 2\,\mathrm{Re}\,\{w_N^{-k/2} Y_k\}, \\ &\tilde{X}_{(N/2)-k-1} = 2\,\mathrm{Im}\,\{w_N^{-k/2} Y_k\}, \end{aligned} \qquad 0 \leqq k \leqq (N/4)-1 \tag{30}$$

where Y_k is the transform of one of the dual sequences which is formed when x_n is split. Again only half of the $\tilde{X}_k$ sequence needs to be computed and it requires only half of the complex sequence Y_k.

Finally, let Y_k be the transform of one of a pair of dual sequences y_n of length N. Assume that y_n is split into subsequences $y_n^{(1)}$ and $z_n^{(1)}$ with transforms $Y_k^{(1)}$ and $Z_k^{(1)}$, respectively. Then the combined formulas for Y_k, $0 \leqq k \leqq (N/2)-1$, are

$$\begin{aligned} &Y_k = Y_k^{(1)} + w_N^{-k} Z_k^{(1)}, \\ &Y_{(N/2)-k-1} = -iw_N^{-(k/2)-1/4}(Y_k^{(1)} - w_N^{-k} Z_k^{(1)}), \end{aligned} \qquad 0 \leqq k \leqq (N/4)-1. \tag{31}$$

These expressions may be evaluated in real terms and may be done in-place.

The combine formulas (29), (30) and (31) define the transform which takes a bit-reversed wO sequence into a naturally ordered QO sequence. By formally inverting these combine formulas, a real, in-place algorithm for a naturally ordered QO sequence results.

5. Conclusions. Using the fact that symmetries of the input sequence are preserved by certain in-place FFT algorithms, two compact versions of the FFT (one the inverse of the other) have been developed. The forward transform takes a real input sequence in natural order and produces the (conjugate symmetric) transform in scrambled order. This algorithm complements the original Edson's algorithm for real data. Like Edson's algorithm, it requires half the operations and storage of the complex FFTs. In applications, such as the solution of boundary value problems or the filtering of large data sets, the explicit scrambling (bit reversal) of data may be avoided by using this transform and its inverse. Although this algorithm has been presented for the case $N = 2^m$, the extension to other and mixed radix cases is immediate.

Compact FFTs for even and odd sequences (i.e., cosine and sine transforms) already exist and may be modified to take the input data in natural order. Finally, the analogous in-place algorithms which produce the transforms of naturally ordered quarter wave even and quarter wave odd sequences are given. This provides a complete set of compact FFTs which take real data with various symmetries and compute their transforms in-place and with real arithmetic.

Acknowledgment. I am grateful to Paul Swarztrauber for sharing his insights on symmetric FFTs.

REFERENCES

[1] G. D. BERGLAND, *A fast Fourier Transform for real-valued series*, Comm. ACM, 11 (1968), pp. 703-710.
[2] E. O. BRIGHAM, *The Fast Fourier Transform*, Prentice-Hall, Englewood Cliffs, NJ, 1974.
[3] J. W. COOLEY AND J. W. TUKEY, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp., 19 (1965), pp. 297-301.
[4] W. M. GENTLEMAN AND G. SANDE, *Fast Fourier Transforms for fun and profit*, Proc. AFIPS Joint Computer Conference, 29 (1966), pp. 563-578.
[5] P. N. SWARZTRAUBER, FFT *algorithms for vector computers*, Parallel Comput., 1 (1984), pp. 45-63.
[6] ———, *Symmetric* FFTs, Math. Comp., to appear.
[7] C. TEMPERTON, *Implementation of a self-sorting in-place prime factor* FFT *algorithm*, J. Comput. Phys., 58 (1985), pp. 283-299.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 4, July 1987
© 1987 Society for Industrial and Applied Mathematics
013

A CONTINUATION APPROACH FOR SOLVING LARGE-RESIDUAL NONLINEAR LEAST SQUARES PROBLEMS*

DOUGLAS E. SALANE†

Abstract. This paper is concerned with the solution of the nonlinear least squares problem. A continuation method is used to develop a new framework for the model trust region approach for solving nonlinear least squares problems. This framework gives a motivation for the direct selection of the trust region parameter. It also provides a natural safeguard for trust region methods and leads to a very robust algorithm.

A class of algorithms based on the continuation method will be presented. In addition, the implementation details for one member of the new class will be examined. The convergence and descent properties of this algorithm will also be discussed. Numerical evidence will be given showing that the new algorithms are competitive with existing model trust region algorithms. For large-residual problems or problems in which a good initial starting guess is not available, the performance of the new algorithm is very promising.

Key words. nonlinear least squares, large residual problems, continuation

AMS (MOS) subject classifications. 65, 49

1. Introduction. Assume that $f: R^n \to R^m$ is a continuously differentiable function and consider the nonlinear least squares problem of minimizing $\phi(x)$, where

$$\phi(x) = \tfrac{1}{2}\|f(x)\|^2. \tag{1.1}$$

Let us introduce the following notation. Let J denote the Jacobian of the residual function f. The gradient of ϕ is

$$\nabla\phi(x) = J^t(x)f(x),$$

and its Hessian is

$$\nabla^2\phi(x) = J^t(x)J(x) + \sum_{i=1}^{m} f_i(x)\nabla^2 f_i(x), \tag{1.2}$$

where f_i is the ith component of f and $\nabla^2 f_i$ is its Hessian. In a subsequent discussion, we shall consider the two terms in (1.2) separately; hence we define J^TJ as the first-order part of the Hessian, and

$$S(x) = \sum_{i=1}^{m} f_i(x)\nabla^2 f_i(x)$$

as the second-order part. If $\|f\| = 0$, the Hessian is simply given by the first-order part; this suggests that if $\|f\|$ is "small," the first-order part provides a good approximation to the Hessian.

Many iterative methods solve (1.1) by attempting to find an x^* such that $\nabla\phi(x^*) = 0$. The most powerful and rapidly convergent methods of this type are based on Newton's method and require a good approximation to the Hessian (1.2). However, exact second derivatives of f are not available in many practical problems. Unless $\|f(x^*)\|$ is small and the initial point is close to x^*, methods that attempt to approximate (1.2) using only first-order information tend to exhibit slow convergence or unreliability.

* Received by the editors November 13, 1984; accepted for publication (in revised form) April 29, 1986. This work was performed at Sandia National Laboratories and was supported by the U.S. Department of Energy under contract DE-AC04-76DP00789.

† Numerical Mathematics Division, Sandia National Laboratories, Albuquerque, New Mexico 87185.

This paper presents an approach intended to overcome the difficulties in solving large-residual problems by constructing a sequence of related small-residual problems that are solved approximately. A good initial estimate of the solution is available for each subproblem so that a reasonable approximation to (1.2) can be obtained using only first-order information. In addition, the technique to be described contains a mechanism for controlling the contribution of $S(x)$ to $\nabla^2\phi(x)$, and this makes the subproblems easier to solve.

The methods to be described are based on continuation techniques [3] and will be given in detail in § 3. Our approach was motivated by the work of de Villiers and Glasser [8], who used the continuation method to minimize an objective function of the form

$$\tfrac{1}{2}\|f(x)-\lambda f(b_0)\|^2, \tag{1.3}$$

where λ is the continuation parameter and b_0 is the initial approximation to x^*. Note that if $\lambda = 1$, b_0 is the minimum of (1.3), and if $\lambda = 0$, (1.3) reduces to (1.1).

For small values of λ, (1.3) can be difficult to minimize because there is no control on the contribution of $S(x)$ to $\nabla^2\phi$. In the methods presented here, frequent updating of the subproblems makes it unnecessary to follow the continuation path to $\lambda = 0$. Furthermore, the subproblem in the new algorithms combines a regularized objective function with maintenance of a trust region [16], [24].

Section 2 gives a brief overview of some of the algorithms currently used for difficult nonlinear least squares problems. In § 3, the new class of methods is derived, and a stable way of implementing these algorithms is mentioned. In § 4, the new algorithms are shown to be descent methods and aspects of convergence are examined. The method for choosing the trust region for each subproblem is discussed in § 5. Numerical experiments with a test code appear in § 6.

2. Methods for difficult nonlinear least squares problems. In order to motivate subsequent algorithmic developments, we first examine certain general unconstrained minimization techniques. (For further details, see, e.g., Dennis and Schnabel [7].) A typical iterative method for minimizing (1.1) as a general unconstrained function develops a sequence of quadratic functions that locally approximate ϕ. If x is the current estimate of the solution, assume that the quadratic function

$$q(x) = \phi(x) + \nabla\phi(x)^T p + \tfrac{1}{2}p^T B p \tag{2.1}$$

is a good approximation to $\phi(x+p)$ for $\|p\|$ sufficiently small. If B is symmetric and positive definite, the quadratic model function is minimized by $p = -B^{-1}\nabla\phi(x)$, and $\bar{x} = x + p$ is a new approximation to x^*.

If B in (2.1) is $\nabla^2\phi(x)$, the model is derived from the Taylor series expansion of ϕ, and the resulting p is the Newton direction. If we take B as the first-order part of the Hessian of ϕ, the associated p is called the Gauss-Newton search direction, and satisfies

$$J^T J p = -J^T f, \tag{2.2}$$

where J and f are evaluated at x.

Based on the discussion in the previous section, we see that the first-order part is a poor approximation of the Hessian when $S(x)$ is not "small" relative to $J^T J$, and hence the Gauss-Newton method will be ineffective in such cases. For example, when J has small singular values, the Gauss-Newton correction p defined by (2.2) may contain large inaccurate components (see, e.g., [25]). For further discussion of the limitations of the Gauss-Newton method, see the papers of Dennis [4] and Meyer [15].

We shall henceforth use the term "difficult" to mean a nonlinear least-squares problems in which the optimal residual is large, or for which a good initial estimate is not available.

Model trust region methods (see, e.g., Dennis and Schnabel [7]) can sometimes improve the reliability of the Gauss–Newton method on difficult problems. In a model trust region method, the quadratic model (2.1) is minimized subject to constraints on the size of p. The Levenberg–Marquardt method [13], [14] is the most widely used method of this type, and chooses p by minimizing (2.1) subject to the ellipsoidal constraint

$$\|D^{1/2}p\| \leqq \Delta,$$

where Δ is called the size of the trust region. The Levenberg–Marquardt correction is given by

$$p(\lambda) = [B + \lambda D]^{-1} J^t(x) f(x), \tag{2.3}$$

where λ depends on Δ and $B = J^t(x)J(x)$.

An effective Levenberg–Marquardt implementation requires a strategy for adjusting the size of the trust region. In most trust region codes [5], [9], [11] and [19], a measure of the linearity of f about the current iterate x determines the size of the trust region. Moré [16] suggests the measure

$$\rho(p) = \frac{\|f(x)\|^2 - \|f(x+p)\|^2}{\|f(x)\|^2 - \|f(x) + J(x)p\|^2},$$

where p is the correction computed at x. If $\rho(p)$ is close to unity, a larger value of Δ is used at the next iterate. If $\rho(p) < a < 1$ for some specified value a, a smaller value of Δ is used at the next iterate. If $\rho(p)$ is unacceptably small, Δ is reduced and a new correction p is computed.

Once Δ is chosen, the computation of a correction p that minimizes the quadratic (2.1) and satisfies the trust region constraint is a formidable part of any trust region algorithm. If the correction is given by (2.3), an iterative procedure is used to choose λ so that $\|D^{1/2}p(\lambda)\| \leqq \Delta$. The iterative procedure must converge quickly, since each evaluation of $p(\lambda)$ requires the solution of an overdetermined least squares problem. The reader is referred to the papers of Moré [16] and Sorensen [24] for further details.

Hanson and Krogh [11] (hereafter called "HK") have developed a trust region algorithm that minimizes (2.1) with $B = J^TJ$ subject to interval constraints on the components of p, which are adjusted based on progress and a measure of the linearity of f. The HK algorithm is not a descent method, and therefore ϕ may increase at some iterations. If necessary, the growth in ϕ is controlled by returning to the previous "best" point and reducing the trust region. Another version of the HK algorithm uses a tensor-quadratic model function [23] in order to accelerate convergence.

An alternative approach has been to use quasi-Newton methods to approximate $S(x)$ with only first-order information (see Nazareth [20] for a brief survey). For example, in Dennis, Gay and Welsch [5], a quasi-Newton approximation to $S(x)$ is used in the quadratic model function (2.1).

3. A new approach motivated by the continuation method. We first describe the application of a straightforward continuation method to minimize (1.1). Let b be an approximation to the minimizer of (1.1), and consider the regularized problem

$$\min \phi(x, \lambda) = \tfrac{1}{2}(\|f(x) - \lambda f(b)\|^2 + \lambda\alpha\|D^{1/2}(x-b)\|^2), \tag{3.1}$$

where λ is a continuation parameter ($0 \leqq \lambda \leqq 1$), α is a positive weight, and $D^{1/2}$ is a positive diagonal matrix of scaling factors. Note that b is the solution to (3.1) if $\lambda = 1$, and that (3.1) reduces to (1.1) if $\lambda = 0$.

Consider the use of a continuation method to solve (3.1). The gradient of the objective function in (3.1) with respect to x for a particular λ is given by

$$\nabla\phi(x, \lambda) = J^T(x)(f(x) - \lambda\tilde{f}) + \alpha\lambda D(x - b), \tag{3.2}$$

where $\tilde{f}$ denotes $f(b)$.

The solution of (3.1) for a fixed value of λ (say λ_l) will be denoted $x(\lambda_l)$, and satisfies the following system of nonlinear equations:

$$\nabla\phi(x, \lambda_l) = 0. \tag{3.3}$$

If $\nabla\phi$ is sufficiently smooth and its Jacobian is invertible, a Newton-like iteration scheme can be used to solve (3.3). In addition, the implicit function theorem (see, e.g., [21]) guarantees that (3.3) can be used to parameterize (with respect to λ) a path of solutions to (3.1).

The Jacobian of $\nabla\phi$ with respect to x is given by

$$\nabla_x\phi(x, \lambda_l) = F(x, \lambda_l) + \bar{S}(x, \lambda_l) \tag{3.4a}$$

where

$$F(x, \lambda_l) = J^T(x)J(x) + \lambda_l\alpha D \tag{3.4b}$$

and

$$\bar{S}(x, \lambda_l) = \sum_{i=1}^{m} (f_i(x) - \lambda_l\tilde{f}_i(x))\nabla^2 f_i(x).$$

By analogy with the Gauss–Newton method, $F(x, \lambda_l)$ (the first-order part) will be used to approximate $\nabla_x\phi(x, \lambda_l)$. The corrector iterations then define a sequence of iterates $\{x_j\}$ by

$$x_{j+1} = x_j - \mu p_j \tag{3.5a}$$

for a positive steplength (damping parameter) μ, where p_j satisfies

$$F(x_j, \lambda_l)p_j = \nabla\phi(x_j, \lambda_l). \tag{3.5b}$$

Given a new value of the continuation parameter, λ_{l+1}, an initial approximation to $x(\lambda_{l+1})$ can be computed by the predictor formula

$$x(\lambda_{l+1}) \approx x(\lambda_l) + (\lambda_{l+1} - \lambda_l)\dot{x}(\lambda_l). \tag{3.6a}$$

In order to estimate $\dot{x}(\lambda_l)$, we differentiate the identity $\nabla\phi(x(\lambda), \lambda) \equiv 0$ with respect to λ, which gives

$$\nabla_x\phi(x, \lambda)\dot{x}(\lambda) = -\nabla_\lambda\phi(x, \lambda). \tag{3.6b}$$

Since

$$\nabla_\lambda\phi(x, \lambda) = -J^T(x)\tilde{f} + \alpha D(x - b), \tag{3.7}$$

we approximate $\dot{x}(\lambda_l)$ by solving (3.6b) with $F(x(\lambda), \lambda)$ as an approximation to $\nabla_x\phi(x, \lambda)$. For convenience of notation, in the remainder of this section we will simply use $\dot{x}(\lambda_l)$ to denote the approximation to the solution of (3.6b).

Equations (3.5) and (3.6) define the classical continuation procedure applied to (3.1). The method of de Villiers and Glasser [8] is obtained by setting $\alpha = 0$ in (3.1) and following the continuation path $x(\lambda)$ from $\lambda = 1$ to $\lambda = 0$, using (3.5) and (3.6a).

For problems in which $\|f\|$ is not small, a major drawback of a straightforward continuation method is that for small values of λ, $F(x)$ is not an accurate approximation to $\nabla_x \phi(x, \lambda)$. The result is that $|\lambda_{l+1} - \lambda_l|$ must be small to ensure convergence of the corrector iterations. This behavior is displayed in [8, Ex. 2].

The new continuation technique differs from that of [8] as follows. The continuation path for a given set of values b, α and D is followed for a decreasing sequence of continuation parameters $\{\lambda_1, \cdots, \lambda_K\}$, where $\lambda_K = \lambda_{\min}$, and $\lambda_{\min}$ is large enough so that only a few corrector iterations are required to obtain a good approximate solution to

$$\nabla \phi(x, \lambda_{\min}) = 0. \tag{3.8}$$

Given $\bar{x}$, a "good enough" solution to (3.8), a new regularized objective function (3.1), i.e., a new subproblem, is created by setting $b = \bar{x}$ and by choosing new values for α and D. The continuation parameter λ is then reset to unity and the procedure is repeated. This process generates a sequence of subproblems and, eventually, the subproblem solutions approximate the minimizer of (1.1).

Since $\nabla \phi(b, 1) = 0$, a good initial approximation to the solution of (3.1) is available for values of λ close to unity. In addition, for such λ values, the residuals in (3.1) are small, and hence $F(x, y)$ is a good approximation to $\nabla_x \phi(x, \lambda)$; therefore, the corrector iterations should converge quickly. If a correction procedure fails by requiring too many iterations or failing to reduce the objective function in (3.1), $\bar{\lambda}$ is replaced by a value closer to λ, and the value of $\lambda_{\min}$ is adjusted for the next subproblem. (The procedures for adjusting $\lambda_{\min}$ and choosing α are discussed in § 5.)

Frequent updates of the subproblem (3.1) allow us to avoid large-residual subproblems. In contrast to the HK and quasi-Newton alternatives to the Gauss-Newton method, the methods just described are not designed to attain a superlinear rate of convergence, since no effort is made to approximate $S(x)$. Rather, the new algorithms simply attempt to determine when the neglected terms become significant, and then pose a new subproblem. The performance of the corrector part of the continuation procedure indicates the significance of the neglected terms, and hence the validity of the predictor step. The corrector iterates reduce the original objective function (1.1). Furthermore, as we shall see, λ can be chosen without the need for solving an inverse problem, as in trust region methods.

In order to compute $\dot{x}(\lambda)$ in a numerically stable manner, note that $\dot{x}(\lambda)$ is the solution of the least squares problem

$$\begin{bmatrix} J(x) \\ \Gamma \end{bmatrix} \dot{x}(\lambda) = \begin{bmatrix} -\tilde{f} \\ (\alpha D)^{1/2} \lambda^{-1/2} (x - b) \end{bmatrix} \tag{3.9}$$

where $\Gamma = (\alpha \lambda D)^{1/2}$. Also, p_j in (3.5b) is the least squares solution to the problem

$$\begin{bmatrix} J(x) \\ \Gamma \end{bmatrix} p_j = -\begin{bmatrix} f(x) - \lambda \tilde{f} \\ \Gamma(x - b) \end{bmatrix}. \tag{3.10}$$

Problems (3.9) and (3.10) can be solved using orthogonal transformations [1]. Householder transformations are used to construct a QR factorization of $J(x)$, and (3.9) or (3.10) can be reduced to an equivalent least squares problem with a coefficient matrix of the form $[R^t \Gamma]^t$. In order to minimize storage and complete the solution in a stable fashion, a sequence of Givens' rotations can be used to eliminate Γ and reduce the problem to an equivalent least squares problem whose coefficient matrix is $[\bar{R}^t 0]^t$, where $\bar{R}$ is upper triangular.

The procedure just described is used in the MINPACK routines QRFAC and QRSOLV (see Moré, Garbow and Hillstrom [18] and Moré [16]). These routines handle systems of the form (3.9) where the last m components of the right-hand size are zero. With minor modifications QRFAC and QRSOLV can be used to solve (3.9). The modified forms of these routines are used in the test code examined in § 6.

4. Descent properties. Throughout the remainder of this work, we consider an implementation of the new approach that solves each subproblem using a single predictor iteration followed by a sequence of corrector iterations. The following is an outline of such an algorithm applied to solve a given subproblem. Let us assume that the predictor-corrector process has been performed on $k-1$ subproblems to yield b_k, the current approximation to the solution of (1.1).

ALGORITHM 4.1.

1. Given b_k, choose parameters α_k and $\lambda_{\min}$, such that $0 \leqq \lambda_{\min} < 1$, and choose a scaling matrix D_k. Define the kth subproblem by setting $b = b_k$, $\alpha = \alpha_k$, $D = D_k$ and $\lambda = \lambda_{\min}$ in (3.1).
2. Determine $\dot{x}(1)$ by solving

$$F(b_k, 1)\dot{x}(1) = -\nabla_\lambda \phi(b_k, 1),$$

where $F(x, \lambda)$ is defined in (3.4).

Set $$\Delta = 1 - \lambda_{\min}.$$

Compute $$x_0 = b_k - \Delta \dot{x}(1).$$

3. Use (3.5a) with x_0 as an initial guess and $\lambda_l = \lambda_{\min}$ to complete the solution of the kth subproblem.

We discuss how to choose α_k, $\lambda_{\min}$ and D_k in § 5. Note that it follows from (3.6) and (3.7) that the predictor step in Algorithm 4.1 is a damped Levenberg-Marquardt step. The algorithm can be viewed as a Levenberg-Marquardt method with corrector iterations used to determine the trust region and thus protect the algorithm. We now examine briefly the descent properties of Algorithm 4.1.

In an iteration of the form $\bar{x} = x - \mu p$, p is called a descent direction at x for the objective function ψ if there exists an $a > 0$, such that, for $0 < \mu \leqq a$, $\psi(\bar{x}) < \psi(x)$. Lemma 8.2.1 of Ortega and Rheinboldt [21] shows that if ψ is differentiable, then p is a descent direction if

$$p^t \nabla \psi(x) > 0. \tag{4.1}$$

We first show that $\dot{x}(1)$, the predictor step of Algorithm 4.1, is a descent direction for the nonlinear least squares objective function (1.1) and the regularized objective function (3.1). We then discuss the descent properties of the corrector steps used in Step 3 of Algorithm 4.1. Throughout this section we assume that the residual function f is twice continuously differentiable; hence, ϕ given by (1.1) is twice continuously differentiable. We also assume that $J^t(x)J(x)$ is positive definite. For notational convenience, we suppress the subscript k for the remainder of this section.

LEMMA 4.1. *Let $\phi(x)$ be the nonlinear least squares objective function* (1.1). *Let $\dot{x}(1)$ and x_0 be computed as in Step* 2 *of Algorithm* 4.1. *Then, if $\nabla \phi(b) \neq 0$, there exists a value $\lambda_{\min}$, where $0 \leqq \lambda_{\min} < 1$, such that*

$$\phi(x_0) < \phi(b). \tag{4.2}$$

Furthermore, inequality (4.2) *holds if x_0 is computed with any value of λ such that $\lambda_{\min} \leqq \lambda < 1$.*

Proof. Since $[J^t(b)J(b)+D]^{-1}$ is positive definite, it then follows from the definition of the gradient $\nabla\phi$ and $\dot{x}(1)$ that

$$\nabla\phi(b)^t\dot{x}(1)>0.$$

The fact that $\dot{x}(1)$ is a descent step for ϕ follows immediately from Lemma 8.2.1 of [21]. □

We next show that if λ is kept sufficiently close to unity, x_0 computed in Step 2 of the algorithm reduces the value of the regularized objective function $\phi_\lambda(x)$ in (3.1). In this section, we use the notation $\nabla\phi_\lambda(x)$ to denote the gradient of $\phi_\lambda(x)$ with respect to x and $\nabla^2\phi_\lambda(x)$ for the Hessian of $\phi_\lambda(x)$ with respect to x. The descent properties of both the predictor and corrector steps depend on the following lemma.

LEMMA 4.2. *Let ϕ_λ denote the regularized objective function in* (3.1). *Assume that the components of the residual function f are twice continuously differentiable on an open convex set $D_0\subset R^n$. In addition, assume that the component functions are defined and twice continuously differentiable on $\bar{D}_0$, where $\bar{D}_0$ is a compact set containing D_0. Let $x\in D_0$. If $p\in R^n$ and $\|p\|$ is sufficiently small, then the vector $\bar{x}$ defined as $\bar{x}=x-p$ is in D_0 and we have the following equality*:

$$\phi_\lambda(\bar{x})=\phi_\lambda(x)-\nabla\phi_\lambda^t(x)p+\tfrac{1}{2}p^t\nabla^2\phi_\lambda(x)p+\frac{\theta}{2}\|p\|^2\varepsilon_p, \tag{4.3}$$

where $-1\leqq\theta\leqq 1$, and ε_p is some function of p such that $\varepsilon_p\to 0$ as $p\to 0$.

Proof. From the mean value theorem (see [21, Lemma 3.3.10]), it follows that

$$\phi_\lambda(\bar{x})=\phi_\lambda(x)-\nabla\phi_\lambda^t(x)p+\tfrac{1}{2}p^t\nabla^2\phi_\lambda(x)p+\tfrac{1}{2}p^t(\nabla^2\phi_\lambda(v)-\nabla^2\phi_\lambda(x))p \tag{4.4}$$

where $v=x+t^*p$ and $0<t^*\leqq 1$. Applying standard inequalities for matrix and vector norms, we observe that

$$|p^t(\nabla^2\phi_\lambda(v)-\nabla^2\phi_\lambda(x))p|\leqq\|p\|^2\|\nabla^2\phi_\lambda(v)-\nabla^2\phi_\lambda(x)\|. \tag{4.5}$$

It follows from the definition of the regularized objective function that

$$\|\nabla^2\phi_\lambda(v)-\nabla^2\phi_\lambda(x)\|\leqq\|\nabla^2\phi(v)-\nabla^2\phi(x)\|+\sum_{i=1}^{n}\lambda|f_i(b)|\,\|\nabla^2 f_i(v)-\nabla^2 f_i(x)\|.$$

Since the f_i are twice continuously differentiable on the compact set $\bar{D}_0$, it follows that $\nabla^2\phi(x)$ and $\nabla^2 f_i(x)$ are uniformly continuous on D_0. Hence, given an $\varepsilon>0$, if $\|p\|$ is sufficiently small, we see that

$$\|\nabla^2\phi_\lambda(v)-\nabla^2\phi_\lambda(x)\|\leqq\varepsilon. \tag{4.6}$$

The existence of θ satisfying equality (4.3) follows from (4.4), (4.5) and (4.6). The existence of ε_p in the last statement in the lemma follows from (4.6) and the definition of v in (4.4). □

Let us now assume that $b\in D_0$, and that Δ in Step 2 of Algorithm 4.1 is small enough so that $x_0\in D_0$. If we assume the hypotheses of Lemma 4.2 and choose Δ sufficiently small, we observe that

$$\phi_\lambda(x_0)=\phi_\lambda(b)-\Delta\nabla\phi_\lambda^t(b)\dot{x}(1)+\frac{\Delta^2}{2}\dot{x}(1)^t\nabla^2\phi_\lambda(b)\dot{x}(1)+\theta_1\frac{\Delta^2}{2}\varepsilon_\lambda\|\dot{x}(1)\|^2, \tag{4.7}$$

where $-1\leqq\theta_1\leqq 1$. In addition, $\varepsilon_\lambda\to 0$ as $\lambda\to 1$.

We now note that

$$\nabla\phi_\lambda^t(b)\dot{x}(1)=\Delta f^t(b)J(b)\dot{x}(1), \tag{4.8}$$

$$\dot{x}(1)^t\nabla^2\phi_\lambda(b)\dot{x}(1)=\dot{x}(1)^t(J^t(b)J(b)+D)\dot{x}(1)+A\Delta, \tag{4.9}$$

where

$$A = \dot{x}(1)^t \left(\sum_{i=1}^{m} f_i(b) \nabla^2 f_i(b) - D \right) \dot{x}(1)$$

and

$$(J^t(b)J(b) + D)\dot{x}(1) = J^t(b)f(b).$$

Substituting expressions (4.8) and (4.9) into (4.7), we obtain the following equality:

$$\phi_\lambda(x_0) = \phi_\lambda(b) - \frac{(1-\lambda)^2}{2} f^t(b)J(b)\dot{x}(1) + \frac{(1-\lambda)^3}{2} A + \theta_1 \frac{(1-\lambda)^2}{2} \varepsilon_\lambda \|\dot{x}(1)\|^2, \tag{4.10}$$

where $-1 \leqq \theta_1 \leqq 1$. Moreover, $\varepsilon_\lambda \to 0$ as $\lambda \to 1$.

Equation (4.10) is used in the next lemma to show that the predictor step of Algorithm 4.1 decreases the regularized objective function if λ is sufficiently close to unity.

LEMMA 4.3. *Let ϕ_λ satisfy the hypotheses of Lemma 4.2, let $b \in D_0$, and let Δ, $\dot{x}(1)$ and x_0 be defined as in Step 2 of Algorithm 4.1. Assume that $\nabla\phi(b) \neq 0$. Then, there exists a value $\lambda_{\min}$, $0 \leqq \lambda_{\min} < 1$, such that for any $\lambda \in [\lambda_{\min}, 1]$, we have*

$$\phi_\lambda(x_0) < \phi_\lambda(b). \tag{4.11}$$

Proof. Let us first assume that λ is chosen sufficiently close to unity so that $x_0 \in D_0$ and also so that the conclusions of Lemma 4.2 hold. Since $J^tJ + D$ is positive definite, it follows that

$$f^t(b)J(b)\dot{x}(1) > 0.$$

Hence, if $\lambda_{\min}$ is chosen so that

$$f^t(b)J(b)\dot{x}(1) > |(1-\lambda)A + \theta_1 \varepsilon_\lambda \|\dot{x}(1)\|^2|,$$

then, in view of (4.10), such a choice of λ ensures that (4.11) holds. □

We will next examine descent properties of the corrector iteration (3.5). Note that if $\lambda > 0$ and $\nabla\phi_\lambda^t(x) \neq 0$, then

$$\nabla\phi_\lambda^t(x)[F(x, \lambda)]^{-1}\nabla\phi_\lambda(x) > 0.$$

Therefore, if μ is sufficiently small in iteration (3.5), Lemma 8.2.1 of [21] guarantees that corrector iterations reduce the value of the regularized objective function.

If Δ is sufficiently small, then $\|x_0 - x^*(\lambda_{\min})\| \leqq \delta$, where δ is small and $x^*(\lambda_{\min})$ is the solution to problem (3.1) with $\lambda = \lambda_{\min}$. By controlling the size of Δ, $F_x(x, \lambda_{\min})$ is forced to be an accurate approximation to the derivative (3.4a) (see (5.2)). Hence the iteration (3.5) behaves like a Newton method when used to solve problem (3.1). Theorem 6.4 of Dennis and Moré [6] shows that close to the solution such an iteration produces descent steps with $\mu = 1$. However, Theorem 6.4 applies only to methods where the correction vector approaches the Newton correction in magnitude and direction. It is possible to show that if λ is suitably restricted in Algorithm 4.1, the corrector iterations are descent steps for the regularized objective function with values of $\mu = 1$.

Let p be the correction term in iteration (3.5), that is,

$$p = [F(x, \lambda)]^{-1}\nabla\phi(x, \lambda). \tag{4.12}$$

The next lemma shows that if λ is sufficiently close to unity, then

$$\bar{x} = x - p \tag{4.13}$$

reduces the value of the regularized objective function.

LEMMA 4.4. *Assume that $\phi_\lambda(x)$ satisfies the hypotheses of Lemma* 4.2 *and that $\nabla\phi_\lambda(x) \neq 0$. Let $x^*(\lambda)$ denote the solution of problem* (3.1) *for a given value of λ and assume that $x^*(\lambda) \in D_0$ for any value of λ in some interval $(\lambda_0, 1]$. Let p and $\bar{x}$ be given by* (4.12) *and* (4.13), *respectively. If λ is sufficiently close to unity and x is sufficiently close to $x^*(\lambda)$, then*

$$\phi_\lambda(\bar{x}) < \phi_\lambda(x). \tag{4.14}$$

Proof. We first assume that $\|p\|$ is small enough so that the conclusions of Lemma 4.2 hold. In view of (4.12), such a choice of $\|p\|$ is possible if $\|x - x^*(\lambda)\|$ is sufficiently small.

We next note that

$$\nabla^2\phi_\lambda(x)p = \nabla\phi_\lambda(x) + R(\lambda)p, \tag{4.15}$$

where

$$R(\lambda) = \sum_{i=1}^{m} (f_i(x) - \lambda f_i(b))\nabla^2 f_i(x).$$

It follows from (4.3) and (4.15) that

$$\frac{\phi_\lambda(\bar{x}) - \phi_\lambda(x)}{p^t p} = -\frac{1}{2}\frac{\nabla\phi_\lambda^t(x)p}{p^t p} + \frac{1}{2}\frac{p^t R(\lambda)p}{p^t p} + \frac{\theta}{2}\varepsilon_\lambda, \tag{4.16}$$

where $-1 \leqq \theta \leqq 1$. In addition, $\varepsilon_\lambda \to 0$ as $\lambda \to 1$.

From the definition of p, we see that

$$\frac{\nabla\phi_\lambda^t(x)p}{p^t p} = \frac{\nabla\phi_\lambda^t(x)[F(x,\lambda)]^{-1}\nabla\phi_\lambda(x)}{\nabla\phi_\lambda^t(x)[F(x,\lambda)]^{-2}\nabla\phi_\lambda(x)}.$$

Since $J^t(x)J(x)$ is positive definite, it follows that $F(x, y)$ is positive definite, and it is not difficult to show that

$$\frac{\nabla\phi_\lambda^t(x)p}{p^t p} > K > 0 \tag{4.17}$$

for some constant K. Moreover, it follows from the continuity of $J(x)$ that (4.17) holds for a fixed value K and any pair (x, λ) in some neighborhood of the pair $(x^*, 1)$.

We next observe that

$$|f_i(x) - \lambda f_i(b)| \leqq |f_i(x) - f_i(b)| + (1-\lambda)|f_i(b)|.$$

Hence, we can ensure $\|R(\lambda)\|$ is small as long as (x, λ) is in a small enough neighborhood of $(x^*, 1)$. Therefore, (4.14) follows from (4.16) and (4.17) as long as $|1-\lambda|$ and $\|x^*(\lambda) - x\|$ are sufficiently small. □

By choosing λ close to unity in Step 2 of Algorithm 4.1, Lemma 4.3 shows that the predictor step reduces the value of the regularized objective function. Also, by taking λ close enough to unity, we keep $\|x_0 - x^*(\lambda)\|$ small. Lemma 4.4 shows that the corrector iterations reduce the value of $\phi_\lambda(x)$ if $\|x_0 - x^*\|$ is sufficiently small.

In view of (4.3) and the fact that the corrector iterations (3.5) are descent steps for the regularized objective function (3.1), we have that

$$\frac{1}{2}\|f(x_{j+1})-\lambda f(b)\|^2+\frac{\alpha\lambda}{2}\|D(x_{j+1}-b)\|^2<\frac{(1-\lambda)^2}{2}\|f(b)\|^2,$$

and hence,

$$\|f(x_{j+1})-\lambda f(b)\|<(1-\lambda)\|f(b)\|.$$

The last inequality implies that

$$\|f(x_{j+1})\|<\|f(b)\|. \tag{4.18}$$

Therefore, if λ is not too small, $\|f(x(\lambda))\|<\|f(b)\|$. In other words, the parameterized path for the subproblem associated with b is a descent path for the objective function ϕ in (1.1). Inequality (4.18) is the basis for the steplength procedure presented in the next section.

A detailed convergence analysis will not be given here. One way of proving convergence is to use the standard analysis for descent methods (see [21, Chap. 8]). Such an analysis is outlined in the report by Salane [22].

5. Parameter selection and steplength control. In Algorithm 4.1, the parameter α_k plays the role of model trust region parameter. In selecting α_k, two factors must be considered. First, $F(x, \lambda)$ should be an accurate approximation to the derivative (3.4). Second, if (1.1) has a small-residual, the damping terms in problem (3.1) should not dominate, especially when b_k is close to the solution.

Let $\Delta_\lambda = 1-\lambda$. For Δ_λ sufficiently small, it was shown in § 4 that

$$\|f(x_0)-\lambda f(b_k)\| \leqq \Delta_\lambda \|f(b_k)\|, \tag{5.1}$$

where x_0 is given by the predictor step in Algorithm 4.1, Step 2. From (5.1) it follows that

$$\left\| \sum_{i=1}^{m} (f_i(x_0)-\lambda f_i(b_k))\nabla^2 f_i(x_0) \right\| \leqq KM\Delta_\lambda \|f(b_k)\|, \tag{5.2}$$

where $M=\sup_i \|\nabla^2 f_i(x_0)\|$ and K is the equivalence-of-norm factor for the l_1 and l_2 (Euclidean) norms on R^n. The contribution of the second derivative term in (3.4) should not dominate; therefore, it is reasonable to choose α_k so that $\alpha_k\lambda D_k$ behaves somewhat like the second derivative terms. The choice

$$\alpha_k = \|f(b_k)\| \tag{5.3}$$

appears to give very good results in experiments with a test code presented in § 6.

Finer control of the high order terms in (3.4) is achieved by restricting λ so that $KM\Delta_\lambda$ is small. In practice M cannot be estimated, so the accuracy of $F(x_j, \lambda)$ is determined by the performance of the corrector step at x. In other words, if any corrector iteration fails to reduce ϕ, it should be assumed that $F(x, \lambda)$ is not an accurate approximation to the derivative (3.4). The value of Δ_λ should be reduced and the corrector cycle restarted.

The steplength control procedure for Δ_λ employed in the test code is straightforward. An initial value of λ for any corrector iteration is specified by the user. If at any predictor or corrector step in the kth subproblem, either

$$\|f(x_j)\| \geqq \|f(b_k)\|$$

or

$$\|\nabla\phi(x_j)\| \geqq \|\nabla\phi(b_k)\|, \tag{5.4}$$

the steplength Δ_λ is reduced by half and the predictor-corrector cycle for the kth subproblem is restarted. In the $(k+1)$st subproblem, Δ_λ is computed, at least initially, with the value of λ specified by the user. Note that the simple steplength strategy used here is not adaptive. An adaptive strategy for these methods would recall that the user specified value of λ was inappropriate for solving the kth subproblem, and in the $(k+1)$st subproblem it would reduce or increase λ accordingly.

The initial value of λ depends on the difficulty of the problem. In the numerical examples presented in § 6, λ is restricted to the values .25, .50, .75, and 1.0. For certain test examples in § 6, results are presented for several values of λ to illustrate the sensitivity of the algorithm to changes in λ.

In § 6, the test code was able to solve a variety of difficult examples with λ chosen from this limited set of values. Using bisection and the condition (5.4), we have developed an adaptive strategy for selecting λ. The adaptive method relieves the user of having to choose an initial value for λ, and it increases the rate of convergence close to the solution. In order to illustrate the efficacy of the proposed approach, the simple steplength strategy is used in the test code implementation.

In a small enough neighborhood of x^*, $F(x, \lambda)$ is an accurate approximation to the derivative (3.4), since $\|f(x)\|$ is close to $\|f(x^*)\|$. The descent path defined by (3.3) can be adequately followed using only predictor iterations. The test code in § 6 contains a parameter TOLC which will terminate corrector iterations as soon as the condition

$$\|\nabla\phi(x, \lambda)\| \leqq \text{TOLC} \tag{5.5}$$

is satisfied. If condition (5.5) is not satisfied in a subproblem, corrector iterations will be performed. Note that a good choice of TOLC will be highly dependent on the scaling of the problem.

In the regularized objective function (3.1), the diagonal matrix D_k scales the corrections in a neighborhood of b_k. In applications such as chemical kinetics parameter estimation, a direct scaling of the x variables is essential [2]. This means that the change of variables

$$x = \Lambda y \tag{5.6}$$

is made, where Λ is a diagonal matrix chosen so that the components of y are all of the same order of magnitude (usually unity). In the new class of algorithms, Λ_k can be determined for each subproblem by letting $b_k = x$ in (5.6) and determining Λ_k and y accordingly. The changes in the predictor and corrector formulas are straightforward. Note that the matrix D_k in (3.1) scales the corrections but not the variables. The direct scaling of the variables can be used in lieu of correction scaling or in conjunction with it.

6. Numerical experiments. The test code examined in this section is an implementation of Algorithm 4.1. The following is a more detailed description of the predictor-corrector iteration used to solve the kth subproblem in the test code.

Predictor-Corrector Iteration:

Given b_k, $\lambda_{\min}$, $f(b_k)$, $J(b_k)$, D_k and α_k

1. Compute $\dot{x}(1)$ (see Step 2 of Algorithm 4.1).
   Set $\lambda = \lambda_{\min}$.
2. Set $\Delta = 1 - \lambda$, compute $x_0 = b_k - \Delta_\lambda \dot{x}(1)$.
   Evaluate $f(x_0)$. If (5.4a), $\lambda = 1 - (\Delta_\lambda * .5)$, go to 2.
   Evaluate $J(x_0)$. If (5.4b), $\lambda = 1 - (\Delta_\lambda * .5)$, go to 2.
   If $(\|\nabla\phi(x_0, \lambda)\| < \text{TOLC}) b_{k+1} = x_0$, go to 4.

3. For $j = 0, 1, 2, \cdots$, Limc
   Compute x_{j+1} by (3.5) with $\lambda_l = \lambda$.
   Evaluate $f(x_{j+1})$. If (5.4a), $\lambda = 1 - (\Delta_\lambda * .5)$, go to 2.
   Evaluate $J(x_{j+1})$. If (5.4b), $\lambda = 1 - (\Delta_\lambda * .5)$, go to 2.
   If $(\|\nabla\phi(x_{j+1}, \lambda)\| < \text{TOLC}) b_{k+1} = x_j$, go to 4.
   END FOR
   Set FLAG to indicate TOLC not satisfied.
   $b_{k+1} = x_{j+1}$.
4. Set $\lambda = \lambda_{\min}$.
   Exit to solve $(k+1)$st problem.

In the test code implementation of this predictor-corrector iteration, a limit is placed on the number of times Δ_λ may be reduced in solving a given subproblem. If the condition

$$\Delta_\lambda < \Delta_{\lambda\,\min} * (.5)^I$$

is satisfied with $I = 5$, the test code will terminate. This reduction limit test (R) provides a natural stopping criterion for the new methods but is far too stringent for practical use. The R-test forces the algorithm to attempt to solve problems to machine accuracy, or until noise in the data makes further reduction of $\phi(x)$ and $\|\nabla\phi(x)\|$ impossible.

The test code has solved the five standard nonlinear least squares problems (see Moré, Garbow and Hillstrom [17]) shown in Table 6.1.

TABLE 6.1

	Problem	m	n
1	Brown-Dennis function (BD)	20	4
2	Jenrich and Sampson function (JS)	10	2
3	Meyer function (M)	16	3
4	Osborne function (O)	33	5
5	Powell singular function (P)	4	4

Problems 1-5 were chosen because they are nonzero residual problems that seem to require large numbers of function and gradient evaluations to solve (note the performance of NLSQ1 and NLSQ2 in [17]). Problem 5 was selected in order to see how the new approach performed on a problem with a singular Jacobian at the solution. In addition, the test code has solved two problems used by de Villiers and Glasser [8] to test their continuation algorithm. These problems (6 and 7) were included in the test set because they can be difficult to solve if a good initial approximation is not used.

Problem 6, DG1. Define

$$g(b, x) = x_1 x_2^b \sin(x_3 b + x_4).$$

Let

$$y_i = g(b_i, x^*)$$

where

$$b_i = .1(i-1) \quad \text{for } i = 1, 2, \cdots, 24$$

and

$$x^* = (60.137,\ 1.371,\ 3.112,\ 1.761).$$

(a) $m = 24$, $n = 4$,
(b) $f_i = y_i - g(b_i, x)$,
(c) starting point—(1, 8, 4, 4.412).

Problem 7, DG2. Define

$$g(b, x) = x_1 x_2^b \tanh(x_3 b + \sin(x_4 b)) \cos(b e^{x_5}).$$

Let

$$y_i = g(b_i, x^*)$$

where

$$b_i = .1(i-1) \quad \text{for } i = 1, 2, \cdots, 16$$

and

$$x^* = (53.81,\ 1.27,\ 3.012,\ 2.13,\ .507).$$

(a) $m = 16$, $n = 5$,
(b) $f_i = y_i - g(b_i, x)$,
(c) starting point—(2.5, 1.7, 1.0, 1.0).

Problems 8 and 9 are large-residual problems due to Hanson [10].

Problem 8, H1. Define

$$f_i = x_1 \sin(x_2 a_i) - b_i,$$

where a_i and b_i are given in Table 6.2.

(a) $m = 16$, $n = 2$,
(b) starting point = (10.0, .15).

Problem 9, H2. Define

$$f_i = x_1 \exp(x_2 a_i) \sin(x_3 a_i) - b_i.$$

(a) $m = 16$, $n = 3$,
(b) starting point = (25.0, −.1, .1).

Table 6.3 presents the results of using the MINPACK routine LMDER to solve these 9 test problems. MINPACK uses several stopping criteria: size of $\|J^t f\|$, GTOL; change in residual, FTOL; and change in the parameters, XTOL. For precise descriptions, the reader is referred to Moré, Garbow and Hillstrom [18]. The values GTOL = 10^{-4}, FTOL = 10^{-6} and XTOL = 10^{-6} were used for MINPACK runs on all problems. MINPACK solved Problems 1-6, 8 and 9. In Problem 7, it terminated at a point where the FTOL criterion was satisfied, but the residuals were not zero.

Table 6.4 presents the results of the test code for each of the 9 problems. No correction scaling was used in any test (i.e., $D = I$). The variable scaling strategy described at the end of § 5 was used in Problem 3. In many tests presented in Table

TABLE 6.2

i	a_i	b_i	i	a_i	b_i	i	a_i	b_i	i	a_i	b_i
1	0	0	5	4	5	9	10	13	13	25	3
2	1	1	6	5	6	10	12	12	17	30	2
3	2	2	7	6	8	11	15	9	15	40	1.5
4	3	3	8	8	11	12	20	6	16	50	1

TABLE 6.3
MINPACK *results.**

	Problem	NFE	NJE	$\|\nabla\phi\|$	$\|\phi\|$	SC
1	BD	214	198	$4 * 10^{1}$	$2.92956 * 10^{2}$	FTOL
2	JS	20	11	$3 * 10^{-1}$	$1.1152 * 10^{1}$	FTOL
3	M	126	115	$4 * 10^{0}$	9.37795	XTOL
4	O	18	15	$3 * 10^{-5}$	$7.392 * 10^{-3}$	GTOL
5	P	19	18	$5 * 10^{-8}$	$7.385 * 10^{-10}$	GTOL
6	DG_1	19	15	$3 * 10^{-9}$	$8.4509 * 10^{-12}$	XTOL
7	DG_2	37	22	$2 * 10^{-3}$	2.37047	FTOL
8	H_1	12	10	$3 * 10^{1}$	$1.444651 * 10^{1}$	FTOL
9	H_2	15	12	$4 * 10^{-1}$	6.29796	FTOL

* MINPACK internal scaling was used in all problems.
NFE - number of f evaluations;
NJE - number of J evaluations;
$\|\phi\|$, $\|\nabla\phi\|$ - residual norm and gradient norm when the algorithm terminated;
SC - stopping criteria used.

TABLE 6.4
Test code results.

	Problem	NFE	NJE	Δ_λ	TOLC	$\|\phi\|$	$\|\nabla\phi\|$	KSUB
1	BD	46	46	.25	10^{2}	$2.92954 * 10^{2}$	$.8 * 10^{1}$	33
2	JS	12	9	1.0	10^{-3}	$.111518 * 10^{2}$	$.1 * 10^{-1}$	7
3	M	31	31	.75	10^{-5}	9.3779452	$.3 * 10^{3}$	14
4	O	11	11	1.0	10^{-5}	$.739249 * 10^{-2}$	$.1 * 10^{-7}$	11
5	P	22	22	1.0	10^{-5}	$.26507 * 10^{-9}$	$.9 * 10^{-14}$	17
6	DG_1	48	46	.75	10^{-5}	$.2438 * 10^{-5}$	$.3 * 10^{-4}$	25
7	DG_2	20	16	1.0	10^{-5}	$.7034 * 10^{-10}$	$.1 * 10^{-11}$	7
8	H_1	14	14	1.0	10^{-5}	$.144465 * 10^{2}$	$.4 * 10^{1}$	6
9	H_2	15	15	1.0	10^{-5}	$.629796 * 10^{2}$	$.7 * 10^{-1}$	6

6.4, the test code terminated at a more accurate solution than the MINPACK solution (note the corresponding values for $\|\nabla\phi\|$ in Tables 6.3 and 6.4). The test code was terminated when the values of ϕ and $\|\nabla\phi\|$ were equivalent to those produced by MINPACK with the stopping criteria given in Table 6.3. This was done to try to compare the amount of work necessary for each method to produce a solution of the same accuracy. The same type of stopping criterion was used in Problem 6 in order to compare the test code with the de Villiers and Glasser continuation method.

A comparison of the number of function and gradient evaluations required by LMDER and the test code to solve each problem shows that the test code performs at least as well as LMDER on this test set. On the more difficult large-residual problems, the performance of the test code is significantly better.

Problems 1, 2 and 3 were run with different values of $\Delta\lambda$ to demonstrate the test code's sensitivity to this parameter (see Table 6.5). Note that on the difficult Problems 1 and 3, for any values of the parameter $\Delta\lambda$, the test code required significantly fewer function and Jacobian evaluations to produce a solution. It should again be pointed out that, in tests 1a and 3a, the test code obtained solutions that are more accurate than those produced by the MINPACK routine (compare the corresponding gradient values in Tables 6.3 and 6.5).

TABLE 6.5
Additional test code results.

Problem	NFE	NJE	Δ_λ	TOLC	$\|\phi\|$	$\|\nabla\phi\|$	KSUB	SC
1a BD	80	76	.25	10^2	$2.92954 * 10^2$	$.8 * 10^{-3}$	63	R
1b BD	137	89	.50	10^2	$2.92954 * 10^2$	$.3 * 10^{-3}$	58	R
1c BD	170	94	.75	10^2	$2.92954 * 10^2$	$.6 * 10^{-4}$	55	R
2a JS	22	21	.75	10^{-5}	$.111518 * 10^2$	$.1 * 10^{-1}$	11	M
3a M	58	36	1.0	10^{-5}	9.3779451	$.2 * 10^{-3}$	17	R
3b M	47	41	.75	10^{-5}	9.3779451	$.3 * 10^{1}$	19	R
3c M	81	72	.50	10^{-5}	9.3779451	$.7 * 10^{1}$	35	R

NFE - number of f evaluations;
NJE - number of J evaluations;
$\Delta_\lambda = 1 - \lambda_{\min}$, the user-selected parameter;
$\|\phi\|$, $\|\nabla\phi\|$ - residual norm and gradient norm when the algorithm terminated;
KSUB - number of subproblems required by the test code;
SC - stopping criteria used for the run;
R - reduction criteria;
DG - de Villiers and Glasser equivalent criteria;
M - MINPACK equivalent criteria.

Table 6.6 presents the results of the DG continuation algorithm reported in [8]. The performance of this continuation method has been improved by Haübler [12] through the use of an adaptive step length strategy. Note that the test code solves Problems 6 and 7 using far fewer function and Jacobian evaluations than the DG continuation method.

In Step 4 of the predictor-corrector iteration description, at the start of the $(k+1)$st subproblem, $\lambda_{\min}$ is reset to the initial value supplied by the user. If the user supplies too large a value for $\lambda_{\min}$, the algorithm will not be efficient. For example, this is shown in Table 6.6 in runs 1b, 1c, 3a and 3c where the number of function evaluations greatly exceeds the number of Jacobian evaluations required to solve each problem. This indicates that an adaptive strategy could even further improve the performance of the test code.

TABLE 6.6
DG *method.*

Problem	NFE	NJE	$\|\nabla\phi\|$
DG_1	300	93	less than 10^{-6}
DG_2	260	773	less than 10^{-6}

NFE - number of f evaluations;
NJE - number of J evaluations;
$\|\phi\|$, $\|\nabla\phi\|$ - residual norm and gradient norm when the algorithm terminated.

7. Concluding remarks. The algorithms presented in this paper provide an alternative method for solving large-residual nonlinear least squares problems and problems for which a good initial approximation is not available. The algorithms are based on applying several steps of a continuation method to a regularized small-residual problem. By solving a sequence of such problems, the algorithms approximate the solution to the original nonlinear least squares problem. The performance of the corrector part of the continuation procedure is used to control the contribution of the second-order

derivative terms in the Hessian of the nonlinear least squares objective function. The resulting algorithms appear to be very stable descent algorithms which are less dependent on ad hoc strategies for control than many methods currently in use. Numerical evidence on some difficult test problems indicates that the new class of methods is very competitive with a well developed implementation of the Levenberg–Marquardt method.

The approach presented here provides a framework for improving the capability of almost any nonlinear least squares algorithm.

Acknowledgment. The author is grateful to the editor for a number of thoughtful comments that have improved the exposition of this material.

REFERENCES

[1] P. BUSINGER AND G. H. GOLUB, *Linear least-squares solution by Householder transformations*, Numer. Math., 7 (1965), pp. 269–276.

[2] G. D. BYRNE, A. J. DEGREGORIA AND D. E. SALANE, *A program for fitting rate constants in gas phase chemical kinetics models*, this Journal, 5 (1984), pp. 642–657.

[3] D. DAVIDENKO, *On a new method of numerically integrating a system of nonlinear equations*, Dokl. Akad. Nauk, SSSR, 88 (1953), pp. 601–604.

[4] J. E. DENNIS, *Nonlinear least squares and equations*, in The State of the Art in Numerical Analysis, D. A. H. Jacobs, ed., Academic Press, London, 1977, pp. 229–268.

[5] J. E. DENNIS, D. M. GAY AND R. E. WELSCH, *An adaptive nonlinear least-squares algorithm*, ACM Trans. Math. Software, 17 (1981), pp. 348–368.

[6] J. E. DENNIS AND J. J. MORÉ, *Quasi-Newton methods motivation and theory*, SIAM Rev., 19 (1977), pp. 46–89.

[7] J. E. DENNIS AND R. B. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[8] N. DE VILLIERS AND D. GLASSER, *A continuation method for nonlinear regression*, SIAM J. Numer. Anal., 18 (1981), pp. 1139–1154.

[9] D. M. GAY, *Subroutines for unconstrained optimization using a model trust region approach*, ACM Trans. Math. Software, 9, pp. 503–525.

[10] R. J. HANSON (June 1984), personal communication.

[11] R. J. HANSON AND F. T. KROGH, *A new algorithm for constrained nonlinear least squares problems*, Sandia National Laboratories, Rept. SAND83-0936, Albuquerque, NM, 1983.

[12] W. M. HAÜBLER, *Variable schrittweitensteuerugen für homotopiemethode bei adaquaten nichtlinearean Ausgleichsproblemes*, Computing, 29 (1982), pp. 309–326.

[13] K. LEVENBERG, *A method for the solution of certain problems in least squares*, Quart. Appl. Math., 2 (1944), pp. 164–168.

[14] D. W. MARQUARDT, *An algorithm for least-squares-estimation of nonlinear parameters*, SIAM J. Appl. Math., 11 (1963), pp. 431–441.

[15] R. R. MEYER, *Theoretical and computational aspects of nonlinear regression*, in Nonlinear Programming, J. B. Rosen, O. L. Mangasarian and K. Ritter, eds., Academic Press, New York, 1970.

[16] J. J. MORÉ, *The Levenberg–Marquardt algorithm: implementation and theory*, in Lecture Notes in Mathematics, No. 630—Numerical Analysis, G. Watson, ed., Springer-Verlag, New York, 1978, pp. 105–116.

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

[18] ———, *User guide for* MINPACK-1, Argonne National Laboratories, Rept. ANL-80-74, Argonne, IL 60439, 1980.

[19] ———, *The* MINPACK *Project*, in Sources and Development of Mathematical Software, W. R. Cowell, ed., Prentice-Hall, Englewood Cliffs, NJ, 1984.

[20] L. NAZARETH, *Some recent approaches to solving large residual nonlinear least squares problems*, SIAM Rev., 22 (1980), pp. 1–11.

[21] J. M. ORTEGA AND W. C. RHEINBOLDT, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.

[22] D. E. SALANE, *A new approach for solving large residual nonlinear least squares problems*, Sandia National Laboratories, Rept. SAND84-1564, Albuquerque, NM, 1984.

[23] R. B. SCHNABEL AND P. D. FRANK, University of Colorado, Department of Computer Science, Rept. CU-CS-243-83, Boulder, CO, 1983.

[24] D. C. SORENSEN, *Newton's method with a model trust region modification*, SIAM J. Numer. Anal., 19 (1982), pp. 409-426.

[25] J. M. VARAH, *On the numerical solution of ill-conditioned linear systems with applications to ill-posed problems*, SIAM Rev., 10 (1973), pp. 257-267.

SIAM J. SCI. STAT. COMP.
Vol. 8, No. 2, September 1987

© 1987 Society for Industrial and Applied Mathematics

TWO MULTIGRID METHODS FOR THREE-DIMENSIONAL PROBLEMS WITH DISCONTINUOUS AND ANISOTROPIC COEFFICIENTS*

J. E. DENDY, JR.†

Abstract. The subject of this paper is two multigrid methods for the numerical solution of

$$-\nabla \cdot (D(x,y,z)\nabla U(x,y,z)) + \sigma(x,y,z)U(x,y,z) = F(x,y,z)$$

in a bounded region Ω of R^3, where $D = (D^1, D^2, D^3)$, D^i is positive, $i = 1,2,3$, and D^i, σ, and F are allowed to be discontinuous across internal boundaries of Ω. The first method differs in two important ways from previously considered multigrid methods for such problems; first, if standard coarsening is used, then anisotropic problems require plane relaxation to obtain a good smoothing factor; for problems with discontinuous coefficients, this was previously done with an ICCG method; here it is done with a two-dimensional multigrid method. Second, a slightly different form of interpolation is used which improves performance for nearly singular problems. Previously considered multigrid methods for problems with discontinuous coefficients have defined the coarse grid operators by Galerkin coarsening; in the second method, such coarsening is also used but with auxiliary intermediate grids obtained by semicoarsening successively in each of the three independent variables; this artifice simplifies coding considerably and results in comparable convergence.

Key words. multigrid method, diffusion equation, three-dimensional, discontinuous coefficients

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

1. Introduction.

The subject of this paper is a multigrid method for the numerical solution of

$$-\nabla \cdot (D(x,y,z)\nabla U(x,y,z)) + \sigma(x,y,z)U(x,y,z) = F(x,y,z), \quad (x,y,z) \in \Omega,$$

$$\nu(x,y,z) \cdot D(x,y,z)\nabla U(x,y,z) + \gamma(x,y,z)U(x,y,z) = 0, \quad (x,y,z) \in \partial\Omega,$$

where Ω is a bounded region in R^3 with boundary $\partial\Omega$, $D = (D^1, D^2, D^3)$, D^i is positive, i=1,2,3, σ and γ are nonnegative, and D^i, $i = 1,2,3$, σ, and F are allowed to be discontinuous across internal boundaries of Ω. This subject was previously considered in [ABDP],[D1] for two-dimensional problems and was extended to nonsymmetric problems and systems in [D2] and [D3], respectively.

The method in [D1] was extended to three dimensions in [BF] and [S]. For the standard multigrid coarsening (in which, for a given grid, the next coarser grid has 1/8 as many points), anisotropic problems require plane relaxation to obtain a good smoothing factor. In [BF] this plane relaxation was performed by ICCG (an incomplete Cholesky conjugate gradient method), whereas in [S] only alternating line relaxation was used. The other relevant reference is [T], in which plane relaxation is performed by a two-dimensional multigrid method; this paper considers only constant coefficient problems but is nevertheless an interesting and thorough study of the pathologies that can occur.

In this paper we consider two methods. The first method is basically the method considered in [BF] with two differences: first, we perform plane relaxation by a two-dimensional multigrid method, and second, we use a slightly different choice of

*Received by the editors April 14, 1986; accepted for publication (in revised form) January 26, 1987. This work was supported by the U.S. Department of Energy under contract W-7405-ENG-36 and by the Air Force Office of Scientific Research under contract W31RPD-63-3132.

† Theoretical Division, Los Alamos National Laboratory, Los Alamos, New Mexico 87545.

interpolation operator, which improves performance for nearly singular problems. In the second method coarsening is done by successively coarsening in each of the three independent variables and then ignoring the intermediate grids; this artifice simplifies coding considerably.

We describe the two methods in § 2. In § 3 we discuss some remaining details. In § 4 we present some numerical experiments, and in § 5 we present some conclusions.

2. The two methods. In the multigrid method one attempts to solve a discrete approximation

$$L^M U^M = F^M \tag{2.1}$$

to a continuous equation

$$LU = F.$$

To do this one constructs a sequence of grids $G^1, \ldots, G^M$ with corresponding mesh sizes $h_1 > \cdots > h_M$ (usually $h_k = 2h_{k+1}$). In its simplest mode of operation, one does a fixed number, IM, of relaxation sweeps on Eq. (2.1) and then drops down to grid G^{M-1} and the equation

$$L^{M-1}V^{M-1} = f^{M-1} = I_M^{M-1}(F^M - L^M v^M), \tag{2.2}$$

where V^{M-1} is to be the coarse grid approximation to $V^M \equiv U^M - u^M$, where $v^M = u^M$ is the last iterate on grid G^M,and where I_M^{M-1} is an interpolation operator from G^M to G^{M-1}. To solve Eq. (2.2) approximately, one resorts to recursion, taking ID relaxation sweeps on grid G^k before dropping down to grid G^{k-1}, $M-1 \geq k \geq 2$ and the equation

$$L^{k-1}V^{k-1} = f^{k-1} = I_k^{k-1}(f^k - L^k v^k).$$

When grid G^1 is reached, the equation $L^1V^1 = f^1$ can be solved directly and $v^2 \leftarrow v^2 + I_1^2 V^1$ performed. Then one does IU relaxation sweeps on grid G^{k-1} before forming $v^k \leftarrow v^k + I_{k-1}^k V^{k-1}$, $3 \leq k \leq M$. (This description assumes $M \geq 3$, the cases $M = 1$ or 2 being trivial.)

We discuss first the choice for I_{k-1}^k made in [BF] and [S], which is a generalization of the choice made in [ABDP], [D1]. We assume that G^{k-1} is obtained from G^k by standard coarsening; that is, if G^k is a tensor product grid $G_x^k \times G_y^k \times G_z^k$, $G^{k-1} = G_x^{k-1} \times G_y^{k-1} \times G_z^{k-1}$, where G_x^{k-1} is obtained by deleting every other grid point of G_x^k and similarly for G_y^k and G_z^k. We also assume that L^k uses at most nearest and next-nearest neighbors (so that the difference template is at most a twenty-seven point one). For fine points corresponding to coarse points I_{k-1}^k is just the identity. For a fine point p on a coarse grid line between two coarse points, define $(L^k)^{C_1}$ at p to be the difference template obtained by summing the difference template of L^k at p in the two directions perpendicular to the coarse grid line; $(L^k)^{C_1}$ is a one-dimensional operator with a three-point template; thus, the equation

$$(L^k)^{C_1} v^k = 0 \tag{2.3a}$$

gives I_{k-1}^k at p as a weighted average of data at the two coarse grid points.

For a fine point p not on a coarse grid line but in the same plane as two coarse grid lines, define $(L^k)^{C_2}$ at p to be the difference template obtained by summing the difference template of L^k at p in the direction perpendicular to the plane containing the coarse grid lines; thus, the equation

$$(L^k)^{C_2} v^k = 0 \tag{2.3b}$$

gives I_{k-1}^k at p as a weighted average of data at the eight nearest neighbors. Four of these neighbors are fine grid points which are related via $(L^k)^{C_1}$ to the four nearest neighbor coarse grid points; thus (2.3*b*) gives I_{k-1}^k at p as a weighted average of data at the four nearest neighbor grid points.

The remaining fine grid points have twenty-seven nearest neighbors and next-nearest neighbors, which by use of the equation

$$L^k v^k = 0 \tag{2.3c}$$

and $(L^k)^{C_1}$ and $(L^k)^{C_2}$ give I_{k-1}^k at such fine grid points as a weighted average of the eight nearest neighbor coarse grid points.

The coarse grid operators are defined by

$$L^{k-1} = (I_{k-1}^k)^T L^k I_{k-1}^k, \tag{2.4}$$

and the restriction operator I_k^{k-1} is $(I_{k-1}^k)^T$. Note that even if the fine grid difference operator L^M is a seven-point operator (using only nearest neighbors), the coarse grid difference operators L^k, $k < M$, are twenty-seven point operators.

Let A^k be the matrix associated with L^k and let a_{ij}^k be the entries of A^k, with $a_{ii}^k > 0$. In testing the above method of [BF] for isotropic problems, we noticed very poor performance if

$$0 < a_{ii}^k - \sum_{j \neq i} a_{ij}^k \ll a_{ii}^k; \tag{2.5}$$

this condition occurs if σ is nearly zero. Such poor performance was also observed in [BF]. We also observed mild degradation in two-dimensional problems in the method of [ABDP], [D1] in this case. One cure is to define $\tilde{A}^k$ to be the matrix with entries $\tilde{a}_{ij}^k$, where

$$\tilde{a}_{ij}^k = a_{ij}^k \quad \text{if } i \neq j;$$

$$\tilde{a}_{ii}^k = \begin{cases} \sum_{j \neq i} a_{ij}^k & \text{if } a_{ii}^k - \sum_{j \neq i} a_{ij}^k \ll a_{ii}^k, \\ a_{ii}^k & \text{otherwise.} \end{cases} \tag{2.6}$$

Let $\tilde{L}^k$ be the associated difference operator, and define I_{k-1}^k as above using $\tilde{L}^k$ instead of L^k. For reference we define

$$L^{k-1} = (I_{k-1}^k)^T L^k I_{k-1}^k, \ I_{k-1}^k \text{defined by (2.3) using } \tilde{L}^k \text{ instead of } L^k. \tag{2.7}$$

The computation of (2.4) or (2.7) is very involved. If L^k is defined on a grid with n points, it requires $116n$ [$309n$] floating point operations if L^k is a seven [twenty-seven] point operator. More importantly, if one takes advantage of the special structure of L^k so as to avoid needless multiplies by zeroes but without using pointers, then the coding is a major effort.

As an alternative, consider the following. Define an auxiliary grid $\tilde{G}_z^{k-1} = G_x^k \times G_y^k \times G_z^{k-1}$; that is, coarsen in z only. Define $(J_{k-1}^k)_z : \tilde{G}_z^{k-1} \to G^k$ as follows. For fine points corresponding to coarse points, $(J_{k-1}^k)_z$ is just the identity. For a fine point p on a plane between two coarse planes, define $(\tilde{L}^k)^{C_3}$ at p to be the difference template

obtained by summing the difference template of $\tilde{L}^k$ at p in xy planes; $(\tilde{L}^k)^{C_3}$ is a one-dimensional difference operator with a three-point template; thus the equation

$$(\tilde{L}^k)^{C_3} v^k = 0 \tag{2.8a}$$

gives $(J^k_{k-1})_z$ at p as a weighted average of the coarse grid point below p in the xy plane below and the coarse grid point above p in the xy plane above. Now define

$$(L^{k-1})_z = ((J^k_{k-1})_z)^T L^k (J^k_{k-1})_z. \tag{2.9a}$$

In a similar manner define $\tilde{G}^{k-1}_{yz} = G^k_x \times G^{k-1}_y \times G^{k-1}_z$,

$$(J^k_{k-1})_{yz} : \tilde{G}^{k-1}_{yz} \to \tilde{G}^{k-1}_z, \tag{2.8b}$$

and

$$(L^{k-1})_{yz} = ((J^k_{k-1})_{yz})^T (L^{k-1})_z (J^k_{k-1})_{yz}. \tag{2.9b}$$

Finally in a similar manner, define $G^{k-1} = G^{k-1}_x \times G^{k-1}_y \times G^{k-1}_z$,

$$J^k_{k-1} : G^{k-1} \to \tilde{G}^{k-1}_{yz}, \tag{2.8c}$$

and

$$L^{k-1} = (J^k_{k-1})^T (L^{k-1})_{yz} J^k_{k-1}. \tag{2.9c}$$

The natural choice for interpolation would be $(J^k_{k-1})_z (J^k_{k-1})_{yz} J^k_{k-1}$, but in numerical experiments, this choice did not perform as well as I^k_{k-1} as defined in (2.3) using $\tilde{L}^k$ instead of L^k. For reference we define

(2.10) L^{k-1} defined by (2.9), where J^k_{k-1} is defined by (2.8) and I^k_{k-1} is defined by (2.3) using $\tilde{L}^k$ instead of L^k.

Note that (2.9) does not require the enormous amount of storage that it appears to since it can be performed three xy planes at a time.

On a grid with n points, (2.9) requires $80n[153n]$ floating point operations if L^k is a seven [twenty-seven] point operator. The savings in coding is more dramatic with (2.4) requiring about four times as much FORTRAN as (2.9).

3. Further details. For isotropic problems, point Gauss-Seidel provides a good smoothing factor, and once the coarse grid operators have been generated, the algorithm is not much more complicated than in the two-dimensional case [D1]; as before, $IM = 2$, $IU = 1$, and $ID = 1$ are usual choices, and the same cycling options—V-cycles, W-cycles, multiple-V-cycles—are available, with V-cycles being the usual best choice. One can also bootstrap the calculation by starting on the coarsest grid with the problem $L^1 V^1 = I^1_2 \cdots I^{M-1}_M F^M$ and by bootstrapping to the finest grid in an F-cycle pattern; for three grids, for example, the pattern is

$$G^1 \to G^2 \to G^1 \to G^2 \to G^3.$$

For anisotropic problems, point Gauss-Seidel no longer provides a good smoothing factor. For constant coefficient problems, plane relaxation provides a good smoothing factor, and xy or yz or xz plane relaxation is sufficient. But for variable coefficient anisotropic problems, one must resort to alternating plane relaxation. Each plane problem need not be solved exactly; we have found that, similar to the situation in

[T], each plane problem need only be solved approximately by one cycle of a two-dimensional multigrid method. For variable coefficient anisotropic problems, the two-dimensional multigrid method will be required to solve a two-dimensional anisotropic variable coefficient problem, for which alternating line relaxation will be required. For problems with strongly discontinuous coefficients, the two-dimensional multigrid method will be handed two-dimensional problems with strongly discontinuous coefficients.

A two-dimensional multigrid method which can handle these demands is BOXMG [D1]. There are a few changes, however, that had to be made in BOXMG particular to this situation. First, as commented in § 2, we had previously noticed mild degradation in convergence in BOXMG in the case that (2.5) holds. However, we observed that many of the problems which BOXMG must solve for plane relaxation for three-dimensional problems exhibited severe degradation in the case that (2.5) holds and in particular for anisotropic problems. Hence, we changed BOXMG to use (2.6) in its definition of I_{k-1}^k.

Second, BOXMG must be called repeatedly for successive xy, yz, and xz planes, for each $k = 2, \ldots, M$. For computational efficiency it makes sense to call BOXMG initially for each such plane and to compute and store the auxiliary two-dimensional coarse grid operators. For storage efficiency it makes sense to call BOXMG for each plane on each plane relaxation sweep and to recompute auxiliary coarse grid operators. These two approaches are discussed further in § 4.

With regard to saving storage one idea is to use plane sections of the three-dimensional operators when available. For example, for a given xy plane relaxation for $k = M$, BOXMG would be called with the part of L^M that lies in the given xy plane, call it $L_{z_0}^M$. If $z = z_0$ is one of the planes that contain coarse grid points, then instead of using BOXMG to generate two-dimensional auxiliary coarse grid operators, one could consider using $L_{z_0}^k$, $k = 1, \ldots, M-1$ and similarly $(I_{k-1}^k)_{z_0}$, $k = 1, \ldots, M-1$. We had hoped that this approach would be good enough for government work— and it does work pretty well for a lot of problems— but unfortunately very poorly for severely anisotropic problems.

The algorithm used when BOXMG is called for plane relaxation is to inject the appropriate plane of the current three-dimensional solution as a guess for the solution of the two-dimensional problem. One V-cycle is then performed with alternating line relaxation and with $IM = 2$, $IU = 1$, and $ID = 1$; the two relaxation sweeps on the finest grid are divided between one before and one after the coarse grids are visited. In the problems we have tried, additional V-cycles have never helped appreciably; one V-cycle has provided sufficient convergence.

Storage is a big problem for three-dimensional problems. If one is concerned with problems in which the difference coefficients are expensive to generate, the storage of the coefficients on the finest grid is necessary, which with storage of F^M and the answer requires $6n[16n]$ words for a seven [twenty-seven] point operator on a grid with n points. Storage of the I_{k-1}^k requires $26n(\frac{1}{8} + \frac{1}{64} + \cdots) \leq 3\frac{5}{7}n$ words. Storage of the answer, F^k, and L^k, $k = 1, \ldots, M-1$ requires $16n(\frac{1}{8} + \frac{1}{64} + \cdots) \leq 2\frac{2}{7}n$ words. This storage is required even for point Gauss-Seidel relaxation. If $n = n_1 n_2 n_3$, storage for xy plane relaxation is bounded by $n_3(17(\frac{n_1}{2}\frac{n_2}{2}(1+\frac{1}{4}+\cdots)+\frac{1}{2}\frac{n_1}{4}\frac{n_2}{4}(1+\frac{1}{4}+\cdots)+\cdots)) \leq \frac{136}{21}n$. (Here 17 = 5 for storage of nine point operator +8 for storage of interpolation coefficients +2 for storage of factors for alternating line relaxation +1 for storage of the answer +1 for the storage of the right-hand side.) Hence, storage for alternating plane relaxation requires $3(\frac{136}{21}n) = 19\frac{3}{7}n$ words plus additional storage of $3n$ words on

the finest grid to store the factors for alternaing line relaxation (needed by BOXMG). Storage must also be provided for the LU decomposition for the coarsest grid for each plane. Thus if storage is provided for the coefficients for xy, yz, and xz plane relaxation, storage is about two and a half [two] times as much as that required for point relaxation for a seven [twenty-seven] point operator. An alternative to the approach in this paper is algebraic multigrid (AMG) [BMR]; in AMG there is overhead for pointers and the drawback of no vectorization without the use of scatter-gathers, but the payoff is the need to do only point Gauss-Seidel relaxation and the versatility to handle nonrectangular grids. Further research is required to make the relative merits of the two methods clearer.

Vectorization is as easy in the three-dimensional case as it is in the two-dimensional case. The coarse grid operator generation, interpolation, and residual weighting routines vectorize immediately. Point Gauss-Seidel is vectorized by using red-black [four color] relaxation in the seven [twenty-seven] point operator case. Plane relaxation uses two-dimensional black box multigrid routines which are already vectorized. Note that the use of (2.6) need not inhibit vectorization since an IF statement can be avoided by the use of the AMAX1 function.

4. Numerical examples. All of the examples given here use V-cycles and $IM = 2$, $IU = 1$, $ID = 1$. All start on the coarsest grid and bootstrap up to the finest grid with an F-cycle.

4.1. Example 1. We begin with a simple problem

$$\begin{cases} -\Delta U = 1. \text{ in } \Omega = (0.,32.) \times (0.,32.) \times (0.,32.), \\ \dfrac{\partial U}{\partial \nu} = 0. \text{ on } x = 0., \text{ on } y = 0., \text{ and on } z = 0, \\ \dfrac{\partial U}{\partial \nu} = -\dfrac{1}{2}U \text{ on } x = 32., \text{ on } y = 32., \text{ and on } z = 32. \end{cases} \tag{4.1}$$

The discretization is the standard seven point one with the boundary conditions incorporated into the operator. We use point red-black Gauss-Seidel relaxation and iterate until

$$\|r^M\| = \|F^M - L^M U^M\| = \left(\sum_{i,j,k} (F^M - L^M U^M)^2_{i,j,k} \right)^{\frac{1}{2}} \leq 10^{-6}.$$

The comparison between (2.7) and (2.10) for (4.1) is given in Tables 1 and 2. Column 1 gives the number of unknowns, columns 2 and 4 give the reduction in $\|F^M - L^M U^M\|$ in the first and last cycles, respectively, and column 3 gives the number of cycles. Note that the convergence factors are not as good as one would expect for a constant coefficient operator, and the reason is that the implementation of the third kind boundary condition gives rise to a coefficient of U^M which jumps from 0. to h_M^2 at the boundary.

TABLE 1
Results for (4.1) *using* (2.7)

Size of Problem	Convergence Factor for First Cycle	Number of Cycles	Convergence Factor for Last Cycle
$9 \times 9 \times 9$	.08	10	.19
$17 \times 17 \times 17$	.06	11	.21
$33 \times 33 \times 33$	.04	12	.26

TABLE 2
Results for (4.1) *using* (2.10)

Size of Problem	Convergence Factor for First Cycle	Number of Cycles	Convergence Factor for Last Cycle
$9 \times 9 \times 9$	.13	11	.18
$17 \times 17 \times 17$	.09	13	.28
$33 \times 33 \times 33$	.04	12	.26

4.2. Example 2. The second problem is

$$\text{(4.2)} \qquad \begin{cases} -\nabla \cdot (D\nabla U) + \sigma U = F \text{ in } \Omega = (0.,32.) \times (0.,32.) \times (0.,32.), \\ \dfrac{\partial U}{\partial \nu} = 0. \text{ on } x = 0., \text{ on } y = 0., \text{ and on } z = 0, \\ \dfrac{\partial U}{\partial \nu} = -\dfrac{1}{2D} U \text{ on } x = 32., \text{ on } y = 32., \text{ and on } z = 32. \end{cases}$$

D is defined as follows: let

$$\begin{aligned} \mathcal{O}^1(w_1,w_2) &= \{(x,y,z) : 0 < x < w_1, 0 < y < w_1, 0 < z < w_1\}, \\ \mathcal{O}^2(w_1,w_2) &= \{(x,y,z) : w_1 < x < w_2, 0 < y < w_1, 0 < z < w_1\}, \\ \mathcal{O}^3(w_1,w_2) &= \{(x,y,z) : 0 < x < w_1, w_1 < y < w_2, 0 < z < w_1\}, \\ \mathcal{O}^4(w_1,w_2) &= \{(x,y,z) : w_1 < x < w_2, w_1 < y < w_2, 0 < z < w_1\}, \\ \mathcal{O}^5(w_1,w_2) &= \{(x,y,z) : 0 < x < w_1, 0 < y < w_1, w_1 < z_1 < w_2\}, \\ \mathcal{O}^6(w_1,w_2) &= \{(x,y,z) : w_1 < x < w_2, 0 < y < w_1, w_1 < z < w_2\}, \\ \mathcal{O}^7(w_1,w_2) &= \{(x,y,z) : 0 < x < w_1, w_1 < y < w_2, w_1 < z < w_2\}, \\ \mathcal{O}^8(w_1,w_2) &= \{(x,y,z) : w_1 < x < w_2, w_1 < y < w_2, w_1 < z < w_2\}; \end{aligned}$$

then

$$D(x,y,z) = \bar{D}^l \quad \text{if } (x,y,z) \in \mathcal{O}^l(16.,32.),$$

where

$$\bar{D}^l = \begin{cases} 1000. & \text{for } l = 1,4,6,7, \\ 1. & \text{for } l = 2,3,5,8, \end{cases}$$

$\sigma \equiv 0.$, and

$$F(x,y,z) = \begin{cases} 1. & \text{for } (x,y,z) \in \mathcal{O}^l(16.,32.), l = 2,3,5,8, \\ 0. & \text{otherwise.} \end{cases}$$

The difference scheme is

$$\begin{aligned} &-\frac{D^1_{i+1/2,j,k}}{h^2}(U_{i+1,j,k} - U_{i,j,k}) + \frac{D^1_{i-1/2,j,k}}{h^2}(U_{i,j,k} - U_{i-1,j,k}) \\ &-\frac{D^2_{i,j+1/2,k}}{h^2}(U_{i,j+1,k} - U_{i,j,k}) + \frac{D^2_{i,j-1/2,k}}{h^2}(U_{i,j,k} - U_{i,j-1,k}) \\ &-\frac{D^3_{i,j,k+1/2}}{h^2}(U_{i,j,k+1} - U_{i,j,k}) + \frac{D^3_{i,j,k-1/2}}{h^2}(U_{i,j,k} - U_{i,j,k-1}) \\ &+ \sigma_{i,j,k} U_{i,j,k} = F_{i,j,k}. \end{aligned}$$

$D^1_{i+1/2,j,k}$ is the arithmetic average of D^1 at the four nearest adjacent cell centers,etc.; $\sigma_{i,j,k}$ and $F_{i,j,k}$ are the arithmetic averages of σ and F at the eight nearest adjacent cell centers; h is the mesh spacing. The comparison between (2.7) and (2.10) for (4.2)

is given in Tables 3 and 4; the last row gives the result of changing $w_1 = 16.$ to $w_1 = 15.$, moving the interfaces so that they lie on grid lines only on the finest grid. To test problems with jumps in σ as well, Tables 5 and 6 compare (2.7) and (2.10) for this same problem but with $\sigma = 1/3D$. (In all problems in § 4.2, we use point red-black Gauss-Seidel and iterate only until $\|r^M\| \leq 10^{-4}$, since the number of cycles is sufficiently large to test asymptotic behavior.)

TABLE 3
Results for (4.2) *using* (2.7)

Size of Problem	Convergence Factor for First Cycle	Number of Cycles	Convergence Factor for Last Cycle
$9 \times 9 \times 9$	.25	15	.43
$17 \times 17 \times 17$	.30	20	.53
$33 \times 33 \times 33$	.33	27	.61
$33 \times 33 \times 33$, *breakpoint* = 15.	.21	11	.33

TABLE 4
Results for (4.2) *using* (2.10)

Size of Problem	Convergence Factor for First Cycle	Number of Cycles	Convergence Factor for Last Cycle
$9 \times 9 \times 9$	.24	15	.43
$17 \times 17 \times 17$	.30	20	.53
$33 \times 33 \times 33$	.33	27	.61
$33 \times 33 \times 33$, *breakpoint* = 15.	.18	11	.30

TABLE 5
Results for (4.2) *with* $\sigma = 1/(3D)$ *using* (2.7)

Size of Problem	Convergence Factor for First Cycle	Number of Cycles	Convergence Factor for Last Cycle
$9 \times 9 \times 9$	.12	12	.42
$17 \times 17 \times 17$	.16	16	.52
$33 \times 33 \times 33$	.28	22	.61
$33 \times 33 \times 33$, *breakpoint* = 15.	.18	10	.31

TABLE 6
Results for (4.2) *with* $\sigma = 1/(3D)$ *using* (2.10)

Size of Problem	Convergence Factor for First Cycle	Number of Cycles	Convergence Factor for Last Cycle
$9 \times 9 \times 9$	.09	12	.42
$17 \times 17 \times 17$	.11	14	.52
$33 \times 33 \times 33$	.24	21	.61
$33 \times 33 \times 33$, *breakpoint* = 15.	.16	10	.33

4.3. Examples 3 and 4. So far there is not much difference between the convergence factors of (2.7) and (2.10) so that if one can tolerate the coding complexity of (2.7), either one is acceptable. We now consider example 3, a problem on $\Omega = (0., 17.6) \times (0., 17.6) \times (0., 17.6)$ in which $D = 1.$ in the first octant $\mathcal{O}^1(1.6, 17.6)$ and $D = 100.$ otherwise, but we zone this problem with x mesh spacing

$$\triangle x = \begin{cases} .1, 0 \le x \le 1.6, \\ 1., 1.6 \le x \le 17.6 \end{cases}$$

and similarly for y mesh spacing $\triangle y$ and for z mesh spacing $\triangle z$ and express the result as a problem like (4.2) on which uniform meshing is used:

$$D^i = D^{i,l} \quad \text{for } (x, y, z) \in \mathcal{O}^l(16., 32.), \quad i = 1, 2, 3,$$

where

$$D^{1,l} = \begin{cases} .1 & \text{for } l = 1, \\ 1. & \text{for } l = 2, \\ 100. & \text{for } l = 3, 5, 8, \\ 10. & \text{for } l = 4, 6, \\ 1000. & \text{for } l = 7, \end{cases}$$

$$D^{2,l} = \begin{cases} .1 & \text{for } l = 1, \\ 100. & \text{for } l = 2, 5, 8, \\ 1. & \text{for } l = 3, \\ 10. & \text{for } l = 4, 7, \\ 1000. & \text{for } l = 6, \end{cases} \tag{4.3}$$

$$D^{3,l} = \begin{cases} .1 & \text{for } l = 1, \\ 100. & \text{for } l = 2, 3, 8, \\ 1000. & \text{for } l = 4, \\ 1. & \text{for } l = 5, \\ 10. & \text{for } l = 6, 7, \end{cases}$$

$$F = \begin{cases} .001 & \text{for } (x, y, z) \in \mathcal{O}^1(16., 32.), \\ 0. & \text{otherwise.} \end{cases}$$

For example 3, we use alternating red-black plane relaxation and iterate until $\|r^M\| \le 10^{-6}$. The comparison of (2.7) and (2.10) for (4.3) is given in Tables 7 and 8. In these tables we also give the corrected convergence factor, the cube root of the convergence factor, to account for the work in doing alternating plane relaxation. Again, the last row gives the result of changing $w_1 = 16.$ to $w_1 = 15.$ For this problem, (2.7) and (2.10) give equal performance except for the last row, where (2.7) outperforms (2.10).

TABLE 7
Results for (4.3) *using* (2.7)

Size of Problem	Convergence Factor = CF for First Cycle and $CF^{1/3}$	Number of Cycles	Convergence Factor = CF for Last Cycle and $CF^{1/3}$
$9 \times 9 \times 9$	.06, .39	4	.06, .40
$17 \times 17 \times 17$	.03, .33	4	.04, .34
$33 \times 33 \times 33$	.02, .26	3	.04, .35
$33 \times 33 \times 33$, *breakpoint* = 15.	.11, .47	5	.15, .58

TABLE 8
Results for (4.3) *using* (2.10)

Size of Problem	Convergence Factor = CF for First Cycle and $CF^{1/3}$	Number of Cycles	Convergence Factor = CF for Last Cycle and $CF^{1/3}$
$9 \times 9 \times 9$	.06, .39	4	.06, .40
$17 \times 17 \times 17$	.03, .33	4	.04, .34
$33 \times 33 \times 33$	.02, .26	3	.04, .35
$33 \times 33 \times 33$, *breakpoint* = 15.	.17, .56	8	.23, .62

We also ran (2.7) and (2.10) for this problem, computing the coefficients for plane relaxation as needed rather than storing them. The additional execution time was 30%, 42%, and 63% for the $9 \times 9 \times 9$, $17 \times 17 \times 17$, and $33 \times 33 \times 33$ problem, respectively; thus, recomputation of coefficients is not advisable unless the charge for storage is exorbitant. There will be problems, of course, which will exceed available storage if the coefficients for plane relaxation are stored but will fit in memory if they are recomputed.

Example 4 is again like (4.2) with

$$D^i = D^{i,l} \quad \text{for } (x,y,z) \in \mathcal{O}^l(16.,32.), \quad i = 1,2,3,$$

but with

$$\text{(4.4)} \qquad \begin{aligned} D^{1,l} &= \begin{cases} 1. & \text{for } l = 1,3,5,7, \\ .01 & \text{for } l = 2,4,6,8, \end{cases} \\ D^{2,l} &= \begin{cases} 100. & \text{for } l = 1,2,5,6, \\ 1. & \text{for } l = 3,4,7,8, \end{cases} \\ D^{3,l} &= \begin{cases} .01 & \text{for } l = 1,2,3,4, \\ 100. & \text{for } l = 5,6,7,8, \end{cases} \\ F &= \begin{cases} 1. & \text{for } (x,y,z) \in \mathcal{O}^1(16.,32.), \\ 0. & \text{otherwise.} \end{cases} \end{aligned}$$

For example 4, we use alternating red-black plane relaxation and iterate until $\|r^M\| \leq 10^{-6}$. The comparison of (2.7) and (2.10) for (4.4) is given in Tables 9 and 10. For this problem (2.10) is superior to (2.7) except for the third row.

TABLE 9
Results for (4.4) *using* (2.7)

Size of Problem	Convergence Factor = CF for First Cycle and $CF^{1/3}$	Number of Cycles	Convergence Factor = CF for Last Cycle and $CF^{1/3}$
$9 \times 9 \times 9$	.05, .36	6	.06, .38
$17 \times 17 \times 17$	.02, .28	6	.09, .46
$33 \times 33 \times 33$	.02, .27	6	.15, .53
$33 \times 33 \times 33$, *breakpoint* = 15.	.05, .36	7	.13, .51

TABLE 10
Results for (4.4) *using* (2.10)

Size of Problem	Convergence Factor = CF for First Cycle and $CF^{1/3}$	Number of Cycles	Convergence Factor = CF for Last Cycle and $CF^{1/3}$
$9 \times 9 \times 9$	.05, .36	5	.05, .38
$17 \times 17 \times 17$	.02, .29	5	.06, .39
$33 \times 33 \times 33$	.04, .33	7	.14, .51
$33 \times 33 \times 33$, *breakpoint* = 15.	.04, .33	5	.05, .38

4.4. Examples 5 and 6. As a final exercise, we consider the two problems which were the hardest in [BF]. They are both problems on $\Omega = (0,1) \times (0,1) \times (0,1)$ of the form

$$\begin{cases} \sum_1^3 \frac{\partial}{\partial x_i}(D^i \frac{\partial U}{\partial x_i}) + 10^{-4}U = F & \text{in } \Omega, \\ \frac{\partial U}{\partial \nu} = 0 & \text{on } \partial\Omega, \end{cases} \tag{4.5}$$

and for both

$$F(x,y,z) = \begin{cases} 1. & \text{if } (x,y,z) = (0.,0.,0.), \\ 0. & \text{otherwise.} \end{cases}$$

As in [BF], cell-centered differencing is used, with the first and last cell centers in each coordinated direction being on $\partial\Omega$; since interfaces are assumed to occur on cell edges, harmonic averaging is used; for example, $D^1_{i+1/2,j,k}$ is the harmonic average of D^1 at $(i+1,j,k)$ and (i,j,k),etc. For both problems we use alternating red-black plane relaxation and iterate until $\|r^M\| \leq 10^{-6}$. Example 5 is problem number 8 in [BF] and has

$$D^1 = 1. \quad D^2 = 10^3 \quad D^3 = 10^{-3}. \tag{4.5a}$$

The comparison between (2.7) and (2.10) for (4.5*a*) is given in Tables 11 and 12. Example 6 is problem number 10 in [BF] and has

$$D^i = D^{i,l} \quad \text{if } (x,y,z) \in \mathcal{O}^l(w,1.),$$

where

$$\begin{aligned} D^{1,l} &= \begin{cases} 1. & \text{for } l = 1,3,5,7, \\ .01 & \text{for } l = 2,4,6,8, \end{cases} \\ D^{2,l} &= \begin{cases} 100. & \text{for } l = 1,2,5,6, \\ 1. & \text{for } l = 3,4,7,8, \end{cases} \\ D^{3,l} &= \begin{cases} .01 & \text{for } l = 1,2,3,4, \\ 100. & \text{for } l = 5,6,7,8, \end{cases} \\ w &= \begin{cases} .4375 & \text{for the } 9 \times 9 \times 9 \text{ case}, \\ .40625 & \text{for the } 17 \times 17 \times 17 \text{ case}, \\ .390625 & \text{for the } 33 \times 33 \times 33 \text{ case}. \end{cases} \end{aligned} \tag{4.5b}$$

(We remark that the description of problem 6 in [BF] does not specify it completely, and we acknowledge the help of the authors in providing the additional details.)

The comparison between (2.7) and (2.10) for (4.5*b*) is given in Tables 13 and 14.

TABLE 11
Results for (4.5*a*) *using* (2.7)

Size of Problem	Convergence Factor = CF for First Cycle and $CF^{1/3}$	Number of Cycles	Convergence Factor = CF for Last Cycle and $CF^{1/3}$
$9 \times 9 \times 9$	.007, .19	1	.007, .19
$17 \times 17 \times 17$	.003, .14	1	.003, .14
$33 \times 33 \times 33$	.005, .17	1	.005, .17

TABLE 12
Results for (4.5*a*) *using* (2.10)

Size of Problem	Convergence Factor = CF for First Cycle and $CF^{1/3}$	Number of Cycles	Convergence Factor = CF for Last Cycle and $CF^{1/3}$
$9 \times 9 \times 9$	.0005, .08	1	.0005, .08
$17 \times 17 \times 17$	.0007, .09	1	.0007, .09
$33 \times 33 \times 33$	.001, .11	1	.001, .11

TABLE 13
Results for (4.5*b*) *using* (2.7)

Size of Problem	Convergence Factor = CF for First Cycle and $CF^{1/3}$	Number of Cycles	Convergence Factor = CF for Last Cycle and $CF^{1/3}$
$9 \times 9 \times 9$	.009, .21	2	.06, .39
$17 \times 17 \times 17$	.006, .18	2	.02, .29
$33 \times 33 \times 33$	.02, .26	2	.05, .37

TABLE 14
Results for (4.5*b*) *using* (2.10)

Size of Problem	Convergence Factor = CF for First Cycle and $CF^{1/3}$	Number of Cycles	Convergence Factor = CF for Last Cycle and $CF^{1/3}$
$9 \times 9 \times 9$	.02, .26	2	.08, .43
$17 \times 17 \times 17$	.005, .18	1	.005, .18
$33 \times 33 \times 33$	.01, .23	1	.01, .23

Examples 5 and 6 were the two hardest in [BF], and they are both solved easily here with both methods, indeed too easily to provide much of a comparison. The better performance of (2.7) compared with the results of [BF] is probably explained by our use of $\tilde{L}^k$ in (2.3) instead of L^k; indeed, we also obtain very poor convergence if L^k is used instead of $\tilde{L}^k$ in (2.3).

5. Conclusion. In this paper we have developed two multigrid methods for solving three-dimensional diffusion equations with discontinuous and anisotropic coeffcients. The first of these is a straightforward generalization of the method in [D1] for two-dimensional problems. The second uses auxiliary intermediate grids obtained by semi-coarsening successively in each of the three independent variables. Numerical evidence suggests that both methods perform about the same for isotropic problems. For anisotropic problems, we have exhibited an example for which the first method is significantly better than the second and an example where the reverse is true. Thus, since the coding complexity of the first method far exceeds that of the second, it would

appear that the second method is to be preferred, particularly if one is considering extensions to nonsymmetric problems and to systems of equations.

Acknowledgments. We gratefully acknowledge the helpful suggestions of Alda Behie, Achi Brandt, and the referees.

REFERENCES

[ABDP] R. E. ALCOUFFE, A. BRANDT, J. E. DENDY, JR. AND J. W. PAINTER, *The multigrid method for the diffusion equation with strongly discontinuous coefficients*, this Journal, 2(1981), pp. 430–454.

[BF] A. BEHIE AND P. A. FORSYTH, JR., *Multi-grid solution of three-dimensional problems with discontinuous coefficients*, Appl. Math. Comp., 13(1983), pp. 229–240.

[BMR] A. BRANDT, S. MCCORMICK AND J. RUGE, *Algebraic multigrid (AMG) for automatic multigrid solution with application to geodetic computations*, this Journal, submitted for publication.

[D1] J. E. DENDY, JR., *Black box multigrid*, J. Comp. Phys., 48(1982), pp. 366–386.

[D2] J. E. DENDY, JR., *Black box multigrid for nonsymmetric problems*, Appl. Math. Comp., 13(1983), pp. 261–283.

[D3] J. E. DENDY, JR., *Black box multigrid for systems*, Appl. Math. Comp., to appear.

[S] T. SCOTT, *Multi-grid methods for oil reservoir simulation in two and three dimensions*, J. Comp. Phys., 59(1985), pp. 290–307.

[T] C.-A. THOLE, *Basic smoothing procedures for the multigrid treatment of elliptic 3D-operators*, to appear.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 5, September 1987
© 1987 Society for Industrial and Applied Mathematics

ALTERNATING DIRECTION METHODS ON MULTIPROCESSORS*

S. LENNART JOHNSSON†, YOUCEF SAAD‡ AND MARTIN H. SCHULTZ§

Abstract. We propose several implementations of the Alternating Direction Method (ADM) for solving parabolic partial differential equations on multiprocessors. A complexity analysis of these implementations shows that the method can be made highly efficient on parallel architectures by using pipelining and variations of the classical Gaussian elimination algorithm for solving tridiagonal systems. Previously, we showed that we could obtain linear speedups for moderate numbers of processors in a ring architecture. In this paper we discuss extensions to a large number of processors in a 2-D grid architecture and a hypercube.

Key words. parabolic equations, Alternating Direction Methods (ADM), concurrent computation, ensemble architectures

AMS(MOS) subject classifications. 15AO6, 58G11, 65F05, 68PO5, 68Q25

1. Introduction. We propose a few implementations of the Alternating Direction Method (ADM) for solving parabolic partial differential equations on multiprocessors. A complexity analysis of these implementations shows that the method can be made highly efficient on parallel architectures by using pipelining and variations of the classical Gaussian elimination algorithm for solving tridiagonal systems. In [22] we showed that we could obtain linear speedups for moderate numbers of processors in a ring architecture. In this paper we discuss extensions to a large number of processors in a 2-D grid architecture and a hypercube.

Consider the partial differential equation $u_t = Lu$ where L is some elliptic operator on a unit square. The domain is discretized into a square mesh with n points in each direction. For illustration we assume in this Introduction that $n = 64$ which may yield a reasonable accuracy in practice. Thus the total number of grid points is n^2, i.e., 4096. The first question that arises when implementing algorithms on parallel processors is how to map the data into the processors. Obviously, the number of possibilities is extremely large and we can consider only a few that appear reasonable. The purpose of this paper is to introduce and compare a few mappings and corresponding algorithms, rather than to be exhaustive, which is impossible.

We can distinguish three separate cases. The first is when k, the number of processors, is smaller than n, i.e., in our illustration $k \leqq 64$. This is the small number of processors case. We will describe algorithms for the shared memory and ring architectures that are devised for this case and only for this case, i.e., only for $k \leqq n$. The second situation is when $n \leqq k \ll n^2$. This is the medium number of processors case. For example, we will see a case of importance when $k = n^{4/3}$, i.e., $k = 256$ in our illustration. New algorithms for the grid topology are proposed that take advantage of the larger number of processors. These algorithms are now defined for every k such that $1 \leqq k \leqq n^2$, but as will be seen they are only efficient when $k \leqq O(n^{4/3})$, i.e., for the medium number of processors case and they can achieve a time of $O(n^{2/3})$ per step of ADM. Finally, when $k = O(n^2)$, i.e., when $k = n^2 = 4096$ in our illustration, we will see that a simple adaptation of the grid algorithm to hypercubes yields an algorithm

* Received by the editors November 18, 1985; accepted for publication (in revised form) September 17, 1986. This work was supported in part by Office of Naval Research grants N00014-82-K-0184 and N00014-84-K-0043 and in part by a joint study with IBM/Kingston.

† Department of Computer Science, Yale University, New Haven, Connecticut 06520 and Thinking Machines Corporation, Cambridge, Massachusetts 02142.

‡ Center for Supercomputer Research and Development, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801.

§ Department of Computer Science, Yale University, New Haven, Connecticut 06520.

that breaks the barrier of $O(n^{2/3})$ of the grid topology to yield a time of the order $O(\log_2 n)$.

Each of the three cases is realistic in that a number of processors of the order, say, $k = n^2 = 4096$ is perfectly achievable with present day technology. We must distinguish among these cases because when implementing the ADM algorithm in a parallel computing environment the real question is: if we have k processors and an $n \times n$ mesh, which mapping and which algorithm are likely to be the best if we want to use ADM? It may be the case that not all processors should be used because, for example, the overhead in communication may make use of a larger number of processors that slow down the algorithm. This is in fact what happens with the grid of processors: although the algorithm is defined even for $k = n^2$, the optimal number of processors for our algorithm is $O(n^{4/3})$ and the best time that can be achieved is $O(n^{2/3})$.

Before proceeding with the algorithms we should point that the variety in possible algorithms for given architectures is so immense that we cannot possibly compare all possible algorithms on all possible architectures. We will examine a few interesting combinations, but acknowledge that improvements to some of the algorithms may still be possible.

2. The Alternating Direction Method and its parallel implementations. We consider the partial differential equation:

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(a(x, y)\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(b(x, y)\frac{\partial u}{\partial y}\right)$$

on the domain $(x, y, t) \in \Omega \times [0, T] \equiv [0, 1] \times [0, 1] \times [0, T]$, with the initial and boundary conditions:

$$u(x, y, 0) = u_0(x, y) \quad \forall (x, y) \in \Omega,$$
$$u(\bar{x}, \bar{y}, t) = g(\bar{x}, \bar{y}, t) \quad \forall (\bar{x}, \bar{y}) \in \partial\Omega, \quad t > 0,$$

where $\partial\Omega$ is the boundary of the unit square Ω. After the description and analysis of algorithms for the two-dimensional problem we generalize the results to higher-dimensional domains.

A common approach to the solution of the above problem is the Alternating Direction Method. First the equations are discretized with respect to the space variables x and y using a mesh of n interior points in each direction. The result is the system of $N \equiv n^2$ ordinary differential equations:

$$\frac{d\underline{u}}{dt} = A_x \underline{u} + B_y \underline{u} \tag{2.1}$$

in which the matrices A_x and B_y represent the 3-point central difference approximations to the operators $\partial/\partial x(a(x, y)\,\partial/\partial x)$ and $\partial/\partial y(b(x, y)\,\partial/\partial y)$, respectively.

The Alternating Direction Method algorithm consists of stepping (2.1) forward in time alternately in the x and y directions as follows [27]:

$$(I - \tfrac{1}{2}\Delta t A x)u^{i+1/2} = (I + \tfrac{1}{2}\Delta t B_y)u^i, \tag{2.2}$$

$$(I - \tfrac{1}{2}\Delta t B_y)u^{i+1} = (I + \tfrac{1}{2}\Delta t A_x)u^{i+1/2}. \tag{2.3}$$

We observe that if the mesh points are ordered by lines in the x direction, then (2.2) constitutes a set of n independent tridiagonal systems whose solution is perfectly parallelizable. However, (2.3) constitutes one tridiagonal system in which the nonzero diagonals are the main diagonal and the $(n+1)$st super- and sub-diagonals. It is important to note that this second system can also be recast into a set of n independent tridiagonal systems by *reordering the grid points* by lines, this time in the y direction. This essentially amounts to transposing the matrix representing the solution at the

$n \times n$ grid points and is an expensive data permutation operation, which is often cited as the main drawback of the Alternating Direction Method in regard to its implementation on parallel machines having only distributed memory. However, it is also a potential source of performance degradation on shared memory systems, since such systems must have banked memory in order to achieve an overall high memory bandwidth. With different access schemes, like row and column access, and data layed out by a standard Fortran compiler, the access rate for one of the access schemes may be limited by the bandwidth of individual banks. However, though not used in currently available architectures, there exist architectural solutions to this problem [15], [14]. The other difficulty that has been traditionally associated with parallelizing the Alternating Direction Method is that the classical algorithms for solving tridiagonal systems are sequential in nature. The arithmetic complexity of the Alternating Direction Method on an idealized parallel architecture without contention for storage or access conflicts has been investigated by Sameh et al. [24]. Various parallel implementations of the Alternating Direction Method have been discussed by Gannon and van Rosendale [3], Lambiotte [13] and Ortega and Voigt [17].

The main purpose of this paper is to describe some data structures and algorithms that efficiently use some candidate multiprocessor configurations for future parallel computer systems. We limit the discussion to the solution of the two tridiagonal systems (2.2) and (2.3). The communication required for the computation of the right-hand side is almost identical to that required for the tridiagonal system, and the arithmetic complexity is approximately half of that for the tridiagonal system. Since the choice of data structure and algorithm is unaffected by the inclusion of the computations for the right-hand side we omit them. On a single processor each half step costs $5n^2$ operations for the forward and $3n^2$ operations for the backward sweep. Assuming that s arithmetic operations can be done per second, the time for a half step on a single processor is approximately

$$T_1 = \frac{8n^2}{s}. \tag{2.4}$$

Throughout the paper it is assumed that communication and arithmetic computations cannot take place concurrently. This assumption is essentially made for convenience and simplicity of our analysis and does not affect the qualitative aspects of the conlusion.

3. Alternating Direction Methods on the shared memory and the broadcast bus models. In this section we assume that k processors, where $k \leqq n$, are connected to a shared memory with total bandwidth B words per second and startup time τ. Moreover, we assume that the I/O bandwidth of each processor is b. Thus B/b different processors can simultaneously access the shared memory. Each processor also has a local memory.

For the Alternating Direction Method we assume that the coefficients of the equations are distributed over the local memories of the processors, and that the shared memory is used to exchange the iterates of u. Processors access data from the memory, use it to do computation and write into the shared memory the results of the computation. If we assume that there are n/k similar tridiagonal systems to solve in each processor, then in each half step each processor has to read n^2/k data, solve n/k systems, and then write the data back to memory. The total time required for performing the arithmetic operations of one half-step is approximately

$$T_{S,\text{Arith}} \approx \frac{n}{k}\frac{8n}{s} = \frac{8n^2}{ks}.$$

Concerning the data movements, at every step the processors must access their share of the n^2 values of the function u, and then after completion of the arithmetic operations, they must update these values, i.e., write them to memory. This requires the communication time

$$T_{S,\text{Comm}} \approx 2\frac{k}{B/b}\left(\tau+\frac{n^2}{kb}\right)$$
$$\approx 2k\frac{b}{B}\tau+2\frac{n^2}{B}.$$

Hence in the shared memory model, it takes a total time of

$$T_S \approx 2k\frac{b}{B}\tau+2\frac{n^2}{B}+\frac{8n^2}{ks}$$

to perform one half step of the Alternating Direction Method.

Assume now that instead of sharing a memory, all k processors are linked to each other via a global bus, which allows for broadcasting. The matrix is distributed among the private memories of the processors. The time to perform the arithmetic operations is the same as for the shared memory model.

In order to communicate the iterates of u each processor broadcasts its data to all the other processors. With a bandwidth B of the bus the data exchange requires a total communication time of

$$T_{B,\text{Comm}} \approx \frac{k}{B/b}\left(\tau+\frac{n^2}{kb}\right) = k\frac{b}{B}\tau+\frac{n^2}{B}.$$

Hence we have the total time

$$T_B \approx k\frac{b}{B}\tau+\frac{n^2}{B}+\frac{8n^2}{ks}$$

for performing one half-step of the Alternating Direction Method.

Observe that neither of the above two models achieves a reasonable speedup when k increases because of the limiting term n^2 in the communication cost. This is the source of the belief that Alternating Direction Methods do not parallelize well.

4. Alternating Direction Methods on a multiprocessor ring. In the ring architecture the k processors are arranged in a ring and each processor can communicate with its two nearest neighbors. In this section we will again assume that $k \leqq n$. It is assumed throughout that the processors are numbered so that the processor numbered i is connected to processors numbered $i+1$ and $i-1$ modulo k. In order to implement the algorithm efficiently, we will first assign the grid points so as to minimize data communication costs. The assignment, which is described in Fig. 1 for a ring of $k=4$ processors, will essentially avoid transposing the data at each half-step.

This assignment results in each of the tridiagonal systems (2.2) being split into k equal parts, one part per processor. We consider the cost of the half step involving the sweep in the x-direction. Looking at the leftmost column of subsquares in Fig. 1, each of the k processors of that column can start performing the forward sweeps simultaneously on the n/k tridiagonal systems that it holds. Thus, the sweep eliminates the variables horizontally from left to right. When the boundary of the first subsquare is reached, each processor sends to its right neighbor in the figure the data that is necessary for that processor to continue its forward sweep on its part of its tridiagonal systems, i.e., the processor numbered j sends to its neighbor numbered $(j+1)$ Mod k

1	2	3	4
2	3	4	1
3	4	1	2
4	1	2	3

FIG. 1. *Domain decomposition and assignment of the unit square into* 4 *processors.*

data for the $(j-1)n/k+1$ to jn/k tridiagonal systems. The forward sweep can now be pursued on the second column of subsquares, and this process is repeated until the forward sweep is completed on the variables of the last column of subsquares. The backward sweep is accomplished in a similar fashion except that we proceed from right to left. The half step in the y direction is performed similarly, with the forward sweep proceeding from bottom to top and the backward sweep from top to bottom. We observe that no processor is idle at any time. It is clear that the time to perform the arithmetic operations is again of the form:

$$T_{R,\text{Arith}} \approx \frac{n}{k}\frac{8n}{s} = \frac{8n^2}{ks}.$$

If an entire row of the tridiagonal system is stored in a processor, then when we cross an interface in the forward sweep, we must transfer the pivot row plus the corresponding element of the right-hand side of each of the n tridiagonal systems in each processor numbered j to the processor numbered $(j+1)$ Mod k. If instead for a complete row a processor is assigned the diagonal matrix element, and the elements to the left and above, it suffices to communicate $2n$ elements in the forward elimination instead of $3n$ elements. These I/O operations can be done in parallel for each of the processors on the same column. A total of $k-1$ interfaces must be crossed. For the backward sweep, we transfer only one element of the current solution per system. Therefore,

$$T_{R,\text{COMM}} = (k-1)\left(\tau+\frac{2n}{kb}\right)+(k-1)\left(\tau+\frac{n}{kb}\right) = 2(k-1)\tau+\left(1-\frac{1}{k}\right)\frac{3n}{b}.$$

Thus, the total execution time comes to

$$T_R(k) = 2(k-1)\tau+\left(1-\frac{1}{k}\right)\frac{3n}{b}+\frac{8n^2}{ks} = 2(k-1)\tau+\frac{3n}{b}+\left(\frac{8n}{s}-\frac{3}{b}\right)\frac{n}{k}.$$

Clearly, the time for solving in the other direction is identical.

The function $T_R(k)$ has a minimum which is achieved for

$$k_{\text{opt}} = \left[\frac{1}{2\tau}\left(\frac{8n^2}{s}-\frac{3n}{b}\right)\right]^{1/2} \tag{4.1}$$

and the corresponding optimal time is

$$T_{R,\text{opt}} = 4\sqrt{\left(\frac{4n^2}{s}-\frac{3n}{2b}\right)\tau}-2\tau+\frac{3n}{b}.$$

There is an important exception to the above formulas that should be made here. The optimal number of processors as given by (4.1) may exceed n, the maximum allowable number of processors for which this algorithm is defined. However, it is

easy to see that in that situation the derivative of the function $T_R(k)$ considered as a function of a real variable is negative for $1 \leqq k \leqq n$, i.e., the optimal value in this case is $k_{\text{opt}} = n$ and the corresponding time is

$$T_{R,\text{opt}} = 2(n-1)\tau + \left(1 - \frac{1}{n}\right)\frac{3n}{b} + \frac{8n}{s} \quad \text{when } k_{\text{opt}} = n.$$

Thus by an appropriate choice of algorithm on the ring, with $O(n)$ processors we can achieve an $O(n)$ speedup. Note that this could also be shown by taking $k = n$ directly without any reference to the best achievable time.

5. Alternating Direction Methods on a square grid of processors. In this section we consider a square two-dimensional grid of processors. Here we only assume that $k \leqq n^2$ and, for convenience, we define $\kappa \equiv \sqrt{k}$. We assign the subsquares defined in the previous section naturally onto the grid, as shown in Fig. 2 in which the numbers in each square indicate the processor number to which the subsquare is assigned.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

FIG. 2. *Assignment of the subdomains to the* 16 *processors of a* 4×4 *grid.*

This assignment resembles the one of the previous section, but each subsquare is now handled by a different processor. If we proceeded as we did for the ring, we would have many processors inactive at all times. In fact, at any given time we would have only κ processors working simultaneously, namely all those holding the same column of subsquares. To avoid this we can think of pipelining the computation: the forward sweep is started with only one tridiagonal system in each of the processors in the first column of subsquares (Processors 1, 5, 9 and 13 in the illustration of Fig. 2). As soon as these processors are finished with the first n/κ steps of Gaussian elimination of their first tridiagonal systems they start the forward sweep on the κ second tridiagonal systems, while processors to their right in the grid start working on their part of the first tridiagonal system. Thus, after $\kappa - 1$ tridiagonal systems are solved, all processors will be active. It is easy to see that the computation consists of forward sweeps on a total of $(n/\kappa) + \kappa - 1$ tridiagonal systems of size n/κ each: n/κ to complete all the first parts in each of the subsquares of the first column of processors, and then $\kappa - 1$ similar eliminations to complete the parts through the last column. Moreover, these are nonsimultaneous eliminations. Each of the partial eliminations is followed by a transfer of the pivot element and the right-hand side, i.e., 2 elements, assuming the "skewed" storage described for the ring architecture. The backward sweep is similar. We find a total time of

$$t_{G0} = \left(\frac{n}{\kappa} + \kappa - 1\right)\left[2\tau + \frac{3}{b} + \frac{8n}{\kappa s}\right].$$

If we start by restricting k to be such that $k \leqq n$, then a nearly linear speedup over the run time for a single processor is achieved and the optimal time is linear in n.

While this is asymptotically equivalent to the result for the ring of processors, it requires an order of magnitude less time for just the data movement. Unfortunately, the above time cannot be less than linear in n, no matter how large the number of processors k, i.e., even when $n < k \leqq n^2$. Thus, it is not possible to outperform significantly the ring architecture for implementing the Alternating Direction Method with this algorithm. The reason is that we are solving tridiagonal systems with the classical Gaussian elimination algorithm. With Gaussian elimination, each tridiagonal system cannot be solved in time less than $8n/s$, independent of the number of processors used, because of its sequential nature.

To exploit the more powerful interconnection features of the grid architecture, we must use a different algorithm. Lambiotte [13] suggested using Gaussian elimination in one direction and odd-even cyclic reduction in the other. This, however, will still require a linear time because of the Gaussian elimination in one direction. More recently, Gannon and van Rosendale [3] have proposed a scheme based on what mechanical and structural engineers call problem substructuring or condensation techniques.

There are many substructuring Gaussian elimination algorithms. The one briefly described next is a version of incomplete nested dissection [4] inspired by Wang's [28] vectorized tridiagonal system solver. The advantage of Wang's variant over the algorithm presented by Kuck and Sameh [25], [16] is its lower arithmetic complexity with almost no increase in communication complexity and its improved numerical properties [8]. The algorithm can be briefly described as follows. Assume that we want to solve *one* tridiagonal system $Tx = b$ of size n in κ processors, each of which holds n/κ successive rows of the system. The algorithm consists of three phases. In the first phase a variant of Gaussian elimination is performed with almost no interprocessor communication. In the jth processor, $j = \{1, 2, \cdots, \kappa\}$, a forward elimination is carried out on equations $(j-1)n/\kappa + i$, $i = \{2, 3, \cdots, n/\kappa\}$, in order to eliminate the subdiagonal elements within each $n/\kappa \times n/\kappa$ diagonal block of T. This requires no interprocessor communication and all the processors compute in parallel. Then, a backward elimination is carried out to eliminate the superdiagonal elements in rows $jn/\kappa - 2$ through $(j-1)n/\kappa$ for $j = \{2, 3, \cdots, \kappa - 1\}$, in rows $n/\kappa - 2$ to 1 for $j = 1$, and in rows $n-1$ through $(\kappa-1)n/\kappa$ for $j = \kappa$. All processors compute in parallel. Interprocessor communication is required for the last equation in partitions 1 through $\kappa - 1$. The result of this first phase is illustrated in Fig. 3 for a system of size 20 distributed in 4 processors. Communication of three nonzero matrix elements and one right-hand side element is required between adjacent processors if an entire row is stored in a processor. However, if the element to the right of the last diagonal element in a partition and the corresponding right-hand side are stored in the processor for the next partition (except for the last partition), then the communication of a modified right-hand side is needed in the forward phase, and one communication of one element for the modification of the diagonal element in the backward elimination phase. The first storage scheme yields a cost of

$$t_1 = \left(\frac{n}{\kappa} - 1\right)\frac{12}{s} + \tau + \frac{4}{b},$$

and the second a cost of

$$t_1 = \left(\frac{n}{\kappa} - 1\right)\frac{12}{s} + 2\tau + \frac{2}{b},$$

for the first phase. This modified storage scheme corresponds to a one step upward

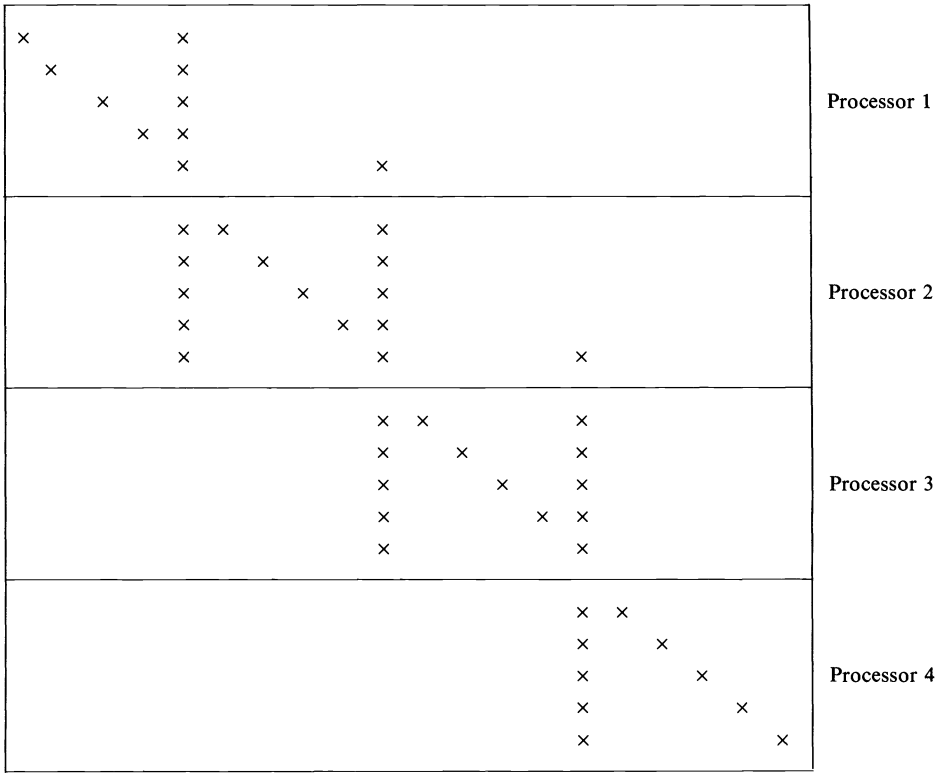


FIG. 3. *A substructuring Gaussian elimination algorithm on four processors.*

jog to the right of the diagonal elements in the partition boundaries shown in Fig. 3. The number of element transfers is reduced by a factor of 2 at the expense of an additional communication startup.

We observe that the unknowns jn/κ, $j=\{1, 2, \cdots, \kappa-1\}$, satisfy an independent tridiagonal system of $\kappa-1$ equations distributed over κ processors. Clearly, the second phase will consist in solving for these unknowns. If the processors form a ring or linear array, the total cost, using a regular Gaussian elimination algorithm, is roughly

$$t_2 \approx (\kappa-1)\left(2\tau+\frac{4}{b}+\frac{8}{s}\right) \quad \text{or} \quad t_2 \approx (\kappa-1)\left(2\tau+\frac{3}{b}+\frac{8}{s}\right).$$

In the third and final phase, each processor j solves for the other variables by subtracting multiples of the fill-in columns. This operation involves no communication, except for the transfer of the variables jn/κ already computed in phase 2 from processors j to processors $j+1$ for $j=\{1, 2, \cdots, \kappa-1\}$. Therefore, the cost of the third phase on a linear array or ring is approximately

$$t_3 \approx \frac{5n}{\kappa s}+\tau+\frac{1}{b}.$$

Going back to implementing the Alternating Direction Method on our grid of processors, we observe that each row of processors can be considered as a linear array of $\kappa \equiv \sqrt{k}$ processors. In each row of processors we will solve n/κ tridiagonal systems

by the substructuring Gaussian elimination. In the first pass of the Alternating Direction Method all the rows of the processor grid will work simultaneously, with no communication between the processors of different rows. In the second pass of the Alternating Direction Method all processors of the same column of the processor grid will work simultaneously with no communication between different columns.

Implementing the first phase is straightforward and its cost is

$$t_1' = \frac{n}{\kappa} \cdot \frac{12n}{\kappa s} + \tau + \frac{n}{\kappa} \cdot \frac{4}{b} = \frac{12n^2}{ks} + \tau + \frac{4n}{\kappa b}$$

or

$$t_1' = \frac{n}{\kappa} \cdot \frac{12n}{\kappa s} + 2\tau + \frac{n}{\kappa} \cdot \frac{2}{b} = \frac{12n^2}{ks} + 2\tau + \frac{2n}{\kappa b}.$$

Implementing the second phase requires a little more care. The problem is that we have n/κ tridiagonal systems each of size κ, or $\kappa - 1$ distributed over κ processors. Just as was pointed out in our first implementation of the Alternating Direction Method for the grid described in the very beginning of this section, a naive implementation consisting of sweeping in the horizontal direction for all variables of the first column of subsquares before forward-sweeping in the second column of subsquares and so on, would be inefficient because only one processor of each row will be active at any time. In fact, this naive approach would require solving n/κ tridiagonal systems of size κ each, and therefore the time would again be nearly linear in n. Once again the remedy is to use pipelining. It is interesting that the seemingly minor modification of pipelining leads to a completely different complexity analysis and a different conclusion.

An argument similar to the one in the beginning of this section shows that in order to complete the forward sweep we need a total of $n/\kappa + \kappa - 1$ successive steps, each consisting of one Gaussian elimination of one row of a tridiagonal system plus a transfer of the pivot row. Clearly, the backward sweep can be pipelined likewise. The total time for this second phase is approximately

$$t_2' = \left(\frac{n}{\kappa} + \kappa - 1\right)\left(2\tau + \frac{4}{b} + \frac{8}{s}\right) \quad \text{or} \quad t_2' = \left(\frac{n}{\kappa} + \kappa - 1\right)\left(2\tau + \frac{3}{b} + \frac{8}{s}\right).$$

Finally, the third phase is fully parallelizable and requires a time of

$$t_3' \approx \frac{5n^2}{ks} + \tau + \frac{n}{\kappa} \cdot \frac{1}{b}.$$

The total as a function of k comes to

$$T_G(k) = \frac{17n^2}{ks} + \left(\frac{n}{\kappa} + \kappa - 1\right)\left(2\tau + \frac{4}{b} + \frac{8}{s}\right) + 2\tau + \frac{5n}{\kappa b} \tag{5.1}$$

or

$$T_G(k) = \frac{17n^2}{ks} + \left(\frac{n}{\kappa} + \kappa - 1\right)\left(2\tau + \frac{3}{b} + \frac{8}{s}\right) + 3\tau + \frac{3n}{\kappa b}. \tag{5.2}$$

The important point is that it is now possible to obtain an execution time of order less than $O(n)$. For example, if we take $k = n^{4/3}$ we get

$$T_G(n^{4/3}) = n^{2/3}\left(\frac{25}{s} + \frac{3}{b} + 2\tau\right) + n^{1/3}\left(\frac{8}{s} + \frac{6}{b} + 2\tau\right) + \tau - \frac{3}{b} - \frac{8}{s}$$

which is $O(n^{2/3})$. This is not the optimal time; however, as we now show the order $O(n^{4/3})$ is asymptotically optimal. Writing $T_G(k)$ in the form

$$T_G(k) = \alpha \frac{n^2}{k} + \beta \frac{n}{\sqrt{k}} + \gamma\sqrt{k} + \text{Constant},$$

we can easily see that the *approximate* minimum is achieved for

$$k_* = \left(2\frac{\alpha}{\gamma}n^2\right)^{2/3}$$

and has the value

$$T_{G,*} \approx \tfrac{3}{2}(2\gamma^2\alpha)^{1/3}n^{2/3}.$$

It is possible to show by employing an argument similar to one used in [20] that $T_{G,*}$ is close to the actual minimum of $T_G(k)$ for large enough n.

An exception similar to the one for the ring needs to be made in case the optimal number of processors k_* exceeds n^2, which is the maximum number of processors for which this algorithm is defined. Note that $k_* = O(n^{4/3})$, which will be between n and n^2 if n is large enough. Although we do not know the constant in this number, the situation is illustrated with the nonoptimal value of $k = n^{4/3}$ or $\kappa = n^{2/3}$. As an example for a 64×64 mesh, this pseudooptimal number of processors will be $k = n^{4/3} = 256$, i.e., we need a 16×16 processor grid, which is a grid of moderate size. For a 512×512 mesh the corresponding processor grid will be a 64×64 grid, which is still realistic if we remember that Goodyear Aerospace's MPP [6] is based on a 128×128 two-dimensional grid and that the Connection Machine [5] can embed a 256×256 processor grid.

Note that when $k \leqq n$, we can choose between using the above 2-D algorithm or embedding a ring into the grid of processors and then using the ring algorithm described in § 4. For $k \ll n$ it is clear that the arithmetic complexity dominates in both cases and the ring algorithm will be advantageous. When k is of the order of n, then the comparison is not as clearcut. For example, if $k = n$ and we neglect the $\sqrt{n}$ terms in the resulting time $T_G(n)$, we get $T_G(n) \approx 17n/s$, as compared with $T_R(n) = 8n/s + 3n/b + 2n\tau$, which means that the grid algorithm will be faster whenever $(9/s - 3/b) < 2\tau$. Note that the way the ring algorithm is defined, it is impossible to take advantage of the additional connections of the grid structure. When implementing it on a 2-D grid there are ways of improving the communication time of the ring algorithm on a 2-D grid by using different mappings, but we will not pursue this matter in this paper.

6. Alternating Direction Methods on hypercube architectures. There are currently several existing loosely coupled multiprocessors, labeled under the name hypercube, which are based on the binary n-cube networks [1], [21], [23], [26]. An m-dimensional hypercube, or m-cube, consists of 2^m nodes that are numbered by m-bit binary numbers, from 0 to $2^m - 1$ and interconnected so that there is a link between two processors if and only if their binary representation differs by one and only one bit. For the case $m = 3$, the 8 nodes can be represented as the vertices of a three-dimensional cube. One of the main advantages of hypercubes is that many of the classical topologies such as two-dimensional or three-dimensional meshes [10], [2], [21], [26], [9] can be embedded, preserving proximity in them. An immediate consequence of this is that the algorithms presented for the ring and the grid architectures can be immediately adapted to the hypercube. Thus one step of the Alternating Direction Method can be run on a hypercube in the same time as on a mesh configured multiprocessor with the same

number of processors, i.e., in time $O(n^{2/3})$ for a sufficiently large dimension. All we need is to embed the grid into the hypercube and then use the substructuring Gaussian elimination of the previous section. For a description on how to map a grid into a hypercube, see for example [9], [21]. An important natural question is whether or not there is another algorithm which achieves a better asymptotic time because of the more powerful topological properties of the hypercube. Once again the answer is yes.

A natural candidate for solving tridiagonal systems on multiprocessors is the cyclic reduction algorithm [6], [11]. Using the same mapping as for the two-dimensional grid, i.e., partitioning the grid into squares of size $n/\kappa \times n/\kappa$ and assigning the square in position (i,j) to processor (i,j) of a two-dimensional mesh of $\kappa \times \kappa$ processors embedded in the hypercube by the proximity preserving embedding described above, it is clear that each of the solve phases in the Alternating Direction Method amounts to solving in each row or column of the conceptual processor mesh n/κ independent tridiagonal systems each of which is split into κ equal parts.

It is important to note that when using the cyclic reduction algorithm the distance between the rows of the tridiagonal system will increase if the grid points are not carefully assigned to the nodes. With a mesh embedding based on a binary-reflected Gray code proximity is preserved throughout the reduction process [11]. For *one* tridiagonal system of 2^m equations, equation i is mapped into the node whose binary label is g_i, where $g_0, g_1, \cdots, g_{2^m-1}$ is the m-bit binary-reflected Gray code [18]. It can be easily shown that g_i and $g_{(i+2^j) \bmod 2^m}$ differ in exactly two bits for all $j>0$. This property means that, with this Gray code assignment, the distance between consecutive rows in the first step of cyclic reduction is one and is exactly two in the subsequent steps. If the tridiagonal system is of size n larger than $\kappa \equiv 2^m$, then substructuring is used with substructures of n/κ equations each and substructure j, $j=\{0, 1, \cdots, \kappa-1\}$, assigned to processor g_j. The solution process is similar to the substructured Gaussian elimination described previously, except that cyclic reduction is used for phase 2. The times for phases 1 and 3 are the same as before. The time for phase 2 is [11]

$$t_2=\left(\frac{17}{s}+4\tau+\frac{10}{b}\right)(\log_2 \kappa-1)-2\tau-\frac{5}{b}+\frac{1}{s}.$$

Consider now the problem of employing this technique in the Alternating Direction Method. An important observation is that each column (or row) of processors is a subcube of the hypercube, on which work can be done independently. Thus, one can consider using the above mapping based on Gray codes on each of the subcubes and the κ different subcubes will solve in parallel a set of n/κ tridiagonal systems each of size n and spread in κ processors, n/κ equations per processor. With n/κ independent tridiagonal systems to be solved in each subcube in phase 2 the following estimate is derived assuming no pipelining of communication actions:

$$t_2'=\left[4\tau+\left(\frac{10}{b}+\frac{17}{s}\right)\frac{n}{\kappa}\right](\log_2 \kappa-1)-2\tau-\frac{5n}{b\kappa}+\frac{n}{s\kappa}.$$

If the two successive communications in a reduction step can be pipelined, then the term $10/b$ can be reduced to $5/b$.

By comparing the time for phase 2 using cyclic reduction on a hypercube to that using Gaussian elimination we conclude that if the communication startup time is high with respect to the elemental transfer time $1/b$ and with respect to the arithmetic processing time $1/s$, then cyclic reduction is preferable to Gaussian elimination. If startups can be ignored, then Gaussian elimination is superior when n is large relative to κ while cyclic reduction is superior when n is small relative to κ. More precisely,

let us assume that $\tau = 0$, and define $\alpha \equiv b/s$, the ratio of communication to arithmetic speeds. Then, by comparing the expressions for t_2' for Gaussian elimination and cyclic reduction, respectively, and by neglecting the lower order terms, we find that the break-even point is realized when n is approximately

$$n(\alpha) = \kappa^2 \Big/ \frac{10+17\alpha}{4+8\alpha} \log_2 \kappa - \frac{19+24\alpha}{4+8\alpha}. \tag{6.1}$$

What is interesting is that the two fractional coefficients in the denominator of the above expression lie in the intervals [2.125, 2.5] and [3, 4.75], respectively, for all values of α between zero and infinity. Therefore, for the sake of simplicity we replace the above expression of $n(\alpha)$ by

$$n(\alpha) \approx \frac{\kappa^2}{2\log_2 \kappa - 4} = \frac{k}{\log_2 k/16}.$$

Note that, conversely, given n the break even number of processors $k(\alpha)$ beyond which cyclic reduction becomes faster than Gaussian elimination is approximately

$$k(\alpha) \approx n \log_2 \left[\frac{n}{16} \log_2 \frac{n}{2} \right],$$

for large n. Thus, under the above assumptions, when $\kappa = 16$, ($k = 256$) cyclic reduction is faster for all values of n not exceeding $n \approx 64$. When $\kappa = 64$, ($k = 4096$) the break-even occurs at $n \approx 512$. Of course, if communication startups were to be included, then the competitiveness of cyclic reduction is further improved.

It should be noticed that with the above simplistic implementation of cyclic reduction the reduction process for all equations converge to the same processor and this leads to a bottleneck. The implementation can be improved by making the reduction process for different equations converge to different processors in a *balanced cyclic reduction* algorithm [12]. That the reduction process can be made to converge to an arbitrary processor is easily seen if, conceptually, a tridiagonal system is extended with the appropriate number of leading, or trailing equations with a corresponding 0 solution (0 off-diagonal elements and right-hand side). For κ tridiagonal systems solved in a subcube of κ nodes the load balancing is perfect, and each node performs the same number of arithmetic operations in the reduction of all κ equations as is required for the reduction of a single equation. Hence, for this improved implementation of cyclic reduction we arrive at the following estimate for the time of one step of the Alternating Direction Method

$$\hat{t}_2 = 2(2\log_2 \kappa - 1)\tau + \left(2\frac{5\kappa - \log_2 \kappa}{b} + \frac{17\kappa - 18\log_2 \kappa + 2}{s} \right) \left[\frac{n}{\kappa^2} \right].$$

The number of startups is the same as in the naive implementation of the cyclic reduction method, but the communication and arithmetic bandwidth requirements are reduced. This improved implementation of cyclic reduction is of lower complexity than Gaussian elimination if $n \leqq \kappa^2$, approximately, ignoring startups. The reason for the advantage of Gaussian elimination over cyclic reduction for $n/\kappa > \kappa$ is due to its lower arithmetic complexity. The solution of a tridiagonal system by cyclic reduction requires approximately twice the number of arithmetic operations of Gaussian elimination, and the sequential nature of Gaussian elimination is outweighed by its arithmetic efficiency for sufficiently many equations per node. The improved implementation of cyclic reduction is competitive for $n \leqq 64$ for $\kappa = 8$, and $n \leqq 1024$ for $\kappa = 32$. Performing a few cyclic reduction steps, a "matrix transpose" to collect all equations for a system

in one processor, solve the reduced systems locally by Gaussian elimination, a new "transpose" to send solution variables back to processors from which the equations were collected followed by cyclic backsubstitution may be optimal on some architectures [12], [7].

Adding the complexities for all three phases (t_1', $\hat{t}_2$, t_3') of one step of the Alternating Direction Method yields

$$T_C(k) = \frac{17n^2}{ks} + 2(2\log_2 \kappa - 1)\tau + \left(2\frac{5\kappa - \log_2 \kappa}{b} + \frac{17\kappa - 18\log_2 \kappa + 2}{s}\right)\left[\frac{n}{\kappa^2}\right] + 2\tau + \frac{n}{\kappa}\frac{5}{b}.$$

The form of $T_C(k)$ is

$$T_C(k) = \alpha\frac{n^2}{k} + \beta\frac{n}{\sqrt{k}} + \gamma\frac{n}{k}\log_2 k + \text{Constant},$$

and its minimum is achieved for $\kappa = n$. For this value of κ

$$T_C \approx \left(4\tau + \frac{10}{b} + \frac{17}{s}\right)\log_2 n.$$

7. Higher-dimensional problems. For the solution of higher-dimensional problems we note that there exists no proximity preserving embedding of an m-dimensional grid in an n-dimensional grid for $m > n$ [19]. Hence, the order of the computational complexity for solving a two-dimensional problem on a linear array cannot be reduced below $O(n)$ by extending the linear array to more than $O(n)$ nodes. Solving an m-dimensional problem with n^m interior grid points, $m \geqq 2$, on a linear array requires a time of order $O(n^{m-1})$.

Similarly, for $m \geqq 3$ and a two-dimensional array of processors, the minimum time is of order $O(n^{m-2})$ and the corresponding number of processors is n^2. In the case of a three-dimensional problem one simple embedding in a two-dimensional array of processors is obtained by identifying all points in one dimension and using the two-dimensional grid embedding described previously. Two of the three computational steps of the Alternating Direction Method for each time step are performed as in the two-dimensional grid case, but repeated n times (for the third dimension). The third step is local to each processor and consists in solving $(n/\kappa)^2$ tridiagonal systems of n equations each. The time for this step is

$$T_G^3 = \frac{8n}{s}\left(\frac{n}{\kappa}\right)^2$$

and the time for each of the other two steps is

$$T_G^1 = T_G^2 = \frac{17n}{s}\left(\frac{n}{\kappa}\right)^2 + \left(n\frac{n}{\kappa} + \kappa - 1\right)\left(2\tau + \frac{4}{b} + \frac{8}{s}\right) + 2\tau + \frac{5n}{b}\frac{n}{\kappa}$$

or

$$T_G^1 = T_G^2 = \frac{17n}{s}\left(\frac{n}{\kappa}\right)^2 + \left(n\frac{n}{\kappa} + \kappa - 1\right)\left(2\tau + \frac{3}{b} + \frac{8}{s}\right) + 3\tau + \frac{3n}{b}\frac{n}{\kappa}$$

assuming that the reduced system is solved by Gaussian elimination. It is clear that for this embedding $\kappa = n$ minimizes T_G^i, $i = \{1, 2, 3\}$.

Higher-dimensional meshes can be embedded in sufficiently large hypercubes preserving proximity by subdividing the cube into m subcubes for an m-dimensional grid. As for the two-dimensional grid the m-dimensional grid is partitioned into regions

and the regions embedded in the hypercube by encoding their indices in a binary-reflected Gray code. With an m-dimensional grid partitioned uniformly into κ partitions in each dimension, the grid having n interior points in each dimension, and the subcubes being of size κ, the complexity estimate for each of the m steps of the Alternating Direction Method is

$$T_C^m = \frac{17}{s}\left(\frac{n}{\kappa}\right)^m + 2(2\log_2 \kappa - 1)\tau + \left(2\frac{5\kappa - \log_2 \kappa}{b} + \frac{17\kappa - 18\log_2 \kappa + 2}{s}\right) \cdot \left[\frac{1}{\kappa}\left(\frac{n}{\kappa}\right)^{m-1}\right] + 2\tau + \frac{5}{b}\left(\frac{n}{\kappa}\right)^{m-1}$$

using cyclic reduction for the solution of the reduced system. The minimum is the same as in the two-dimensional grid case, namely

$$T_c^m \approx \left(4\tau + 2\frac{5}{b} + \frac{17}{s}\right)\log_2 n.$$

8. Conclusion. In this paper we have proposed several implementations of the Alternating Direction Method on multiprocessors. Estimates of the time to execute one half-step of the algorithm lead us to the following conclusions.

- For the shared memory model and broadcast bus model our implementations of the Alternating Direction Method are not highly efficient.
- If $k \leqq n$ then the ring architecture is sufficient to obtain an asymptotically optimal speedup. The optimal time is of order $O(n^{m-1})$ for an m-dimensional grid of $O(n^m)$ interior points. The optimum is attained for n processors.
- In order to reduce the computational complexity further when $k > n$, it is necessary to use an array of processors of a dimensionality closer to that of the problem. For a two-dimensional grid a two-dimensional array of processors suffices to achieve a complexity less than $O(n)$. By using a substructuring technique and solving the reduced system by either Gaussian elimination or cyclic reduction a minimum complexity of order $O(n^{2/3})$ is obtained. The number of processors for which the minimum is attained is $O(n^{4/3})$. The method which has the lowest absolute complexity is dependent on the ratio of the communication and arithmetic bandwidths and the ratio n/κ [11]. For an m-dimensional grid, $m \geqq 3$, the minimum time for a two-dimensional array of processors is of order $O(n^{m-2})$ and the corresponding number of processors $O(n^2)$.
- The hypercube architecture makes it possible to exploit very large scale parallelism in the Alternating Direction Method. One step of ADI on an m-dimensional problem can be performed in time proportional to $\alpha \log_2 n$ by using an n^m-node hypercube. This time is of the order of the lower bound.
- For $k < n$, i.e., when the number of processors is less than the number of grid points in one dimension, the lower arithmetic complexity of Gaussian elimination compared to cyclic reduction may outweigh its inherent sequential nature, since the degree of parallelism is relatively low. For a two-dimensional problem a substructuring algorithm based entirely on Gaussian elimination may be of a lower complexity than combining local Gaussian elimination with "global" cyclic reduction if $n/\kappa > \kappa$. Consequently, a ring or a grid architecture may yield a complexity comparable to that of a hypercube architecture. For higher-dimensional problems the hypercube architecture always yields a lower asymptotic complexity than a ring or a two-dimensional array architecture. The break-even point between Gaussian elimination and a cyclic reduction occurs approximately at $n \approx \kappa^{m/(m-1)}$.

REFERENCES

[1] L. N. BHUYAN AND D. P. AGRAWAL, *Generalized Hypercube and Hyperbus structures for a computer network*, IEEE Trans. Comput., C-33 (1984), pp. 323-333.

[2] T. F. CHAN AND Y. SAAD, *Multigrid algorithms on the hypercube multiprocessor*, Research Report 368, Dept. Computer Science, Yale University, New Haven, CT, 1985.

[3] D. GANNON AND J. VAN ROSENDALE, *On the impact of communication complexity in the design of parallel algorithms*, Technical Report 84-41, Institute for Computer Applications in Science and Engineering, Hampton, VA, 1984.

[4] A. GEORGE, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345-363.

[5] W. D. HILLIS, *The Connection Machine*, MIT Press, Cambridge, MA, 1985.

[6] R. W. HOCKNEY AND C. R. JESSHOPE, *Parallel Computers*, Adam Hilger, Bristol, England, 1981.

[7] S. L. JOHNSSON, C.-T. HO AND F. SAIED, *Solving Multiple Tridiagonal Systems, the Alternating Direction Method, and Boolean Cube configured multiprocessors*, Technical Report YALEU/CSD/RR-552, Dept. of Computer Science, Yale University, New Haven, CT, 1986.

[8] S. L. JOHNSSON, *Solving narrow banded systems on ensemble architectures*, ACM Trans. Math. Software, 11/3 (1985).

[9] ———, *Communication efficient basic linear algebra computations on hypercube architectures*, *J. Parallel Distributed Computing*, 4 (1987), pp. 133-172; (Technical Report YALEU/CSD/RR-361, Dept. of Computer Science, Yale University, New Haven, CT, 1985).

[10] ———, *Data permutations and basic linear algebra computations on ensemble architectures*, Technical Report YALEU/CSD/RR-367, Dept. of Computer Science, Yale University, New Haven, CT, 1985.

[11] ———, *Odd-even cyclic reduction on ensemble architectures*, Technical Report, YALEU/DCS/RR-339, Yale University, New Haven, CT, 1984.

[12] ———, *Solving tridiagonal systems on ensemble architectures*, this Journal, 8 (1986), pp. 354-392; (Technical Report YALEU/CSD/RR-436, Yale University, New Haven, CT, 1985).

[13] J. J. LAMBIOTTE, *The solution of linear systems of equations on a vector computer*, Ph.D. thesis, University of Virginia, Charlottesville, VA, 1975.

[14] D. H. LAWRIE AND C. R. VORA, *The prime memory system for array access*, IEEE Trans. Comput., C-31 (1982), pp. 1435-442.

[15] D. H. LAWRIE, *Access and alignment of data in an array processor*, IEEE Trans. Comput., C-24 (1975), pp. 1145-55.

[16] D. H. LAWRIE AND A. H. SAMEH, *The computation and communication complexity of a parallel banded linear system solver*, ACM Trans. Math. Software, 10 (1984), pp. 185-95.

[17] J. M. ORTEGA AND R. G. VOIGT, *Solution of partial differential equations on vector and parallel computers*, SIAM Rev., 27 (1985), pp. 149-240.

[18] E. M. REINGOLD, J. NIEVERGELT AND N. DEO, *Combinatorial Algorithms*, Prentice-Hall, New York, 1977.

[19] A. L. ROSENBERG, *Preserving proximity in arrays*, SIAM J. Comput., 4 (1975), pp. 443-460.

[20] Y. SAAD AND M. H. SCHULTZ, *Direct parallel methods for solving banded linear systems*, Research Report 387, Dept. Computer Science, Yale University, New Haven, CT, 1985.

[21] ———, *Topological properties of hypercubes*, Research Report 389, Dept. Computer Science, Yale University, New Haven, CT, 1985.

[22] ———, *Alternating direction methods on multiprocessors: an extended abstract*, Research Report 381, Dept. Computer Science, Yale University, New Haven, CT, 1985.

[23] ———, *Data communication in hypercubes*, Research Report 428, Dept. Computer Science, Yale University, New Haven, CT, 1985.

[24] A. H. SAMEH, S. CHEN AND D. KUCK, *Parallel Poisson and biharmonic solvers*, Computing, 17 (1976), pp. 219-230.

[25] A. H. SAMEH AND D. J. KUCK, *On stable parallel linear system solvers*, J. Assoc. Comput. Mach., 25 (1978), pp. 81-91.

[26] C. L. SEITZ, *The cosmic cube*, Comm. ACM, 28 (1985), pp. 22-33.

[27] R. S. VARGA, *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1962.

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

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 5, September 1987
© 1987 Society for Industrial and Applied Mathematics

ON THE PERFORMANCE OF SOME RAPID ELLIPTIC SOLVERS ON A VECTOR PROCESSOR*

J. R. KIGHTLEY† AND C. P. THOMPSON†‡

Abstract. This paper compares the performance of two software packages which use multigrid techniques, together with the preconditioned conjugate gradient method and direct methods, for solving certain classes of elliptic partial differential equations. The advantages and drawbacks of each code are discussed, and indication given of the circumstances in which each type of code will work efficiently on a CRAY-1 computer and on a conventional serial processor.

Key words. elliptic partial differential equations, software performance, vector processing, CRAY-1, multigrid, preconditioned conjugate gradients

AMS(MOS) subject classifications. 65N20, 35J05, 65F10

1. Introduction. Recent years have seen the development of several new approaches to the problem of the solution of elliptic partial differential equations. Much research has been done into the application of multigrid methods to this topic. There are now several pieces of software generally available for the two-dimensional case.

The aim of this paper is to compare some of these algorithms and some of the software. There are many possible bases on which to perform such comparisons: computational complexity, generality of application, ease of use and speed are all common alternatives. It is our intention here to concentrate on speed, and in particular on performance on a vector computer. Our study has been restricted to one machine, the CRAY-1S. Although, of course, comparisons of this nature can extend across very large ranges of algorithms and computers, we feel that this machine is in the mainstream of vector processing development and is sufficiently widely used to make the comparison of general interest.

The current investigation examines four groups of codes: the multigrid codes MGD1V and MGD5V [1], [2]; the multigrid code MG00 [3]; preconditioned conjugate gradient (PCG) algorithms; and two cyclic reduction programs. These include some very general algorithms and some which are very specific in terms of admissible discretizations, user interface, types of domain and boundary conditions.

The test problem selected to measure the speed of these programs is that of Poisson's equation with Dirichlet boundary conditions on a square. The number of boundary points is relatively small and therefore, with proper treatment, the type of boundary conditions will not affect the rate of convergence significantly. Poisson's equation was chosen because it is very important in many fields of mathematical physics and engineering, because it is one of the few cases which all the programs can attempt, and because it is mathematically well behaved. There is an implication of this final point which requires further comment. Each of the codes uses a different algorithm and addresses a different problem class. This means that in some cases extra computations are performed which are redundant for this particular test problem. In other words, the technique is less efficient but more robust than it need be. However, for other types of elliptic problems—giving, for example, anisotropic discretizations

* Received by the editors March 22, 1986; accepted for publication (in revised form) September 25, 1986.
† Computer Science and Systems Division, Harwell Laboratory, UKAEA, Oxfordshire, United Kingdom OX11 0RA.
‡ Present address, Argonne National Laboratory, Argonne, Illinois 60439-4844.

or more complex stencils—the benefits of these more general algorithms become apparent. We have tried to outline some of these distinctions in the next section. This subject is further discussed in [4].

2. Brief description of programs.

2.1. MGD1V/MGD5V. These programs are variants on two members, MGD1 and MGD5, of the family of multigrid codes developed by Wesseling et al. [1], [2] and enhanced for performance on vector processors by de Zeeuw et al. [5], [6]. Extensive tests of MGD1 have been carried out by McCarthy [7]. The MGD codes are intended as "black box" solvers. The user discretizes the differential equation employing a seven-point stencil: the north-west and south-east points are available. Subject to this restriction any discretization can be used, although convergence rates will be affected if an h-ellipticity condition is strongly violated. In particular the class of geometrical domains can be extended by inserting trivial equations, and second-order approximations to mixed derivatives can be incorporated. The resulting arrays of coefficients are then passed to the program, with the user remaining unaware of the underlying multigrid algorithm.

Prolongation is by linear interpolation, and the restriction operator is its adjoint. Coarse grid operators are obtained by a "Galerkin" approximation; that is, the fine grid operator is transferred to coarser grids using the same transfer operators as for residuals, without further reference to the continuous problem. A "sawtooth" iteration strategy is used, whereby the equations are only relaxed on each grid after transfer of the correction from the next coarser grid.

The only major difference between MGD1 and MGD5 is the smoothing algorithm: the former uses a straightforward ILU (incomplete LU) factorisation of the coefficient matrix, while the latter uses an ILLU (incomplete line or incomplete block) factorisation. For anisotropic problems MGD5 can exhibit higher rates of convergence than MGD1, although this depends upon the alignment of the grid in a complex fashion. In the isotropic case MGD1 is slightly more efficient; see [7] for details. Exactly one iteration of the ILU or ILLU smoother is performed on each grid.

The majority of work in multigrid algorithms is spent in the relaxations on the finest grid. Transfers should involve comparatively little arithmetic, and the decrease in the number of unknowns (a factor 4 in two dimensions) means that the relative amount of work on coarse grids is small. Therefore the question of vectorizability of a multigrid algorithm is essentially restricted to consideration of the relaxation schemes. For MGD1 and MGD5 recursions inhibit the vectorization of some sections of the smoothing step.

Principal advantages of these codes are: their efficiency and robustness, the range of applicability (any suitable seven-point star), and the truly "black-box" nature of the modules. A drawback of most multigrid codes, including all those considered here, is a restriction on the number of meshpoints in each dimension on the finest grid. This must equal (small integer) $\times$ (power of two) $+ 1$, where the small integer is the dimension of the coarsest grid. The rate of convergence increases with the number of grids (i.e. the exponent of two).

The performance of these subroutines on other vector computers has been described by Hemker [8].

2.2. MG00. The approach of Foerster and Witsch in their multigrid program MG00 [3] is somewhat different. The user is required to specify the continuous problem by supplying four Fortran subroutines which will evaluate the coefficients and right-hand side of the p.d.e. and boundary conditions at any given point in the domain of

interest. The task of discretizing, as well as that of solving the problem, is therefore transferred to the multigrid program, so it may be a more useful tool from the point of view of economy of the user's time. This approach fits into the high-level framework of the ELLPACK system [9] for solving elliptic partial differential equations, in which package this program is included.

Central differences are used for second order terms in the p.d.e. In Release 1.0 of the code, no mixed derivatives or first-order derivatives are permitted. The form of the most general equation which can be solved is

$$-a(x, y)\frac{\partial^2 u}{\partial x^2} - b(x, y)\frac{\partial^2 u}{\partial y^2} + c(x, y)u = f(x, y). \tag{1}$$

Coarse grid operators are obtained from the continuous problem in the same manner as the fine grid operators.

More control over the algorithm is possible than with the MGD programs. The user must choose between the full multigrid algorithm (FMG) and simple multigrid iterations (MGI). The former algorithm starts with an approximate solution on the coarsest grid and performs a single multigrid iteration on each successively finer grid. Fourth order interpolation (decreasing to third order at boundaries with non-Dirichlet conditions) is used at this stage to obtain an initial fine grid approximation, which means that the truncation error accuracy of the coarser levels should be preserved. Additional multigrid iterations may then be applied to reduce the algebraic error further. When all coefficients are smoothly varying throughout the domain, so that FMG may be expected to work well, the number of relaxation sweeps can be reduced; this feature was not exploited. The user may also select either *V*-type or *W*-type multigrid cycles; *V*-cycles were used for the numerical tests in § 3.

The MG00 program actually comprises a set of three solution modules, two of which have been tailored to deal efficiently with special problems. These modules are:

(i) MG00M1: Poisson's equation and Helmholtz-type equations where $c(x, y)$ is a constant;
(ii) MG00M2: Helmholtz-type equations where c may vary;
(iii) MG00M3: the general case, as defined in (1).

Details of the algorithms used in each of these instances may be found in [3].

For Poisson's equation on a grid with the same mesh spacing in each direction, the module MG00M1 is used. Except on the coarsest grid, smoothing is by point Gauss-Seidel with a red-black (checkerboard) ordering of the gridpoints (RB-GS). This section of the code produces vector loops, the length of which is $\frac{1}{2}n$ (where n is the number of points in one dimension of the current grid). On the coarsest grid a direct solver is used, this being necessary for the solution of indefinite problems where conventional relaxation schemes are inadequate. Normally one relaxation sweep is performed before a coarse grid correction and two sweeps afterwards. Prolongation and restriction are by bilinear interpolation and half weighting respectively. The program eliminates many computations by taking account of the special form of the discrete equations for the Poisson problem, so this module is considerably more efficient than that for more general problems.

In MG00M3, the smoother is one of several forms of zebra line relaxation, which entails the solution of n tridiagonal systems. A zebra ordering decouples these tridiagonal solutions into two sets of size $\frac{1}{2}n$. It would be possible to vectorize across each of these sets by reorganizing the data, but this has not been done. Work is in hand to improve the vector performance of these modules [10], but it falls outside the scope of this comparison. For the test problem in question, alternating zebra (AZ)

relaxation is used. One double relaxation sweep is performed before a coarse grid correction and one double sweep afterwards. Prolongation and restriction are by linear interpolation and full weighting respectively. Of course, AZ relaxation is a very robust method, but computationally expensive. This accounts for the poorer efficiency of this version of the algorithm on the test problem described in § 3.

The main advantage of this program is that it handles more of the process of setting up a problem for solution than the MGD programs do. On the other hand, altering the prescribed choice of grid points and differencing scheme then becomes a major task. This may be a drawback when it is anticipated that a strongly stretched grid is necessary for accurate solution of a particular problem. Other advantages are efficiency on the special cases of Poisson and Helmholtz-type equations and the ability of the user to influence the algorithm (to a small extent) according to whether he anticipates his problem to be easy or difficult. On the negative side, Release 1.0 of the code is severely limited by only being able to treat equations of the form in (1). General anisotropic and self-adjoint problems cannot be tackled. Also there is a restriction on the number of gridpoints on the coarsest grid similar to that for the MGD codes.

2.3. Preconditioned conjugate gradients. As a comparison of these multigrid programs with another popular iterative method, results are also presented for the preconditioned conjugate gradient (PCG) algorithm (many authors: see, for example, Concus, Golub and O'Leary [11]). The method can be applied to problems which give rise to a symmetric and positive definite coefficient matrix, such as arise when the differential operator is in self-adjoint form. A slightly more general case may also be treated; see [12] for a more detailed discussion of the applicability of PCG methods.

Kightley and Jones [13] have shown that preconditioning by the diagonal of the coefficient matrix works well for Poisson's equation, at least for small problems. If the problem has Dirichlet boundary conditions, all terms on the diagonal are the same and diagonal scaling reduces to classical, unpreconditioned CG. It will be denoted "DCG."

For more general problems, preconditioning by an incomplete Cholesky factorization of the coefficient matrix is preferable. Because of storage considerations due to the need to tackle much larger problems than those quoted here, possibly in three dimensions, only the ICCG(0) algorithm (i.e. without fill-in; denoted "ICCG" hereafter) of Meijerink and van der Vorst [14] has been considered. The diagonal of the preconditioning matrix is chosen such that sums of elements by columns are the same in coefficient and preconditioning matrices (sometimes called "modification" and hence termed "MICCG"). The problem was rescaled by this diagonal to eliminate two floating point operations. Eisenstat's implementation [15] was not used, however, because of its greater complexity, its incompatibility with other types of preconditionings and difference molecules, and because for low accuracy the extra cost of initialisation may not be worthwhile.

The ICCG algorithm entails forward and backward substitution with the triangular factors of the preconditioning matrix. If the grid nodes are ordered lexicographically, then a recursion is present which prevents vectorization. A vectorizable variant called Truncated ICCG has been introduced by van der Vorst [16], and is denoted here by "ICCG(T)." The technique used is to expand the scalar recursion in a power series, truncated at the third term. It is assumed that sufficient diagonal dominance is present in the matrix that the neglected terms are small; hence the preconditioning is not degraded by much. However, each scalar operation is replaced by two vector operations. On certain regular problems, the slower convergence is more than offset by a faster

execution speed when compared with the coding of normal ICCG with recursions [13], [16].

The problem of dependencies inhibiting vectorization is directly related to the ordering of the nodes which has been employed. ICCG(T) uses a mathematical technique to circumvent this problem, which is essentially one of data flow. However, the sequence of operations to implement ICCG may be reordered by running along diagonals of the grid, rather than along gridlines. The actual arithmetic is unaltered but the recursion is eliminated, thus the calculations may be vectorized [17]. The results for ICCG in the next section use this diagonal ordering. As demonstrated by these results, the fact that the lengths of vector loops range from n down to 1 does not weigh heavily against this technique, since the CRAY-1S achieves reasonable efficiency even on short loops.

The class of PCG methods has no limitations on grid sizes or on the geometry of the domain. It is much simpler to program, and the work of initialisation is typically much less than for a multigrid method. However, the coefficient matrix must be symmetric and positive definite, or nearly so as mentioned above. There are variants of the algorithm which perform more arithmetic but allow for an unsymmetric matrix, e.g., CG-squared [18], or for indefinite systems, but they are not considered here.

The code to implement PCG is the authors' own test program, which is not yet commercially available. It has been written in standard Fortran, except for two assembler-coded BLAS subroutines [19] which have been used on the CRAY in the vectorized case. These subroutines evaluate the inner product of two vectors (SDOT) and add a multiple of one vector to another (SAXPY). The BLAS are available for a wide range of computers, or equivalent Fortran code may be used at the cost of a modest degradation in speed of execution.

2.4. Cyclic reduction algorithms. For completeness we also present results for two direct methods from Version 3 of the FISHPAK library [20]. This is a package of Fortran subprograms for the solution of separable elliptic partial differential equations.

The program HWSCRT solves the Helmholtz equation on a square. Like MG00 this code addresses the continuous problem (the discretization is by central differences). The program calls the solution module GENBUN, which assumes that the differential equation is Poisson-like in one direction and separable in the other. In other words the discrete equations have the form

$$A_i u_{i-1,j} + B_i u_{i,j} + C_i u_{i+1,j} + u_{i,j-1} - 2u_{i,j} + u_{i,j+1} = Y_{i,j}.$$

The second solver, BLKTRI, is also designed for systems arising from separable elliptic partial differential equations, the discrete equations now taking the form

$$A_i u_{i-1,j} + B_i u_{i,j} + C_i u_{i+1,j} + D_j u_{i,j-1} + E_j u_{i,j} + F_j u_{i,j+1} = Y_{i,j}.$$

Both solvers can deal with combinations of Dirichlet, Neumann and periodic boundary conditions and a wide range of drivers has been provided to perform the discretization and error checking.

The basis of both algorithms is Buneman's Cyclic Reduction Algorithm which is both fast and direct. It is, however, restricted to problems on a logically rectangular domain, and suffers in efficiency if the number of meshpoints is chosen inappropriately, in a similar way to the multigrid codes. The limitation to separable equations is also fundamental to this method; multigrid algorithms are substantially more general, though of course the codes considered in this paper are restricted.

The present version of the cyclic reduction subroutines has not been coded with vector computers in mind, and indeed the gains from vectorization are slight. This is principally because the sequences of tridiagonal systems which arise are solved singly. However, new versions are in preparation which have a high degree of vectorization [21].

3. Numerical tests.

3.1. Test problem. The Dirichlet boundary-value problem for Poisson's equation in 2-D on the unit square was used to assess the performance of the above programs. Results are presented for grid spacings of $h = \frac{1}{32}, \frac{1}{64}$ and $\frac{1}{128}$ in both dimensions. Boundary conditions were eliminated from the discrete equations passed to the PCG routines. The initial guess was zero in all cases, and the solution to the problem used for timing purposes was

$$u = x(1-x)+y(1-y). \tag{2}$$

This problem suffices also to give representative convergence rates for all except MG00. The latter uses high order interpolation for the FMG algorithm; hence the exact solution is obtained from an interpolated coarse grid approximation without need for additional multigrid iterations. Thus more realistic convergence rates for MG00 were obtained from the Poisson problem with solution

$$u = 1 - e^{-10(x+y)}. \tag{3}$$

As convergence rates depend only on the differential operator, these results are directly comparable to those of the other programs, as long as possible differences due to the initial guess are accounted for by considering only residual reductions and not absolute sizes of residuals.

Normally the efficient Poisson solver module of MG00 (namely MG00M1) would be selected automatically. In order to illustrate the efficiency of the program for a more general case, a flag was set to indicate varying coefficients in the differential equation, forcing selection of the MG00M3 module. The coefficients themselves were the same as before.

The test cases were run on the CRAY-1S at Harwell using the CFT compiler version 1.11 with vectorization enabled (vector mode) and disabled (scalar mode). The scalar results should give an indication of relative performance on serial machines. Table 1 displays results for vector mode, and Table 2, those for scalar mode with ratios of scalar to vector times given in brackets. t_0 is the cpu time required for initialisation and t_{iter} the cpu time per iteration; units are milliseconds.

Let $R = \log_{10}(r_0/r)$, where r is the residual L_2-norm at any stage of the calculation, and r_0 is the residual norm due to the initial approximation. R gives an indication of the number of significant figures improvement in the accuracy of the solution. Figures 1-6 show graphs of $(-R)$ plotted against time, since it is more usual to think of a decreasing residual norm. More than sufficient iterations were performed by each iterative method to reduce the residual to rounding error level. Having done this, the required number of iterations to achieve a fixed level in the residual, close to rounding error, was estimated a posteriori, and an average change in R per iteration evaluated. This quantity, which is labelled ΔR_{av} in the tables, is the number of significant figures gained per iteration on average; in other words, the residual is reduced by a factor $10^{-\Delta R_{\text{av}}}$ each iteration.

A useful measure of efficiency is the average slope of a graph of R against cpu time, that is, the number of significant figures gained on average for a unit amount of work (1 cpu second). This quantity is given in the column of the tables labelled dR/dt,

TABLE 1

Timings (ms) *and residual reductions for vector mode. Entries in each row refer to grid spacings of* $\frac{1}{32}$ (*top*), $\frac{1}{64}$ (*middle*) *and* $\frac{1}{128}$ (*bottom*).

Program	t_0	t_{iter}	ΔR_{av}	dR/dt
	15.0	3.73	1.02	274
MGD1V	36.6	12.3	1.02	82.9
	102	44.4	1.02	23.0
	17.7	5.93	1.33	224
MGD5V	44.6	20.7	1.32	64.1
	130	77.0	1.32	17.1
	3.2	2.01	1.47	731
MG00M1(MGI)	10.9	4.85	1.44	300
	40.1	13.2	1.43	109
	15.2	15.4	2.09	136
MG00M3(MGI)	57.7	58.1	2.13	36.7
	226	225	2.16	9.6
	0.5	0.39	.173	444
DCG	1.0	1.54	.087	57
	2.9	6.13	.044	7.2
	2.3	1.17	.391	334
ICCG	7.1	3.47	.265	76.4
	25.4	11.6	.181	15.6
	2.4	1.09	.347	318
ICCG(T)	7.3	3.70	.231	62.4
	26.1	13.9	.157	11.3

and is calculated from

$$\frac{dR}{dt} = \frac{\Delta R_{\text{av}}}{t_{\text{iter}}} \times 1000.$$

As explained in § 2.2, the full multigrid (FMG) algorithm is a fixed set of computations, which may be regarded as an alternative starting procedure for one multigrid iteration. In Table 3 the respective times for MG00 using FMG and the resulting reductions in residual norm (R) due to this improved startup are shown. Further multigrid iterations may then be performed by MG00, yielding residual reduction factors as given in the entries for MG00(MGI) in Tables 1 and 2.

It does not make sense to talk of convergence rates for the direct methods and therefore these do not appear either in Tables 1 and 2 or on the graphs. Instead Table 4 displays the time taken by all the programs to obtain a solution with accuracy only slightly above machine precision; the tolerance corresponds to a residual reduction factor of 10^{-10}. Note that HWSCRT performs a fixed amount of work for all problems, while BLKTRI is called once to factorize the matrix and then may be called many times to solve problems with the same matrix but with different source terms.

3.2. Bases for comparison. In order to assign a ranking among the programs considered in this study, one must decide what constitutes a satisfactory solution to the problem. One could ask for the full precision available from the computer in order to ensure that errors in solving the linear equations are minimised.

TABLE 2

Timings (ms) *and residual reductions for scalar mode. Figures in brackets are ratios of scalar/vector times.*

Program	t_0	t_{iter}	ΔR_{av}	dR/dt
	36.8 (2.5)	10.4 (2.8)	1.02	98.1
MGD1V	125 (3.4)	38.9 (3.2)	1.02	26.2
	454 (4.4)	151 (3.4)	1.02	6.8
	41.8 (2.4)	14.5 (2.5)	1.33	91.7
MGD5V	144 (3.2)	54.9 (2.7)	1.32	24.0
	528 (4.1)	214 (2.8)	1.32	6.2
	3.5 (1.1)	5.30 (2.6)	1.47	277
MG00M1(MGI)	11.7 (1.1)	19.7 (4.1)	1.44	73.1
	43.1 (1.1)	75.8 (5.7)	1.43	18.9
	15.6 (1.0)	20.6 (1.3)	2.09	101
MG00M3(MGI)	59.0 (1.0)	81.3 (1.4)	2.13	26.2
	229 (1.0)	323 (1.4)	2.16	6.7
	1.1 (2.2)	2.50 (6.4)	.173	69.5
DCG	3.2 (3.2)	10.2 (6.6)	.087	8.5
	11.8 (4.1)	41.2 (6.7)	.044	1.1
	8.8 (3.8)	4.62 (3.9)	.391	84.6
ICCG	34.8 (4.9)	18.8 (5.4)	.265	14.1
	139 (5.5)	76.1 (6.6)	.181	2.4
	9.2 (3.8)	5.81 (5.3)	.347	59.7
ICCG(T)	36.3 (5.0)	23.8 (6.4)	.231	9.7
	146 (5.6)	96.1 (6.9)	.157	1.6

However, local truncation error limits the accuracy with which the exact continuous problem may be calculated. So one may argue that iterating beyond the point at which numerical errors are significantly less than truncation error gains nothing. The norm of the local truncation error for the problem in (3) corresponds to a residual reduction factor (R) of 3.8, 4.8 and 5.8 for the three grid sizes in order of decreasing h, when allowance has been made for the scaling of the problem and the normalization of the residuals. The full multigrid scheme aims to reduce numerical errors to this level in one or two cycles. It is possible to see that this is achieved by studying Table 3.

Another possibility arises when the problem being solved is just one step within an outer iteration, for example, the solution of a nonlinear system of equations; in this case only low accuracy may be required. The experience of the authors in a wide range of fluid flow predictions, involving the solution of many Poisson-like problems, is that roughly one order of magnitude reduction in the residual norm is sufficient for convergence of the overall iterative procedure.

The results must therefore be compared with a particular convergence requirement in mind.

3.3. Results from individual classes of programs.

MGD. MGD1V reduces the residual norm by less per iteration than MGD5V, but faster execution speed makes MGD1V the more efficient overall. This is especially true in vector mode, since the ILLU smoother contains more scalar work than the ILU

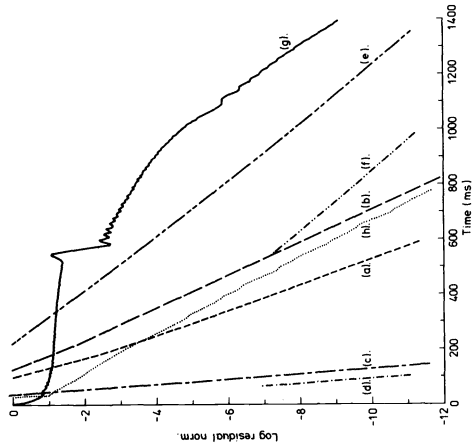


FIG. 3. *Vector mode,* $h = \frac{1}{128}$.

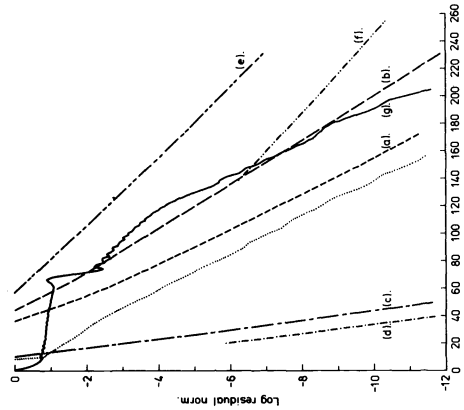


FIG. 2. *Vector mode,* $h = \frac{1}{64}$.

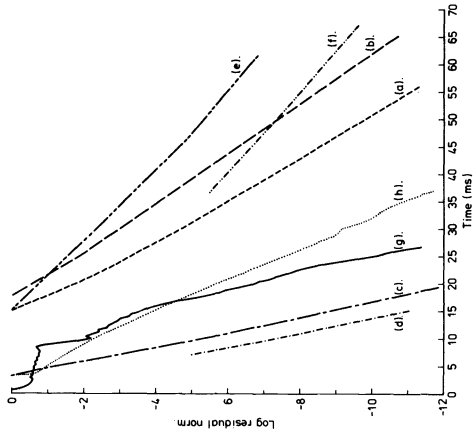


FIG. 1. *Vector mode,* $h = \frac{1}{32}$.

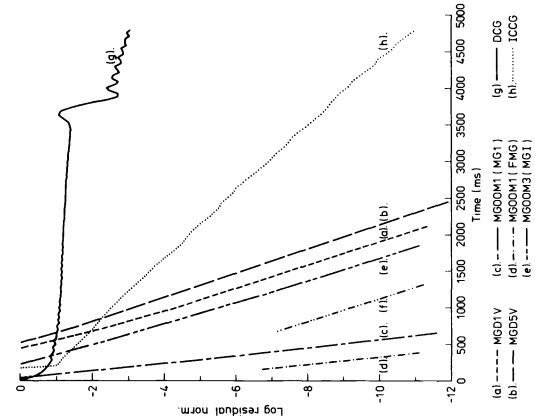


FIG. 6. *Scalar mode,* $h = \frac{1}{128}$.

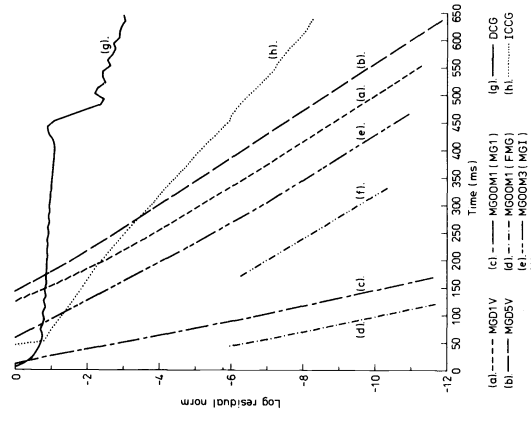


FIG. 5. *Scalar mode,* $h = \frac{1}{64}$.

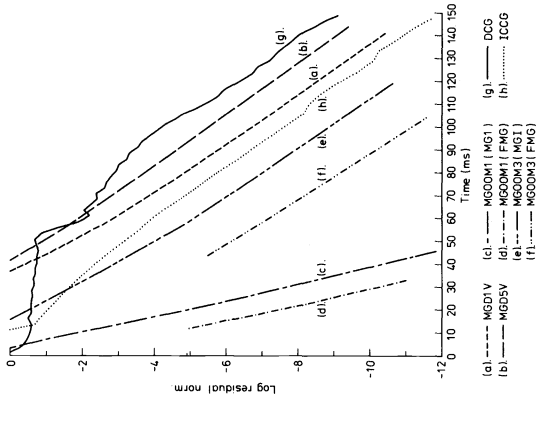


FIG. 4. *Scalar mode,* $h = \frac{1}{32}$.

TABLE 3
Timings (ms) *and residual reductions for* MG00 *using the full multigrid algorithm.*

Program	Times			R
	Vector	Scalar	Ratio	
	7.1	11.8	1.7	4.9
MG00M1(FMG)	20.3	42.1	2.1	5.8
	65.5	159	2.4	6.8
	36.5	43.6	1.2	5.5
MG00M3(FMG)	138	171	1.2	6.2
	536	675	1.3	7.1

smoother. Consequently, MGD1V runs up to 3.4 times faster in vector mode than in scalar mode, while MGD5V is only up to 2.8 times faster. The speedup improves as h decreases because vector loops become longer.

MG00. Much more work is done in MG00M3 than in the specialized Poisson module, MG00M1. Also the RB-GS smoother vectorizes completely, while AZ line relaxation as implemented, involving sequential solution of tridiagonal systems, is all scalar work. This explains the huge difference in t_{iter} between the two cases in vector mode. Since vector loop lengths for RB-GS are half the number of gridpoints in one dimension, the speedup of MG00M1 due to vectorization is only a factor 2.6 for the smallest test problem, but this rises to 5.7 for the largest case. MG00M3 does not vectorize automatically to any significant extent, although some improvement in this respect may be obtained by recoding certain loops [22]. The FMG algorithm for MG00 is highly effective; the work involved is little more than that for a normal setup plus one iteration, yet a large residual reduction results. While this is so for Poisson's equation, more work may be necessary to produce an FMG algorithm which is as profitable when coefficients of the differential equation or boundary conditions vary discontinuously.

PCG. As predicted [13], diagonal scaling (DCG) is effective for the smallest test case in vector mode. In all other cases, though, the residual norm decreases faster and more monotonically for ICCG. DCG is totally vectorized with vector lengths equal to the total number of unknowns (N), so vectorization gives a speedup by a factor of over 6. ICCG uses shorter loops, some very short indeed; hence it is speeded up by a comparable factor only on the largest problem. ICCG(T) is not a worthwhile improvement over ICCG on any of the problems considered, hence the graphs of results omit those for ICCG(T) for clarity. There is more work in the ICCG(T) algorithm than in straightforward ICCG, so it is not sensible even to consider it in scalar mode, though results are included for completeness.

Cyclic reduction. As can be seen from Table 4, the simpler code (HWSCRT) is particularly fast. Comparison of the scalar and vector timings shows that there is very little vectorization occurring automatically.

The results show that ΔR_{av} for multigrid iterations is independent of h, as theory predicts. For DCG, $\Delta R_{\text{av}} = O(h) = O(N^{-1/2})$, so the number of iterations required for a given accuracy is $O(N^{1/2})$. Since the work per iteration is $O(N)$, total work is $O(N^{3/2})$. For ICCG, $\Delta R_{\text{av}} = O(h^{1/2})$, hence the total work is $O(N^{5/4})$, again confirming theoretical results. The work involved in the cyclic reduction routines is $O(N \log N)$.

TABLE 4
Time (ms) *to attain machine accuracy.*

Program	Times		
	Vector	Scalar	Ratio
MGD1V	52.4	141	2.7
	160	514	3.2
	546	1960	3.6
MGD5V	65.2	158	2.4
	210	583	2.8
	746	2240	3.0
MG00M1(MGI)	17.3	40.6	2.3
	44.9	149	3.3
	132	573	4.3
MG00M1(FMG)	15.1	33.0	2.2
	34.9	101	2.9
	105	386	3.7
MG00M3(MGI)	92.1	119	1.3
	348	466	1.3
	1350	1840	1.4
MG00M3(FMG)	82.2	105	1.3
	254	333	1.3
	987	1320	1.3
DCG	25.0	156	6.2
	191	1270	6.6
	1510	10100	6.7
ICCG	32.6	129	4.0
	142	769	5.4
	688	4480	6.5
ICCG(T)	35.2	184	5.2
	170	1080	6.4
	927	6390	6.9
HWSCRT	7.6	9.8	1.3
	33.8	43.0	1.3
	154	192	1.2
BLKTRI (factorize and solve)	29.2	35.9	1.2
	129	162	1.3
	581	737	1.3
BLKTRI (solve only)	19.1	25.2	1.3
	92.4	123	1.3
	449	594	1.3

For sufficiently large N these programs will be slower than the multigrid codes, although it is not clear how large the problems must be before this occurs.

3.4. Comparison of results for different programs. We consider first the total times for convergence close to machine accuracy. Relative to the iterative methods HWSCRT performs well; less so in vector mode since this program is not well vectorized. However, MG00M1(FMG) vectorizes well and performs better for large problems. This can be seen in the case $h = \frac{1}{128}$ in vector mode. The scalar performance of the PCG methods is particularly poor if high accuracy is needed.

If one is interested in solving the equations to the level of truncation error only, then HWSCRT is still competitive. MG00M1(FMG) is clearly highly efficient in obtaining residuals around this level, both in vector and scalar mode and for all grid sizes. It would be possible to eliminate computational work from the other programs in a way similar to that done in MG00M1; this would redress the balance. If results from this fast Poisson solver module of MG00 are set aside as a special case for this particular problem, the remaining results cover a much smaller range. MGD1V is consistently better than MG00M3 in vector mode, because the latter does not vectorize well. In scalar mode there is not much to choose between MGD1V and MG00M3 when performing multigrid iterations, except for the longer set-up time of the former. By using the FMG algorithm, MG00M3 obtains a converged solution significantly more quickly. The PCG methods prove to be better than MGD1V on the smallest test case in vector mode, but performance deteriorates as h decreases.

If low accuracy only is required, those algorithms with the shortest setup times are competitive. For this purpose, for problems of the size considered here, the PCG methods are the cheapest in vector mode and at least as good as any of the multigrid codes in scalar mode, if the fast Poisson solver of MG00 is again ignored.

4. Conclusions. As a result of numerical tests, we conclude that multigrid software which is available now can more efficiently compute accurate solutions to simple elliptic partial differential equations in rectangular domains than an effective and widely-used iterative method (PCG) which does not use multigrid techniques. The multigrid programs which have been considered, namely MG00, MGD1V and MGD5V, all exhibit faster convergence rates for a given amount of work than PCG when large numbers of grid points are used, particularly when vectorization is not available. So if high accuracy is needed in the solution of such a problem, a multigrid method is to be preferred. MG00 is the most efficient on Poisson's equation because of special coding for this particular problem. Excepting this special case, MGD1V is as good as the other modules of MG00, or possibly better because it is better vectorized. On problems where coefficients are smoothly varying, the full multigrid algorithm implemented in MG00 is highly effective; the other multigrid codes do not have this facility.

If, however, the problem yields a symmetric positive definite coefficient matrix and only limited accuracy is required in the solution, PCG with incomplete Cholesky preconditioning is competitive, especially in vector mode and for problems of moderate size. At the other end of the spectrum, if high accuracy is required, the direct methods perform well.

All of the programs have restrictions on the classes of problems which they can tackle, some more stringent than others. Apart from the PCG methods, careful choice of the numbers of grid points to be used is important from the point of view of efficiency.

Availability. Enquiries about the codes described in this paper should be directed to their respective authors. A version of MGD1 will be available in Mk 12 of the NAG Fortran Library as subroutine D03EDF. MG00 is part of the ELLPACK software system. Both FISHPAK codes are available from the National Center for Atmospheric Research, USA, or through NETLIB.

Acknowledgments. The authors are indebted to P. W. Hemker, P. Wesseling and P. M. de Zeeuw for providing them with the MGD1V and MGD5V codes; to K. Solchenbach for supplying the MG00 program; and to P. Swarztrauber for help in obtaining the FISHPAK timings.

The authors are also grateful to I. P. Jones for many helpful comments, and to H. A. van der Vorst for drawing our attention to diagonal orderings for ICCG.

REFERENCES

[1] P. WESSELING, *A robust and efficient multigrid method*, in Multigrid Methods, W. Hackbush and U. Trottenberg, eds., Springer-Verlag, Berlin-New York, 1982.

[2] P. W. HEMKER, R. KETTLER, P. WESSELING AND P. M. DE ZEEUW, *Multigrid methods: development of fast solvers*, Appl. Math. Comput., 13 (1983), pp. 311-326.

[3] H. FOERSTER AND K. WITSCH, *Multigrid software for the solution of elliptic problems on rectangular domains:* MG00 (*Release* 1), GMD-Studien 70, 1982.

[4] J. RUGE AND K. STUBEN, *Efficient solution of finite difference and finite element equations by algebraic multigrid* (AMG), in Multigrid Methods for Integral and Differential Equations, D. J. Paddon and H. Holstein, eds., Oxford University Press, London, 1985.

[5] P. W. HEMKER AND P. M. DE ZEEUW, *Some implementations of multigrid linear system solvers*, in Multigrid Methods for Integral and Differential Equations, D. J. Paddon and H. Holstein, eds., Oxford University Press, London, 1985.

[6] P. W. HEMKER, P. WESSELING AND P. M. DE ZEEUW, *A portable vector-code for autonomous multigrid modules*, in PDE Software: Modules, Interfaces and Systems, B. Enquist and T. Smedsaas, eds., North-Holland, Amsterdam-New York, 1984, pp. 29-40.

[7] G. J. MCCARTHY, *Investigation into the multigrid code*-MGD1, Harwell Lab., AERE-R 10889, 1983.

[8] P. W. HEMKER, *Performance of multigrid software on vector machines*, Supercomputer, 1984.

[9] J. R. RICE AND R. F. BOISVERT, *Solving elliptic problems using* ELLPACK, Springer-Verlag, Berlin-New York, 1985.

[10] M. LEMKE, private communication.

[11] P. CONCUS, G. H. GOLUB AND D. P. O'LEARY, *A generalised conjugate gradient method for the numerical solution of elliptic partial differential equations*, in Sparse Matrix Communications, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1976, pp. 309-332.

[12] V. FABER AND T. MANTEUFFEL, *Necessary and sufficient conditions for the existence of conjugate gradient methods*, SIAM J. Numer. Anal., 21 (1984), pp. 352-362.

[13] J. R. KIGHTLEY AND I. P. JONES, *A comparison of conjugate gradient preconditionings for three-dimensional problems on a* CRAY-1, Comput. Phys. Comm., 37 (1985), pp. 205-214. (Preprint Harwell Lab., AERE-CSS 162).

[14] 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.

[15] S. C. EISENSTAT, *Efficient implementation of a class of preconditioned conjugate gradient methods*, this Journal, 2 (1981), pp. 1-4.

[16] H. A. VAN DER VORST, *A vectorizable variant of some* ICCG *methods*, this Journal, 3 (1982), pp. 350-356.

[17] ———, *The performance of* FORTRAN *implementations for preconditioned conjugate gradients on vector computers*, Parallel Comput., 3 (1986), pp. 49-59.

[18] P. SONNEVELD, P. WESSELING AND P. M. DE ZEEUW, *Multigrid and conjugate gradient methods as convergence acceleration techniques*, in Multigrid Methods for Integral and Differential Equations, D. J. Paddon and H. Holstein, eds., Oxford University Press, London, 1985.

[19] C. LAWSON, R. HANSON, D. KINCAID AND F. KROGH, *Basic linear algebra subprograms for* Fortran *usage*, ACM Trans. Math. Software, 5 (1979), pp. 308–323.

[20] P. N. SWARZTRAUBER AND R. A. SWEET, *Efficient* Fortran *subprograms for the solution of separable elliptic partial differential equations*, [D3], ACM Trans. Math. Software, 5 (1979), pp. 352–364.

[21] R. A. SWEET, private communication.

[22] P. WEIDNER AND B. STEFFEN, *Vektorisierung von mehrgitterverfahren: tests auf einer* CRAY X-MP, KfA Juelich, 1984.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 5, September 1987
© 1987 Society for Industrial and Applied Mathematics
004

SCALED GIVENS ROTATIONS FOR THE SOLUTION OF LINEAR LEAST SQUARES PROBLEMS ON SYSTOLIC ARRAYS*

JESSE L. BARLOW† AND ILSE C. F. IPSEN‡

Abstract. A class of Scaled Givens rotations, to be applied to the solution of weighted multiple linear least squares problems on systolic arrays, is discussed.

In comparison to Fast Givens transformations, properly scaled rotations for weighted problems exhibit the same stability, require fewer divisions, and avoid square roots as well as pivoting. Consequently, with a suitable elimination strategy, the algorithm is amenable to parallel linear-time implementation on systolic arrays in VLSI. Round off error and stability analyses are presented, indicating slightly less accumulation of round off error than known sequential methods.

Key words. least squares computations, Scaled Givens rotations, systolic arrays, VLSI

1. Introduction.

The weighted linear least squares problem is that of finding an $n \times 1$ vector v^* such that

$$\|D(Av^* - s)\| = \min_{v \in R^n} \|D(Av - s)\|, \tag{1.1}$$

where A is an $m \times n$ matrix, s is an $m \times 1$ vector, v is a $n \times 1$ vector, and $D = \operatorname{diag}(d_1, \cdots, d_m)$ is the diagonal $m \times m$ matrix wherein, usually, $d_i > 0$, $1 \leqq i \leqq m$. The symbol $\| \cdot \|$ denotes the Euclidean vector or matrix norm depending on context.

It will be presumed throughout the discussion that $m \geqq n$ (usually $m \gg n$) and that A has rank n to machine precision.

A parallel linear time solution to problem (1.1) will be presented. The triangular decomposition of A and corresponding modifications on s are obtained from a VLSI (very large scale integration) implementation of a systolic array of processors. A systolic device for back substitution can be found in [13].

According to VLSI philosophy, processor interconnections are planar and of nearest neighbour type, while processing elements are primitive, i.e., perform only elementary arithmetic operations. Pipelining replaces global broadcasting of data, so only local exchange of data takes place. Of course, computations are to be evenly distributed over the processors for good hardware utilisation. The above restrictions together with numerical aspects rule out the following methods:

- Normal equations approach: squaring a matrix constitutes an ill-conditioned task, hence, methods must involve orthogonal transformations.
- Householder-Golub factorisation [4]: its global data communication is not suited to implementation in VLSI, rendering a rather inflexible architecture [12].
- Standard Givens rotations: they are easily adapted to systolic arrays [1], [3], [7], [10], but require a square root computation per transformation. Besides consuming a major part of the chip area, they also constitute a severe bottleneck with regard to computation time.
- Fast Givens rotations: they are faster than standard Givens rotations when solving (1.1) sequentially. As the matrix is considered in factored form, square roots are obviated and the number of multiplications is halved. These advantages carry over

* Received by the editors April 3, 1984; accepted for publication (in revised form) December 3, 1984.

† Department of Computer Science, The Pennsylvania State University, University Park, Pennsylvania 16802. This work was supported by the National Science Foundation under contract MCS-8201065 and by the Office of Naval Research under contract N0014-80-0517.

‡ Department of Computer Science, Yale University, New Haven, Connecticut 06520. This work was supported by the Office of Naval Research under contract N000014-82-K-0184.

to parallel devices, as a systolic array implementation [11] makes clear. Yet, unlike standard Givens rotations, the Fast rotations demand "nearest neighbour pivoting" which greatly increases the complexity and area of processing elements. Furthermore, careful monitoring of element growth, important for the prevention of under- and overflow, is not conveniently achieved [8, p. 160].

Consequently, the parallel solution of (1.1) on a VLSI device must rely on the development of Scaled rotations which do without square roots and pivoting. Additionally, the Scaled rotations to be presented here involve only one division as opposed to two for the standard transformations. A decrease in the amount of multiplications is not really crucial if a good hardware utilisation and a balanced distribution of the computation among the processors is achievable.

The Scaled rotations are easily adapted to the systolic structure for the standard [10] and Fast [11] rotations by slight reprogramming of the processing elements (in fact, the scaling technique is best used in a parallel context). The resulting decomposition is as stable as any method available, with an error bound at least as good as one based on standard Givens transformations.

Scaled Givens rotations are most useful for forms of problem (1.1) wherein the weight matrix $W = D^2$ is not the identity matrix, because the premultiplication of A by the square root of the weight matrix, D, is unnecessary. Such problems arise in signal processing and least squares filtering when earlier signals are weighted less compared to later ones. Another context in which Scaled Givens transformations have advantages over standard rotations occurs when a row of the decomposed matrix is to be removed. In this case the weight of the row is set to -1 (cf. [14]). It remains to be investigated how practical these techniques are in a parallel implementation.

There are six sections after this introductory one. Section two contains a review of standard Givens rotations, Fast Givens rotations and a 2×2 version of Bareiss' G-transformation [2]. Section 3, describing and motivating the Scaled rotations, is followed by an error analysis in § 4. Before a short summary in § 7, the suitability of Scaled rotations in a parallel environment is discussed (§ 5) and a systolic VLSI implementation of the triangular decomposition with Scaled rotations is given (§ 6).

2. Standard and Fast Givens rotations. The Givens (or Jacobi) plane rotation is a device for inserting zeros into a matrix selectively. Let $M = (m_{ij})$ be a $2\times n$ matrix. The Givens rotation H that annihilates the $(2, 1)$ entry of $M' = HM$ is given by

$$H = \begin{pmatrix} \gamma & \sigma \\ -\sigma & \gamma \end{pmatrix} \tag{2.1}$$

where

$$\gamma = \frac{m_{11}}{r}, \qquad \sigma = \frac{m_{21}}{r}, \qquad r = \sqrt{m_{11}^2 + m_{21}^2}.$$

Note that

$$\gamma^2 + \sigma^2 = 1,$$

and H is orthogonal. The $2\times n$ matrix $M' = HM$ is computed according to

$$m'_{1j} = \gamma m_{1j} + \sigma m_{2j}, \qquad m'_{2j} = -\sigma m_{1j} + \gamma m_{2j}, \qquad j = 2, \cdots, n, \tag{2.2a}$$

$$m'_{11} = r, \qquad m'_{21} = 0. \tag{2.2b}$$

This computation requires one square root, two divisions, $4n-2$ multiplications, and $2n-1$ additions. Even if the square root were implemented in hardware, it would

still take as long as two or three multiplications, thus contributing significantly to the computation time involved in forming H.

Fast Givens rotations [5], [9] require only half the number of multiplications and avoid the formation of square roots. These operations are saved by considering M in the form

$$M = KB \tag{2.3}$$

where $K = \operatorname{diag}(k_1, k_2)$ is a diagonal matrix and $B = (b_{ij})$. One then determines a transformation G of the form

$$G = (K')^{-1} HK \tag{2.4}$$

where H is given in (2.1) and $K' = \operatorname{diag}(k_1', k_2')$. If

$$B' = GB, \tag{2.5}$$

it is easily seen that

$$K'B' = K'GB = K'(K')^{-1}HKB = HM = M'. \tag{2.6}$$

The saving in multiplications and square roots results from the proper choice of K'. A number of such choices is given in [9]. Gentleman [5] gives the following procedure for forming the matrix G in (2.4). K' is chosen implicitly.

Define

$$W = K^2 = \operatorname{diag}(w_1, w_2) = \operatorname{diag}(k_1^2, k_2^2)$$

and

$$W' = (K')^2 = \operatorname{diag}(w_1', w_2') = \operatorname{diag}((k_1')^2, (k_2')^2).$$

The matrices M' and K' are not explicitly computed, so M and K are not used. Instead one forms G from W and B and computes only W' and B'. Regarding the first column of B, three cases are distinguished when forming G.

Case I. $b_{21} = 0$.

$$W' = W, \quad B' = B, \quad G = I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{2.7}$$

Case II. $w_1 b_{11}^2 \geqq w_2 b_{21}^2 > 0$. Let $t = w_2 b_{21}^2 / w_1 b_{11}^2$; then

$$\begin{aligned} G &= \begin{pmatrix} 1 & w_2 b_{21} / w_1 b_{11} \\ -b_{21}/b_{11} & 1 \end{pmatrix}, \\ B' &= GB, \\ W' &= (1+t)^{-1} W. \end{aligned} \tag{2.8}$$

Note that

$$b_{11}' = b_{11}(1+t), \qquad b_{21} = 0.$$

Case III. $w_2 b_{21}^2 \geqq w_1 b_{11}^2 \geqq 0$. Let $t = w_1 b_{11}^2 / w_2 b_{21}^2$; then

$$\begin{aligned} G &= \begin{pmatrix} w_1 b_{11} / w_2 b_{21} & 1 \\ -1 & b_{21}/b_{11} \end{pmatrix}, \\ B' &= GB, \\ W' &= (1+t)^{-1} \operatorname{diag}(w_2, w_1). \end{aligned} \tag{2.9}$$

Note that

$$b'_{11} = b_{21}(1+t), \qquad b'_{21} = 0.$$

In Cases II and III the computation of GB takes three divisions, $2n+5$ multiplications, and $2n$ additions. The distinction between Cases II and III is essentially a pivot. This "pivot" is necessary since for small $w_1 b_{11}^2$ computations with the matrix G in (2.8) are unstable. The same can be said of the matrix G in (2.9) if $w_2 b_{21}^2$ is small. Unfortunately, this "pivot" adds considerably to the complexity of the processing elements in a parallel device [11].

In a matrix decomposition algorithm based on Fast Givens rotations the matrix to be decomposed must be periodically rescaled. Yet, there is no completely satisfactory method for performing this rescaling.

The Scaled rotations to be discussed are based in part upon the 2×2 version of Bareiss' G-transformation [2],

$$G = \begin{pmatrix} w_1 b_{11}/\rho & w_2 b_{21}/\rho \\ -b_{21}/b_{11} & 1 \end{pmatrix}, \tag{2.10}$$

where $\rho = r^2$, which can have "pivoting" and scaling problems similar to those of Fast Givens rotations.

In the next section, a Scaled Givens rotation is developed in order to prevent problems of pivoting and scaling during parallel computations.

3. Scaled Givens rotations for parallel computation. Before constructing a Scaled Givens rotation, it is necessary to discuss a computational algorithm which, when based on this rotation, solves problem (1.1). Such an algorithm would decompose the $m\times n$ matrix A in (1.1) as follows:

$$A_0 = A, \qquad s_0 = s, \tag{3.1a}$$

$$A_{i+1} = G_{i+1}A_i, \quad s_{i+1} = G_{i+1}s_i, \quad i = 1, \cdots, l-1. \tag{3.1b}$$

$l = mn - (n^2+n)/2$ is the number of rotations necessary to transform A into an upper trapezoidal matrix. Each G_i has the form

$$G_i = D_i^{-1} H_i D_{i-1}, \qquad i = 1, \cdots, l \tag{3.2}$$

with $D_0 = D$ in (1.1), D_i properly chosen diagonal matrices, and H_i Givens rotations, $1 \leqq i \leqq l$.

An important concern for algorithms of the form (3.1) is the growth of elements of A; overflows or spurious underflows are to be avoided.

Define the norm

$$\|S\|_{D_0} \overset{\text{def}}{=} \|D_0 S D_0^{-1}\| \tag{3.3}$$

for an $m\times m$ matrix S. If S is an $m\times n$ matrix where $n<m$ then

$$\|S\|_{D_0} \overset{\text{def}}{=} \|(S,0)\|_{D_0}, \tag{3.4}$$

where $(S,0)$ is an $m\times m$ matrix whose last $m-n$ columns are zero.

After combining (3.1) and (3.2) one obtains

$$\begin{aligned} A_i &= G_i \cdots G_1 A \\ &= (D_i^{-1}H_i D_{i-1})(D_{i-1}^{-1}H_i D_{i-2}) \cdots (D_1^{-1}H_1 D_0)A \\ &= D_i^{-1} H_i \cdots H_1 D_0 A = D_i^{-1} Q_i D_0 A. \end{aligned} \tag{3.5}$$

The matrix Q_i is orthogonal if the effects of machine rounding are ignored. Classical norm inequalities with the norm in (3.3) yield

$$\|A_i\|_{D_0} \leqq \|D_0 D_i^{-1} Q_i\| \|A\|_{D_0} = \|D_0 D_i^{-1}\| \|A\|_{D_0}. \tag{3.6}$$

A similar derivation results in the lower bound

$$\|A_i\|_{D_0} \geqq \|A\|_{D_0} / \|D_i D_0^{-1}\|. \tag{3.7}$$

Thus if $\|D_0 D_i^{-1}\|$ does not grow large, then overflows in A_i are unlikely (they may still occur if D_0 is chosen badly enough, but such cases seldom arise in practice). Spurious underflows are unlikely if $\|D_i D_0^{-1}\|$ does not become too large. The Fast Givens rotation method guarantees that the elements of A_i will not underflow. However, overflow in the elements of A_i is a strong possibility, as is underflow in the elements of D_i. Since the Euclidean norm of $D_i A_i$ is invariant with respect to i, it suffices to monitor the diagonal matrix D_i for underflows. Appropriate procedures are discussed in [9] and [14, p. 62]. Unfortunately, none of them is computationally convenient.

The Scaled Givens rotation of this section is designed to be easily implementable on systolic devices; it rescales subject to the values of $\|D_0 D_i^{-1}\|$ and $\|D_i D_0^{-1}\|$. With the same number of multiplications as in a standard Givens rotation it avoids the pivoting of a Fast Givens rotation. It requires no square roots and only one division—two less than the Fast rotation.

To observe the effect of *one* rotation, let G be the 2×2 nontrivial portion of the Scaled Givens rotation G_i, let K be the portion of D_{i-1} pertaining to the two rows altered by G_i, K' the corresponding portion of D_i, $K^{(0)}$ the corresponding portion of D_0, and B the submatrix of A_{i-1} that is altered by G_i. As in § 2, it is the judicious choice of K' that gives the Scaled Givens rotation its desirable properties.

The 2×2 matrix G for the Scaled Givens rotation is

$$G = (K')^{-1} H K,$$

where H is the standard Givens rotation which inserts a zero in the (2, 1) entry of B. The matrix H has the form

$$H = \begin{pmatrix} \gamma & \sigma \\ -\sigma & \gamma \end{pmatrix},$$

where

$$\gamma = \frac{k_1 b_{11}}{r}, \quad \sigma = \frac{k_2 b_{21}}{r}, \quad r = \sqrt{k_1^2 b_{11}^2 + k_2^2 b_{21}^2}.$$

The choice of K' given by

$$K' = \begin{pmatrix} 2^{-\alpha}/r & 0 \\ 0 & 2^{-\beta} k_1 k_2 / r \end{pmatrix}, \tag{3.8}$$

where α and β are integers chosen for scaling purposes, yields the matrix

$$\begin{aligned} G = (K)^{-1} H K &= \begin{pmatrix} r2^{\alpha} & 0 \\ 0 & r2^{\beta}/k_1 k_2 \end{pmatrix} \begin{pmatrix} k_1 b_{11}/r & k_2 b_{21}/r \\ -k_2 b_{21}/r & k_1 b_{11}/r \end{pmatrix} \begin{pmatrix} k_1 & 0 \\ 0 & k_2 \end{pmatrix} \\ &= \begin{pmatrix} 2^{\alpha} k_1^2 b_{11} & 2^{\alpha} k_2^2 b_{21} \\ -2^{\beta} b_{21} & 2^{\beta} b_{11} \end{pmatrix}. \end{aligned}$$

Thus,

$$G = \begin{pmatrix} 2^{\alpha} w_1 b_{11} & 2^{\alpha} w_2 b_{21} \\ -2^{\beta} b_{21} & 2^{\beta} b_{11} \end{pmatrix}. \tag{3.9}$$

If

$$B' = GB,$$

then

$$b'_{11} = 2^{\alpha} r^2, \qquad b'_{21} = 0.$$

As with Fast Givens rotations, the matrix K is not used, and the matrix K' is not explicitly computed. Instead $W = K^2 = \operatorname{diag}(w_1, w_2) = \operatorname{diag}(k_1^2, k_2^2)$ serves to determine

$$W' = (K')^2 = \operatorname{diag}(w'_1, w'_2) = \begin{pmatrix} 2^{-2\alpha}/r^2 & 0 \\ 0 & 2^{-2\beta} k_1^2 k_2^2 / r^2 \end{pmatrix} = \begin{pmatrix} 2^{-2\alpha}/\rho & 0 \\ 0 & 2^{-2\beta} w_1 w_2 / \rho \end{pmatrix} \tag{3.10}$$

and $\rho = r^2 = w_1 b_{11}^2 + w_2 b_{21}^2$. Thus throughout (3.1), D_i^2 rather than D_i is stored.

The integers α and β are chosen according to the algorithm below, where

$$W^{(0)} = \operatorname{diag}(w_1^{(0)}, w_2^{(0)}) = (K^{(0)})^2 = \operatorname{diag}((k_1^{(0)})^2, (k_2^{(0)})^2).$$

ALGORITHM 3.1.

1. Find the integer α such that $w_1^{(0)} \rho 2^{2\alpha} \in [\frac{1}{2}, 2)$ {This insures that $w_1^{(0)}/w'_1 \in [\frac{1}{2}, 2)$}.
2. Compute $\zeta = 1/\rho$ {This is the only division necessary}.
3. Compute $\xi = w_1 w_2 \zeta$.
4. Subtract the floating point exponent of ξ from that of $w_2^{(0)}$. Call this integer ω.
5. If ω is even then $\beta = \omega/2$, otherwise $\beta = (\omega + 1)/2$ {This insures that $w_2^{(0)}/w'_2 \in [1/4, 2)$}.

These values of α and β ascertain

$$\| W^{(0)} (W')^{-1} \| \leqq 2$$

and

$$\| W' (W^{(0)})^{-1} \| \leqq 4.$$

Therefore,

$$\| K^{(0)} (K')^{-1} \| \leqq \sqrt{2} \tag{3.11}$$

and

$$\| K' (K^{(0)})^{-1} \| \leqq 2. \tag{3.12}$$

Since

$$\| D_0 D_i^{-1} \| \leqq \max \{ \| D_0 D_{i-1}^{-1} \|, \| K^{(0)} (K')^{-1} \| \}$$

and

$$\| D_i D_0^{-1} \| \leqq \max \{ \| D_{i-1} D_0^{-1} \|, \| K' (K^{(0)})^{-1} \| \},$$

a simple induction argument renders the inequalities

$$\| D_0 D_i^{-1} \| \leqq \sqrt{2}, \qquad i = 1, 2, \cdots, l, \tag{3.13a}$$

$$\| D_i D_0^{-1} \| \leqq 2, \qquad i = 1, 2, \cdots, l. \tag{3.13b}$$

Combining (3.13a, b) with (3.6) and (3.7) yields

$$\tfrac{1}{2} \| A \|_{D_0} \leqq \| A_i \|_{D_0} \leqq \sqrt{2} \| A \|_{D_0}, \qquad i = 1, 2, \cdots, l.$$

Hence, the method based on (3.1) using the Scaled Givens rotations is resistant to underflows and overflows; its advantages in a parallel implementation are discussed in § 5.

In the next section, a parallel algorithm for solving problem (1.1) using Scaled rotations is proved to be reasonably stable, and as accurate as extant methods.

4. Error analysis. The following algorithm will be used to solve (1.1):

$$A_{l_0} = A, \qquad s_{l_0} = s, \tag{4.1a}$$

$$A_{l_{k+1}} = \Gamma_{k+1} A_{l_k}, \qquad s_{l_{k+1}} = \Gamma_{k+1} s_{l_k}, \qquad k = 0, 1, \cdots, \nu - 1, \tag{4.1b}$$

where the Greek equivalent of G,

$$\Gamma_k = G_{l_k} \cdots G_{l_{k-1}+1},$$

denotes the product of the disjoint Scaled Givens rotations G_i, $i = l_{k-1}+1, \cdots, l_k$ (two Givens rotations are disjoint if they affect disjoint pairs of rows). The indices l_k, $k = 1, 2, \cdots, \nu$, denote the number of Scaled Givens rotations performed at the kth stage. The indices $l_0 = 0$ and $l_\nu = l$ are the upper and lower limits of equations (3.1). Each G_i has the form (3.9). Equations (4.1) are simply a special version of equations (3.1). Therefore,

$$\begin{aligned} \Gamma_k &= (D_{l_k}^{-1} H_{l_k} D_{l_k - 1})(D_{l_k - 1}^{-1} H_{l_k - 1} D_{l_k - 2}) \cdots (D_{l_{k-1}+1}^{-1} H_{l_{k-1}+1} D_{l_{k-1}}) \\ &= D_{l_k}^{-1} Q_k D_{l_{k-1}}, \qquad k = 1, 2, \cdots, \nu, \end{aligned}$$

where $Q_k = H_{l_k} \cdots H_{l_{k-1}+1}$ is orthogonal. A simpler form for Γ_k is given by

$$\Gamma_k = D_{l_k}^{-1} Q_k D_{l_{k-1}}, \qquad k = 1, 2, \cdots, \nu. \tag{4.2}$$

Presumably, the final matrix A_{l_ν} has the form

$$A_{l_\nu} = \begin{pmatrix} R \\ 0 \end{pmatrix},$$

where R is an $n \times n$ nonsingular and upper triangular matrix. Neglecting rounding errors, the algorithm guarantees

$$\| D(Av - s) \| = \| D_{l_\nu}(A_{l_\nu} v - s_{l_\nu}) \|.$$

If s_{l_ν} is partitioned according to

$$s_{l_\nu} = \begin{pmatrix} s_{l_\nu}^{(1)} \\ s_{l_\nu}^{(2)} \end{pmatrix},$$

where $s_{l_\nu}^{(1)}$ is an $n \times 1$ and $s_{l_\nu}^{(2)}$ an $(m-n) \times 1$ vector, then the solution v^* to (1.1) is given by

$$v^* = R^{-1} s_{l_\nu}^{(1)}$$

with the residual

$$\| D_{l_\nu}(A_{l_\nu} v^* - s_{l_\nu}) \| = \| D_{l_\nu}(0, -s_{l_\nu}^{(2)})^T \|.$$

In order to estimate the effect of rounding errors in (4.1) on the solution, (1.1) is posed as an unweighted least squares problem and the effect of the algorithm on the solution to that problem will be considered. With

$$\hat{A} = \hat{A}_0 = DA, \qquad \hat{s} = \hat{s}_0 = Ds,$$

and

$$\hat{A}_k = D_{l_k} A_{l_k}, \quad \hat{s}_k = D_{l_k} s_{l_k}, \quad k = 1, 2, \cdots, \nu,$$

(4.1), in effect, performs the sequence of operations

$$\hat{A}_0 = \hat{A} = DA, \qquad \hat{s}_0 = \hat{s} = Ds, \tag{4.3a}$$

$$\hat{A}_{k+1} = Q_{k+1}\hat{A}_k, \qquad \hat{s}_{k+1} = Q_{k+1}\hat{s}_k, \qquad k = 1, 2, \cdots, \nu \tag{4.3b}$$

to solve the problem of finding v^* such that

$$\|\hat{A}v^* - \hat{s}\| = \min_{v \in R^n} \|\hat{A}v - \hat{s}\|. \tag{4.4}$$

If the error is analysed for the sequence of orthogonal transformations in (4.3), rather than the Scaled rotations (4.1), the results on error analyses of orthogonal transformations in [5], [6], [16] can be directly applied.

The classical techniques of backward error analysis from [16] will be employed to bound $\|u^* - v^*\|$, where u^* satisfies the perturbed problem

$$\|(\hat{A} + \delta\hat{A})u^* - (\hat{s} + \delta\hat{s})\| = \min_{u \in R^n} \|(\hat{A} + \delta\hat{A})u - (\hat{s} + \delta\hat{s})\|. \tag{4.5}$$

In order to obtain this bound four quantities must be estimated:

- the condition number $\|\hat{A}\|\|\hat{A}^\dagger\|$ where $\hat{A}^\dagger$ is the Moore-Penrose inverse of $\hat{A}$,
- the norm of the residual $\|\hat{A}v^* - \hat{s}\|$,
- the norm of the backward error in $\hat{A}$, $\|\delta\hat{A}\|$,
- the norm of the backward error in $\hat{s}$, $\|\delta\hat{s}\|$.

The first two quantities are dependent on the problem (4.4), not on the algorithm used to solve it. The last two quantities depend upon the algorithm and the precision of the underlying arithmetic. The relation between these four quantities and the forward error $\|u^* - v^*\|$ is given in [14, p. 50].

Thus it is enough to bound the quantities $\|\delta\hat{A}\|$ and $\|\delta\hat{s}\|$ to show that the Scaled Givens rotations produce an algorithm that is as stable as any available.

The first theorem discusses the rounding error from a single Scaled Givens rotation. The symbol $fl(\cdot)$ will denote the floating point computation performed on its argument (for the use of this convention, see [16]).

THEOREM 4.1. *Let G be the 2×2 Scaled Givens rotation defined by the first column of the $2 \times n$ matrix B and the weight matrix $W = \operatorname{diag}(w_1, w_2)$; and $M = KB$ where $K = W^{1/2} = \operatorname{diag}(k_1, k_2)$. Assume G is given by*

$$G = (K')^{-1}HK,$$

where H is the standard Givens rotation defined by the first column of M; and $K' = (W')^{1/2}$ is the new weight matrix defined by (3.8). *If $\tilde{B}'$ is the machine computation of GB and $\tilde{M}' = \tilde{K}'\tilde{B}'$, then*

$$\tilde{M}' + \delta M' = \tilde{K}'\tilde{B}' + \delta M' = HKB = HM$$

where $\|\delta M'\|_F \leqq 5.5\mu\|M\|_F + O(\mu^2)$ and μ is the machine unit.

Two lemmata are needed for the proof of the theorem. The first bounds the perturbations in the weight matrix K' while the second bounds the perturbations in the Givens rotation matrix H.

LEMMA 4.1. *Let K' and W' be defined as in Theorem* 4.1 *and $\tilde{K}' = (\tilde{W}')^{1/2}$, where $\tilde{W}'$ is the machine computation of W'. Then*

$$\tilde{K}' - K' = (\delta K')K',$$

and $\|\delta K'\| \leqq 2.5\mu + O(\mu^2)$.

Proof. The first order Taylor expansion

$$\sqrt{1+\varepsilon} = 1 + \frac{\varepsilon}{2} + O(\varepsilon^2)$$

for ε small is used to bound the error in

$$\tilde{K}' = \begin{pmatrix} fl(k_1') & 0 \\ 0 & fl(k_2') \end{pmatrix},$$

where

$$fl(k_1') = \sqrt{fl(w_1')}, \qquad fl(k_2') = \sqrt{fl(w_2')}$$

since instead of k_1' and k_2', w_1' and w_2' are actually computed.

Performing the computations according to Algorithm 3.1,

$$fl(w_1') = 2^{-2\alpha} fl(1/(w_1 b_{11}^2 + w_2 b_{21}^2)) = w_1'(1+\delta w_1'),$$

where $|\delta w_1'| \leqq 3\mu + O(\mu^2)$ and

$$fl(w_2') = 2^{-2\beta} fl(w_1 w_2/\sigma) = w_2'(1+\delta w_2'),$$

where $|\delta w_2'| \leqq 5\mu + O(\mu^2)$. With the above Taylor expansion of $\sqrt{1+\varepsilon}$ one has

$$fl(k_1') = k_1'(1+\delta k_1'), \qquad |\delta k_1'| \leqq 1.5\mu + O(\mu^2),$$
$$fl(k_2') = k_2'(1+\delta k_2'), \qquad |\delta k_2'| \leqq 2.5\mu + O(\mu^2).$$

This gives

$$\tilde{K}' = (I + \delta K')K', \qquad \|\delta K'\| \leqq 2.5\mu + O(\mu^2)$$

which is the desired result. □

LEMMA 4.2. *Let G, K, and K' be as defined in Theorem* 4.1 *and let $\tilde{G}$ be the machine computation of G. Then*

$$\|K'(G-\tilde{G})K^{-1}\|_F \leqq \mu.$$

Thus also

$$\|K'\tilde{G}K^{-1}\| \leqq 1+\mu.$$

Proof.

$$G = \begin{pmatrix} 2^{\alpha} w_1 b_{11} & 2^{\alpha} w_2 b_{21} \\ -2^{\beta} b_{21} & 2^{\beta} b_{11} \end{pmatrix} = \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix}$$

gives

$$\tilde{G} = fl(G) = \begin{pmatrix} \tilde{g}_{11} & \tilde{g}_{12} \\ \tilde{g}_{21} & \tilde{g}_{22} \end{pmatrix},$$

where

$$\tilde{g}_{11} = 2^{\alpha} w_1 b_{11}(1+\varepsilon_{11}), \qquad \tilde{g}_{12} = 2^{\alpha} w_2 b_{21}(1+\varepsilon_{12}),$$
$$\tilde{g}_{21} = -2^{\beta} b_{21} = g_{21}, \qquad \tilde{g}_{22} = g_{22}.$$

The multiplications by 2^{α} and 2^{β} are just shifts. They cause no rounding error. Hence $\tilde{g}_{21}$ and $\tilde{g}_{22}$ are exact. Now

$$K'\tilde{G}K^{-1} = \begin{pmatrix} \tilde{\gamma} & \tilde{\sigma} \\ -\sigma & \gamma \end{pmatrix} = \begin{pmatrix} \gamma(1+\varepsilon_{11}) & \sigma(1+\varepsilon_{12}) \\ -\sigma & \gamma \end{pmatrix},$$

where

$$\gamma = \frac{k_1 b_{11}}{r}, \qquad \sigma = \frac{k_2 b_{21}}{r},$$

and

$$r = \sqrt{w_1 b_{11}^2 + w_2 b_{21}^2}.$$

Note again that the bottom row of $K'\tilde{G}K^{-1}$ is exact. Thus

$$\|K'(\tilde{G}-G)K^{-1}\|_F = \left\| \begin{pmatrix} \gamma\varepsilon_{11} & \sigma\varepsilon_{12} \\ 0 & 0 \end{pmatrix} \right\|_F \leqq \mu,$$

and

$$\|K'\tilde{G}K^{-1}\| \leqq \|K'GK^{-1}\| + \|K'(\tilde{G}-G)K^{-1}\| = \|H\| + \|K'(\tilde{G}-G)K^{-1}\| \leqq 1+\mu. \quad \square$$

It is now possible to prove Theorem 4.1.

Proof. By definition,

$$\begin{aligned} \|\delta M'\|_F &= \|\tilde{M}' - HM\|_F = \|\tilde{K}'\tilde{B}' - K'B'\|_F = \|\tilde{K}'fl(\tilde{G}B) - K'GB\|_F \\ &= \|\tilde{K}fl(\tilde{G}B) - \tilde{K}'\tilde{G}'B + \tilde{K}'\tilde{G}'B - K'\tilde{G}B + K'\tilde{G}B - K'GB\|_F \qquad (4.6) \\ &\leqq \|\tilde{K}'fl(\tilde{G}B) - \tilde{K}'\tilde{G}B\|_F + \|(\tilde{K}'-K')\tilde{G}B\|_F + \|K'(\tilde{G}-G)B\|_F. \end{aligned}$$

The last two terms of (4.6) can be bounded using Lemmata 4.1 and 4.2. The second term has the bound

$$\begin{aligned} \|(\tilde{K}'-K')GB\|_F &= \|(\delta K')K'G'B\|_F \leqq \|\delta K'\|\,\|K'\tilde{G}B\|_F \\ &= \|\delta K'\|\,\|K'\tilde{G}K^{-1}M\|_F \leqq \|\delta K'\|\,\|K'\tilde{G}K^{-1}\|\,\|M\|_F. \end{aligned}$$

With Lemmata 4.1 and 4.2,

$$\|(\tilde{K}'-K')\tilde{G}B\|_F \leqq 2.5\mu(1+\mu)\|M\|_F + O(\mu^2) = 2.5\mu\|M\|_F + O(\mu^2).$$

The third term is bounded by

$$\begin{aligned} \|K'(\tilde{G}-G)B\|_F &= \|K'(\tilde{G}-G)K^{-1}M\|_F \\ &\leqq \|K'(\tilde{G}-G)K^{-1}\|_F\|M\|_F = \mu\|M\|_F. \end{aligned}$$

It now suffices to bound the first term of (4.6), whereby the following inequality will be useful:

$$\begin{aligned} \|\tilde{K}'(fl(\tilde{G}B) - GB)\|_F &\leqq (\|K'\| + \|\delta K'\|)\|fl(\tilde{G}B) - GB\|_F \\ &\leqq (1+2.5\mu)\|K'fl(\tilde{G}B) - GB\|_F \qquad (4.7) \\ &= (1+2.5\mu)\|K'(fl(\tilde{G}B) - GB)\|_F. \end{aligned}$$

Let

$$\tilde{M}' = (\tilde{m}_1', \tilde{m}_2', \cdots, \tilde{m}_n'), \qquad M = (m_1, m_2, \cdots, m_n),$$

$$\tilde{B}' = (\tilde{b}_1', \tilde{b}_2', \cdots, \tilde{b}_n'), \qquad B = (b_1, b_2, \cdots, b_n)$$

where $\tilde{m}_i'$, m_i, $\tilde{b}_i'$, and b_i, $1 \leqq i \leqq n$, are 2×1 column vectors. The first column of $\tilde{B}'$ is computed in a different manner from columns $2, \cdots, n$ since it defines the transformation G.

Let

$$\tilde{b}_1' = \begin{pmatrix} \tilde{b}_{11}' \\ 0 \end{pmatrix} = fl(\tilde{G}b_1).$$

Then

$$\tilde{b}_{11}' = fl(2^{\alpha}w_1 b_{11}^2 + 2^{\alpha}w_2 b_{21}^2) = 2^{\alpha} fl(w_1 b_{11}^2 + w_2 b_{21}^2).$$

Assume that the computations in $\tilde{b}_{11}'$ are associated as follows:

$$(w_1 b_{11})b_{11} + (w_2 b_{21})b_{21}.$$

This is a sensible assumption since $w_1 b_{11}$ and $w_2 b_{21}$ must be computed to form G.

In that case,

$$\tilde{b}_{11}' = fl(\tilde{g}_{11}b_{11} + \tilde{g}_{12}b_{21}) = \tilde{g}_{11}b_{11}(1+\eta_1) + \tilde{g}_{12}b_{21}(1+\eta_2),$$

where $|\eta_1|, |\eta_2| \leqq 2\mu + O(\mu^2)$. As

$$0 = (\tilde{g}_{21} \quad \tilde{g}_{22}) \begin{pmatrix} b_{11} \\ b_{21} \end{pmatrix}$$

is an exact relation, there is no error in the insertion of the zero. Thus,

$$fl(\tilde{G}b_1) = \begin{pmatrix} \tilde{g}_{11}(1+\eta_1) & \tilde{g}_{12}(1+\eta_2) \\ \tilde{g}_{21} & \tilde{g}_{22} \end{pmatrix} b_1$$

which implies

$$K' fl(\tilde{G}K^{-1}m_1) = \begin{pmatrix} \tilde{\gamma}(1+\eta_1) & \tilde{\sigma}(1+\eta_2) \\ -\sigma & \gamma \end{pmatrix} m_1,$$

and the annihilation of the $(2, 1)$ entry of M is exact. Hence,

$$K'(fl(\tilde{G}K^{-1}m_1) - \tilde{G}K^{-1}m_1) = \begin{pmatrix} \tilde{\gamma}\eta_1 & \tilde{\sigma}\eta_2 \\ 0 & 0 \end{pmatrix} m_1,$$

from which it can be concluded that

$$\begin{aligned} \|K'(fl(\tilde{G}K^{-1}m_1) - \tilde{G}^{-1}m_1)\| &\leqq 2\mu \|K'\tilde{G}K^{-1}\| \|m_1\| + O(\mu^2) \leqq 2\mu(1+\mu)\|m_1\| + O(\mu^2) \\ &= 2\mu\|m_1\| + O(\mu^2). \end{aligned}$$

For the remaining columns, $\tilde{G}b_k$, $1 \leqq k \leqq n$, is just a matrix vector multiplication. Therefore,

$$fl(\tilde{G}b_k) = \begin{pmatrix} \tilde{g}_{11}(1+\eta_{11}) & \tilde{g}_{12}(1+\eta_{12}) \\ \tilde{g}_{21}(1+\eta_{21}) & \tilde{g}_{22}(1+\eta_{22}) \end{pmatrix} b_k,$$

where $|\eta_{ij}| \leqq 2\mu + O(\mu^2)$, $1 \leqq i, j \leqq 2$. The η_{ij} depend on b_k. Consequently,

$$K' fl(\tilde{G}K^{-1}m_k) = \begin{pmatrix} \tilde{\gamma}(1+\eta_{11}) & \tilde{\sigma}(1+\eta_{12}) \\ -\sigma(1+\eta_{21}) & \gamma(1+\eta_{22}) \end{pmatrix} m_k,$$

implying

$$\|K'(fl(\tilde{G}K^{-1}m_k) - \tilde{G}K^{-1}m_k)\| \leqq 2\mu \|K'\tilde{G}K^{-1}\| \|m_k\| + O(\mu^2).$$

From

$$\|K'\tilde{G}K^{-1}\| \leqq 1 + \mu$$

it follows that

$$\|K'(fl(\tilde{G}K^{-1}m_k)-\tilde{G}K^{-1}m_k)\| \leqq 2\mu(1+\mu)\|m_k\|+O(\mu^2)$$
$$=2\mu\|m_k\|+O(\mu^2), \qquad k=2,\cdots,n.$$

Therefore,

$$\|K'(fl(\tilde{G}K^{-1}M)-\tilde{G}K^{-1}M)\|_F = \|K'(fl(\tilde{G}B)-\tilde{G}B)\|$$
$$\leqq 2\mu\|M\|_F+O(\mu^2)$$

which along with (4.7) gives

$$\|\tilde{K}'(fl(\tilde{G}B)-\tilde{G}B)\|_F \leqq (1+2.5\mu)2\mu\|M\|_F+O(\mu^2)=2\mu\|M\|_F+O(\mu^2).$$

Employing the inequality in (4.6), one has

$$\|\delta M'\|_F=\|\tilde{M}'-HM\|_F \leqq 5.5\mu\|M\|_F+O(\mu^2). \qquad \square$$

The following two lemmata are adapted from [6]; they allow the construction of a bound for the backward error $\|\delta\hat{A}\|_F$ in (4.5).

LEMMA 4.3. *Let H be a Givens transformation rotating a $2\times n$ matrix M and implemented in such a way that*

$$\|fl(HM)-HM\|_F \leqq \tau\|M\|_F+O(\mu^2),$$

where μ is the machine unit and $\tau=O(\mu)$. Then a transformation Q made up of $k\geqq 1$ disjoint Givens transformations implemented in the same way and applied to $2k$ rows selected from the $m\times n$ matrix $\hat{A}$ will also satisfy

$$\|fl(Q\hat{A})-Q\hat{A}\|_F \leqq \tau\|\hat{A}\|_F+O(\mu^2).$$

In particular, this bound is independent of k and m.

LEMMA 4.4. *If a sequence of Givens transformations can be written as a sequence of ν stages, where each stage consists of the simultaneous application of disjoint Givens transformations, then the final computed matrix obtained when this sequence of Givens transformations is applied to a given matrix $\hat{A}$ will be the exact transformation of $\hat{A}+\delta\hat{A}$ where*

$$\|\delta\hat{A}\|_F \leqq \tau\nu(1+\tau)^{\nu-1}\|\hat{A}\|_F+O(\mu^2).$$

THEOREM 4.2. *The algorithm specified by* (4.1) *obtains an approximate solution to problem* (4.4) *which solves* (4.5) *exactly for some backward error matrix $\delta\hat{A}$ and backward error vector $\delta\hat{s}$ such that*

$$\|\delta\hat{A}\| \leqq \|\delta\hat{A}\|_F \leqq 5.5\mu\nu(1+5.5\mu)^{\nu-1}\|\hat{A}\|_F+O(\mu^2), \tag{4.8}$$

$$\|\delta\hat{s}\| \leqq 5.5\mu\nu(1+5.5\mu)^{\nu-1}\|\hat{s}\|+O(\mu^2). \tag{4.9}$$

Proof. Simply combine the results of Theorem 4.1 and Lemmata 4.3 and 4.4. $\square$

The factor $(1+5.5\mu)^{\nu-1}$ results from the fact that the computed transformations Q_k may not be exactly orthogonal. However, as pointed out by Wilkinson [16, p. 138] this factor is seldom important enough to affect the stability.

5. The advantages of Scaled rotations for systolic architectures. The systolic processor array, designed for implementation in silicon with VLSI (very large scale integration), solves multiple linear least squares problems, where $z\geqq 1$ different

right-hand side vectors are given,

$$\|D(Av_h^* - s_h)\| = \min_{v_h \in R^n} \|D(Av_h - s_h)\|, \qquad 1 \leqq h \leqq z.$$

Constraints imposed by VLSI technology and fabrication demand regular (and if possible planar) processor interconnections as well as processor communication on a nearest neighbour basis. The obvious choice of structure is a rectangular array of synchronously operating processors, a so-called systolic array. Furthermore, to keep the chip area minimal, processors should perform no other than the elementary operations: addition, multiplication and division.

The constraint of *local* data exchange suggests a combination of pipelining and multiprocessing to achieve good processor utilisation and speed up in computation time. Pipelining is efficient for long matrices which often possess a narrow dense band. The hardware should reflect the features of the matrix: a processor count proportional to the bandwidth, not the order of the matrix. Consequently, input and output occur by codiagonals rather than rows or columns.

Proper scaling procedures for Fast Givens transformations have been a persistent problem in their implementation; continuous monitoring of the growth of the weight matrix elements has, in the sequential case, led to slower and more complicated implementations [8, p. 160]. On account of the "global" nature of the monitoring process its implementation would be inefficient in a parallel environment as well.

Moreover, although the "pivoting" process for Fast rotations requires simple pointer manipulation on a sequential processor, it greatly increases the complexity of processing elements in a systolic array. As each processor performs a rotation on a single 2×1 vector at a time, it would have to either distinguish between the cases in (2.8) and (2.9) for each matrix-vector multiplication or store all four elements of the matrix G and perform four multiplications, see [11]. The first approach greatly increases the complexity of the processing elements, while the second completely ignores the principal advantage of Fast rotations.

On the contrary, the scaling procedure proposed here is extremely cheap with regard to a systolic implementation. The costly parts of Algorithm 3.1, the division in step 2 and the multiplication in step 3, must be performed for the determination of the new weight matrix W' in a Fast rotation anyway. The operations added for mere scaling purposes consist of shifts and subtractions in steps 1, 4 and 5 and cause very little overhead in computation time and area of a VLSI implementation. Consequently, the disadvantage of an increased number of multiplications in a Scaled rotation is offset by the homogeneity and simplicity of the processing elements, which are of utmost importance in present VLSI design.

6. A systolic array implementation. The following paragraphs will illustrate a systolic array for the triangular decomposition of DA with corresponding modifications of the right-hand side vectors Ds_h. The discussion will omit the backsubstitution, as a systolic array for it can be found in [13].

Matrices to be considered have a presumably narrow and dense band of width $w = p + q + 1$, $q \geqq 0$ being the number of subdiagonals and $p \geqq 0$ the number of superdiagonals. Sameh and Kuck [15] proposed a simple strategy for elimination of subdiagonal elements which preserves the bandwidth w (i.e., annihilation of $a_{q+i',i'}$ causes fill-in $a_{q+i'-1,i'+w-1}$) and removes elements by rotations in adjacent planes (that is, the elimination of $a_{q+i',i'}$ takes place by rotating planes $q+i'-1$ and $q+i'$). Therefore, it adheres to regular data flow and local communication.

Banded matrices are reduced to (banded) triangular form through removal of elements by subdiagonals, starting from without toward the main diagonal while proceeding from top to bottom within a subdiagonal. Formally, if $a_{q+1,1}$ is removed at time $t=1$ then, for all $1 \leqq i' \leqq m+q-2$, the elements $a_{q+i'-2(j'-1),i'-(j'-1)}$ are removed at the same time $t=i'$ if

$$1 \leqq j' \leqq \min\{i', q\} \quad \text{and} \quad q+i'-2(j'-1) \leqq m$$

(see the j'-loop in Algorithm 6.1 below). In the example below for $q=3$, $p=2$, $m=9$ and $n=8$, matrix entry $(q+i'-2(j'-1),\ i'-(j'-1))$ contains the time of removal of the corresponding (subdiagonal) element:

$$\begin{bmatrix}
x & x & x & & & & & \\
3 & x & x & x & & & & \\
2 & 4 & x & x & x & & & \\
1 & 3 & 5 & x & x & x & & \\
 & 2 & 4 & 6 & x & x & x & \\
 & & 3 & 5 & 7 & x & x & x \\
 & & & 4 & 6 & 8 & x & x \\
 & & & & 5 & 7 & 9 & x \\
 & & & & & 6 & 8 & 10
\end{bmatrix}.$$

Algorithm 6.1, which is by and large the same as in [11] for Fast Givens transformations, describes a parallel implementation of equations (4.1) for $m \geqq n$. Input consists of the matrix A, the right-hand side vectors s_h (whose ith element will be denoted by s_{hi}) and *two* copies of the weight matrix W. The elements of one copy, denoted by $w_i^{(0)}$, merely serve scaling purposes of Algorithm 3.1 and are consequently never altered during the whole algorithm. As outputs are obtained the upper triangular factor of A, the modified right-hand side vectors, the new weight matrix as well as the original matrix $W^{(0)}$. For brevity's sake the pair of elements $(w_i^{(0)}, w_i)$ is denoted by the Greek letter ω_i in the figures.

Observe, that during the elimination of *one* subdiagonal each row of the matrices A, W and the vectors s_h is modified by *two* plane rotations. Accordingly, during the elimination of the j'th subdiagonal the quantity $\Delta_{j'}$ represents an element of the weight matrix which has been altered by the first but not yet by the second rotation, $1 \leqq j' \leqq q$; the same is true for $\Delta_{h,j'}$ which holds an element of s_h. $\Delta_{j'}^{(0)}$ represents elements of the original weight matrix $W^{(0)}$. In a hardware implementation, $\Delta_{j'}$, $\Delta_{j'}^{(0)}$ and $\Delta_{h,j'}$ correspond to storage cells.

ALGORITHM 6.1.

Initially, $\Delta_{j'} := \Delta_{j'}^{(0)} := w_{j'}^{(0)}$; $\Delta_{h,j'} := s_{h,j'}$; $1 \leqq j' \leqq q$, $1 \leqq h \leqq z$;
  for $i' = 1 \cdots m+q-2$ *do sequentially,*
    for $j' = 1 \cdots \min\{i', q\}$ *do in parallel,*
      if $q+i'-2(j'-1) \leqq m$ *then*

$$i := q+i'-2(j'-1); \qquad j := i'-(j'-1);$$

(6.1a) $$\rho_j := \Delta_{j'}(a_{i-1,j})^2 + w_i(a_{i,j})^2,$$

(6.1b) *Compute* α *and* β *according to Algorithm* 3.1
where $\zeta_j := 1/\rho_j$, $\xi_j := \Delta_{j'} w_i \zeta_j$,

(6.1c) $$G_{i,j} := \begin{pmatrix} 2^{\alpha}\Delta_{j'}a_{i-1,j} & 2^{\alpha}w_i a_{i,j} \\ -2^{\beta}a_{i,j} & 2^{\beta}a_{i-1,j} \end{pmatrix},$$

(6.1d) $$a_{i-1,j} := 2^{\alpha}\rho_j, \qquad a_{i,j} := 0,$$

(6.1e) $$w_{i-1} := 2^{-2\alpha}\zeta_j, \quad \Delta_{j'} := 2^{-2\beta}\xi_j, \quad \Delta_{j'}^{(0)} := w_i^{(0)},$$

for $l = 1 \cdots w-1$ *do sequentially by pipelining* $G_{i,j}$,

(6.2) $$\begin{pmatrix} a_{i-1,j+l} \\ a_{i,j+l} \end{pmatrix} := G_{i,j} \begin{pmatrix} a_{i-1,j+l} \\ a_{i,j+l} \end{pmatrix};$$

for $h = 1 \cdots z$ *do sequentially by pipelining* $G_{i,j}$,

(6.3) $$\begin{pmatrix} s_{h,i-1} \\ \Delta_{h,j'} \end{pmatrix} := G_{i,j} \begin{pmatrix} \Delta_{h,j'} \\ s_{h,i} \end{pmatrix}.$$

The systolic array that implements Algorithm 6.1 consists of q processor vectors, where vector j' (counting from top to bottom) is responsible for eliminating subdiagonal j', $1 \leqq j' \leqq q$. Each vector consists of $w+z$ processors, since each matrix row consists of w nonzero elements and there are z right-hand side vectors. Input occurs at the bottom (processor vector q) of the array while the results are available at the top (processor vector 1), see Fig. 1. Within vector j', processors are numbered from left to right, $(j', -z), \cdots, (j', -1), (j', 1), \cdots, (j', w)$ of which the first z apply $G_{i,j}$ to elements of the right-hand side vectors and the remaining w to elements of the matrix A; note that $j = i' - (j' - 1)$. Consequently, $G_{i,j}$ is computed in processors $(j', 1)$ and thereafter pipelined to the left across processors $(j', -1), \cdots, (j', -z)$ and at the same time to the right across $(j', 2), \cdots, (j', w)$.

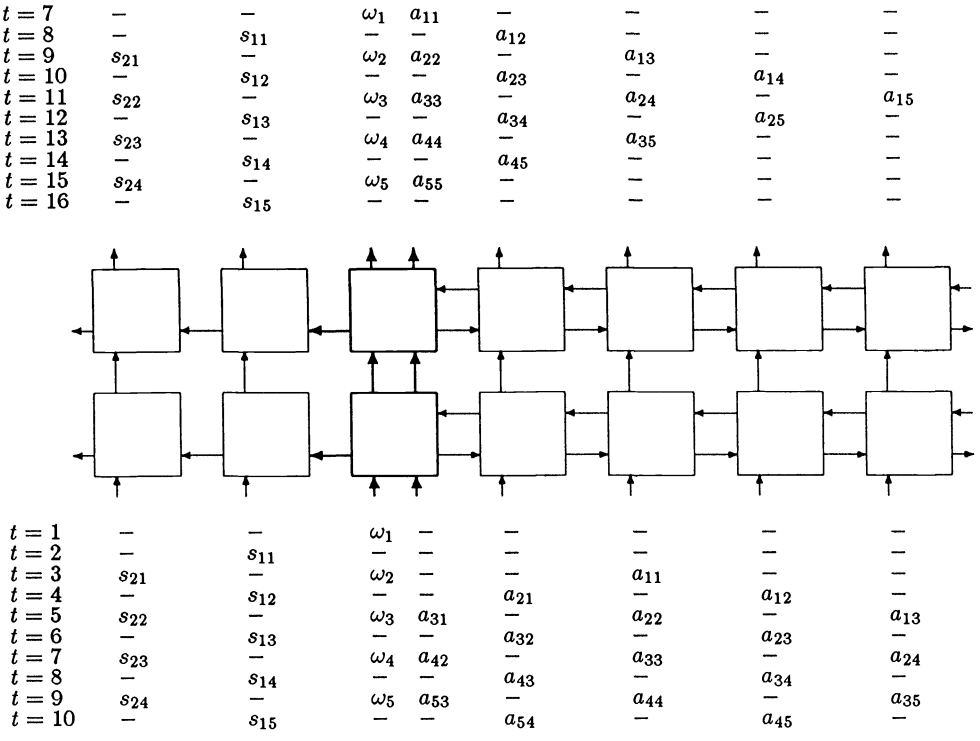
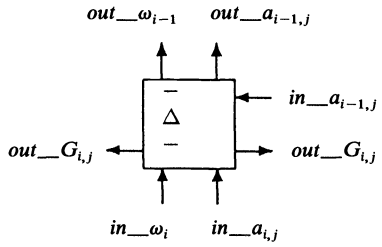


FIG. 1. *Processor array for* $q = p = z = 2$ *and* $m = n = 5$.

Subject to the three different sets of equations, three different types of processors, sketched in Fig. 2, accomplish the triangularisation. For $1 \leqq j' \leqq q$, "matrix processors" $(j', 1)$ and $(j', l+1)$ compute (6.1) and (6.2), respectively, while "right-hand side processors" $(j', -h)$ determine (6.3), $1 \leqq h \leqq z$. When no input is available for a processor, the input data lines carry a distinct value, represented by "—", and no computations take place (see also the remarks below regarding Fig. 1). In Fig. 2, the storage cells which store a value for more than one time step are depicted in a processor.

Data flow in and out of a matrix processor (j', l), $1 \leqq l \leqq w$ occurs by codiagonals so that succeeding codiagonal elements enter and leave a processor every other time step (in Fig. 1, "empty" time intervals during which a processor receives no input or delivers no output are indicated by "—". The times when items enter or leave the array are given to the left of the data streams). Input and output for right-hand side

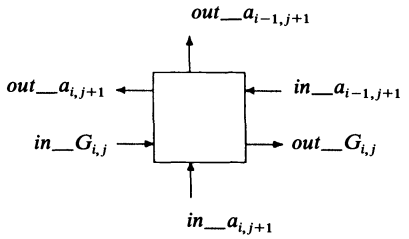


if $in_a_{i,j} \neq$ "—" *then*
$\omega_i := in_\omega_i$; $a_{i,j} := in_a_{i,j}$;
compute (6.1);
$out_\omega_{i-1,j} := \omega_{i-1}$; $out_a_{i,j} := a_{i,j}$;
$out_G_{i,j} := G_{i,j}$.

$\omega_{i-1} \equiv (w_{i-1}^{(0)}, w_{i-1})$, $\omega_i \equiv (w_i^{(0)}, w_i)$.

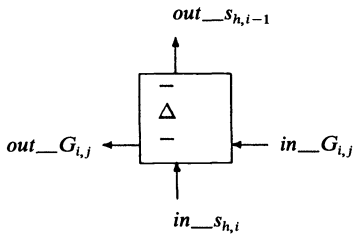
Δ Represents the storage cells $\Delta_{j'}$ and $\Delta_{j'}^{(0)}$.

(*a*) Matrix processor $(j', 1)$ in processor vector j'.



if $in_a_{i,j} \neq$ "—" *then*
$a_{i-1,j} := in_a_{i-1,j}$; $a_{i,j} := in_a_{i,j}$;
$G_{i,j} := in_G_{i,j}$;
compute (6.2);
$out_a_{i-1,j} := a_{i-1,j}$; $out_a_{i,j} := a_{i,j}$;
$out_G_{i,j} := G_{i,j}$.

(*b*) Matrix processors $(j', 2) \cdots (j', w)$ in processor vector j'.

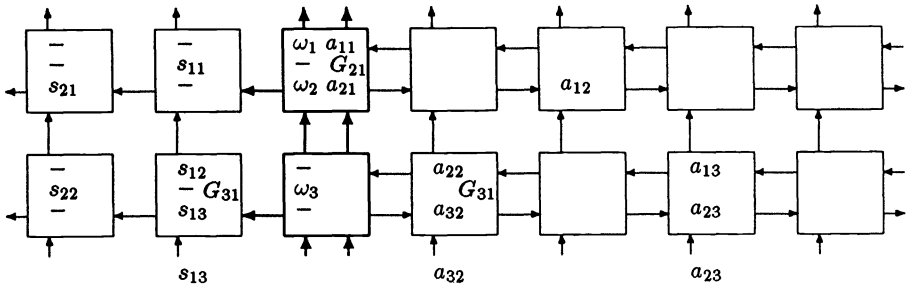


if $in_a_{i,j} \neq$ "—" *then*
$s_{i,j} := in_s_{i,j}$;
$G_{i,j} := in_G_{i,j}$
compute (6.3);
$out_s_{i-1,j} := s_{i-1,j}$;
$out_G_{i,j} := G_{i,j}$.

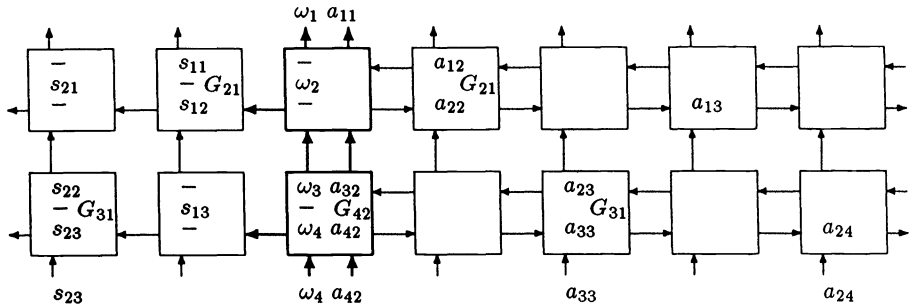
Δ Represents the storage cells $\Delta_{h,j'}$ and $\Delta_{h,j'}^{(0)}$.

(*c*) Right-hand side processors $(j', -z) \cdots (j', -1)$ in processor vector j'.

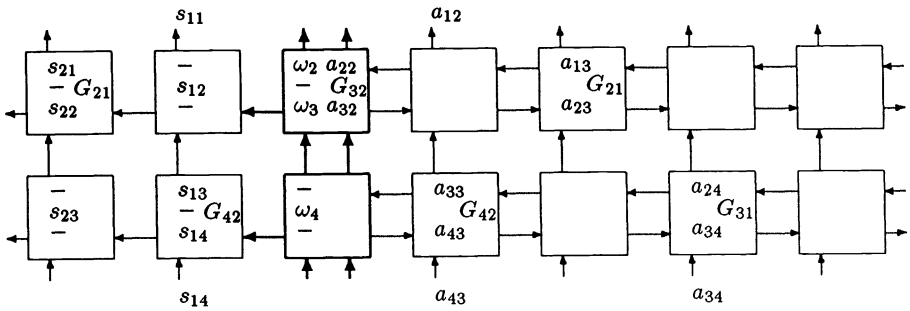
FIG. 2. *Processors.*



(*a*) $t=6$, Elimination of a_{21} by G_{21}.



(*b*) $t=7$, Elimination of a_{42} by G_{42}.



(*c*) $t=8$, Elimination of a_{32} by G_{32}.

$\omega_i \equiv (w_i^{(0)}, w_i)$

FIG. 3. *Partial execution trace for* $p=3$, $q=z=2$.

processors $(j', -h)$, $1 \leqq h \leqq z$, is by columns, again with successive column elements entering and exiting every other time step. A partial execution trace for the example $p=3$, $q=z=2$ can be found in Fig. 3.

In particular, $s_{h,1}$ enters processor $(q, -h)$ $q-h$ steps before a_{11} enters $(q, q+1)$ and leaves processor $(1, -h)$ h steps after a_{11} leaves $(1, 1)$. The weight matrix elements $\omega_1 = (w_1^{(0)}, w_1)$ enter processor $(q, 1)$ q steps before a_{11} enters the array while they exit $(1, 1)$ together with a_{11}. Starting the input with ω_1 at $t=1$ the computation time on $q(w+z)$ processors comes to

$$2m+3(q-1)+z+1.$$

7. Summary. A class of Scaled rotations for the parallel solution to weighted multiple linear least squares problems of finding v_h^* such that

$$\|D(Av_h^* - s_h)\| = \min_{v_h \in R^n} \|D(Av_h - s_h)\|, \qquad 1 \leqq h \leqq z,$$

for $m \times n$, w-banded matrices, $m > n$, in time $O(m+z)$ has been presented.

The planar orthogonally connected systolic array for this problem consists of $O(w^2+zw)$ processing elements which perform only the elementary operations $\{+, *, /\}$. To satisfy VLSI requirements, data exchange among processors proceeds on a nearest neighbour basis, and square root computations are avoided.

Proper scaling, which has been a persistent problem in implementations of Fast Givens rotations, prevents under- and overflow, obviating the need to pivot. A backward error analysis has shown the algorithm to be as stable as any available.

REFERENCES

[1] H. M. AHMED, J. M. DELOSME AND M. MORF, *Highly concurrent structures for matrix arithmetic and signal processing*, Computer, 15 (1982), pp. 65-82.

[2] E. H. BAREISS, *Numerical solution of the weighted least squares problem by G-transformations*, Tech. Report 82-03-NAM-03, Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, 1982.

[3] A. BOJANCZYK, R. P. BRENT AND H. T. KUNG, *Numerically stable solution of dense systems of linear equations using mesh-connected processors*, this Journal, 5 (1984), pp. 95-104.

[4] P. A. BUSINGER AND G. H. GOLUB, *Linear least squares solutions by Householder transformations*, Numer. Math., 7 (1965), pp. 269-278.

[5] W. M. GENTLEMAN, *Least squares computations by Givens rotations without square roots*, J. Inst. Math. Appl., 12 (1973), pp. 329-336.

[6] ———, *Error analysis of QR decompositions by Givens transformations*, Linear Algebra Appl., 10 (1975), pp. 189-197.

[7] W. M. GENTLEMAN AND H. T. KUNG, *Matrix triangularization by systolic arrays*, Proc. SPIE Real Time Signal Processing IV, SPIE, San Diego, CA, 1981, pp. 19-26.

[8] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins Press, Baltimore, MD, 1973.

[9] S. HAMMARLING, *A note on modifications to the Givens plane rotation*, J. Inst. Math. Appl., 13 (1974), pp. 215-218.

[10] D. E. HELLER AND I. C. F. IPSEN, *Systolic networks for orthogonal decompositions*, this Journal, 4 (1983), pp. 261-269.

[11] I. C. F. IPSEN, *A parallel QR method using Fast Givens' rotations*, Research Report 299, Department of Computer Science, Yale University, New Haven, CT, 1984.

[12] L. JOHNSSON, *A computational array for the QR-method*, Proc. Conf. Advanced Research in VLSI, Massachusetts Institute of Technology, Cambridge, MA, 1982, pp. 123-129.

[13] H. T. KUNG AND C. E. LEISERSON, *Systolic arrays (for VLSI)*, Sparse Matrix Proceedings, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1978, pp. 256-282.

[14] C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[15] A. H. SAMEH AND D. J. KUCK, *On stable parallel linear system solvers*, J. Assoc. Comput. Mach., 25 (1978), pp. 81-91.

[16] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Oxford University Press, London, 1965.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 5, September 1987
© 1986 Society for Industrial and Applied Mathematics

FAST TOEPLITZ ORTHOGONALIZATION USING INNER PRODUCTS*

GEORGE CYBENKO†

Abstract. A new method for the orthogonalization of complex m by n Toeplitz matrices of full rank is presented. An inverse QR factorization is computed in $9mn+(27/2)n^2$ multiplications and divisions. This method uses inner products and projections in the same spirit as lattice algorithms for linear prediction do.

Key words. Toeplitz matrix, orthogonal decomposition

AMS(MOS) subject classifications. Primary 65F25; secondary 62M20

1. Introduction. Toeplitz matrices arise in numerous applications of current interest. In a large class of problems, rectangular Toeplitz matrices are specially structured in that they have zeros in the upper right and lower left triangular corners. These matrices arise commonly in linear prediction problems [4], [8] when the *autocorrelation* approach is used. Linear prediction problems and discretizations of integral equations can also involve rectangular Toeplitz matrices without zero corners. (In signal processing contexts these are called covariance approaches.)

In [9], Sweet described an efficient method for the fast computation of QR decompositions of general Toeplitz matrices. An improvement of Sweet's method has recently been proposed by Bojanczyk, Brent and de Hoog [1]. Another method with potential for simple parallel implementation has recently been proposed by Chun, Kailath and Lev-Ari [3]. As was noted by Sweet, there are algorithms in the literature for solving closely related least squares problems involving Toeplitz matrices. In particular, the paper [6] describes an efficient algorithm for solving the normal equations arising from linear least squares problems involving Toeplitz matrices. Implicit in that work, and related work on *close-to-Toeplitz* matrices, are efficient orthogonalization methods. This paper derives one such fast orthogonalization for complex Toeplitz matrices that uses inner products and so is quite distinct from the methods of [1], [3], [9].

Our repeated use of the terms *fast* and *efficient* is meant to mean that an order of magnitude less work is involved so that while a general m by n matrix (with $m > n$) can be orthogonalized using $O(mn^2)$ operations, the fast methods use $O(mn)$ operations.

The method of this paper actually computes an *inverse orthogonal* factorization in the sense that instead of computing

$$T = QR$$

with Q having orthonormal columns and R upper triangular, the method presented here computes a Q and an R for which

$$TR = Q.$$

The implication of this for solving least squares problems is quite simple. Instead of backsolving a triangular system, a triangular matrix is multiplied against a vector. Insofar as numerical stability is concerned, the computation of the inverse triangular

* Received by the editors October 15, 1986; accepted for publication (in revised form) July 29, 1986. This work was partially supported by National Science Foundation grants DCR-8505634 and DCR-8619103, and U.S.-Spain Joint Committee for Scientific and Technical Cooperation grant 84-019.

† Department of Computer Science, Tufts University, Medford, Massachusetts 02155.

factor presents a potential pitfall. An error analysis of the classical lattice algorithm was presented in [5] and indicated that the excellent numerical properties exhibited in practice by the lattice method can be analytically substantiated. The similarity between the method of this paper and lattice algorithms suggests that some analytical attention ought to be paid to the accuracy of the former. By the same token, the numerical properties of Sweet's method have been subject to study recently [7]. Preliminary investigations suggest that that method is unstable in spite of the use of orthogonal plane rotations (it is the updating and downdating that can present problems there).

In the next section we present a simple derivation of this new orthogonalization method together with a motivation that arises naturally from analogies with lattice algorithms.

2. An algorithm for Toeplitz orthogonal factorization. Let T be a rectangular Toeplitz matrix with entries $t_{i,j} = t_{i-j}$ for simplicity ($1 \leqq i \leqq m$, $1 \leqq j \leqq n$, $n \leqq m$). If $t_k = 0$ for $k < 0$ and $m - n < k \leqq m$, such matrices can be efficiently orthogonalized using the so-called "lattice algorithm" [4], [9]. Since our method for the general case is derived directly from this important special case, it is useful to present this lattice algorithm.

DEFINITION. $\mathbf{Z}$ denotes the unit cyclical shift operator

$$(\mathbf{Z}t)_i = \begin{cases} t_{i-1} & \text{if } 1 < i \leqq m, \\ t_m & \text{if } i = 1. \end{cases}$$

Note that $T = [t, \mathbf{Z}t, \mathbf{Z}^2t, \cdots, \mathbf{Z}^{n-1}t]$ has a rectangular Toeplitz structure for any m-vector t (in fact, a rectangular circulant structure). We shall assume that $t_j = 0$ for $m - n < j \leqq m$ temporarily.

LATTICE ALGORITHM

Input:

$$\{t_j\}_{j=1}^{m} \quad \text{with } t_j = 0 \text{ for } m - n < j \leqq m.$$

Output:

Orthogonal vectors, q_j with $Q = [q_1, \cdots, q_n]$
and $TR = Q$, R upper triangular.

Initialization:

$$q_1 = \hat{q} = t.$$

Main Loop:

FOR $j = 2$ TO n DO

$$k_j = \frac{-(\mathbf{Z}q_{j-1}, \hat{q})}{(\hat{q}, \hat{q})}, \tag{1}$$

$$q_j = \mathbf{Z}q_{j-1} + k_j\hat{q}, \tag{2}$$

$$\hat{q} = k_j\mathbf{Z}q_{j-1} + \hat{q}. \tag{3}$$

Here (x, y) denotes the complex inner product $\sum \bar{x}_i y_i$. We have not included the steps that would also compute R or R^{-1} although that is possible. See [4], [8] for details. Note that the matrix T constructed as above from some nonzero t, always has full rank.

A key observation is that general Toeplitz matrices can be imbedded into such special Toeplitz matrices of larger dimension $(m + 2n - 2)$ by n as follows.

$$\hat{T} = \begin{bmatrix} t_{-n+1} & 0 & \cdot & \cdot & 0 \\ t_{-n+2} & t_{-n+1} & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ t_{-1} & t_{-2} & \cdot & \cdot & 0 \\ t_0 & t_{-1} & \cdot & \cdot & t_{-n+1} \\ t_1 & t_0 & \cdot & \cdot & t_{-n+2} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ t_{m-1} & t_{m-2} & \cdot & \cdot & t_{m-n} \\ 0 & t_{m-1} & \cdot & \cdot & t_{m-n+1} \\ \cdot & 0 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & 0 & t_{m-1} \end{bmatrix} = \begin{bmatrix} T_1 \\ T \\ T_2 \end{bmatrix}$$

where T is the original m by n Toeplitz matrix and T_1 and T_2 are lower and upper triangular Toeplitz matrices, respectively.

It is convenient to write

$$\hat{T} = [t, \mathbf{Z}t, \mathbf{Z}^2 t, \cdots, \mathbf{Z}^{n-1} t]$$

for the obvious $m+2n-2$ vector t.

Now this $\hat{T}$ has a form suitable for orthogonalization by the previously described lattice algorithm. However, we desire to orthogonalize the columns of $\hat{T}$ with respect to the Euclidean inner product over the m interior entries only. We can view this as orthogonalizing the big matrix $\hat{T}$ with respect to a non-Euclidean inner product defined by a weighting say W. This diagonal weighting matrix is an $m+2n-2$ by $m+2n-2$ diagonal matrix

$$W = \text{diag}\,(0, \cdots, 0, 1, \cdots, 1, 0, \cdots, 0)$$

where there are $n-1$ zeros, m ones and $n-1$ zeros, respectively. Clearly, if $\hat{T} = \hat{Q}\hat{R}$ is a decomposition of $\hat{T}$ such that $\hat{Q}^* W \hat{Q}$ is diagonal and $\hat{R}$ is upper trangular, then this gives an orthogonal decomposition of T by simply picking off the middle interior m rows of $\hat{Q}$. Our algorithm is based on this observation and really orthogonalizes $\hat{T}$ with respect to the W-inner product.

For notational convenience, let $x^* Wy = [x, y]$. Before proceeding, note the following fundamental identity satisfied by the W-inner product:

$$[\mathbf{Z}x, \mathbf{Z}y] = [x, y] + \bar{x}_{n-1} y_{n-1} - \bar{x}_{n+m-1} y_{n+m-1} \tag{4}$$

where $x = (x_j)$ and $y = (y_j)$ are two $m+2n-2$ vectors. This is easily verified by direct expansion using the definitions. A key element of this identity is that it can be rewritten as

$$[\mathbf{Z}x, \mathbf{Z}y] = [x, y] + x^* e_{n-1} e_{n-1} e_{n-1}^* y - x^* e_{n+m-1} e_{n+m-1}^* y = [x, y] + x^* D y$$

where e_{n-1} and e_{n+m-1} are the obvious standard basis vectors and $D = e_{n-1} e_{n-1}^* - e_{n+m-1} e_{n+m-1}^*$ is a rank two "displacement" correction. It will be easy to see

shortly that the efficiency of this method depends precisely on the fact that D has small rank, namely two in this case. More general situations where D has rank greater than two, say rank α, will have $O(\alpha nm)$ orthogonalization methods also.

This algorithm uses one main iteration with two basic loop invariants. They are stated as follows:

Invariant A. For each $j \geqq 1$, we have

$$q_j \in \text{span}\,[t, \mathbf{Z}t, \cdots, \mathbf{Z}^{j-1}t],$$

$$[q_j, q_i] = 0, \quad 1 \leqq i < j, \quad q_1 = t.$$

Invariant B. For each $j \geqq 1$, we have

$$\hat{q}_j \in \text{span}\,[t, \mathbf{Z}t, \cdots, \mathbf{Z}^{j-1}t],$$

$$[\hat{q}_j, x] = 0 \quad \text{for all } x \in \text{span}\,[\mathbf{Z}t, \cdots, \mathbf{Z}^{j-1}t].$$

Given these invariants as true for $1, 2, \cdots, j$, we now demonstrate that they are valid for $j+1$ also. This involves constructing q_{j+1} and $\hat{q}_{j+1}$ from q_i, $i \leqq j$ and $\hat{q}_i$, $i \leqq j$. By analogy with (2), let

$$q_{j+1} = \mathbf{Z}q_j + \delta_j \hat{q}_j + \sum_{k=2}^{j} \alpha_{j,k} q_k. \tag{5}$$

First, select δ_j so that

$$[q_1, q_{j+1}] = 0.$$

Thus let

$$\delta_j = \frac{-[q_1, \mathbf{Z}q_j]}{[q_1, \hat{q}_j]}. \tag{6}$$

(Note that by Invariants A and B above, the denominator in (6) is nonzero providing the first column of the Toeplitz matrix is linearly independent of the remaining columns.) Now pick i with $2 \leqq i \leqq j$ and observe that by construction

$$q_i = \sum_{k=1}^{i} \rho_{k,i} \mathbf{Z}^{k-1} t$$

for some $\{\rho_{k,i}\}$. Let

$$\mathbf{Z}p_i = \sum_{k=2}^{i} \rho_{k,i} \mathbf{Z}^{k-1} t$$

$$= \mathbf{Z} \sum_{k=2}^{i} \rho_{k,i} \mathbf{Z}^{k-2} t$$

(that is $\mathbf{Z}p_i$ is the same as q_i but with the single term $\rho_{1,i}q_1 = \rho_{1,i}t$ subtracted). Note that $p_i \in \text{span}\,[t, \mathbf{Z}t, \cdots, \mathbf{Z}^{i-2}t]$. Using the definitions of δ_j and p_i, we have

$$\begin{aligned}
[q_i, q_{j+1}] &= [q_i, \mathbf{Z}q_j] + \delta_j [q_i, \hat{q}_j] + \alpha_{j,i} [q_i, q_i] \\
&= [\mathbf{Z}p_i + \rho_{1,i}q_1, \mathbf{Z}q_j] + \delta_j [\mathbf{Z}p_i + \rho_{1,i}q_1, \hat{q}_j] + \alpha_{j,i} [q_i, q_i] \\
&= [\mathbf{Z}p_i, \mathbf{Z}q_j] + \alpha_{j,i} [q_i, q_i]
\end{aligned}$$

because both

$$[\rho_{1,i}q_1, \mathbf{Z}q_j] + \delta_j[\rho_{1,i}q_1, \hat{q}_j] = 0$$

(by the previous choice of δ_j) and

$$[\mathbf{Z}p_i, \hat{q}_j] = 0$$

(by Invariant B above). Now, we have

$$\begin{aligned}[q_i, q_{j+1}] &= [\mathbf{Z}p_i, \mathbf{Z}q_j] + \alpha_{j,i}[q_i, q_i] \\ &= [p_i, q_j] + \bar{p}_{n-1,i}q_{n-1,j} - \bar{p}_{n+m-1,i}q_{n+m-1,j} + \alpha_{j,i}[q_i, q_i].\end{aligned}$$

By Invariant A, $[p_i, q_j] = 0$ and so selecting

$$\alpha_{j,i} = \frac{\bar{p}_{n+m-1,i}q_{n+m-1,j} - \bar{p}_{n-1,i}q_{n-1,j}}{[q_i, q_i]}$$

will yield a q_{j+1} with the desired properties. In this form, the method would be an $O(mn^2)$ algorithm and the key to getting an $O(mn)$ algorithm lies in the fact that the summation in (5) reduces to

$$\begin{aligned}\sum_{k=2}^{j} \alpha_{j,k}q_k &= q_{n+m-1,j} \sum_{k=2}^{j} \bar{p}_{n+m-1,k} \frac{q_k}{[q_k, q_k]} - q_{n-1,j} \sum_{k=2}^{j} \bar{p}_{n-1,k} \frac{q_k}{[q_k, q_k]} \\ &= q_{n+m-1,j}f_j - q_{n-1,j}b_j.\end{aligned}$$

Here f_j and b_j are easily updated to give f_{j+1} and b_{j+1} which are required at the next iteration. Specifically, the updating is

$$f_{j+1} = f_j + \bar{p}_{n+m-1,j+1} \frac{q_{j+1}}{[q_{j+1}, q_{j+1}]},$$

$$b_{j+1} = b_j + \bar{p}_{n-1,j+1} \frac{q_{j+1}}{[q_{j+1}, q_{j+1}]}.$$

Finally, $\hat{q}_{j+1}$ must be obtained and to this end, we note that both $\hat{q}_j$ and q_{j+1} are W-orthogonal to

$$\mathbf{Z}t, \mathbf{Z}^2t, \cdots, \mathbf{Z}^{j-1}t$$

and so any linear combination of these two vectors will also enjoy this orthogonality property. In particular, select γ_j so that

$$\hat{q}_{j+1} = \hat{q}_j + \gamma_j q_{j+1}$$

is W-orthogonal to $\mathbf{Z}^j t$. Thus

$$\gamma_j = \frac{-[\mathbf{Z}^j t, \hat{q}_j]}{[\mathbf{Z}^j t, q_{j+1}]}.$$

This outlines the basic algorithm and an actual implementation requires a few additional calculations for bookeeping purposes. They are:

(1) Maintaining ortho*normality* (in the W inner product) of the q_j;

(2) Keeping track of R for which $TR = Q$. That is, keeping track of coefficients $\rho_{i,j}$ for which

$$\sum_{j=1}^{i} \rho_{j,i} \mathbf{Z}^{j-1} t = q_i.$$

In light of all this, we have the following formal description of this orthogonalization method. Assume that $t_{i,j} = t_{i-j}$ are the entries of an m by n Toeplitz matrix. The conventions used are:

(a) $\mathbf{Z}$ denotes the unit shift operator as described before, with dimension appropriate to the context;
(b) **bold roman** letters denote matrices with leading dimensions $n+m+1$ that are used in the computation of Q in the QR factorization;
(c) *italic roman* letters denote scalar quantities, either real or integer;
(d) lower case Greek characters $(\beta, \phi, \rho, \hat{\rho})$ denote matrices with leading dimensions n that are used to maintain the coefficients of various important linear combinations (that is, are related to R^{-1} in the QR factorization);
(e) as before, [,] denotes the W-inner product.

TOEPLITZ ORTHOGONALIZATION ALGORITHM

:INITIALIZATIONS:

$k = m + n$
$\mathbf{q}_{j,1} = t_{j-n}$ for $1 \leqq j \leqq k-1$, $\mathbf{q}_{k,1} = \mathbf{0}$
$\mathbf{f} = \mathbf{b} = \mathbf{0}$, $\phi = \beta = \rho = \mathbf{0}$
$r = [\mathbf{q}_1, \mathbf{q}_1]^{1/2}$
$\rho_{1,1} = 1/r$
$\hat{\rho}_1 = 1/r$
$\mathbf{q}_1 = \mathbf{q}_1 / r$
$\hat{\mathbf{q}} = \mathbf{q}_1$

:MAIN LOOP:

FOR $J = 2$ TO n DO
BEGIN
$v = -[\mathbf{Z}\mathbf{q}_{j-1}, \mathbf{q}_1]$
$u = -\mathbf{q}_{n-1,j-1}$
$w = \mathbf{q}_{n+m-1,j-1}$
$\mathbf{q}_j = \mathbf{Z}\mathbf{q}_{j-1} + v\hat{\mathbf{q}} + u\mathbf{b} + w\mathbf{f}$
$\rho_j = \mathbf{Z}\rho_{j-1} + v\hat{\rho} + u\beta + w\phi$
$v = [\mathbf{q}_j, \mathbf{q}_j]$
$\mathbf{q}_j = \mathbf{q}_j / v$
$\rho_j = \rho_j / v$
$v = \mathbf{q}_{n,j} - \rho_{1,j} t_0$
$w = \mathbf{q}_{n+m,j} - \rho_{1,j} t_m$
$\mathbf{f} = \mathbf{f} + v\mathbf{q}_j$
$\phi = \phi + v\rho_j$
$\mathbf{b} = \mathbf{b} + w\mathbf{q}_j$
$\beta = \beta + w\rho_j$
$v = r\rho_{1,j}$
$\hat{\mathbf{q}} = \hat{\mathbf{q}} + v\mathbf{q}_j$
$\hat{\rho} = \hat{\rho} + v\rho_j$
END

The output of this algorithm contains the orthonormal columns in the vectors consisting of the entries

$$\mathbf{q}_{j,i}$$

as i goes from n to $n+m-1$ inclusively and the inverse triangular factor in the matrix ρ.

3. Conclusion. A single algorithm for computing the inverse orthogonal factorization of a general complex Toeplitz matrix is presented. The method is more efficient for solving linear least squares problems than is the method of Sweet [9]. Our method uses $9mn+(27/2)n^2$ multiplies while Sweet's method uses $25mn+O(n^2)$ multiplies. The Bojanczyk-Brent-de Hoog method uses $13mn+6n^2+O(n)$ multiplies. The numerical properties of these methods are not yet well understood.

A Fortran program, implementing this method and using calls to LINPACK BLAS [2], can be obtained from the author.

REFERENCES

[1] A. W. BOJANCZYK, R. P. BRENT AND F. R. DE HOOG, *QR factorization of Toeplitz matrics*, Numerische Mathematik, to appear, 1986.

[2] J. R. BUNCH, J. DONGARRA, C. MOLER AND G. W. STEWART, LINPACK *User's Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

[3] J. CHUN, T. KAILATH AND H. LEV-ARI, *Fast parallel algorithm for QR factorization of structured matrices*, preprint, 1986.

[4] G. CYBENKO, *A general orthogonalization method with applications to time series analysis and signal processing*, Math. Comp., 40 (1983), pp. 323-336.

[5] ———, *The numerical stability of lattice algorithms for least squares linear prediction problems*, BIT, 24 (1984), pp. 441-455.

[6] T. KAILATH, S. Y. KUNG AND M. MORF, *Displacement ranks of matrices and linear equations*, J. Math. Anal. Appl., 68 (1979), pp. 395-407.

[7] F. LUK AND S. QIAO, *A fast but unstable orthogonal traingularization technique for Toeplitz matrices*, Cornell University, Ithaca, NY, Department of Electrical Engineering Technical Report EE-CEG-85-5, July 1985.

[8] J. MAKHOUL, *Linear prediction: a tutorial review*, Proceedings IEEE, 63 (1975), pp. 564-580.

[9] D. R. SWEET, *Fast Toeplitz orthogonalization*, Numer. Math., 43 (1984), pp. 1-21.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 5, September 1987
© 1987 Society for Industrial and Applied Mathematics
006

ON JACOBI METHODS FOR SINGULAR VALUE DECOMPOSITIONS*

VJERAN HARI† AND KREŠIMIR VESELIĆ‡

Abstract. An improvement of the Jacobi singular value decomposition algorithm is proposed. The matrix is first reduced to a triangular form. It is shown that the row-cyclic strategy preserves the triangularity. Further improvements lie in the convergence properties. It is shown that the method converges globally and a proof of the quadratic convergence is indicated as well. The numerical experiments confirm these theoretical predictions. Our method is about 2-3 times slower than the standard QR method but it almost reaches the latter if the matrix is diagonally dominant or of low rank.

Key words. singular values, Jacobi method

AMS(MOS) subject classification. 65F15

1. Introduction. As is well known, the Jacobi eigenreduction of real symmetric matrices is generally inferior to the QR method, the former being about three times slower than the latter [9] at least on the conventional serial computers. In spite of this, there are certain circumstances under which the Jacobi method can be preferable, e.g., if approximate vectors are known. This happens as a rule in the course of modeling the parameters of a system where one has to reduce a sequence of matrices differing only slightly from each other. Another interesting application of the Jacobi method along these lines was described recently by Stewart [12], [13].

In the case of the singular value decomposition (SVD)

$$U^{-1}AV = D, \quad U, V \text{ orthogonal}, \quad D \text{ diagonal} \tag{1.1}$$

of a general square matrix A, the superiority of the QR algorithm [3], [4] is even more expressed. In fact, the Jacobi algorithm, as described by Forsythe and Henrici [2], has the following additional shortcomings:

(i) It works all the time on a full nonsymmetric matrix.

(ii) There are global convergence problems with the row or column cyclic strategy since some rotation angles can run over the interval $(-\pi/2, \pi/2)$.

(iii) There are quadratic convergence problems in the case of multiple singular values.

In fact, if the off-diagonal elements are $O(\varepsilon)$ then, in contrast to the standard symmetric case [14], the off-diagonal elements corresponding to multiple singular values will not necessarily be $O(\varepsilon^2)$ [7]. This, together with possible large rotation angles, can inhibit the ultimate quadratic convergence.

Similar objections can be made for the Jacobi generalized singular value decomposition (GSVD) on a pair of matrices A, B, giving

$$\begin{gathered} U^{-1}AT = D_1, \qquad V^{-1}BT = D_2, \\ T \text{ nonsingular}, \qquad U, V \text{ orthogonal}, \qquad D_1, D_2 \text{ diagonal}. \end{gathered} \tag{1.2}$$

For a Jacobi-like algorithm of this type, see § 4.

The algorithms which we propose in this paper are based on the following simple observation: *If the matrix (or matrices) is triangular, then the triangularity is essentially*

* Received by the editors June 21, 1984; accepted for publication (in revised form) September 23, 1986.

† Department of Mathematics, University of Zagreb, pp. 187, YU-41000 Zagreb, Yugoslavia.

‡ Lehrgebiet. Mathematische Physik, Fernuniverersität-GHS-Hagen, Postfach 940, D-5800 Hagen, West Germany.

preserved by the Jacobi algorithm, provided that certain strategies are used (e.g. the row or column cyclic strategy).

As we shall see below in this case *all* shortcomings listed above are removed. On the other hand, the requirement for the initial matrix to be triangular is not a strong one since in many cases the matrices to be processed are already in this form.

In § 2 we describe how the column (row) cyclic strategy essentially preserves the triangularity.[1] In addition, another strategy is presented which could be used in parallel computing. In our opinion, however, the method is interesting even on a classical serial computer.

In § 3 we discuss the effect of the initial triangularity to the singular value decomposition of a single matrix A. As we shall see, the theory of the Jacobi SVD on a triangular matrix is about as good as the standard symmetric one. In particular, we show that there exists a convenient angle choice which insures general global convergence of the column cyclic method. We also make plausible that ultimate quadratic convergence holds in full analogy with that on real symmetric matrices.

In § 4 we define a generalized singular value decomposition (GSVD) algorithm of Jacobi type, also realizable on triangular matrices. This algorithm is related to the classical generalized eigenvalue algorithm of Falk and Langemeyer [1].

In § 5 we give some preliminary numerical results confirming our theoretical predictions. For general matrices our Jacobi SVD method was about 2–3 times slower than the QR SVD method. For special classes of matrices, like rank deficient or diagonally dominant matrices, both methods were about equally efficient.

2. Jacobi transformations on triangular matrices. Let A be a square matrix of order n.[2] *A Jacobi transformation* (step) on A is any transformation of the form

$$A \to A' = FAG$$

where F and G differ from the unit matrix at most at the positions

$$\begin{matrix} (l, l) & (l, m) \\ (m, l) & (m, m) \end{matrix}$$

and such that zeros are created on the (l, m) and (m, l) position of A'.

Let A be, e.g., upper triangular and let the pairs (l, m) be chosen following the column cyclic strategy:

$$(1,2), (1,3), (2,3) \cdots (1,n), \cdots, (n-1, n).$$

Then the movement of the zero elements of A during one cycle can be illustrated in the following example of dimension four (see Fig. 2.1.1).

During the second cycle the movement of the zeros is just the opposite so that after two cycles the matrix is again upper triangular. Analogous behaviour takes place if the row cyclic strategy is used. In any case, however, the matrix A is *essentially triangular*, i.e.,

$$a_{ij}a_{ji} = 0, \qquad i \neq j \tag{2.1}$$

holds at every stage. Thus, A can be permanently stored on the starting triangle which halves the usual computational and storage effort.

Not many strategies exist which preserve the essential triangularity. However one certainly deserves to be mentioned. We shall call it the *parallel strategy* since it could

[1] The same idea has been exploited by C. Paige [19].

[2] For simplicity we restrict ourselves to real matrices. All considerations in this paper can be carried over to the complex case as well.

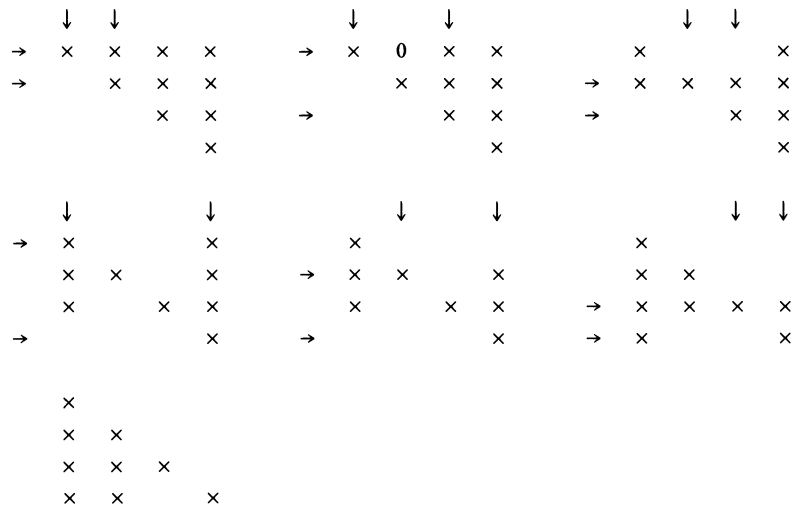


FIG. 2.1.1

be appropriate for parallel computing [11]. Here at every step several transformations are performed simultaneously. The strategy has two phases. The *preliminary phase* consists of $n-1$ steps. We illustrate it (see Fig. 2.1.2) on a matrix of order eight (the elements denoted by $\boxtimes$ are going to be annihilated in the next step).

Starting with the last matrix above, the *cyclic phase* begins each cycle consisting of $2n$ steps. We illustrate the form of the matrix (see Fig. 2.2) after the rth step ($r \leqq n$) for the example $n = 10$, $r = 5$.

After the $(n+r)$th step the picture is just the transpose of the one above. At any stage, however, the essential triangularity (2.1) holds.

3. The Jacobi singular value decomposition. The Jacobi singular value decomposition is based on the following elementary transformation of the pivot submatrix:

$$\begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \begin{pmatrix} a_{ll} & a_{lm} \\ a_{ml} & a_{mm} \end{pmatrix} \begin{pmatrix} \cos\psi & \sin\psi \\ -\sin\psi & \cos\psi \end{pmatrix} = \begin{pmatrix} a'_{ll} & 0 \\ 0 & a'_{mm} \end{pmatrix}.$$

The rotation angles φ and ψ are determined by the formulae

$$\tan 2\varphi = \frac{2(a_{ml}a_{ll} + a_{mm}a_{lm})}{a_{mm}^2 + a_{ml}^2 - a_{ll}^2 - a_{lm}^2}, \tag{3.1a}$$

$$\tan\psi = -\frac{a_{lm} - a_{mm}\tan\varphi}{a_{ll} - a_{ml}\tan\varphi} = \frac{a_{ml} + a_{ll}\tan\varphi}{a_{mm} + a_{lm}\tan\varphi} \tag{3.1b}$$

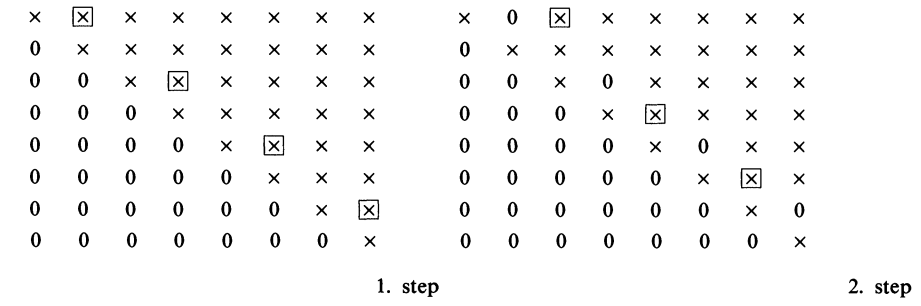
or, equivalently, by the formula

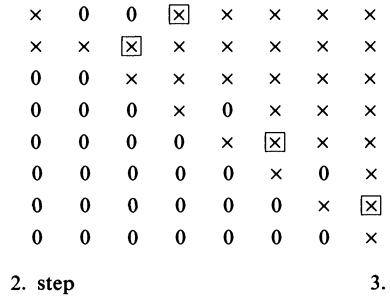
$$\tan 2\psi = \frac{2(a_{ll}a_{lm} + a_{ml}a_{mm})}{a_{mm}^2 + a_{lm}^2 - a_{ll}^2 - a_{ml}^2}, \tag{3.2a}$$

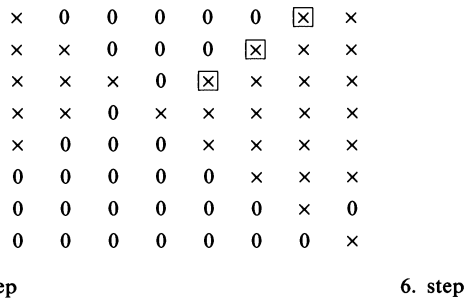
$$\tan\varphi = \frac{a_{lm} + a_{ll}\tan\psi}{a_{mm} + a_{ml}\tan\psi} = -\frac{a_{ml} - a_{mm}\tan\psi}{a_{ll} - a_{lm}\tan\psi}. \tag{3.2b}$$

For obvious stability reasons in (3.1b) and (3.2b), respectively, the quotients of absolutely larger numbers are always taken.[3]

[3] If, e.g., both quotients in (3.1b) were 0/0 then the pivot matrix would be zero and both rotations could be skipped.







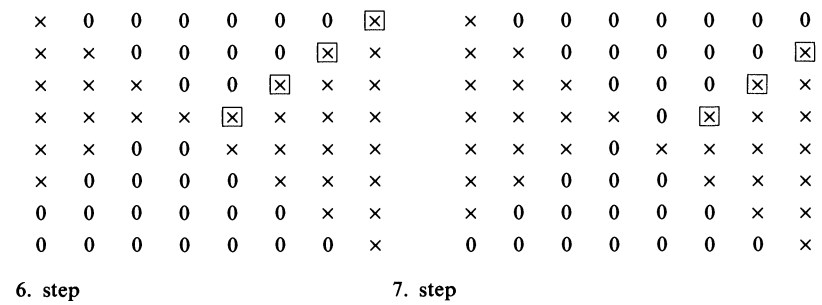


FIG. 2.1.2

We start with an upper triangular matrix A and take the following particular choice between (3.1) and (3.2): *In any even (odd) cycle, i.e., when the pivot submatrix is lower (upper) triangular the formula* (3.1) ((3.2)) *is used.* The advantages of this choice will be seen below.

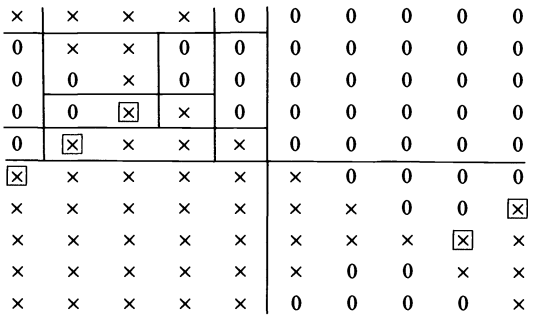


FIG. 2.2

With this choice the formulae (3.1) and (3.2) yield

$$\tan 2\varphi = \frac{2a_{ml}a_{ll}}{a_{mm}^2 - a_{ll}^2 + a_{ml}^2}, \tag{3.3a}$$

$$\tan \psi = \frac{a_{mm}\tan\varphi}{a_{ll} - a_{ml}\tan\varphi} = \frac{a_{ml} + a_{ll}\tan\varphi}{a_{mm}} \tag{3.3b}$$

for a lower triangular pivot matrix and

$$\tan 2\psi = \frac{2a_{ll}a_{lm}}{a_{mm}^2 - a_{ll}^2 + a_{lm}^2}, \tag{3.4a}$$

$$\tan \varphi = \frac{a_{lm} + a_{ll}\tan\psi}{a_{mm}} = \frac{a_{mm}\tan\psi}{a_{ll} - a_{lm}\tan\psi}, \tag{3.4b}$$

for an upper triangular pivot matrix, respectively. Since the whole time the current matrix is stored in the *upper* triangle of the array (a_{ij}) the formulae (3.4) *are in fact used in all cases.* Similarly, under the same storage convention the transformation formulae read in all cases as follows:

$$i = 1, \cdots, l-1; \begin{cases} a_{il} = a_{il}\cos\varphi - a_{im}\sin\varphi, \\ a'_{im} = a_{il}\sin\varphi - a_{im}\cos\varphi, \end{cases} \tag{3.5}$$

$$i = l+1, \cdots, m-1: \begin{cases} a'_{li} = a_{li}\cos\psi - a_{im}\sin\psi, \\ a'_{im} = a_{li}\sin\psi + a_{im}\cos\psi, \end{cases} \tag{3.6}$$

$$i = m+1, \cdots, n: \begin{cases} a'_{li} = a_{li}\cos\varphi - a_{mi}\sin\varphi, \\ a'_{mi} = a_{li}\sin\varphi + a_{mi}\cos\varphi, \end{cases} \tag{3.7}$$

$$a'_{ll} = a_{ll}\cos\psi\cos\varphi + a_{mm}\sin\psi\sin\varphi - a_{lm}\cos\varphi\sin\psi, \tag{3.8}$$

$$a'_{mm} = a_{mm}\cos\psi\cos\varphi + a_{ll}\sin\psi\sin\varphi + a_{lm}\cos\psi\sin\varphi, \tag{3.9}$$

$$a'_{lm} = 0. \tag{3.10}$$

This simplicity of the transformation formulae is the first advantage of our angle choice. The second which is much less trivial will lie in the convergence proof of Theorem 3.4 below.

Instead of (3.1)-(3.9) modified formulae in the spirit of the known Rutishauser code [10] are used in order to improve the accuracy. Also a somewhat faster variant

of the algorithm can be constructed which uses the elementary matrices of the type

$$\begin{pmatrix} 1 & \alpha \tan \varphi \\ -(\tan \varphi)/\alpha & 1 \end{pmatrix}. \tag{3.11}$$

We now consider the convergence properties of the algorithm obtained. Note that here optimal or threshold strategies are out of the question, since they destroy the essential triangular form. We present first a simple result for nonsingular matrices.

THEOREM 3.1. *If A is nonsingular then the row (and column) cyclic algorithm is convergent.*

Proof. First we note that the essential triangularity (2.1) does not describe all nice properties of the matrices with which we are working. The general form of such a matrix (or its transpose) before the l, m-transformation is given below in Fig. 3.1.

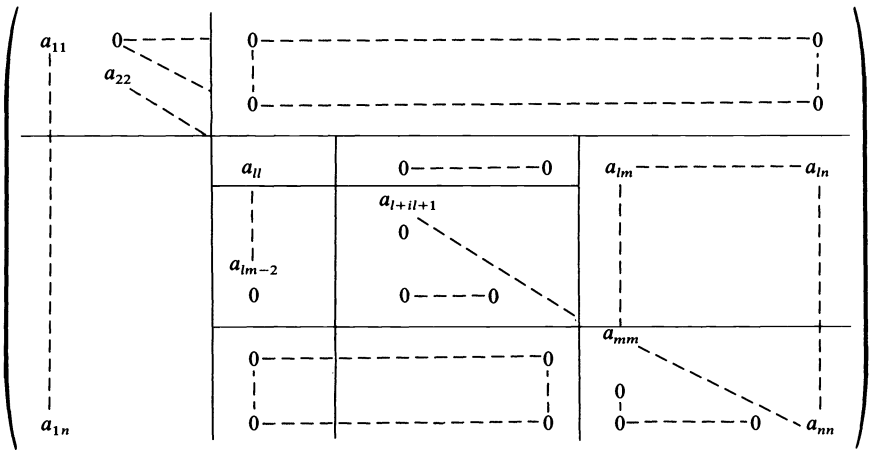


FIG. 3.1

LEMMA 3.2. *The eigenvalues of any matrix as in Fig.* 3.1, *counting their multiplicities, lie on its diagonal.*

Proof of Lemma 3.2. The assertion follows from the simple observation that such matrices are *hierarchically triangular*, i.e., block triangular such that the diagonal blocks are again block triangular, etc.

Remark 3.3. The same assertion could be proved for the matrices on Fig. 2.2 but we shall not need this here.

Proof of Theorem 3.1. Denote by $\sigma_{\min}$ and $\sigma_{\max}$ the smallest and the largest singular value of A, respectively. Then $|a_{ij}| \leqq \sigma_{\max}$ and by Lemma 3.2 also $|a_{ii}| \geqq \sigma_{\min}$. Now from (3.7), (3.8) it follows that

$$|\tan \varphi| \leqq \frac{|a_{ll}||\tan \psi| + |a_{lm}|}{|a_{ll}|} \leqq \frac{2\sigma_{\max}}{\sigma_{\min}}.$$

The same estimate follows from (3.9), (3.10). Now φ is uniformly separated from $\pm\pi/2$ and by a result of Forsythe and Henrici [2] the convergence follows. This proves the theorem. Note that in this rather simple proof our choice of the rotation angles was not used. This choice, however, is crucial in our general convergence result.

THEOREM 3.4. *Let A be a general triangular matrix. The column cyclic algorithm with our choice of angles is always convergent.*

Proof.[4] We take an upper triangular $A=(a_{ij})$. Note that in this proof a_{ij} are the very matrix elements of A and not the elements of the upper triangular storage array we mentioned before. However, we consider only one cycle of the algorithm and thus the formulae (3.4)-(3.10) are valid. Set

$$S^2(A)=\sum_{\substack{i,j\\ i\neq j}} a_{ij}^2.$$

Let A' be the current matrix after the full cycle. We shall prove that

$$S^2(A')\leqq c_n S^2(A) \tag{3.12}$$

where $0\leqq c_n<1$ and c_n depends only on n. Note that (3.12) is trivially true for $n=2$. The proof will proceed by induction with respect to n. Suppose that (3.12) holds for all upper trangular A's of order $n-1$, where $c_2,\cdots,c_{n-1}<1$. Denote by $\tilde{A}=(\tilde{a}_{ij})$ the current matrix after the first $(n-1)(n-2)/2=\tilde{n}$ steps and by $\tilde{A}_{n-1}$ its leading submatrix of order $n-1$. Observe that at this stage $\tilde{a}_{nj}=0$ for $j=1,\cdots.n-1$ (see Fig. 2.1) and that

$$\sum_{i=1}^{n-1} a_{in}^2=\sum_{i=1}^{n-1}\tilde{a}_{in}^2. \tag{3.13}$$

Define ε by

$$(1-\varepsilon^2)S^2(A)=\sum_{i=1}^{n-1} a_{in}^2, \qquad \varepsilon\geqq 0.$$

Then $0\leqq\varepsilon\leqq 1$ and by the induction hypothesis

$$\begin{aligned} S^2(\tilde{A}) &= S^2(\tilde{A}_{n-1})+(1-\varepsilon^2)S^2(A)\\ &\leqq c_{n-1}\varepsilon^2 S^2(A)+(1-\varepsilon^2)S^2(A)\\ &=[1-\varepsilon^2(1-c_{n-1})]S^2(A). \end{aligned} \tag{3.14}$$

We now estimate the effect of the last $n-1$ steps. Denote by $\tilde{A}^k=(\tilde{a}_{ij}^k)$, ψ_k the current matrix and the due angle after $\tilde{n}+k-1$ steps, respectively. We have

$$\begin{aligned} \tilde{a}_{in}^{(2)} &= \tilde{a}_{in}\cos\psi_1+\tilde{a}_{i1}\sin\psi_i,\\ \tilde{a}_{in}^{(3)} &= \tilde{a}_{in}^{(2)}\cos\psi_2+\tilde{a}_{i2}\sin\psi_2,\\ &\vdots\\ \tilde{a}_{in}^{(i)} &= \tilde{a}_{in}^{(i-1)}\cos\psi_{i-1}+\tilde{a}_{ii-1}\sin\psi_{i-1}, \end{aligned}$$

for $2\leqq i\leqq n-1$. These relations imply

$$\tilde{a}_{in}^{(i)}=\tilde{a}_{in}\cos\psi_1\cdots\cos\psi_{i-1}+\sum_{k=1}^{i-1}\tilde{a}_{ik}\sin\psi_k\cos\psi_{k+1}\cdots\cos\psi_{i-1}. \tag{3.15}$$

[4] The proof is modeled after an analogous proof for the eigenreduction of symmetric matrices [14].

Using the inequality $(a+b)^2 \geqq \frac{1}{2}a^2 - b^2$ and $\cos^2 \psi_k \geqq \frac{1}{2}$ $(|\psi_k| \leqq \pi/4!)$, we obtain

$$
\begin{aligned}
(\tilde{a}_{in}^{(i)})^2 &\geqq 2^{-i}(\tilde{a}_{in})^2 - \left[\sum_{k=1}^{n-2} |\sin \psi_k| |a_{ik}|\right]^2 \\
&\geqq 2^{-i}(\tilde{a}_{in})^2 - \sum_{k=1}^{n-2} \sin^2 \psi_k \sum_{k=1}^{i-1} (\tilde{a}_{ik})^2, \qquad 2 \leqq i \leqq n-1.
\end{aligned} \tag{3.16}
$$

Now the reduction of S^2 in the last $n-1$ steps is

$$
\begin{aligned}
\sum_{i=1}^{n-1} (\tilde{a}_{in}^{(i)})^2 &\geqq \sum_{i=1}^{n-1} 2^{-i}(\tilde{a}_{in})^2 - \sum_{k=1}^{n-2} \sin^2 \psi_k \sum_{i=2}^{n-1} \sum_{k=1}^{i-1} (\tilde{a}_{ik})^2 \\
&\geqq 2^{-n+1}(1-\varepsilon^2) S^2(A) - \tfrac{1}{2}(n-2) S^2(\tilde{A}_{n-1}) \\
&\geqq [2^{-n+1}(1-\varepsilon^2) - \tfrac{1}{2}(n-2) c_{n-1} \varepsilon^2] S^2(A) \\
&\equiv f(\varepsilon) S^2(A).
\end{aligned}
$$

In the estimate (3.16) we used the Schwarz inequality and the inequality $|\psi_k| \leqq \pi/4$ $1 \leqq k \leqq n$. This inequality follows from the formula (3.4a) and this is the point of the proof where our angle choice enters crucially. Note that $f(\varepsilon)$ is a decreasing function of ε for $0 \leqq \varepsilon \leqq 1$ with its maximum at zero. This function vanishes at $\varepsilon = \varepsilon_n = 1/(1+2^{n-2}(n-2)c_{n-1})^{1/2} \in [0, 1]$. Set

$$
f_+(\varepsilon) = \begin{cases} f(\varepsilon), & 0 \leqq \varepsilon \leqq \varepsilon_n, \\ 0, & \varepsilon_n \leqq \varepsilon \leqq 1. \end{cases}
$$

Then, of course

$$
S^2(A') \leqq S^2(\tilde{A}) - f_+(\varepsilon) S^2(A).
$$

This together with (3.14) gives

$$
S^2(A') \leqq g(\varepsilon) S^2(A)
$$

with

$$
g(\varepsilon) = 1 - f_+(\varepsilon) - \varepsilon^2(1 - c_{n-1}), \qquad 0 \leqq \varepsilon \leqq 1.
$$

To complete the proof we have only to show that

$$
c_n \equiv \max_{0 \leqq \varepsilon \leqq 1} g(\varepsilon) < 1. \tag{3.17}
$$

Indeed we have

$$
\begin{aligned}
1 - g(\varepsilon) &= f_+(\varepsilon) + \varepsilon^2(1 - c_{n-1}) \\
&= \begin{cases} 2^{-n+1} - \varepsilon^2\left(2^{-n+1} + \dfrac{n}{2} c_{n-1} - 1\right), & 0 \leqq \varepsilon \leqq \varepsilon_n, \\ \varepsilon^2(1 - c_{n-1}), & \varepsilon_n \leqq \varepsilon \leqq 1. \end{cases}
\end{aligned}
$$

For $2^{-n+1} + (n/2)c_{n-1} - 1 \geqq 0$ we have

$$
\min (1 - g(\varepsilon)) = 1 - g(\varepsilon_n) = \frac{1 - c_{n-1}}{1 + 2^{n-2}(n-2)c_{n-1}} > 0.
$$

For $2^{-n+1} + (n/2)c_{n-1} - 1 < 0$ we have

$$
\min (1 - g(\varepsilon)) = 1 - g(0) = 2^{-n+1} > 0
$$

which gives

$$c_n = \begin{cases} 1-\varepsilon_n^2(1-c_{n-1}), & 2^{-n+1}+\frac{n}{2}c_{n-1} \geqq 1, \\ 1-2^{-n+1}, & 2^{-n+1}+\frac{n}{2}c_{n-1} < 1, \end{cases} \tag{3.18}$$

and (3.17) holds. Therefore, (3.12) holds as well. Since the same proof works on a lower triangular matrix A[5] the formula (3.12) holds after each cycle which proves that the method is at least linearly convergent. This proves the theorem.

Note that (3.18) gives a recursion formula for the sequence of constants c_n which starts by $c_2 = 0$, $c_3 = \frac{3}{4}$ etc. For $n = 4$ the upper formula in (3.18) is valid.

If A is singular, then it has zeros on its diagonal. Every diagonal zero can be removed in a finite number of steps. In fact, let e.g. $a_{11} = 0$:

$$A = \begin{pmatrix} & & l & \\ * & & * & \\ & \ddots & \vdots & \\ & * & * & \\ & & 0 & * \cdots * \\ & & & * \\ & & & \ddots \\ & & & * \end{pmatrix} \begin{matrix} \\ \\ \\ \\ l. \\ \\ \\ \\ \end{matrix}$$

Then

$$R_{n-l}^{-1} \cdots R_1^{-1} A S_1 \cdots S_{l-1} = \begin{pmatrix} * & & & 0 & & \\ & \ddots & & \vdots & & \\ & & * & 0 & * & \\ & & & 0 & 0 \cdots & 0 \\ & & & & * & \\ & & & & & \ddots \\ & & & & & * \end{pmatrix} \tag{3.19}$$

where S_j (R_j) is a rotation in the $(l-j, l)$th plane ($(l, l+j)$th plane). If there are more diagonal zeros the process is repeated. At the end a nonsingular triangular matrix of a smaller dimension remains. Of course, such a deflation can be performed during the real Jacobi diagonalisation process at the end of each cycle if small elements occur on the diagonal. Similar but a little more complicated deflation formulae exist for any step of the algorithm. We omit the further consideraton of such deflation steps since they do not seem to carry significant gains in efficiency (see § 4 below).

The quadratic convergence proof for this triangular Jacobi SVD can be shown to hold with constants similar to those of the symmetric Jacobi method [16] for distinct eigenvalues. On the other hand, it can be shown that in an almost diagonal *essentially triangular matrix* the off-diagonal elements corresponding to a multiple eigenvalue are

[5] Our angle choice, in fact, insures that during the sweeping of a column every element of that column, before its annihilation, is rotated only by angles between $\pm\pi/4$.

quadratically small; therefore, the situation is quite similar to that of the symmetric Jacobi algorithm (see [15]).[6]

If the singular value decomposition of an $n \times m$ matrix A with $n > m$ is required, then as the first step a QR decomposition of A is needed. For a good ordering of the singular values and an early separation of diagonal zeros the pivoting variant of the QR decomposition can be used giving

$$\mathrm{QAP} = \begin{pmatrix} r_{11} & \cdots & r_{1m} \\ \vdots & \ddots & \vdots \\ 0 & \cdots & r_{mm} \\ 0 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{pmatrix}$$

where for each $k = 1, \cdots, m$

$$r_{kk}^2 \geqq r_{kj}^2 + \cdots + r_{jj}, \qquad j = k+1, \cdots, m,$$

and P is a permutation. The algorithm can be modified to try to keep this ordering during the process. Then, however, the global convergence problem is open, at least theoretically.

4. The generalised Jacobi singular value decomposition. The Jacobi GSVC algorithm is based on the reduction formulae for 2×2 matrices A, B:

$$\begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \begin{pmatrix} a_{ll} & a_{lm} \\ a_{ml} & a_{mm} \end{pmatrix} \begin{pmatrix} 1 & \alpha \\ -\gamma & 1 \end{pmatrix} = \begin{pmatrix} a'_{ll} & 0 \\ 0 & a'_{mm} \end{pmatrix}, \tag{4.1}$$

$$\begin{pmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{pmatrix} \begin{pmatrix} b_{ll} & b_{lm} \\ b_{ml} & b_{mm} \end{pmatrix} \begin{pmatrix} 1 & \alpha \\ -\gamma & 1 \end{pmatrix} = \begin{pmatrix} b'_{ll} & 0 \\ 0 & b'_{mm} \end{pmatrix}, \tag{4.2}$$

which gives four equations for the unknown quantities φ, ψ, α, γ.

$$(\alpha a_{ll} + a_{lm}) \cos\varphi - (a_{mm} + \alpha a_{ml}) \sin\varphi = 0, \tag{4.3}$$

$$(a_{ml} - \gamma a_{mm}) \cos\varphi + (a_{ll} - \gamma a_{lm}) \sin\varphi = 0, \tag{4.4}$$

$$(\alpha b_{ll} + b_{lm}) \cos\psi - (b_{mm} + \alpha b_{ml}) \sin\psi = 0, \tag{4.5}$$

$$(b_{ml} - \gamma b_{mm}) \cos\psi + (b_{ll} - \gamma b_{lm}) \sin\psi = 0. \tag{4.6}$$

By first eliminating the variables ψ and φ equations similar to those of [1] are obtained and it follows that

$$\alpha q - \gamma p = 0, \tag{4.7}$$

with

$$q = (b_{ll}b_{lm} + b_{ml}b_{mm})(a_{ll}^2 + a_{ml}^2) - (a_{ll}a_{lm} + a_{ml}a_{mm})(b_{ll}^2 + b_{ml}^2), \tag{4.8}$$

$$p = (b_{ll}b_{lm} + b_{ml}b_{mm})(a_{lm}^2 + a_{mm}^2)(-a_{ll}a_{lm} + a_{ml}a_{mm})(b_{lm}^2 + b_{mm}^2), \tag{4.9}$$

$$\gamma = (a_{ll}^2 + a_{ml}^2)(b_{lm}^2 + b_{mm}^2)(-a_{lm}^2 + a_{mm}^2)(b_{ll}^2 + b_{ml}^2). \tag{4.10}$$

It can be shown that in any case

$$r^2 + 4pq \geqq 0$$

[6] The proofs of these facts can be found in [6], [7].

holds. Now α, β are computed as follows:

Case I: $|q|+|r|>0$ and $|p|+|r|>0$. Then

$$\alpha=\frac{p}{x}, \qquad \gamma=\frac{q}{x}, \tag{4.11}$$

$$x=\left(\frac{|r|}{2}+\sqrt{\frac{r^2}{4}+pq}\right)\operatorname{sign} r \qquad \text{(with e.g. sign } 0=+1\text{)}. \tag{4.12}$$

Case II: $p=r=q=0$. (It can be easily seen that the conditions $p=r=0$ and $q=r=0$ are equivalent so that Cases I and II cover all possibilities.)

In this case we can always take $\alpha\gamma=0$ which has computational advantages.

If

$$a_{lm}^2+a_{mm}^2+b_{lm}^2+b_{mm}^2\neq 0$$

then

$$\alpha=0, \qquad \gamma=\frac{\sigma}{\tau}, \tag{4.13}$$

$$\begin{pmatrix}\sigma\\ \tau\end{pmatrix}\in\left\{\begin{pmatrix}a_{lm}a_{ll}+a_{mm}a_{ml}\\ a_{lm}^2+a_{mm}^2\end{pmatrix},\begin{pmatrix}b_{lm}b_{ll}+b_{mm}b_{ml}\\ b_{lm}^2+b_{mm}^2\end{pmatrix}\right\} \tag{4.14}$$

where the larger $|\sigma|+|\tau|$ has to be taken.

If

$$a_{ll}^2+a_{ml}^2+b_{ll}^2+b_{ml}^2\neq 0$$

then

$$\gamma=0, \qquad \alpha=\frac{\sigma}{\tau}, \tag{4.15}$$

$$\begin{pmatrix}\sigma\\ \tau\end{pmatrix}\in\left\{\begin{pmatrix}-(a_{lm}a_{ll}+a_{mm}a_{ml})\\ a_{ll}^2+a_{ml}^2\end{pmatrix},\begin{pmatrix}-(b_{lm}b_{ll}+b_{mm}b_{ml})\\ b_{ll}^2+b_{ml}^2\end{pmatrix}\right\} \tag{4.16}$$

where again the greater $|\sigma|+|\tau|$ is taken.

If

$$a_{lm}^2+a_{mm}^2+b_{lm}^2+b_{mm}^2=0 \quad \text{or} \quad a_{ll}^2+a_{ml}^2+b_{ll}^2+b_{ml}^2=0$$

then

$$\alpha=\gamma=0. \tag{4.17}$$

In all cases above the matrix

$$\begin{pmatrix}1 & \alpha\\ -\gamma & 1\end{pmatrix}$$

is nonsingular. For φ and ψ we have the following formulae:

$$\tan\varphi=\frac{\sigma}{\tau}, \tag{4.18}$$

$$\begin{pmatrix}\sigma\\ \tau\end{pmatrix}\in\left\{\begin{pmatrix}\alpha a_{ll}+a_{lm}\\ a_{mm}+\alpha a_{ml}\end{pmatrix},\begin{pmatrix}-a_{ml}+\gamma a_{mm}\\ a_{ll}-\gamma a_{lm}\end{pmatrix}\right\}, \tag{4.19}$$

$$\tan\psi=\frac{\sigma}{\tau}, \tag{4.20}$$

with

$$\begin{pmatrix}\sigma\\ \tau\end{pmatrix} \in \left\{ \begin{pmatrix} \alpha b_{ll} + b_{lm} \\ b_{mm} + \alpha b_{lm} \end{pmatrix}, \begin{pmatrix} -b_{ml} + \gamma b_{mm} \\ b_{ll} - \gamma b_{lm} \end{pmatrix} \right\}. \tag{4.21}$$

Here again the greater value $|\sigma|+|\tau|$ is chosen.[7] By the nonsingularity of the matrix

$$\begin{pmatrix} 1 & \alpha \\ -\gamma & 1 \end{pmatrix}$$

the expressions (4.18) or (4.20) are well defined unless $A=0$ or $B=0$ in which case $\varphi=0$ or $\psi=0$ has to be taken.

The above formulae are somewhat simplified if the matrices A, B in (1.2) are triangular. In this case both matrices can be stored essentially in one square array. Here, too, the initial triangularity halves the usual operational and storage effort. The initial triangularity is easily obtained by a direct reduction. If both matrices are rectangular of the type $n \times m (n > m)$ then this reduction is unavoidable and the initial triangularity is obtained at no extra cost. The ordering of diagonal zeros is much more complicated here.

We shall not write here the set of transformation formulae for an upper triangular pair A, B; they are analogous to those in § 3.

For this algorithm the theory is lacking completely but it can be expected that it has many properties of the symmetric algorithm of Falk and Langemeyer [1] which is now well understood [5], [8].

5. Numerical results. As yet only preliminary results are available concerning the Jacobi SVD algorithm described in § 3. We have compared our algorithm with the QR algorithm of Golub and Reinsch [4] on square matrices. In the Jacobi algorithm we have applied fast rotation formulae. On matrices of the order 10–100 our algorithm was 2–3 times slower than the QR algorithm of Golub and Reinsch [4].

We give some tables of typical time relations for matrices of the order $n = 10, 20, 30, 50, 100$.

(1) Random matrices with elements between -1 and 1

JAC/QR	2,5	2,6	2,7	3	3
n	10	20	30	50	100

(2) Random matrices with the rank $n/2$ [8]

JAC/QR	1,9	1,9	1,8	1,8	1,8
n	10	20	30	50	100

(3) Random matrices with the rank $n/10$ [8]

JAC/QR	1,9	1,2	1,1	1,1	1,03
n	10	20	30	50	100

In all experiments both left and right eigenvectors were computed. Without eigenvectors the relative efficiency of the Jacobi method is somewhat weaker.

[7] Note that if both quotients exist, then they are equal. The same is true of (4.14), (4.16) and (4.19).

[8] The matrices were obtained as sums of dyadics whose generating vectors had random components between -1 and 1.

The increased efficiency of the Jacobi method for rank deficient matrices is greatly due to the following features:

(i) The initial triangulation is made with column pivoting;

(ii) Small pivot elements (both diagonal and off-diagonal) are substituted by exact zeros.

This made the deflation steps, described by (3.19), completely superfluous. The quadratic convergence of the Jacobi method was observed in all cases.

On matrices which were initially triangular the relative efficiency of the Jacobi method was slightly better for low dimensions ($n < 50$).

Of course, if applied to $n \times m$ matrices the Jacobi method gets faster than QR very soon (with growing n/m) because the initial bidiagonalization of such matrices is more expensive than their triangularization. But in such cases an initial triangularization can be made for the Golub–Reinsch algorithm as well, and we omit this type of comparison here.

Further gains in the relative efficiency of the Jacobi method are obtained on diagonally dominant matrices appearing, e.g., in small parameter analysis or in subspace iteration decouplings. In any case, according to our experiments the superiority of QR over Jacobi on SVD seems to be somewhat less expressed than that on the symmetric eigenvalue decomposition.

Much less definitive is our experience with the Jacobi GSVD although a running code exists and works satisfactorily. More exhaustive results on this algorithm will be presented in a later work.

Upon finishing this manuscript the authors became aware of the work of Brent, Luk and van Loan [17] and Luk [18] where different parallel strategies are investigated. A different triangularity-preserving GSVD algorithm was considered by C. Paige [19].

Acknowledgments. The authors thank the editor and the anonymous referees for their substantial and constructive comments and suggestions. They also thank H. Deichmöller, Hagen, for implementing ALGOL programs used in the numerical experiments.

REFERENCES

[1] S. FALK AND P. LANGEMEYER, *Das Jacobische Rotations-verfahren für reellsymmetrische Matrizenspaare*, Elektronische Datenverarbeitung, 1960, pp. 30–34.

[2] G. E. 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.

[3] G. GOLUB AND W. KAHAN, *Calculating the singular values and pseudoinverse of a matrix*, SIAM J. Numer. Anal., 2 (1965), pp. 205–224.

[4] G. GOLUB AND C. REINSCH, *Singular value decomposition and least squares solution*, Numer. Math., 14 (1970), pp. 403–420.

[5] G. GOSE, *Das Jacobi Verfahren für $Ax = \lambda Bx$*, Z. Angew. Math. Mech., 59 (1979), pp. 93–101.

[6] V. HARI, *On the quadratic convergence of Jacobi algorithms*, Radovi Matematički, 2(1) (1986), pp. 127–146.

[7] ———, *On pairs of almost diagonal matrices*, Uni. of Zagreb, Zagreb, Yugoslavia, 1985.

[8] ———, *On cyclic Jacobi method for the positive definite generalized eigenvalue problems*, Ph.D. thesis, University of Hagen, Hagen, W. Germany, 1984.

[9] B. PARLETT, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[10] H. RUTISHAUSER, *The Jacobi method for real symmetric matrices*, in Handbook for Automatic Computation (Linear Algebra), Springer, Berlin, 1971, pp. 202–211.

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

[12] G. W. STEWART, *A Method for Computing and Generalized Singular Value Decomposition in Matrix Pencils*, Lecture Notes in Mathematics 973, Springer, Berlin, 1983, pp. 207–220.

[13] G. W. STEWART, *Computing the* CS *decomposition of a partitioned orthogonal matrix*, Numer. Math. (1982), pp. 297-306.
[14] V. V. VOEVODIN, *Čislennye metody Algebry*, Nauka, Moskva, 1966.
[15] J. H. WILKINSON, *Almost diagonal matrices with multiple or close eigenvalues*, Linear Algebra Appl., 1 (1968), pp. 1-12.
[16] ———, *Note on the quadratic convergence of the cyclic Jacobi process*, Numer. Math., 4 (1962), pp. 296-300.
[17] 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.
[18] F. LUK, *A parallel method for computing the generalized singular value decomposition*, J. Parallel and Distributed Computing, 2 (1985), pp. 250-260.
[19] C. C. PAIGE, *Computing the generalized singular value decomposition*, this Journal, to appear.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 5, September 1987
© 1987 Society for Industrial and Applied Mathematics
007

TIME-DEPENDENT BOUNDARY AND INTERIOR FORCING IN LOCALLY ONE-DIMENSIONAL SCHEMES*

D. A. SWAYNE†

Abstract. This paper is concerned with the numerical solution of parabolic partial differential equations with Dirichlet boundary conditions by locally one-dimensional (LOD) methods. When a corresponding system of ordinary differential equations is developed for the numerical time-integration, the prescribed values of the solution (boundary values) and any interior forcing (resulting either from local linearization or from interior source terms in the original equations) must be treated differently to prevent loss in the order of accuracy of the numerical solution as compared to a homogeneous (unforced) problem. A stable second-order scheme and a global extrapolation to third order are demonstrated.

Key words. numerical analysis, parabolic partial differential equations, splitting methods, method of lines, LOD methods

AMS(MOS) subject classifications. 65L05, 65M05, 65M20

1. Introduction. This paper investigates some of the reasons for the difficulty of maintaining the accuracy in the numerical solutions of parabolic equations with time-dependent Dirichlet boundary conditions, when locally one-dimensional (LOD) techniques are employed.

The investigation arose from consideration of an alternative formulation for LOD algorithms for the solution of linear differential equations such as those used in the approximation of the heat equation, $u_t = \nabla^2 u$, by the method of lines (e.g. Zafarullah (1970)). For such equations, approximations may be derived by superimposing a uniform grid over the domain of interest. On such a grid, the spatial operator is replaced by a discrete (finite-difference) operator which produces an ordinary differential equation $(d/dt)\mathbf{u} = \mathbf{L}\mathbf{u}$. Thus over the given domain, the semidiscrete solution $\mathbf{u}$ is given by

$$\mathbf{u}(t+\tau) = \exp(\tau\mathbf{L})\mathbf{u}(t) \tag{1.1}$$

where τ is some convenient time-step.

It is frequently possible to split the matrix $\mathbf{L}$ into two or more constituents $\mathbf{L} = \mathbf{L}_1 + \mathbf{L}_2$, where $\mathbf{L}_1$ and $\mathbf{L}_2$ result in physically meaningful problems which can be solved and recombined into an approximate solution to (1.1). One must assume $\mathbf{L}_1\mathbf{L}_2 \neq \mathbf{L}_2\mathbf{L}_1$. This strategy has a payoff whenever the numbers of operations and the associated computing overhead for the one-dimensional strategy produces numerical solutions with the same accuracy in a shorter time than similar numerical methods associated with the "full" matrix $\mathbf{L}$. LOD methods are now classical (e.g., Gourlay and Mitchell (1972), Mitchell and Griffiths (1980), and Yanenko (1970)). However, there has remained an outstanding problem with such methods, namely a general treatment of time-dependent boundary conditions so that no loss of accuracy occurs. Yanenko (1970) states that the LOD method is only $O(\tau^{0.25})$ accurate when time-dependent boundary conditions are present.

* Received by the editors August 9, 1984; accepted for publication (in revised form) November 13, 1986. This work was supported by the United Kingdom Science and Engineering Research Council in the form of a Visiting Fellowship to the University of Kent at Canterbury.

† Department of Computing and Information Science, University of Guelph, Guelph, Ontario, Canada, N1G2W1.

We propose an approach which corrects the past defect, and permits flexibility in the evaluation of the forcing terms. There are many cases for which freedom in the location of forcing evaluation is advantageous. We can evaluate the forcing terms only for main time-points $\{t_i, i = 1 \cdots N\}$ where accurate values of $\mathbf{u}$ are to be calculated. One situation involves nonlinear equations, where the (possibly nonlinear) semidiscretization $(d/dt)\mathbf{u} = \mathbf{f}(t, \mathbf{u})$ is linearized to produce $(d/dt)\mathbf{u} = \mathbf{Lu} + (\mathbf{f}(t, \mathbf{u}) - \mathbf{Lu})$. A stabilized method of predictor-corrector type becomes feasible; its solution, at time-points for which $\mathbf{u}$ is known, is used to predict $\mathbf{u}(t+\tau)$, and one or more corrector iterations are applied.

The process of developing numerical schemes involves the replacement of $\exp(\tau\mathbf{L})$ in (1.1) by combinations of $\exp(\tau\mathbf{L}_1)$ and $\exp(\tau\mathbf{L}_2)$ in such a way as to obtain **order** in the approximation to $\exp(\tau\mathbf{L})$. For example, it is known that

$$\exp(\tau\mathbf{L}) = \tfrac{1}{2}[\exp(\tau\mathbf{L}_1)\exp(\tau\mathbf{L}_2) + \exp(\tau\mathbf{L}_2)\exp(\tau\mathbf{L}_1)] + \mathbf{O}(\tau^3). \tag{1.2}$$

We say that (1.2) defines a **second-order** approximation to $\exp(\tau\mathbf{L})$ because $\mathbf{O}(1)$, $\mathbf{O}(\tau)$ and $\mathbf{O}(\tau^2)$ terms match on both sides of (1.2).

This approximation to $\exp(\tau\mathbf{L})$ was investigated initially because of the inherent parallelism in the two terms on the right-hand side of (1.2). A more common LOD splitting, defined by

$$\exp(\tau\mathbf{L}) = \exp\left(\frac{\tau}{2}\mathbf{L}_1\right)\exp\left(\frac{\tau}{2}\mathbf{L}_2\right)\exp\left(\frac{\tau}{2}\mathbf{L}_2\right)\exp\left(\frac{\tau}{2}\mathbf{L}_1\right) + \mathbf{O}(\tau^3), \tag{1.3}$$

results in numerical schemes which (without correction terms) have poor convergence as $\tau \to 0$ near the boundaries of the solution domain when forcing is time-dependent. Interior forcing terms are integrated to full accuracy. The schemes derived from (1.2) have the opposite shortcoming. Boundary conditions are accurately incorporated into the numerical solution. It is, rather, the forcing terms affecting equations in the interior of the solution domain which behave poorly. An explanation of this behavior follows, and corrections for methods based on formula (1.2) are developed.

2. An example of boundary forcing. In this section we consider time-dependent Dirichlet boundary conditions for the linear heat conduction equation.

A forced linear differential equation,

$$\frac{d}{dt}\mathbf{u} = \mathbf{Lu} + \mathbf{f}(t), \tag{2.1}$$

has an equivalent representation in integral form:

$$\mathbf{u}(t+\tau) = \exp(\tau\mathbf{L})\mathbf{u}(t) + \tau\int_0^1 \exp(\tau(1-s)\mathbf{L})\mathbf{f}(t+s\tau)\,ds, \tag{2.2}$$

where $\mathbf{u}(t)$ is known and $\mathbf{u}(t+\tau)$ is to be computed. The first term on the right-hand side of (2.2) is called the **complementary function** and it represents the solution to the unforced or **homogeneous** problem ($\mathbf{f} \equiv 0$). The second term of (2.2) is called the **particular integral** and represents a solution which is identically zero at the initial moment, and which satisfies the forced or **inhomogeneous** differential equation ($\mathbf{f} \neq 0$). Our treatment of boundary conditions proceeds as follows. The (known) boundary values for the heat equation are incorporated into inhomogeneous terms through a change in the normal definition of the coefficient matrix $\mathbf{L}$ and the complementary function term is given boundary values identically zero. (Forcing located in the interior of the solution domain is not affected by this discussion, and will be considered as a separate issue.)

As a concrete example, we consider a simple domain with a uniform grid, mesh size h, superimposed along (x, y) coordinate directions. The usual central difference approximation to $\nabla^2 u$ is applied on the grid to give a spatial semidiscretization of the original equation. The unknowns $\{U_{ij},\ i=1,2,\ j=1,2\}$ together with the boundary values are laid out in a table thus:

$$\begin{pmatrix} \cdot & U_{13} & U_{23} & \cdot \\ U_{02} & U_{12} & U_{22} & U_{32} \\ U_{01} & U_{11} & U_{21} & U_{31} \\ \cdot & U_{10} & U_{20} & \cdot \end{pmatrix}.$$

If U_{ij} is a boundary point, the boundary condition is given by $f(t, x_i, y_j) = f_{ij}(t)$. If we organize the solution components into *interior points*

$$\mathbf{U}_I = (U_{11}, U_{21}, U_{12}, U_{22})^T$$

and *boundary points*

$$\mathbf{U}_{1,b} = (U_{01}, U_{31}, U_{02}, U_{32})^T, \qquad \mathbf{U}_{2,b} = (U_{10}, U_{20}, U_{13}, U_{23})^T$$

and define $\mathbf{U}$ by

$$\mathbf{U} = \begin{pmatrix} \mathbf{U}_I \\ \mathbf{U}_{1,b} \\ \mathbf{U}_{2,b} \end{pmatrix}.$$

the discretized differential equation can be written

$$\frac{d}{dt}\mathbf{U} = \mathbf{L}(\mathbf{U} + \mathbf{f}(t)), \tag{2.3}$$

with initial conditions given by

$$\mathbf{U}(0) = (\mathbf{U}_I(0)^T \mathbf{0}_{1\times 8})^T. \tag{2.4}$$

The coefficient matrix $\mathbf{L}$ is built up from

$$\mathbf{L}^{11} = h^{-2}\begin{pmatrix} -4 & 1 & 1 & 0 \\ 1 & -4 & 0 & 1 \\ 1 & 0 & -4 & 1 \\ 0 & 1 & 1 & -4 \end{pmatrix} \tag{2.5}$$

and

$$\mathbf{L}^{12} = h^{-2}\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.6}$$

Then

$$\mathbf{L} = \begin{pmatrix} \mathbf{L}^{11} & \mathbf{L}^{12} \\ \mathbf{0}_{8\times 12} \end{pmatrix}$$

The forcing function is made up of

$$\mathbf{f}_1 = (\mathbf{0}_{1\times 4}, f_{01}, f_{31}, f_{02}, f_{32}, \mathbf{0}_{1\times 4})^T$$

and

$$\mathbf{f}_2 = (\mathbf{0}_{1\times 8}, f_{10}, f_{20}, f_{13}, f_{23})^T,$$

and $\mathbf{f}$ is defined by

$$\mathbf{f}(t) = \mathbf{f}_1(t) + \mathbf{f}_2(t). \tag{2.7}$$

We define $\mathbf{L}_1$ by

$$\mathbf{L}_1^{11} = h^{-2}\begin{pmatrix} -2 & 1 & 0 & 0 \\ 1 & -2 & 0 & 0 \\ 0 & 0 & -2 & 2 \\ 0 & 0 & 1 & -2 \end{pmatrix}, \qquad \mathbf{L}_1^{12} = h^{-2}(\mathbf{I}_{4\times 4}\mathbf{0}_{4\times 4})$$

and

$$\mathbf{L}_1 = \begin{pmatrix} \mathbf{L}_1^{11} & \mathbf{L}_1^{12} \\ \mathbf{0}_{8\times 12} \end{pmatrix}.$$

Finally, $\mathbf{L}_2 = \mathbf{L} - \mathbf{L}_1$. One can verify the following:

$$\mathbf{L}\mathbf{f} = \mathbf{L}_1\mathbf{f}_1 + \mathbf{L}_2\mathbf{f}_2, \tag{2.8}$$

where

$$\mathbf{L}_1\mathbf{f}_2 = \mathbf{L}_2\mathbf{f}_1 = \mathbf{0}. \tag{2.9}$$

For problems similar in structure to our simple example, (2.3) is split into two one-dimensional equations:

$$\frac{1}{2}\frac{d}{dt}\mathbf{U} = \mathbf{L}_1(\mathbf{U} + \mathbf{f}_1) \tag{2.10}$$

and

$$\frac{1}{2}\frac{d}{dt}\mathbf{U} = \mathbf{L}_2(\mathbf{U} + \mathbf{f}_2). \tag{2.11}$$

The reader should note that this layout of the coefficient matrix does not result in an increase in the effective size of the computation. We have not added m unknowns to the problem (m being the number of boundary points). Rather we are attempting to simplify splitting of some of the (known) boundary values in the one-dimensional subproblems. The typical linear equations that will be solved will be similar in size to other implementations of LOD splitting, but with correct boundary terms.

DEFINITION 1. For a differential equation which may be split into locally one-dimensional subproblems, we define **separable boundary forcing** to be a forcing function $\mathbf{f}(t)$ in $(d/dt)\mathbf{U} = \mathbf{L}(\mathbf{U} + \mathbf{f}(t))$ for which $\mathbf{L} = \mathbf{L}_1 + \mathbf{L}_2$ and $\mathbf{f}(t) = \mathbf{f}_1(t) + \mathbf{f}_2(\tau)$ satisfy (2.8) and (2.9).

In the next section, quadrature formulae are developed for the integral term in (2.2), and the formulae developed are applied to boundary forcing in a locally one-dimensional scheme. These quadratures are then shown to integrate boundary forcing correctly in LOD schemes defined by (1.2).

3. One-dimensional approximations. We construct numerical methods from (2.2) by replacing the integral by a finite sum (quadrature rule). Following Lawson (1972), we write

$$\tau \int_0^1 \exp(\tau(1-s)\mathbf{L})\mathbf{f}(t+s\tau)\,ds = \tau[\mathbf{W}_1\mathbf{f}(t+\tau s_1)+\mathbf{W}_2\mathbf{f}(t+\tau s_2)]+\mathbf{O}(\tau^3). \tag{3.1}$$

The exponential is then replaced by an approximation $\mathbf{E}(\tau\mathbf{L})$, to give

$$\mathbf{U}(t+\tau) = \mathbf{E}(\tau\mathbf{L})\mathbf{U}(t)+\tau[\mathbf{W}_1\mathbf{f}(t+\tau s_1)+\mathbf{W}_2\mathbf{f}(t+\tau s_2)]. \tag{3.2}$$

To derive a quadrature rule for particular abscissae $\{s_1, s_2\}$, we solve

$$W_1(z)+W_2(z) = M_0(z), \tag{3.3a}$$

$$s_1 W_1(z)+s_2 W_2(z) = M_1(z). \tag{3.3b}$$

(The scalar parameter z is replaced by the matrix argument $\tau\mathbf{L}$ to define $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{M}_0$ and $\mathbf{M}_1$ in the numerical method.) The right-hand sides of (3.3) are derived from integration by parts of constant and linear forcing:

$$M_0(z) = z^{-1}(E(z)-1), \tag{3.4a}$$

and

$$M_1(z) = z^{-1}(M_0(z)-1). \tag{3.4b}$$

Any exponential approximation $E(z)$ which is at least second order in z may be used to develop a method of LOD type using formula (1.2), but we examine in detail only the exponential approximation $P_{11}(z) = (1-z/2)^{-1}(1+z/2)$.

The moment equations (3.4) are particularly simple for this approximation, namely:

$$M_0(z) = \left(1-\frac{z}{2}\right)^{-1} \quad \text{and} \quad M_1(z) = \left(1-\frac{z}{2}\right)^{-1}\frac{1}{2}.$$

The numerators of M_0 and M_1 are independent of the matrix argument, and so P_{11} is a special case for which $s_1 = s_2 = \frac{1}{2}$ satisfies (3.3). (Most exponential approximations produce quadrature weights which cannot satisfy (3.3) in a one-point rule.) For the mid-point rule together with P_{11}, our numerical method becomes

$$\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}\right)\mathbf{U}(t+\tau) = \left(\mathbf{I}+\frac{\tau}{2}\mathbf{L}\right)\mathbf{U}(t)+\tau\mathbf{f}\left(t+\frac{\tau}{2}\right). \tag{3.5}$$

Noting that $(1-z/2)^{-1}(1+z/2) \equiv 2(1-z/2)^{-1}-1$, we rewrite (3.5) as

$$\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}\right)\mathbf{U}^* = \mathbf{U}(t)+\frac{\tau}{2}\mathbf{f}\left(t+\frac{\tau}{2}\right), \qquad \mathbf{U}(t+\tau) = 2\mathbf{U}^*-\mathbf{U}(t). \tag{3.6}$$

We may also apply $s_1 = 0$ and $s_2 = 1$ (a "trapezoidal rule") which results in

$$\left(\mathbf{I} - \frac{\tau}{2}\mathbf{L}\right)\mathbf{U}^* = \mathbf{U}(t) + \frac{\tau}{4}(\mathbf{f}(t) + \mathbf{f}(t+\tau)), \qquad \mathbf{U}(t+\tau) = 2\mathbf{U}^* - \mathbf{U}(t). \tag{3.6'}$$

We now implement the LOD scheme defined by (1.2), using (3.5). Following (2.10) and (2.11) we consider matrix arguments for the one-dimensional calculations to be $2\mathbf{L}_1$ and $2\mathbf{L}_2$, with time-step $\tau/2$. The results, for boundary forcing, will be shown to be equivalent to within $O(\tau^3)$ to a two-dimensional two-point rule with $s_1 = \frac{1}{4}$ and $s_2 = \frac{3}{4}$.

We proceed, setting $\mathbf{f}_i(\alpha) = \mathbf{f}_i(t + \alpha\tau)$:

$$\left(\mathbf{I} - \frac{\tau}{2}\mathbf{L}_1\right)\mathbf{U}_1 = \left(\mathbf{I} + \frac{\tau}{2}\mathbf{L}_1\right)\mathbf{U}(t) + \tau\mathbf{L}_1\mathbf{f}_1\left(\frac{1}{4}\right), \tag{3.7a}$$

$$\left(\mathbf{I} - \frac{\tau}{2}\mathbf{L}_2\right)\mathbf{U}_2 = \left(\mathbf{I} + \frac{\tau}{2}\mathbf{L}_2\right)\mathbf{U}(t) + \tau\mathbf{L}_2\mathbf{f}_2\left(\frac{1}{4}\right), \tag{3.7b}$$

$$\left(\mathbf{I} - \frac{\tau}{2}\mathbf{L}_2\right)\mathbf{U}_{21} = \left(\mathbf{I} + \frac{\tau}{2}\mathbf{L}_2\right)\mathbf{U}_1 + \tau\mathbf{L}_2\mathbf{f}_2\left(\frac{3}{4}\right), \tag{3.7c}$$

$$\left(\mathbf{I} - \frac{\tau}{2}\mathbf{L}_1\right)\mathbf{U}_{12} = \left(\mathbf{I} + \frac{\tau}{2}\mathbf{L}_1\right)\mathbf{U}_2 + \tau\mathbf{L}_1\mathbf{f}_1\left(\frac{3}{4}\right), \tag{3.7d}$$

and, finally,

$$\mathbf{U}(t+\tau) = \tfrac{1}{2}(\mathbf{U}_{12} + \mathbf{U}_{21}). \tag{3.7e}$$

We substitute (3.7a)-(3.7d) into (3.7e) and collect terms multiplying $\mathbf{U}(t)$, $\mathbf{f}_i(\frac{1}{4})$, and $\mathbf{f}_i(\frac{3}{4})$:

$$\mathbf{U}(t+\tau) = \mathbf{E}\mathbf{U}(t) + \tau\{\mathbf{T}_1\mathbf{f}(\tfrac{1}{4}) + \mathbf{T}_2\mathbf{f}(\tfrac{3}{4})\},$$

where

$$\begin{aligned}\mathbf{E}\mathbf{U}(t) &= \mathbf{E}(\tau, \mathbf{L}_1, \mathbf{L}_2)\mathbf{U}(t) \\ &= \tfrac{1}{2}(\mathbf{P}_{11}(\tau\mathbf{L}_1)\mathbf{P}_{11}(\tau\mathbf{L}_2) + \mathbf{P}_{11}(\tau\mathbf{L}_2)\mathbf{P}_{11}(\tau\mathbf{L}_1))\mathbf{U}(t),\end{aligned} \tag{3.8}$$

$$\begin{aligned}\mathbf{T}_1\mathbf{f}\left(\frac{1}{4}\right) &= \frac{1}{2}\left(\mathbf{I} - \frac{\tau}{2}\mathbf{L}_2\right)^{-1}\left(\mathbf{I} + \frac{\tau}{2}\mathbf{L}_2\right)\left(\mathbf{I} - \frac{\tau}{2}\mathbf{L}_1\right)^{-1}\mathbf{L}_1\mathbf{f}_1\left(\frac{1}{4}\right) \\ &\quad + \frac{1}{2}\left(\mathbf{I} - \frac{\tau}{2}\mathbf{L}_1\right)^{-1}\left(\mathbf{I} + \frac{\tau}{2}\mathbf{L}_1\right)\left(\mathbf{I} - \frac{\tau}{2}\mathbf{L}_2\right)^{-1}\mathbf{L}_2\mathbf{f}_2\left(\frac{1}{4}\right),\end{aligned} \tag{3.9}$$

and

$$\mathbf{T}_2\mathbf{f}\left(\frac{3}{4}\right) = \frac{1}{2}\left[\left(\mathbf{I} - \frac{\tau}{2}\mathbf{L}_1\right)^{-1}\mathbf{L}_1\mathbf{f}_1\left(\frac{3}{4}\right) + \left(\mathbf{I} - \frac{\tau}{2}\mathbf{L}_2\right)^{-1}\mathbf{L}_2\mathbf{f}_2\left(\frac{3}{4}\right)\right]. \tag{3.10}$$

We see immediately that formula (3.8) defines a particular case of (1.2).

Now we expand the moment equations (3.3) for (3.8) directly. Because $\mathbf{L}$ is a singular matrix, it is necessary to work with $\mathbf{M}_0\tau\mathbf{L}$ and $\mathbf{M}_1\tau\mathbf{L}^2$. We first expand $\mathbf{M}_0\tau\mathbf{L} \equiv \mathbf{E}(\tau, \mathbf{L}_1, \mathbf{L}_2) - \mathbf{I}$, using the partial fraction form of P_{11}:

$$\begin{aligned}\mathbf{E}(\tau, \mathbf{L}_1, \mathbf{L}_2) - \mathbf{I} &= \frac{1}{2}\left(2\left(\mathbf{I} - \frac{\tau}{2}\mathbf{L}_1\right)^{-1} - \mathbf{I}\right)\left(2\left(\mathbf{I} - \frac{\tau}{2}\mathbf{L}_2\right)^{-1} - \mathbf{I}\right) \\ &\quad + \frac{1}{2}\left(2\left(\mathbf{I} - \frac{\tau}{2}\mathbf{L}_2\right)^{-1} - \mathbf{I}\right)\left(2\left(\mathbf{I} - \frac{\tau}{2}\mathbf{L}_1\right)^{-1} - \mathbf{I}\right) - \mathbf{I},\end{aligned}$$

which simplifies to

$$(\mathbf{E}-\mathbf{I}) = 2\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1} + 2\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1}$$
$$-2\left(\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1} + \left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1}\right)$$

and eventually to

$$(\mathbf{E}-\mathbf{I}) = \left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1}\tau\mathbf{L}_2 + \left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1}\tau\mathbf{L}_1.$$

The first moment equation is applied to **separable boundary forcing** function $\mathbf{f} \equiv \mathbf{f}(t)$ (Definition 1). There results

$$\text{(3.11)}\quad \mathbf{M}_0\tau\mathbf{L}\mathbf{f} = (\mathbf{E}-\mathbf{I})\mathbf{f} = \left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1}\tau\mathbf{L}_2\mathbf{f}_2 + \left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1}\tau\mathbf{L}_1\mathbf{f}_1.$$

We now examine $\mathbf{M}_1\tau\mathbf{L}^2$, which must be derived from $\mathbf{M}_1\tau\mathbf{L}\mathbf{f}(t)$. The singularity of $\mathbf{L}$ in our formulation of the heat equation, for example, makes it impossible to assume that there exists a $\mathbf{g}(t)$ satisfying $\mathbf{L}\mathbf{g} = \mathbf{f}$. However, for the semidiscretized heat equation, a solution exists for

$$\text{(3.12)}\qquad \mathbf{L}^2\mathbf{g}(t) = \mathbf{L}\mathbf{f}(t).$$

We need an expression for $\mathbf{g}(t)$ only for t-values corresponding to s_1 and s_2, since $\mathbf{g}$ is again replaced by $\mathbf{f}$ in the derivation of $\mathbf{M}_1$ applied to separable boundary forcing. For a fixed t-value, if $\mathbf{g}$ is partitioned into components defined on boundary points, $\mathbf{g}_b$, and interior points, $\mathbf{g}_I$, then $\mathbf{g}_b$ may be freely chosen and $\mathbf{g}_I$ satisfies $\mathbf{L}^{11}\mathbf{L}^{11}\mathbf{g}_I = \mathbf{L}^{12}\mathbf{f}_b - \mathbf{L}^{11}\mathbf{L}^{12}\mathbf{g}_b$. Here $\mathbf{L}^{11}$ and $\mathbf{L}^{12}$ are given by (2.5) and (2.6), and $\mathbf{f}_b$ denotes the restriction of $\mathbf{f}$ on the boundary points. We choose $\mathbf{g}_b \equiv \mathbf{0}$, to get $\mathbf{L}^{11}\mathbf{L}^{11}\mathbf{g}_I = \mathbf{L}^{12}\mathbf{f}_b$.

For separable forcing, we may similarly define components $\mathbf{g}_1(t)$ and $\mathbf{g}_2(t)$ of $\mathbf{g}$ satisfying $\mathbf{L}_1^{11}\mathbf{L}^{11}\mathbf{g}_{1,I} = \mathbf{L}_1^{12}\mathbf{f}_{1,b}$ and $\mathbf{L}_2^{11}\mathbf{L}^{11}\mathbf{g}_{2,I} = \mathbf{L}_2^{12}\mathbf{f}_{2,b}$ and note that

$$\mathbf{g}(t) = \mathbf{g}_1(t) + \mathbf{g}_2(t).$$

$\mathbf{g}_1$ and $\mathbf{g}_2$ also satisfy

$$\text{(3.13)}\qquad \mathbf{L}_1(\mathbf{L}_1+\mathbf{L}_2)\mathbf{g}_1 = \mathbf{L}_1\mathbf{f}_1,$$

$$\text{(3.14)}\qquad \mathbf{L}_2(\mathbf{L}_1+\mathbf{L}_2)\mathbf{g}_1 = \mathbf{O},$$

$$\text{(3.15)}\qquad \mathbf{L}_2(\mathbf{L}_1+\mathbf{L}_2)\mathbf{g}_2 = \mathbf{L}_2\mathbf{f}_2$$

and

$$\text{(3.16)}\qquad \mathbf{L}_1(\mathbf{L}_1+\mathbf{L}_2)\mathbf{g}_2 = \mathbf{O}.$$

From (3.13)-(3.16), we obtain

$$\text{(3.17)}\qquad (\mathbf{L}_1\mathbf{L}_2+\mathbf{L}_1^2)\mathbf{g} = \mathbf{L}_1\mathbf{f}_1,$$

and

$$\text{(3.18)}\qquad (\mathbf{L}_2\mathbf{L}_1+\mathbf{L}_2^2)\mathbf{g} = \mathbf{L}_2\mathbf{f}_2.$$

We substitute and simplify:

$$\begin{aligned}\mathbf{M}_1\tau\mathbf{L}^2 &= (\mathbf{M}_0-\mathbf{I})\mathbf{L},\\ &= \frac{1}{\tau}(\mathbf{E}-\mathbf{I})-\mathbf{L},\\ &= \frac{\tau}{2}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1}\mathbf{L}_2^2+\frac{\tau}{2}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1}\mathbf{L}_1^2\\ &\quad+\frac{\tau}{2}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1}\mathbf{L}_1\mathbf{L}_2+\frac{\tau}{2}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1}\mathbf{L}_2\mathbf{L}_1.\end{aligned} \tag{3.19}$$

We apply $\mathbf{M}_1\tau\mathbf{L}^2$ to $\mathbf{g}$, with the following substitutions deduced from (3.17) and (3.18):

$$\begin{aligned}\mathbf{L}_1^2\mathbf{g} &= \tfrac{1}{2}\mathbf{L}_1^2\mathbf{g}+\tfrac{1}{2}\mathbf{L}_1\mathbf{f}-\tfrac{1}{2}\mathbf{L}_1\mathbf{L}_2\mathbf{g},\\ \mathbf{L}_1\mathbf{L}_2\mathbf{g} &= \tfrac{1}{2}\mathbf{L}_1\mathbf{L}_2\mathbf{g}+\tfrac{1}{2}\mathbf{L}_1\mathbf{f}-\tfrac{1}{2}\mathbf{L}_1^2\mathbf{g},\\ \mathbf{L}_2^2\mathbf{g} &= \tfrac{1}{2}\mathbf{L}_2^2\mathbf{g}+\tfrac{1}{2}\mathbf{L}_2\mathbf{f}-\tfrac{1}{2}\mathbf{L}_2\mathbf{L}_1\mathbf{g},\end{aligned}$$

and

$$\mathbf{L}_2\mathbf{L}_1\mathbf{g} = \tfrac{1}{2}\mathbf{L}_2\mathbf{L}_1\mathbf{g}+\tfrac{1}{2}\mathbf{L}_2\mathbf{f}-\tfrac{1}{2}\mathbf{L}_2^2\mathbf{g}.$$

We further simplify:

$$\begin{aligned}\mathbf{M}_1\tau\mathbf{L}\mathbf{f} &= \frac{1}{4}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1}\tau\mathbf{L}_2\mathbf{f}+\frac{1}{4}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1}\tau\mathbf{L}_1\mathbf{f}+\frac{1}{4}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1}\tau\mathbf{L}_2\mathbf{f}\\ &\quad+\frac{1}{4}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1}\tau\mathbf{L}_1\mathbf{f}+\tau\mathbf{R}(\tau).\end{aligned} \tag{3.20}$$

$\mathbf{R}(\tau)$ is given by:

$$\begin{aligned}\mathbf{R}(\tau) &= \frac{1}{4}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1}(\mathbf{L}_2\mathbf{L}_1-\mathbf{L}_2^2)\mathbf{g}+\frac{1}{4}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1}(\mathbf{L}_1\mathbf{L}_2-\mathbf{L}_1^2)\mathbf{g}\\ &\quad+\frac{1}{4}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1}(\mathbf{L}_2^2-\mathbf{L}_2\mathbf{L}_1)\mathbf{g}\\ &\quad+\frac{1}{4}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1}(\mathbf{L}_1^2-\mathbf{L}_1\mathbf{L}_2)\mathbf{g}.\end{aligned}$$

$\mathbf{R}(\tau)$ further simplifies, when terms in $(\mathbf{L}_z^2-\mathbf{L}_i\mathbf{L}_j)$ are collected, to

$$\begin{aligned}\mathbf{R}(\tau) &= \frac{\tau\mathbf{L}_1}{8}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1}(\mathbf{L}_2^2-\mathbf{L}_2\mathbf{L}_1)\mathbf{g}\\ &\quad+\frac{\tau\mathbf{L}_2}{8}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1}(\mathbf{L}_1^2-\mathbf{L}_1\mathbf{L}_2)\mathbf{g}.\end{aligned}$$

$\mathbf{R}(\tau)$ is an expression whose order is $\mathbf{O}(\tau)$. The variation of $\mathbf{f}(t)$ over the interval is assumed to be $\mathbf{O}(\tau)$ (i.e. $\mathbf{f}$ is assumed to be linear for a two-point quadrature). As a result $\mathbf{M}_1\tau\mathbf{L}\mathbf{f}(t)$ contributes $\mathbf{O}(\tau^2)$ to the numerical integration of the forcing, and the omission of $\mathbf{R}(\tau)$ affects the solution only in the $\mathbf{O}(\tau^3)$ terms.

The solution to (3.3a), (3.3b) for $s_1=\frac{1}{4}$, $s_2=\frac{3}{4}$ is $W_1(z)=2M_1(z)-\frac{1}{2}M_0(z)$, and $W_2(z)=\frac{3}{2}M_0(z)-2M_1(z)$. Equations (3.10), (3.11) and (3.20) may be substituted into

$$\tau\mathbf{T}_1\mathbf{f}(\tfrac{3}{4}) = 2\tau\mathbf{M}_1\mathbf{L}\mathbf{f}(\tfrac{3}{4})-\tfrac{1}{2}\tau\mathbf{M}_0\mathbf{L}\mathbf{f}(\tfrac{3}{4})+\mathbf{O}(\tau^3)$$

to verify the correctness of the result. We also substitute (3.9), (3.11), and (3.20) into

$$\tau \mathbf{T}_2 \mathbf{f}(\tfrac{1}{4}) = \tfrac{3}{2}\tau \mathbf{M}_0 \mathbf{L} \mathbf{f}(\tfrac{1}{4}) - 2\tau \mathbf{M}_1 \mathbf{L} \mathbf{f}(\tfrac{1}{4}) + \mathbf{O}(\tau^3).$$

In order to simplify this second expression, we observe that the six terms resulting from the substitution of $\mathbf{M}_0$ and $\mathbf{M}_1$ simplify to two:

$$\frac{3}{2}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1} - \frac{1}{2}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1} - \frac{1}{2}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1}$$
$$= \frac{1}{2}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1}\left(\mathbf{I}+\frac{\tau}{2}\mathbf{L}_1\right)\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1}$$

and

$$\frac{3}{2}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1} - \frac{1}{2}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1} - \frac{1}{2}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1}$$
$$= \frac{1}{2}\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)^{-1}\left(\mathbf{I}+\frac{\tau}{2}\mathbf{L}_2\right)\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)^{-1}.$$

When the forcing function is entirely due to Dirichlet boundary conditions, and no boundary terms are "shared" between the two systems of one-dimensional differential equations, the methods based on (1.2) have local error $\mathbf{O}(\tau^3)$ at the boundary.

4. Interior forcing. The boundary forcing paradigm of the previous section has the effect of **halving the contributions of the $\mathbf{O}(t)$ term** in the boundary values. In this section we induce the same result for interior forcing.

Suppose that an interior forcing term $\mathbf{f}(t)$ is present in the differential equation (2.3). We have derived the following formula:

$$\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)\mathbf{U}_1 = \left(\mathbf{I}+\frac{\tau}{2}\mathbf{L}_1\right)\mathbf{U}(t) + \frac{\tau}{2}\mathbf{f}(t), \tag{4.1}$$

$$\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)\mathbf{U}_2 = \left(\mathbf{I}+\frac{\tau}{2}\mathbf{L}_2\right)\mathbf{U}(t) + \frac{\tau}{2}\mathbf{f}(t), \tag{4.2}$$

$$\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_2\right)\mathbf{U}_{21} = \left(\mathbf{I}+\frac{\tau}{2}\mathbf{L}_2\right)\mathbf{U}_1 + \frac{\tau}{2}\mathbf{f}(t+\tau), \tag{4.3}$$

$$\left(\mathbf{I}-\frac{\tau}{2}\mathbf{L}_1\right)\mathbf{U}_{12} = \left(\mathbf{I}+\frac{\tau}{2}\mathbf{L}_1\right)\mathbf{U}_2 + \frac{\tau}{2}\mathbf{f}(t+\tau), \tag{4.4}$$

and finally,

$$\mathbf{U}(t+\tau) = \frac{1}{2}(\mathbf{U}_{12}+\mathbf{U}_{21}). \tag{4.5}$$

Note that the weights and abscissae do not relate intuitively to a known quadrature rule. In order to derive (4.1)-(4.4), we return to our original definition of the one-dimensional quadrature using (3.3a), (3.3b), and modify $M_1(z)$ by a factor of one half in the two-dimensional solution without modifying $M_0(z)$. Equations (4.1) and (4.2) are derived by altering the trapezoidal weights W_1 and W_2 by the addition of p_1 and p_2 satisfying

$$p_1 + p_2 = 0 \quad \text{and} \quad s_1 p_1 + s_2 p_2 = -M_1.$$

W_1 and W_2 in (4.3) and (4.4) are corrected by p_1 and p_2 satisfying

$$p_1+p_2=0 \quad \text{and} \quad s_1p_1+s_2p_2=M_0-M_1.$$

The flexibility of the forcing evaluation may now be increased. If (3.7) and (4.1)-(4.4) are solved together, then the interior forcing is incorporated in one-dimensional two-point rules and the boundary forcing in corresponding midpoint rules. Another example with practical importance is a predictor formula using only $(n-1)\tau$ and $n\tau$ for forcing evaluation (Table 1). The corrector is also shown in Table 2. "First step" in the table headings refers to the calculation of $\mathbf{U}_1$ and $\mathbf{U}_2$ and "second step" to the calculation of $\mathbf{U}_{12}$ and $\mathbf{U}_{21}$, in equations such as (3.7) or (4.1)-(4.4).

TABLE 1
Predictor.

Interval	first step		second step	
Time-level	$(n-1)\tau$	$n\tau$	$(n-1)\tau$	$n\tau$
Boundary	−0.25	1.25	−0.75	1.75
Interior	0.0	1.0	−1.0	2.0

TABLE 2
Corrector.

Interval	first step		second step	
Time-level	$n\tau$	$(n+1)\tau$	$n\tau$	$(n+1)\tau$
Boundary	0.75	0.25	0.25	0.75
Interior	1.0	0.0	0.0	1.0

The utilization of the predictor for nonlinear schemes, where the coefficient matrix $\mathbf{L}$ represents a local linearization, remains to be investigated, but is identical to the use in standard ordinary differential equations codes (e.g. Gear (1971), Lawson (1972), Lawson and Swayne (1987)). The corrector form is also useful for linear differential equations, since evaluation of forcing at extra intermediate time-points is unnecessary.

5. Numerical results and conclusions. Three representative test problems are examined using the standard finite-difference approximation to the one-dimensional heat equations $\frac{1}{2}u_t=u_{xx}$ and $\frac{1}{2}u_t=u_{yy}$ on the two-dimensional problem $u_t=u_{xx}+u_{yy}$ on a uniform square mesh. The initial and boundary values chosen are given below.

Problem 1. For

$$(t, x, y)\in[0, 1]\times(0, 1)^2,$$

$$u(0, x, y)=\sin\frac{\pi}{2}x\sin\frac{\pi}{2}y,$$

$$u(t, x, 1)=\exp\left(-\frac{\pi^2}{2}t\right)\sin\frac{\pi}{2}x,$$

$$u(t, 1, y)=\exp\left(-\frac{\pi^2}{2}t\right)\sin\frac{\pi}{2}y,$$

$$u(t, x, 0)=u(t, 0, y)=0$$

the exact solution is

$$u(t, x, y) = \exp\left(-\frac{\pi^2}{2}t\right)\sin\frac{\pi}{2}x \sin\frac{\pi}{2}y.$$

Table 3 shows the results for problem 1 of the new method, labeled PALOD (Parallel Alternating LOD), with a locally one-dimensional midpoint rule for boundary forcing compared to a standard LOD, based on (1.3), at time $t = 1$. The step-size referred to in the tables is that for the total calculation (four individual LOD steps plus averaging). The calculations were performed on a Digital Equipment Corporation VAX in single precision for Problem 1 and the rest in double precision.

Table 4 illustrates the effect of a two-point quadrature based on the trapezoidal rule for the one-dimensional steps compared to the Peaceman-Rachford alternating-direction implicit (ADI) method. Errors are decreasing approximately by a factor of four as τ is halved for PALOD, indicating that the method is indeed locally second order. The per step efforts of ADI and PALOD are such that they provide almost identical results, for this problem, for the same expenditure of effort.

Problem 2 has both boundary and interior forcing. In the literature, LOD methods have been shown to have difficulty with this problem. (See, for example, Verwer and de Vries (1985)). This new scheme appears to have resolved the order problem, allowing even extrapolation to improve order. After n time-steps of length τ, an error of $\mathbf{O}(\tau^3)$ will have increased to $\mathbf{O}(\tau^2)$, when $n\tau \approx 1$. We calculate our extrapolation by computing two solutions to $t = 1$, $\mathbf{U}_\tau$ with step τ and $\mathbf{U}_{2\tau}$ with step 2τ. We then compute for a given t: $\mathbf{U}^* = (4/3)\mathbf{U}_\tau - (1/3)\mathbf{U}_{2\tau}$. Table 5 summarizes the results. The column entitled "work" gives a rough estimate for computational effort, based on numbers of linear system solutions, given that the numbers of function calls to evaluate the forcing terms

TABLE 3
PALOD *versus* LOD (*Midpoint quadrature*).
Problem 1. *Maximum absolute error at* $t = 1$.

	PALOD (1, 1) error				LOD error		
τ \ h	0.1	0.05	0.025	τ \ h	0.1	0.05	0.025
0.2	2.2−3	8.9−3	1.7−2	0.2	1.1−1	6.3−1	2.9
0.1	6.6−5	5.4−4	2.2−3	0.1	2.0−2	2.1−1	1.2
0.05	1.3−5	7.5−6	1.7−5	0.05	6.6−3	3.8−2	4.2−1
0.02	1.9−6	2.1−6	2.3−6	0.02	3.5−3	9.4−3	4.4−2

TABLE 4
ADI *versus* PALOD (*Trapezoidal quadrature*).
Problem 1. *Maximum absolute error at* $t = 1$.

	ADI error				PALOD error		
τ \ h	0.1	0.05	0.025	τ \ h	0.1	0.05	0.025
0.1	5.1−6	7.8−6	8.2−6	0.1	5.1−6	7.8−6	8.5−6
0.05	1.4−6	1.3−6	1.9−6	0.05	1.3−6	1.3−6	2.0−6
0.025	2.9−6	3.4−7	3.1−7	0.025	3.0−6	3.3−7	3.0−7

TABLE 5
Problem 2. Maximum error (40 points).

Method	Work	$\tau=\frac{1}{6}$	$\tau=\frac{1}{12}$	$\tau=\frac{1}{24}$
(New method)				
PALOD	2	4.3−3	1.1−3	2.7−4
PALOD (extrapolated)	3	1.9−3	3.9−4	4.1−5
PALOD (Predictor mode)	2	4.2−2	8.1−3	1.0−3
PALOD (Corrector mode)	2	5.6−3	1.4−3	3.5−4
PALOD (Corr.) extrapolated	3	2.4−3	5.2−4	5.5−5
(Classical Schemes)				
LOD (with $\tau/2$)	2	1.2−2	7.8−3	4.5−3
Lawson-Morris	2	2.3−2	1.5−2	8.7−3
ADI (with $\tau/2$)	3	2.6−6	6.6−7	1.6−7
(Verwer-de Vries)				
Global extrap. LOD	2	9.8−3	4.7−3	2.0−3
Global extrap. ADI	3	3.8−6	2.2−7	1.3−8

are in proportion to the number of linear systems to be solved. The results for the Lawson-Morris algorithm (an extrapolated LOD scheme) are quoted from the literature (Verwer and de Vries (1985)).

Problem 2. For

$$(t, x, y) \in [0, 1] \times (0, 1)^2,$$
$$u(0, x, y) = 1 + x^2 + y^2,$$
$$u(t, x, 1) = 1 + \exp(-t)(1 + x^2),$$
$$u(t, 1, y) = 1 + \exp(-t)(1 + y^2),$$
$$u(t, x, 0) = 1 + \exp(-t)x^2,$$
$$u(t, 0, y) = 1 + \exp(-t)y^2,$$
$$u_t = u_{xx} + u_{yy} - \exp(-t)(x^2 + y^2 + 4).$$

The exact solution is $u(t, x, y) = 1 + \exp(-t)(x^2 + y^2)$.

The PALOD schemes all exhibit second-order behaviour with τ, and the PALOD extrapolation appears to have error one order higher. On this problem, PALOD performs well against other LOD schemes and Lawson-Morris, but it does not match the accuracy of the Peaceman-Rachford ADI or the Verwer-de Vries ADI.

A third test problem illustrates a situation for which LOD produces a significantly better numerical solution than ADI for the same τ and h. It is defined on the same $[0, 1] \times (0, 1)^2$:

$$u_t = u_{xx} + u_{yy} - 4 + \exp(-t) \sin x \sin y,$$

with initial and boundary values consistent with $u(t, x, y) = \exp(-t) \sin x \sin y + x^2 + y^2$ (see Table 6).

We have presented a locally one-dimensional scheme which preserves order for time-dependent boundary and interior forcing. Through it, we have also developed a method of attacking other LOD schemes, to bring them into competition with direct and ADI methods. It appears that LOD schemes are potentially as accurate as other

TABLE 6
Maximum absolute errors.
Problem 3. $h = \frac{1}{40}$, $t = 1$.

Method	$\tau = \frac{1}{6}$	$\tau = \frac{1}{12}$	$\tau = \frac{1}{24}$
PALOD (Corrector)	1.6−5	3.4−6	2.9−7
ADI (Peaceman-Rachford)	6.5−4	1.2−4	3.1−4

splitting methods, but that the one-dimensional solution steps must be modified in a systematic way to preserve order.

Acknowledgment. Support for this work from the United Kingdom Science and Engineering Research Council in the form of a Visiting Fellowship to the University of Kent at Canterbury is gratefully acknowledged. The thorough and patient work of the referees is appreciated.

REFERENCES

C. W. GEAR (1971), *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ.

A. R. GOURLAY AND A. R. MITCHELL (1972), *On the structure of alternating direction implicit* (A.D.I.) *and locally one dimensional* (L.O.D.) *difference methods*, J. Inst. Math. Appl., 9, pp. 80-90.

J. D. LAWSON (1972), *Some numerical methods for stiff ordinary and partial differential equations*, Proc. Second Manitoba Conf. on Numerical Mathematics, Winnipeg, Canada, pp. 27-34.

J. D. LAWSON AND J. LL. MORRIS (1978), *The extrapolation of first order methods for parabolic partial differential equations.* I, SIAM J. Numer. Anal., 15, pp. 1212-1244.

J. D. LAWSON AND D. A. SWAYNE (1987), EPIMK2—*an innovative stiff systems solver*, Congr. Numer., 55, pp. 151-158.

A. R. MITCHELL AND D. F. GRIFFITHS (1980), *The Finite Difference Method in Partial Differential Equations*, John Wiley, London-New York.

D. A. SWAYNE (1975), *Computation of rational functions with matrix argument with application to initial value problems*, Ph.D. thesis, University of Waterloo, Ontario, Canada.

J. G. VERWER AND H. B. DE VRIES (1985), *Global extrapolation of a first order splitting method*, this Journal, 6, pp. 771-780.

N. N. YANENKO (1970), *The Method of Fractional Steps*, Springer-Verlag, Berlin-New York.

A. ZAFARULLAH (1970), *Application of the method of lines to parabolic differential equations with error estimates*, J. Assoc. Comput. Mach., 17, pp. 294-302.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 5, September 1987

© 1987 Society for Industrial and Applied Mathematics
008

SOLVING GALERKIN APPROXIMATIONS TO NONLINEAR TWO-POINT BOUNDARY VALUE PROBLEMS BY A GLOBALLY CONVERGENT HOMOTOPY METHOD*

LAYNE T. WATSON† AND L. RIDGWAY SCOTT‡

Abstract. The Chow-Yorke algorithm is a homotopy method that has been proved globally convergent for Brouwer fixed point problems, certain classes of zero finding and nonlinear programming problems, and two-point boundary value approximations based on shooting, finite differences, and spline collocation. The method is numerically stable and has been successfully applied to a wide range of practical engineering problems. Here the Chow-Yorke algorithm is proved globally convergent for a class of Galerkin approximations to nonlinear two-point boundary value problems. Several numerical implementations of the algorithm are briefly described, and computational results are presented for a fairly difficult magneto-hydrodynamics boundary value problem.

Key words. homotopy method, Chow-Yorke algorithm, globally convergent, two-point boundary value problem, Galerkin method, finite element method, nonlinear equations

AMS(MOS) subject classifications. 65H10, 65L10, 65L60

1. Introduction.

The Chow-Yorke algorithm (actually a family of algorithms) for nonlinear systems of equations was proposed in 1976, and since that time both the theory and the scope of its practical applicability have been greatly extended. This homotopy algorithm is accurately described as a globally convergent, probability-one algorithm. It is truly globally convergent in the sense that it will converge to a solution of the problem from an *arbitrary* starting point. The phrase "probability one" refers to the rigorous theoretical results which guarantee convergence for almost all choices of some parameter vector, i.e., with probability one.

Homotopy methods (both continuous [1] and simplicial [13], [29]) were originally very inefficient, and dismissed by some as inherently inferior to quasi-Newton algorithms. A common misconception is that homotopy algorithms are just continuation or an obvious extension of classical continuation. While this is superficially true, there are fundamental philosophical differences between these probability-one algorithms and standard continuation. These differences result in an elegant and rigorous theoretical foundation for globally convergent probability-one homotopy algorithms, and have subtle yet important implications for computer implementations [60]. Furthermore, there has been a series of practical engineering problems, successfully solved by homotopy methods, on which continuation and quasi-Newton methods either totally failed or experienced great difficulty [32]–[50]. Current implementations (as in

* Received by the editors May 12, 1986; accepted for publication (in revised form) November 24, 1986.

† Departments of Electrical Engineering and Computer Science, Industrial and Operations Engineering, and Mathematics, University of Michigan, Ann Arbor, Michigan 48109. Current address: Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24061. The work of this author was supported in part by Air Force Office of Scientific Research Grant 85-0250.

‡ Department of Mathematics, University of Michigan, Ann Arbor, Michigan 48109.

HOMPACK [59]) of these globally convergent probability-one homotopy algorithms are reasonably efficient, and their robustness, stability, and accuracy have never been in doubt.

There are three distinct, but interrelated, aspects of homotopy methods: 1) construction of the right homotopy map, 2) theoretical proof of global convergence for this homotopy map, and 3) tracking the zero curve of this homotopy map. The first aspect is currently still an art, although this is much better understood now due to the accumulation of computational experience [32]–[50], [53]. Although much remains to be done, significant progress has been made on the second aspect. Global convergence has been proved for Brouwer fixed point problems [5], [52], certain classes of zero finding [52] and nonlinear programming (both unconstrained and constrained) problems [53], and two-point boundary value approximations based on shooting [55], finite differences [56], and spline collocation [58]. Recently Morgan [26]–[27] obtained some elegant results for polynomial systems, and the present work considers Galerkin approximations to nonlinear two-point boundary value problems. Various curve tracking algorithms have been around for a long time (e.g., [18]–[20]), but there is an important distinction between general curve tracking algorithms and homotopy curve tracking algorithms. The object of the former is the zero curve *itself*, whereas the object of the latter is a point at $\lambda = 1$. This difference was emphasized in [51], [52], and [60], and the algorithms described here are more in the spirit of homotopy methods than a general tracker.

There have been various studies on the existence of solutions to discrete approximations of nonlinear boundary value problems. For example, Keller [17] shows that if a two-point boundary value problem has an isolated solution, then any finite difference approximation from some class has an isolated solution also, which is computable by Newton's method with a good initial guess, for a sufficiently small mesh spacing. Brezzi, Rappaz, and Raviart [4] show that if a parametrized operator equation has a nondegenerate curve of solutions, then any Galerkin approximation in some class also has for sufficiently small mesh spacing a nondegenerate curve of solutions close to the solution curve of the operator equation. The results presented here are more particular, in that they pertain to a specific equation, but they are sharper, in that they do not require a mesh spacing restriction. The proofs utilize the Chow, Mallet-Paret, and Yorke homotopy theory [5].

Section 2 outlines the theoretical foundation of globally convergent, probability-one homotopy methods, and proves some convergence theorems for Galerkin approximations to nonlinear two-point boundary value problems. Section 3 discusses several algorithms for tracking the homotopy curves, as well as some pertinent details of the software package HOMPACK used to obtain the numerical results. Section 4 considers a nontrivial two-point boundary value problem from fluid dynamics, presents numerical results, and illustrates the dependence of the solution on the problem parameters.

2. Theory. Consider the two-point boundary value problem

$$y''(x) = f(x, y(x), y'(x)), \qquad 0 \le x \le 1, \tag{1}$$

$$y(0) = y(1) = 0, \tag{2}$$

where $y(x)$ is an $\hat{n}$-dimensional vector function and $f(x,u,v)$ is an $\hat{n}$-dimensional C^2 vector function. This is naturally posed in a weak form via a variational formulation using the space

$$V = \{f \in H^1[0,1] \mid f(0) = f(1) = 0\},$$

where $H^1[0,1]$ is the Hilbert space of all absolutely continuous functions on $[0,1]$ whose derivative lies in $L^2[0,1]$. Let $S_n \subset V$ be a finite-dimensional vector space with basis $\phi_1, \ldots, \phi_n$.

Using the finite-dimensional approximations

$$y_m(x) \approx A_m(x) = \sum_{i=1}^{n} \alpha_{mi}\phi_i(x), \qquad m = 1, \ldots, \hat{n}, \tag{3}$$

the standard Galerkin approximation to equations (1)–(2) is the nonlinear system of equations

$$\sum_{i=1}^{n} \alpha_{mi}\langle \phi_i', \phi_k' \rangle + \langle f_m(\cdot, A, A'),\ \phi_k \rangle = 0, \tag{4}$$

$$k = 1, \ldots, n, \quad m = 1, \ldots, \hat{n}.$$

Here

$$\langle u, v \rangle = \int_0^1 u(t)v(t)\,dt \tag{5}$$

is the inner product.

Example 1. *Continuous piecewise linear functions.* Let $0 = x_0 < x_1 < \cdots < x_n < x_{n+1} = 1$ be a partition of $[0,1]$, and let S_n be the space of continuous, piecewise linear functions $f(x)$ with breakpoints $x_0, \ldots, x_{n+1}$ and $f(0) = f(1) = 0$. The basis functions ϕ_i, $i = 1, \ldots, n$, are defined by $\phi_i(x_j) = \delta_{ij}$. Observe that in this case the equations (4) correspond to the difference equations

$$\frac{2}{h_k + h_{k+1}}\left[\frac{A_m(x_{k+1}) - A_m(x_k)}{h_{k+1}} - \frac{A_m(x_k) - A_m(x_{k-1})}{h_k}\right]$$
$$= \frac{2}{h_k + h_{k+1}}\langle f_m(\cdot, A, A'),\ \phi_k \rangle = f_m\big(x_k, A(x_k), A'(x_k)\big) + O(h)$$

where $h_i = x_i - x_{i-1}$, $h = \max_j h_j$. This is a standard finite difference approximation to (1) (*cf.* [56] for the case of a uniform mesh).

Example 2. *Chebyshev series (spectral methods).* Using the Chebyshev polynomials $\{T_n(x)\}_{n=0}^{\infty}$ for the interval $[-1,1]$, take

$$\phi_{k-1}(x) = \begin{cases} T_k(2x-1) - 1, & k \text{ even}, \\ T_k(2x-1) - (2x-1), & k \text{ odd} \end{cases}$$

for $k = 2, 3, \ldots$. These $\phi_k(x)$ satisfy the zero boundary conditions, are dense in V, and are relatively well conditioned. The equations (4) in this case are not sparse. See Gottlieb and Orszag [15] for details.

A third example, generalizing Example 1, would be a space $S_n \subset C^{k-2}[0,1]$ consisting of piecewise polynomials of order $k \geq 2$ with B-splines for the basis functions ϕ_i. This spline space is developed in §4.

The accuracy and convergence of the Galerkin approximation (3) to the exact solution $\phi(x)$ of (1)–(2) can be shown under various assumptions on the nonlinear right hand side $f(x,u,v)$ and generalizations of y'' to elliptic operators $L[w(x)] = \sum_{j=0}^{N}(-1)^{j+1}D^j\big(p_j(x)D^j w(x)\big)$. To illustrate, some typical theorems from Ciarlet, Schultz, and Varga [6]–[10] will be quoted here. Following [6], assume that $\hat{n} = 1$, a classical solution $\phi(x)$ of (1)–(2) exists, the nonlinear right hand side has the form $f(x,y)$ (i.e., does not involve y'), and

$$\frac{\partial f(x,u)}{\partial u} \geq \gamma > -\Lambda \equiv -\inf_{\substack{w\in V\\ w\neq 0}} \frac{\int_0^1 \big(w'(t)\big)^2 dt}{\int_0^1 \big(w(t)\big)^2 dt} \tag{6}$$

for all $x \in [0,1]$ and all real u. The appropriate norm (derived from an inner product based on a linearization of (4)) in this context is

$$\|w\|_\gamma = \left[\int_0^1 \big(w'(x)\big)^2 + \gamma\big(w(x)\big)^2\, dx\right]^{1/2}, \quad w \in V,$$

where the constant γ is the same as that in (6). The accuracy of the Galerkin approximation $A(x)$ is given by

THEOREM. *Let $\phi(x)$ be the solution of (1)–(2) subject to the assumptions above, let S_n be any finite dimensional subspace of V of dimension n with basis $\phi_1,\ldots,\phi_n$, and let $\hat{A}_n(x) = \sum_{i=1}^{n}\hat{\alpha}_{ni}\phi_i(x)$, where $\hat{\alpha}_n$ is the (unique under the above assumptions) solution to (4). Then there exist constants C and K independent of n and S_n such that*

$$\|\hat{A}_n - \phi\|_\infty \leq K\,\|\hat{A}_n - \phi\|_\gamma \leq C \inf_{w\in S_n}\|w-\phi\|_\gamma.$$

THEOREM. *Let $\phi(x)$ be the solution of (1)–(2) subject to the assumptions above, let $\{S_n\}_{n=1}^{\infty}$ be any sequence of finite dimensional subspaces of V such that $\bigcup_{n=1}^{\infty} S_n$ is dense in V in the norm $\|\cdot\|_\gamma$, and let $\{\hat{A}_n(x)\}_{n=1}^{\infty}$ be the sequence of functions obtained by solving (4) over the subspaces S_n respectively. Then $\{\hat{A}_n(x)\}_{n=1}^{\infty}$ converges* uniformly *to $\phi(x)$.*

Generalizations of these theorems [6]–[10], [30] tend toward generalizing the differential operator on the left hand side in (1), weakening the smoothness assumptions on the right hand side in (1), and generalizing the boundary conditions (2), rather than the form of and growth conditions on $f(x,u,v)$. The homotopy theorems derived here require C^2 smoothness (piecewise C^2 is the absolute minimal requirement), and have rather mild growth restrictions on $f(x,u,v)$ compared to (6). Furthermore the f considered here can involve $y'(x)$ in a highly nonlinear way, whereas the $f = f(x,u)$ in (6) cannot involve $y'(x)$ at all. Roughly speaking, with the exception of smoothness, the limitations of the theoretical foundations for applying globally convergent

homotopy methods to Galerkin approximations to nonlinear two-point boundary value problems derive more from the applicability of the Galerkin method itself than from the convergence conditions of the homotopy methods.

Let $Y = (\alpha_{11}, \alpha_{12}, \ldots, \alpha_{1n}, \alpha_{21}, \ldots, \alpha_{2n}, \ldots, \alpha_{\hat{n}1}, \ldots, \alpha_{\hat{n}n})^t$. Then the system of equations (4) has the form

$$F(Y) = MY + N(Y) = 0, \tag{7}$$

where

$$M = \begin{pmatrix} \bar{M} & 0 & \ldots & 0 \\ 0 & \bar{M} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \bar{M} \end{pmatrix}, \qquad \bar{M}_{ij} = \langle \phi_i', \phi_j' \rangle,$$

$$\begin{aligned} N(Y) = \Big(& \langle f_1(x, A(x), A'(x)), \phi_1(x) \rangle, \ldots, \langle f_1(x, A(x), A'(x)), \phi_n(x) \rangle, \\ & \langle f_2(x, A(x), A'(x)), \phi_1(x) \rangle, \ldots, \langle f_2(x, A(x), A'(x)), \phi_n(x) \rangle, \\ & \vdots \\ & \langle f_{\hat{n}}(x, A(x), A'(x)), \phi_1(x) \rangle, \ldots, \langle f_{\hat{n}}(x, A(x), A'(x)), \phi_n(x) \rangle \Big). \end{aligned}$$

Thus the two-point boundary value problem (1)–(2) is approximated by the nonlinear system of equations (7), which has dimension

$$p = \hat{n}\, n.$$

A homotopy method is used to solve (7).

Since $(u, v) \to \langle u', v' \rangle$ is an inner product on V and $\phi_1, \ldots, \phi_n$ are linearly independent, the Gram matrix $\bar{M}$ is symmetric and positive definite. Therefore M is also symmetric and positive definite.

Let E^p denote p-dimensional real Euclidean space. The following four lemmas from [55], [55], [54], [56] respectively will be useful.

LEMMA 1. *Let $F : E^p \to E^p$ be a C^2 map, $a \in E^p$, and define $\rho_a : [0, 1) \times E^p \to E^p$ by*

$$\rho_a(\lambda, y) = \lambda F(y) + (1 - \lambda)(y - a).$$

Then for almost all $a \in E^p$ there is a zero curve γ of ρ_a emanating from $(0, a)$ along which the Jacobian matrix $D\rho_a(\lambda, y)$ has full rank.

LEMMA 2. *If the zero curve γ in Lemma 1 is bounded, it has an accumulation point $(1, \bar{y})$, where $F(\bar{y}) = 0$. Furthermore, if $DF(\bar{y})$ is nonsingular, then γ has finite arc length.*

LEMMA 3. *Let $F : E^p \to E^p$ be a C^2 map such that for some $r > 0$, $xF(x) \geq 0$ whenever $\|x\| = r$. Then F has a zero in $\{x \in E^p \mid \|x\| \leq r\}$, and for almost all $a \in E^p$, $\|a\| < r$, there is a zero curve γ of*

$$\rho_a(\lambda, x) = \lambda F(x) + (1 - \lambda)(x - a),$$

along which the Jacobian matrix $D\rho_a(\lambda, x)$ *has full rank, emanating from* $(0, a)$ *and reaching a zero* $\bar{x}$ *of* F *at* $\lambda = 1$. *Furthermore,* γ *has finite arc length if* $DF(\bar{x})$ *is nonsingular.*

Lemma 3 is a special case of the following more general lemma.

LEMMA 4. *Let* $F : E^p \to E^p$ *be a* C^2 *map such that for some* $r > 0$ *and* $\tilde{r} > 0$, $F(x)$ *and* $x - a$ *do not point in opposite directions* $(\mu F(x) + (x - a) \neq 0 \ \forall \mu > 0)$ *for* $\|x\| = r$, $\|a\| < \tilde{r}$. *Then* F *has a zero in* $\{x \in E^p \mid \|x\| \leq r\}$, *and for almost all* $a \in E^p$, $\|a\| < \tilde{r}$, *there is a zero curve* γ *of*

$$\rho_a(\lambda, x) = \lambda F(x) + (1 - \lambda)(x - a),$$

along which the Jacobian matrix $D\rho_a(\lambda, x)$ *has full rank, emanating from* $(0, a)$ *and reaching a zero* $\bar{x}$ *of* F *at* $\lambda = 1$. *Furthermore,* γ *has finite arc length if* $DF(\bar{x})$ *is nonsingular.*

THEOREM 1. *Let* $N(Y)$ *in (7) be a* C^2 *mapping, and suppose there exist constants* C *and* ν *such that*

$$\limsup_{\|Y\|_2 \to \infty} \frac{\|N(Y)\|_2}{\|Y\|_2^{\nu}} = C, \qquad 0 \leq \nu < 1. \tag{8}$$

For $W \in E^p$, *define* $\rho_W : [0, 1) \times E^p \to E^p$ *by*

$$\rho_W(\lambda, Y) = \lambda F(Y) + (1 - \lambda)(Y - W).$$

Then for almost all $W \in E^p$ *there exists a zero curve* γ *of* ρ_W, *along which the Jacobian matrix* $D\rho_W(\lambda, Y)$ *has full rank, emanating from* $(0, W)$ *and reaching a zero* $\tilde{Y}$ *of* F *(at* $\lambda = 1$*). Furthermore, if* $DF(\tilde{Y})$ *is nonsingular, then* γ *has finite arc length.*

Proof. As observed above, the matrix M in (7) is symmetric and positive definite. Therefore all its eigenvalues are real and positive. Let $\Gamma > 0$ be the smallest eigenvalue of M. Then a simple argument shows that for $Y \neq 0$,

$$Y^t M Y \geq \Gamma \|Y\|_2^2 > 0. \tag{9}$$

Choose $\epsilon > 0$. Using (8) and (9) yields

$$Y^t F(Y) = Y^t M Y + Y^t N(Y) \geq \Gamma \|Y\|_2^2 - \|Y\|_2 (C + \epsilon) \|Y\|_2^{\nu} > 0$$

for $\|Y\|_2$ sufficiently large. Therefore there exists $r > 0$ such that $Y^t F(Y) \geq 0$ for $\|Y\|_2 = r$. The result now follows directly from Lemma 3. Q.E.D.

THEOREM 2. *Let* $N(Y)$ *in (7) be a* C^2 *mapping and* $\Gamma > 0$ *defined by (9). If*

$$\limsup_{\|Y\|_2 \to \infty} \frac{\|N(Y)\|_2}{\|Y\|_2} = C < \Gamma,$$

then the conclusion of Theorem 1 holds.

Proof. This follows immediately from the proof of Theorem 1. Q.E.D.

Observe that Theorem 2 is stronger than Theorem 1, but since Theorem 2 requires knowledge of Γ, Theorem 1 is easier to verify. The next two theorems, giving conditions on the right-hand side $f(x,u,v)$ in (1), are what one would use in practice.

THEOREM 3. *If $f(x,u,v)$ in (1) is C^2 and bounded, then the conclusion of Theorem 1 holds.*

Proof. $f(x,u,v)$ bounded and C^2 implies that $N(Y)$ is also bounded and C^2, and thus (8) holds with $C=0$ and $\nu = 1/2$. Therefore the conclusion of Theorem 1 holds. Q.E.D.

THEOREM 4. *Let $f(x,u,v)$ in (1) be a C^2 mapping, and suppose there exist constants μ, ξ, and ν such that*

$$\|f(x,u,v)\|_2 \le \mu\big(\xi + (\|u\|_2 + \|v\|_2)^\nu\big), \quad 0 \le \nu < 1,$$

for all $0 \le x \le 1$ and $u, v \in E^{\hat{n}}$. Then the conclusion of Theorem 1 holds.

Proof. Let

$$y_k(x) = \sum_{i=1}^{n} Y_{(k-1)n+i}\,\phi_i(x), \quad k = 1,\dots,\hat{n}, \quad 0 \le x \le 1, \tag{10a}$$

and

$$z_k(x) = \sum_{i=1}^{n} Z_{(k-1)n+i}\,\phi_i(x), \quad k = 1,\dots,\hat{n}, \quad 0 \le x \le 1. \tag{10b}$$

Since the basis functions ϕ_i are in V, it follows from the equivalence of norms in finite dimensional spaces that there exist constants $K_1 > 0$ and $K_2 < \infty$ such that

$$K_1(\alpha^t\alpha)^{1/2} \le \left\| \sum_{i=1}^{n} \alpha_i \phi_i \right\|_{W_2^1[0,1]} \le K_2(\alpha^t\alpha)^{1/2}, \tag{11}$$

where $\alpha = (\alpha_1,\dots,\alpha_n)^t$ is a real n-vector and $W_p^1[0,1]$ is the Sobolev space of functions whose (weak) derivatives lie in $L^p[0,1]$, and the Sobolev norms are (cf. [30])

$$\|v\|_{W_p^1[0,1]} = \left[\int_0^1 |v(x)|^p + |v'(x)|^p\,dx\right]^{1/p}, \quad 1 \le p < \infty,$$

$$\|v\|_{W_\infty^1[0,1]} = \sup_{0\le x<y\le 1} \left\{ |v(x)|,\ \frac{|v(x)-v(y)|}{|x-y|} \right\}.$$

For instance, the constant K_2 corresponding to Example 1 in §2 has the form $K_2 = (\text{constant})/\sqrt{\min_j h_j}$.

Now using (10), (11), the theorem hypothesis, the Cauchy-Schwarz inequality, Jensen's inequality, and the relation $(a+b)^2 \le 2(a^2+b^2)$ gives

(12)

$$
\begin{aligned}
\|N(Y)\|_2 &= \sup_{\|Z\|_2=1} N(Y)^t Z = \sup_{\|Z\|_2=1} \sum_{m=1}^{\hat{n}} \langle f_m(\cdot, y, y'),\, z_m(\cdot)\rangle \\
&\le \sup_{\|Z\|_2=1} \sum_{m=1}^{\hat{n}} \|f_m(\cdot, y, y')\|_2 \, \|z_m(\cdot)\|_2 \\
&\le \sup_{\|Z\|_2=1} \|f(\cdot, y, y')\|_2 \, \|z(\cdot)\|_2 \\
&\le \sup_{\|Z\|_2=1} \left[\int_0^1 f_1(x, y(x), y'(x))^2 + \cdots + f_{\hat{n}}(x, y(x), y'(x))^2 \, dx\right]^{1/2} K_2 \, \|Z\|_2 \\
&\le \mu K_2 \left[\int_0^1 \xi^2 + 2\xi \left(\|y(x)\|_2 + \|y'(x)\|_2\right)^{\nu} \right. \\
&\qquad\qquad \left. + \left(\|y(x)\|_2 + \|y'(x)\|_2\right)^{2\nu} dx\right]^{1/2} \\
&\le \mu K_2 \left[\xi^2 + 2^{1+\nu/2}\xi \int_0^1 \left(\|y(x)\|_2^2 + \|y'(x)\|_2^2\right)^{\nu/2} dx \right. \\
&\qquad\qquad \left. + 2^{\nu} \int_0^1 \left(\|y(x)\|_2^2 + \|y'(x)\|_2^2\right)^{\nu} dx\right]^{1/2} \\
&\le \mu K_2 \left[\xi^2 + 2^{1+\nu/2}\xi \|y\|_{W_2^1}^{\nu} + 2^{\nu}\|y\|_{W_2^1}^{2\nu}\right]^{1/2} \\
&= \mu K_2 \left(\xi + 2^{\nu/2}\|y\|_{W_2^1}^{\nu}\right) \\
&\le \mu K_2 \left(\xi + 2^{\nu/2} K_2^{\nu} \, \|Y\|_2^{\nu}\right).
\end{aligned}
$$

Therefore for $\epsilon > 0$ and $\|Y\|_2$ sufficiently large it follows from (12) that

$$
\frac{\|N(Y)\|_2}{\|Y\|_2^{\nu}} \le 2^{\nu/2}\mu K_2^{\nu+1} + \epsilon,
$$

which is precisely the condition (8) in Theorem 1 for some constant

$$
C \le 2^{\nu/2}\mu K_2^{\nu+1}. \qquad \text{Q.E.D.}
$$

Even though the growth conditions on $f(x,u,v)$ in Theorems 3 or 4 do not hold, it may turn out that for some constant $K > 0$, the solution to (1) is the *same* as the solution to (1) with $f(x,u,v)$ replaced by

$$
\hat{f}(x,u,v) = \begin{cases} f(x,u,v) & \text{if } \|u\|_2 \le K \text{ and } \|v\|_2 \le K, \\ f\big(x, \psi(u), \psi(v)\big) & \text{if } \|u\|_2 > K \text{ or } \|v\|_2 > K, \end{cases}
$$

where $\psi : E^{\hat{n}} \to E^{\hat{n}}$ is C^2, bounded, and $\psi(u) = u$ for $\|u\|_2 \leq K$. Since the theorems apply to $\hat{f}(x, u, v)$, a globally convergent homotopy algorithm can still be successfully applied to the problem.

3. Algorithm. The general idea of the algorithm is apparent from Theorem 1: just follow the zero curve γ emanating from $(0, W)$ until a zero $\tilde{Y}$ of $F(Y)$ is reached (at $\lambda = 1$). Of course it is nontrivial to develop a viable numerical algorithm based on that idea, but at least conceptually, the algorithm for solving the nonlinear system of equations (7) is clear and simple. The homotopy map is

$$\rho_W(\lambda, Y) = \lambda F(Y) + (1 - \lambda)(Y - W), \tag{13}$$

which has the same form as a standard continuation or embedding mapping. However, there are two crucial differences. In standard continuation, the embedding parameter λ increases monotonically from 0 to 1 as the trivial problem $Y - W = 0$ is continuously deformed to the problem $F(Y) = 0$. The present homotopy method permits λ to both increase and decrease along γ with no adverse effect; that is, turning points present no special difficulty. Of course, methods have been developed [18]–[24], [28] to navigate turning points, but the method considered here does so automatically. The second important difference is that there are never any "singular points" which afflict standard continuation methods. The way in which the zero curve γ of ρ_W is followed and the full rank of $D\rho_W$ along γ guarantee this. Observe that Lemma 1 guarantees that γ cannot just "stop" at an interior point of $[0, 1) \times E^p$.

However, "numerically singular" points along γ are possible, and manifest themselves in different ways for the different algorithms described below. Precise details on the detection of and recovery from numerically singular points are in [59], where the algorithms are described in considerable detail. Generally, such ill conditioning causes inconsistent derivatives, slow corrector convergence, rank deficient Jacobian matrices, and drift from γ. At best a reduced step size or a recalculation of the initial point W gets past the ill conditioning with no harm done (virtually always the case in practice); at worst an exactly rank deficient Jacobian matrix is calculated or the zero curve γ is lost.

The zero curve γ of the homotopy map $\rho_W(\lambda, Y)$ in (13) can be tracked by many different techniques; refer to the excellent survey [1] and recent work by Rheinboldt and Burkhardt [28] and Mejia [23]. The numerical results in §4 were obtained with the software package HOMPACK, currently under development at Sandia National Laboratories, General Motors Research Laboratories, Virginia Polytechnic Institute and State University, and the University of Michigan. HOMPACK is a suite of codes for tracking zero curves of probability one homotopy maps, and provides both high-level and low-level subroutines for three different approaches to tracking γ. The three algorithmic approaches provided by HOMPACK are: 1) an ODE-based algorithm derived from that in [51], with several refinements; 2) a predictor-corrector algorithm whose corrector follows the flow normal to the Davidenko flow (a "normal flow" algorithm); 3) a version of Rheinboldt's linear predictor, quasi-Newton corrector algorithm [28] (an "augmented Jacobian" method). There are qualitatively different algorithms for dense and sparse Jacobian matrices; only algorithms for dense Jacobian matrices are discussed here. See [60] for a discussion of sparsity in relation to homotopy methods.

First the ODE-based algorithm will be discussed. Assuming that $F(Y)$ is C^2 and W is such that Theorem 1 holds, the zero curve γ is C^1 and can be parametrized by arc length s. Thus $\lambda = \lambda(s)$, $Y = Y(s)$ along γ, and

$$\rho_W(\lambda(s), Y(s)) = 0 \tag{14}$$

identically in s. Therefore

$$\frac{d}{ds}\rho_W(\lambda(s), Y(s)) = D\rho_W(\lambda(s), Y(s)) \begin{pmatrix} \frac{d\lambda}{ds} \\ \frac{dY}{ds} \end{pmatrix} = 0, \tag{15}$$

$$\left\| \left(\frac{d\lambda}{ds}, \frac{dY}{ds} \right) \right\|_2 = 1. \tag{16}$$

If we take

$$\lambda(0) = 0, \quad Y(0) = W, \tag{17}$$

the zero curve γ is the trajectory of the initial value problem (15)–(17). When $\lambda(\bar{s}) = 1$, the corresponding $Y(\bar{s})$ is a zero of $F(Y)$. Thus all the sophisticated ODE techniques currently available can be brought to bear on the problem of tracking γ [31], [52].

ODE software requires $(d\lambda/ds, dY/ds)$ explicitly, and (15), (16) only implicitly define the derivative $(d\lambda/ds, dY/ds)$. This can be calculated by finding the kernel of the $p \times (p+1)$ Jacobian matrix

$$D\rho_W(\lambda(s), Y(s)),$$

which has full rank by Theorem 1. It is here that a substantial amount of computation is incurred, and it is imperative that the number of derivative evaluations be kept small. Once the kernel has been calculated, the derivative $(d\lambda/ds, dY/ds)$ is uniquely determined by (16) and continuity. Complete details for solving the initial value problem (15)–(17) and obtaining $Y(\bar{s})$ are in [51] and [55]. A brief discussion of the kernel computation for dense Jacobian matrices $D\rho_W$ follows.

The Jacobian matrix $D\rho_W$ is $p \times (p+1)$ with (theoretical) rank p. The crucial observation is that the last p columns of $D\rho_W$, corresponding to $D_Y\rho_W$, may not have rank p, and even if they do, some other p columns may be better conditioned. The objective is to avoid choosing p "distinguished" columns, rather to treat all columns the same (not possible for sparse matrices). Kubicek [20] and Mejia [23] have kernel finding algorithms based on Gaussian elimination and p distinguished columns. Choosing and switching these p columns is tricky, and based on *ad hoc* parameters. Also, computational experience has shown that accurate tangent vectors $(d\lambda/ds, dY/ds)$ are essential, and the accuracy of Gaussian elimination may not be good enough. A

conceptually elegant, as well as accurate, algorithm is to compute the QR factorization with column interchanges of $D\rho_W$,

$$Q\, D\rho_W\, P^t P z = \begin{pmatrix} * & \cdots & * & * \\ & \ddots & \vdots & \vdots \\ 0 & & * & * \end{pmatrix} Pz = 0,$$

where Q is a product of Householder reflections and P is a permutation matrix, and then obtain a vector $z \in \ker D\rho_W$ by back substitution. Setting $(Pz)_{p+1} = 1$ is a convenient choice. This scheme provides high accuracy, numerical stability, and a uniform treatment of all $p+1$ columns. Finally,

$$\left(\frac{d\lambda}{ds}, \frac{dY}{ds}\right) = \pm \frac{z}{\|z\|_2},$$

where the sign is chosen to maintain an acute angle with the previous tangent vector on γ. There is a rigorous mathematical criterion, based on a $(p+1)\times(p+1)$ determinant, for choosing the sign, but there is no reason to believe that would be more robust than the angle criterion.

Several features which are a combination of common sense and computational experience should be incorporated into the algorithm. Since most ordinary differential equation solvers only control the local error, the longer the arc length of the zero curve γ gets, the farther away the computed points may be from the true curve γ. Therefore when the arc length gets too long, the last computed point $(\bar{\lambda}, \bar{Y})$ is used to calculate

$$\overline{W} = \frac{\bar{\lambda} F(\bar{Y}) + (1-\bar{\lambda})\bar{Y}}{1-\bar{\lambda}}.$$

Then $\rho_{\overline{W}}(\bar{\lambda}, \bar{Y}) = 0$ exactly, and the zero curve of $\rho_{\overline{W}}(\lambda, Y)$ is followed starting from $(\bar{\lambda}, \bar{Y})$. A rigorous justification for this strategy was given in [52].

Remember that tracking γ was merely a means to an end, namely a zero $\tilde{Y}$ of $F(Y)$. Since γ itself is of no interest (usually), one should not waste computational effort following it too closely. However, since γ is the only sure way to $\tilde{Y}$, losing γ can be disastrous. The tradeoff between computational efficiency and reliability is very delicate, and a fool-proof strategy appears difficult to achieve. This is the reason HOMPACK provides several algorithms; no single algorithm is superior overall, and each of the three beats the other two (sometimes by an order of magnitude) on particular problems.

The normal flow algorithm due to Georg [14] has three phases: prediction, correction, and step size estimation. (13) and (14) are the relevant equations here. For the prediction phase, assume that several points $P^{(1)} = (\lambda(s_1), Y(s_1))$, $P^{(2)} = (\lambda(s_2), Y(s_2))$ on γ with corresponding tangent vectors $(d\lambda/ds(s_1),\ dY/ds(s_1))$, $(d\lambda/ds(s_2),\ dY/ds(s_2))$ have been found, and h is an estimate of the optimal step (in arc length) to take along γ. The prediction of the next point on γ is

$$Z^{(0)} = p(s_2 + h), \tag{18}$$

where $p(s)$ is the Hermite cubic interpolating $(\lambda(s), Y(s))$ at s_1 and s_2. Precisely,

$$p(s_1) = (\lambda(s_1), Y(s_1)), \quad p'(s_1) = (d\lambda/ds(s_1), dY/ds(s_1)),$$
$$p(s_2) = (\lambda(s_2), Y(s_2)), \quad p'(s_2) = (d\lambda/ds(s_2), dY/ds(s_2)),$$

and each component of $p(s)$ is a polynomial in s of degree less than or equal to 3.

Starting at the predicted point $Z^{(0)}$, the corrector iteration is

$$Z^{(n+1)} = Z^{(n)} - \left[D\rho_W(Z^{(n)})\right]^{\dagger} \rho_W(Z^{(n)}), \qquad n = 0, 1, \ldots \tag{19}$$

where $\left[D\rho_W(Z^{(n)})\right]^{\dagger}$ is the Moore-Penrose pseudoinverse of the $p \times (p+1)$ Jacobian matrix $D\rho_W$. Small perturbations of W produce small changes in the trajectory γ, and the family of trajectories γ for varying W is known as the "Davidenko flow". Davidenko [11] was one of the first to propose solving a nonlinear system of equations by following a trajectory of an ordinary differential equation. Geometrically, the iterates given by (19) return to the zero curve along the flow normal to the Davidenko flow, hence the name "normal flow algorithm".

A corrector step ΔZ is the unique minimum norm solution of the equation

$$\left[D\rho_W\right]\Delta Z = -\rho_W. \tag{20}$$

Fortunately ΔZ can be calculated at the same time as the kernel of $\left[D\rho_W\right]$, and with just a little more work. Normally for dense problems the kernel of $\left[D\rho_W\right]$ is found by computing a QR factorization of $\left[D\rho_W\right]$, and then using back substitution. By applying this QR factorization to $-\rho_W$ and using back substitution again, a *particular* solution v to (20) can be found. Let $u \neq 0$ be any vector in the kernel of $\left[D\rho_W\right]$. Then the minimum norm solution of (20) is

$$\Delta Z = v - \frac{v^t u}{u^t u} u. \tag{21}$$

Since the kernel of $\left[D\rho_W\right]$ is needed anyway for the tangent vectors, solving (20) only requires another $\mathcal{O}(p^2)$ operations beyond those for the kernel. The number of iterations required for convergence of (19) should be kept small (say < 4) since QR factorizations of $\left[D\rho_W\right]$ are expensive. The alternative of using $\left[D\rho_W(Z^{(0)})\right]$ for several iterations, which results in linear convergence, is rarely cost effective.

When the iteration (19) converges, the final iterate $Z^{(n+1)}$ is accepted as the next point on γ, and the tangent vector to the integral curve through $Z^{(n)}$ is used for the tangent–this saves a Jacobian matrix evaluation and factorization at $Z^{(n+1)}$. The step-size estimation described next attempts to balance progress along γ with the effort expended on the iteration (19).

Define a contraction factor

$$L = \frac{\left\|Z^{(2)} - Z^{(1)}\right\|}{\left\|Z^{(1)} - Z^{(0)}\right\|}, \tag{22}$$

a residual factor

$$R = \frac{\left\|\rho_W\left(Z^{(1)}\right)\right\|}{\left\|\rho_W\left(Z^{(0)}\right)\right\|}, \tag{23}$$

a distance factor ($Z^* = \lim_{n\to\infty} Z^{(n)}$)

$$D = \frac{\left\|Z^{(1)} - Z^*\right\|}{\left\|Z^{(0)} - Z^*\right\|}, \tag{24}$$

and ideal values $\bar{L}$, $\bar{R}$, $\bar{D}$ for these three. Let h be the current step-size (the distance from Z^* to the previous point found on γ), and $\bar{h}$ the "optimal" step-size for the next step. The goal is to achieve

$$\frac{\bar{L}}{L} \approx \frac{\bar{R}}{R} \approx \frac{\bar{D}}{D} \approx \frac{\bar{h}^q}{h^q} \tag{25}$$

for some q. This leads to the choice

$$\hat{h} = \left(\min\{\bar{L}/L, \bar{R}/R, \bar{D}/D\}\right)^{1/q} h, \tag{26}$$

a worst case choice. To prevent chattering (oscillation between very large and very small steps) and unreasonable values, constants $h_{\min}$ (minimum allowed step-size), $h_{\max}$ (maximum allowed step-size), $B_{\min}$ (contraction factor), and $B_{\max}$ (expansion factor) are chosen, and $\bar{h}$ is taken as

$$\bar{h} = \min\left\{\max\left\{h_{\min}, B_{\min}h, \hat{h}\right\}, B_{\max}h, h_{\max}\right\}. \tag{27}$$

There are eight parameters in this process: $\bar{L}$, $\bar{R}$, $\bar{D}$, $h_{\min}$, $h_{\max}$, $B_{\min}$, $B_{\max}$, q. HOMPACK permits the user to specify nondefault values for any of these. The choice of $\bar{h}$ from (27) can be refined further. If (19) converged in one iteration, then $\bar{h}$ should certainly not be smaller than h, hence set

$$\bar{h} := \max\{h, \bar{h}\} \tag{28}$$

if (19) only required one iteration.

To prevent divergence from the iteration (19), if (19) has not converged after K iterations, h is halved and a new prediction is computed. Every time h is halved the old value h_{old} is saved. Thus if (19) has failed to converge in K iterations sometime during this step, the new $\bar{h}$ should not be greater than the value h_{old} known to produce failure. Hence in this case

$$\bar{h} := \min\left\{h_{\text{old}}, \bar{h}\right\}. \tag{29}$$

Finally, if (19) required the maximum K iterations, the step-size should not increase, so in this case set

$$\bar{h} := \min\{h, \bar{h}\}. \tag{30}$$

The logic in (28)–(30) is rarely invoked, but it does have a stabilizing effect on the algorithm.

Rheinboldt's augmented Jacobian algorithm together with step size strategies has been described very well elsewhere [2], [28], and so will only be described briefly here. The augmented Jacobian matrix algorithm has four major phases: prediction, correction, step size estimation, and computation of the solution at $\lambda = 1$. The algorithm here is based on Rheinboldt [28], but with some significant differences: (1) a Hermite cubic rather than a linear predictor is used; (2) a tangent vector rather than a standard basis vector is used to augment the Jacobian matrix of the homotopy map; (3) updated QR factorizations and quasi-Newton updates are used rather than Newton's method; (4) different step size control, necessitated by the use of quasi-Newton iterations, is used; (5) a different scheme for locating the target point at $\lambda = 1$ is used which allows the Jacobian matrix of F to be singular at the solution.

The prediction phase is exactly the same as in the normal flow algorithm. Having the points $P^{(1)} = (\lambda(s_1), Y(s_1)), P^{(2)} = (\lambda(s_2), Y(s_2))$ on γ with corresponding tangent vectors

$$T^{(1)} = \begin{pmatrix} \frac{d\lambda}{ds}(s_1) \\ \frac{dY}{ds}(s_1) \end{pmatrix}, \qquad T^{(2)} = \begin{pmatrix} \frac{d\lambda}{ds}(s_2) \\ \frac{dY}{ds}(s_2) \end{pmatrix},$$

the prediction $Z^{(0)}$ of the next point on γ is given by (18).

In order to use this predictor, a means of calculating the tangent vector $T^{(2)}$ at a point $P^{(2)}$ is required. This is done by solving the system

$$\begin{bmatrix} D\rho_W\left(P^{(2)}\right) \\ T^{(1)\,t} \end{bmatrix} z = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \tag{31}$$

for z, where $D\rho_W$ is the $p \times (p+1)$ Jacobian of ρ_W. Normalizing z gives

$$T^{(2)} = \frac{z}{\|z\|}. \tag{32}$$

The last row of (31) insures that the tangent $T^{(2)}$ makes an acute angle with the previous tangent $T^{(1)}$. It is the augmentation of the Jacobian matrix with this additional row which motivates the name "augmented Jacobian matrix algorithm." The solution to (31) is found by computing a QR factorization of the matrix, and then using back substitution [59].

Starting with the predicted point $Z^{(0)}$, the correction is performed by a quasi-Newton iteration defined by

$$Z^{(k+1)} = Z^{(k)} - \begin{bmatrix} A^{(k)} \\ T^{(2)\,t} \end{bmatrix}^{-1} \begin{pmatrix} \rho_W\left(Z^{(k)}\right) \\ 0 \end{pmatrix}, \qquad k = 0, 1, \ldots \tag{33}$$

where $A^{(k)}$ is an approximation to the Jacobian matrix $D\rho_W\left(Z^{(k)}\right)$. The last row of the matrix in (33) insures that the iterates lie in a hyperplane perpendicular to the tangent vector $T^{(2)}$. (33) is the quasi-Newton iteration for solving the augmented nonlinear system

$$\begin{pmatrix} \rho_W(y) \\ T^{(2)\,t}\left(y - Z^{(0)}\right) \end{pmatrix} = 0. \tag{34}$$

A corrector step $\Delta Z^{(k)}$ is the unique solution to the equation

$$\begin{bmatrix} A^{(k)} \\ T^{(2)\,t} \end{bmatrix} \Delta Z^{(k)} = \begin{pmatrix} -\rho_W\left(Z^{(k)}\right) \\ 0 \end{pmatrix}. \tag{35}$$

The matrix on the left side of this equation is produced by successive Broyden rank one updates [59] of the matrix in (31). Precisely, letting $Z^{(-1)} = P^{(2)}$, $A^{(-1)} = D\rho_W\left(P^{(2)}\right)$, and

$$M^{(k)} = \begin{bmatrix} A^{(k)} \\ T^{(2)\,t} \end{bmatrix},$$

the update formulas are

$$M^{(-1)} = \begin{bmatrix} A^{(-1)} \\ T^{(2)\,t} \end{bmatrix} = \begin{bmatrix} D\rho_W\left(P^{(2)}\right) \\ T^{(1)\,t} \end{bmatrix} + e_{p+1}\left(T^{(2)} - T^{(1)}\right)^t, \tag{36}$$

and

$$M^{(k+1)} = M^{(k)} + \frac{\left(\tilde{\Delta}\rho_W - M^{(k)}\Delta Z^{(k)}\right)\Delta Z^{(k)\,t}}{\Delta Z^{(k)\,t}\,\Delta Z^{(k)}}, \qquad k = -1, 0, \ldots \tag{37}$$

where

$$\tilde{\Delta}\rho_W = \begin{pmatrix} \rho_W\left(Z^{(k+1)}\right) - \rho_W\left(Z^{(k)}\right) \\ 0 \end{pmatrix}.$$

These updates can be done in QR factored form, requiring a total of $\mathcal{O}(p^2)$ operations for each iteration in the correction process [59]. When the iteration (33) converges within some tolerance, the final iterate $Z^{(*)}$ is accepted as the next point on the zero curve γ.

The step size estimation algorithm is an adaptation of a procedure developed by Rheinboldt [28]. At each point $P^{(k)}$ with tangent $T^{(k)}$ along γ, the curvature is estimated by the formula

$$\left\|w^{(k)}\right\| = \frac{2}{\Delta s_k}\left|\sin\left(\alpha_k/2\right)\right|, \tag{38}$$

where

$$w^{(k)} = \frac{T^{(k)} - T^{(k-1)}}{\Delta s_k}, \quad \alpha_k = \arccos\left(T^{(k)\,t}T^{(k-1)}\right), \quad \Delta s_k = \left\|P^{(k)} - P^{(k-1)}\right\|.$$

Intuitively, α_k represents the angle between the last two tangent vectors, and the curvature is approximated by the Euclidean norm of the difference between these two tangents divided by Δs_k.

This curvature data can be extrapolated to produce a prediction for the curvature for the next step

$$\hat{\xi}_k = \left\|w^{(k)}\right\| + \frac{\Delta s_k}{\Delta s_k + \Delta s_{k-1}} \left(\left\|w^{(k)}\right\| - \left\|w^{(k-1)}\right\|\right). \tag{39}$$

Since $\hat{\xi}_k$ can be negative, use

$$\xi_k = \max(\xi_{min}, \hat{\xi}_k) \quad \text{for some small} \quad \xi_{min} > 0, \tag{40}$$

as the predicted curvature for the next step.

The goal in estimating the optimal step size is to keep the error in the prediction $\|Z^{(0)} - Z^{(*)}\|$ relatively constant, so that the number of iterations required by the corrector will be stable. This is achieved by choosing the step size as

$$\hat{h} = \sqrt{\frac{2\delta_k}{\xi_k}}, \tag{41}$$

where δ_k represents the ideal starting error desired for the prediction step. δ_k is chosen as a function of the tolerance for tracking the curve and is also restricted to be no larger than half of Δs_k.

As with the normal flow algorithm, additional refinements on the optimal step size are made in order to prevent chattering and unreasonable values. In particular, $\bar{h}$ is chosen to satisfy equations (27) and (29). This $\bar{h}$ is then used as the step size for the next step.

The final phase of the algorithm, computation of the solution at $\lambda = 1$, is entered when a point $P^{(2)}$ is generated such that $P_1^{(2)} \geq 1$. $P^{(2)}$ is the first such point, so the solution must lie on γ somewhere between $P^{(2)}$ and the previous point $P^{(1)}$. The algorithm for finding this solution is a two step process which is repeated until the solution is found. First, starting from a point $P^{(k)}$, a prediction $Z^{(k-2)}$ for the solution is generated such that $Z_1^{(k-2)} = 1$. Second, a single quasi-Newton iteration is performed to produce a new point $P^{(k+1)}$ close to γ, but not necessarily on the hyperplane $\lambda = 1$.

Normally, the prediction $Z^{(k-2)}$ is computed by a secant method using the last two points $P^{(k)}$ and $P^{(k-1)}$:

$$Z^{(k-2)} = P^{(k)} + \left(P^{(k-1)} - P^{(k)}\right) \frac{\left(1 - P_1^{(k)}\right)}{\left(P_1^{(k-1)} - P_1^{(k)}\right)}. \tag{42}$$

However, this formula can potentially produce a disastrous prediction (e.g., if $|P_1^{(k-1)} - P_1^{(k)}| \ll |1 - P_1^{(k)}|$), so an additional scheme is added to ensure that this does not happen. In order to implement this scheme, a point $P^{(opp)}$ must be saved. This

point is chosen as the last point computed from a quasi-Newton step which is on the opposite side of the hyperplane $\lambda = 1$ from $P^{(k)}$. Thus, the points $P^{(opp)}$ and $P^{(k)}$ bracket the solution. The prediction $Z^{(k-2)}$ may be bad whenever the inequality

$$\left\| Z^{(k-2)} - P^{(k)} \right\| > \left\| P^{(k)} - P^{(opp)} \right\| \tag{43}$$

is true. In this case, $Z^{(k-2)}$ is recomputed from the equation

$$Z^{(k-2)} = P^{(k)} + \left(P^{(opp)} - P^{(k)} \right) \frac{\left(1 - P_1^{(k)} \right)}{\left(P_1^{(opp)} - P_1^{(k)} \right)}. \tag{44}$$

This chord method, while much safer than the secant method (42), is used only in the special case (43) because it has a much slower rate of convergence than the secant method.

An exception to these linear prediction schemes occurs with the first step of the final phase. Since the tangents $T^{(1)}$ and $T^{(2)}$ at $P^{(1)}$ and $P^{(2)}$ are available, this information is used to generate a Hermite cubic polynomial $p(s)$ for calculating the first prediction point $Z^{(0)}$. This is done by finding the root $\bar{s}$ of the equation $p_1(s) = 1$. $Z^{(0)}$ is then given by

$$Z^{(0)} = p(\bar{s}). \tag{45}$$

After the predictor $Z^{(k-2)}$ has been determined, a quasi-Newton step is taken to get the point $P^{(k+1)}$. This step is defined by

$$P^{(k+1)} = Z^{(k-2)} + \Delta Z^{(k-2)}, \tag{46}$$

where $\Delta Z^{(k-2)}$ is the solution to (35). Again, the matrix in (35) is produced by the rank one updates (36) and (37).

The alternating process of computing a prediction and taking a quasi-Newton step is repeated until the solution is found.

4. Numerical results. Consider the magneto-hydronamics (MHD) nonlinear two-point boundary value problem [21]:

$$H'''(\eta) = H(\eta)\, H''(\eta) - \frac{(H'(\eta))^2}{2} + m\, H'(\eta) + 2\, G(\eta)^2, \tag{47}$$

$$G''(\eta) = H(\eta)\, G'(\eta) - H'(\eta)\, G(\eta) + m\, G(\eta), \tag{48}$$

$$\theta''(\eta) = \Pr H(\eta)\, \theta'(\eta) \tag{49}$$

with the boundary conditions

$$H(0) = -A, \quad H'(0) = 0, \quad G(0) = 1, \quad \theta(0) = 1, \tag{50}$$

$$H'(\eta) \to 0, \quad G(\eta) \to 0, \quad \theta(\eta) \to 0 \quad \text{as } \eta \to \infty, \tag{51}$$

where Pr denotes the Prandtl number and m is a magnetic parameter. Note that $H(\eta)$ and $G(\eta)$ are determined by equations (47) and (48) independent of $\theta(\eta)$, and thus $\theta(\eta)$

can be uncoupled from the system. Once $H(\eta)$ has been found from (47), (48), (50), and (51), $\theta(\eta)$ can be determined from the one-dimensional two-point boundary value problem constituted by (49) and the θ parts of (50), (51). The (routine) computation for $\theta(\eta)$ will not be discussed here.

In practice the boundary conditions (51) are replaced by

$$H'(\tau) = G(\tau) = \theta(\tau) = 0 \tag{51a}$$

for some τ sufficiently large such that

$$|H(\eta) - H(\tau)| + |G(\eta)| + |\theta(\eta)| \approx 0 \quad \text{for } \tau \leq \eta < \infty.$$

Let S_n be the finite dimensional vector space with basis $\{B_{j,k,\mathbf{t}}(x)\}_{j=1}^{n}$, where $B_{j,k,\mathbf{t}}(x)$ is the jth B-spline of order k (degree $\leq k-1$) defined on the knot sequence $\mathbf{t} = (t_1, t_2, \ldots, t_{n+k})$. When there is no ambiguity $B_{j,k,\mathbf{t}}(x)$ is simply written as $B_j(x)$. For this problem the knot sequence $\mathbf{t}$ is based on the breakpoint sequence

$$\begin{aligned}\Xi = (&0, .25, .50, .75, 1.0, 1.25, 1.50, 1.75, 2.0, 2.25, 2.50, 2.75,\\ &3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 7.0, 8.0, 9.0, 11.0, 13.0, 15.0,\\ &18.0, 21.0, 24.0, 28.0, 32.0, 36.0, 41.0, 46.0, 51.0, 60.0,\\ &70.0, 80.0, 90.0, 100.0),\end{aligned}$$

following the convention

$$t_1 = t_2 = \cdots = t_k \qquad \text{and} \qquad t_{n+1} = t_{n+2} = \cdots = t_{n+k}$$

used by deBoor [12]. These repeated knots essentially mean that the spline is free at the endpoints of the approximation interval $[t_k, t_{n+1}]$. Note the distinction between knots and breakpoints [12]. The rest of the knots are simple, i.e., $t_i < t_{i+1}$ for $i = k, \ldots, n$. The functions $H(\eta)$ and $G(\eta)$ have boundary layers (large derivatives, [22]) near $\eta = 0$, and then asymptotically approach a constant as $\eta \to \infty$. The appropriate value for τ (unknown beforehand) is where the solutions level off (within some error tolerance) at their asymptotic value. Short of dynamically adapting the knot sequence (which, as deBoor [12] points out, is rarely cost effective), a reasonable strategy is to space the knots farther apart as η increases. Depending on the values of n and k, only an initial subsequence of the breakpoint sequence Ξ above is used.

The approximations are

$$H(\eta) = \sum_{j=1}^{N+2} \alpha_j B_j(\eta), \tag{52}$$

$$\alpha_1 = -A, \quad \alpha_2 = \frac{A\, B_1'(0)}{B_2'(0)}, \quad \alpha_{N+2} = \frac{-\alpha_{N+1} B_{N+1}'(\tau-)}{B_{N+2}'(\tau-)}, \tag{53}$$

$$G(\eta) = \sum_{j=1}^{N+2} \beta_j B_j(\eta), \tag{54}$$

$$\beta_1 = 1, \quad \beta_{N+2} = 0. \tag{55}$$

The boundary conditions (50)–(51) force the equations (53) and (55). The Galerkin approximation is the nonlinear system of equations

$$\begin{aligned} &\langle -H''' + H\,H'' - (H')^2/2 + m\,H' + 2G^2,\ B_i\rangle = 0, \qquad i = 3,\ldots,N+1, \\ &\langle -G'' + H\,G' - H'\,G + m\,G,\ B_i\rangle = 0, \qquad i = 2,\ldots,N+1, \end{aligned} \tag{56}$$

where

$$\langle u, v\rangle = \int_0^\tau u(\eta)v(\eta)\,d\eta.$$

Let $Y = (\alpha_3, \alpha_4, \ldots, \alpha_{N+1}, \beta_2, \beta_3, \ldots, \beta_{N+1})^t$ and $F(Y) = 0$ be given by the $p = 2N - 1 = 2n - 5$ equations (56).

This MHD problem is fairly difficult, and both homotopy and standard quasi-Newton methods based on simple shooting and multiple shooting are known to fail for certain ranges of the parameters A and m [21]. Finite difference, collocation, or Galerkin formulations handle the fluid dynamics boundary layers better, but the failure of quasi-Newton algorithms (started distant from the solution) on such large dimensional approximations to fluid dynamics problems is well documented [56], [57]. Thus the alternatives are to combine a homotopy algorithm with a finite difference approximation [56], a collocation approximation [58], or a Galerkin approximation (the present approach).

Table 1 shows some numerical results obtained by applying subroutine FIXPNF (normal flow algorithm) of HOMPACK to (56). The normal flow algorithm was the most efficient of the three algorithms in HOMPACK for this problem. While predicting the best algorithm for a given problem is risky, some general rules of thumb are: the ODE-based algorithm is the most robust but the most expensive; if the zero curve γ has very sharp turns (it doesn't for this problem) then the ODE-based algorithm is the best; if the Jacobian matrices are very expensive to evaluate (they aren't for this problem) then the quasi-Newton augmented Jacobian scheme is best; otherwise the normal flow algorithm is best. The values $H(\infty)$ are from [21], and τ can be inferred from n, k, and the breakpoint sequence Ξ listed above. The integrals in (56) were computed by 10-point Gaussian quadrature over each subinterval, and are thus essentially exact. NFE is the number of Jacobian matrix evaluations, and the format of the CPU time (on a VAX 11/780) is hh:mm:ss. The local curve tracking tolerance was 10^{-4} and the final accuracy (the end game tolerance [59]) was 10^{-8}.

The problem gets easier (weaker boundary layers, asymptotic value reached sooner) as either A or m increases. For example, with $n = N+2 = 12$ and spline order $k = 6$, $\tau = t_{13} = t_{18} = 1.75$, and as the plots in [21] show, this τ is clearly not large enough for $A = -1$, $m = 1.0$, but it is about right for $A = m = 4$. Taking $n = 24$, $k = 6$ gives $\tau = 7.0$, which produces better results for the case $A = -1$, $m = 1.0$, and finally $n = 32$, $k = 6$ gives $\tau = 24.0$ and the theoretically best possible results

TABLE 1

A	m	k	$N+2$	$-H(\tau)$	$-H(\infty)$	NFE	CPU time	arc length
−1.0	1.0	6	12	−.80700	−.43166	29	13:28	1.309
−1.0	2.0	6	12	−.88173	−.78156	29	13:28	1.387
−1.0	4.0	6	12	−.94477	−.93015	30	14:37	1.518
0.0	1.0	6	12	.11991	.25331	25	11:13	1.067
0.0	2.0	6	12	.07372	.10858	25	11:24	1.103
0.0	4.0	6	12	.03558	.04078	20	9:07	1.198
1.0	1.0	6	12	1.06079	1.0898	21	9:38	1.141
1.0	2.0	6	12	1.03959	1.0481	20	8:57	1.187
1.0	4.0	6	12	1.02103	1.0225	15	6:42	1.272
2.0	1.0	6	12	2.02744	2.0318	18	8:00	1.257
2.0	2.0	6	12	2.01968	2.0213	24	10:38	1.296
2.0	4.0	6	12	2.01193	2.0123	15	6:37	1.365
4.0	1.0	6	12	4.00625	4.0064	18	7:58	1.471
4.0	2.0	6	12	4.00530	4.0054	18	7:58	1.491
4.0	4.0	6	12	4.00402	4.0041	15	6:38	1.530
−1.0	1.0	6	24	−.43877	−.43166	32	1:09:37	1.916
−1.0	2.0	6	24	−.78196	−.78156	27	58:20	1.604
−1.0	4.0	6	24	−.93019	−.93015	24	51:32	1.953
0.0	1.0	6	24	.25286	.25331	26	55:45	1.986
0.0	2.0	6	24	.10852	.10858	21	44:53	1.804
0.0	4.0	6	24	.04073	.04078	25	53:27	1.903
−1.0	1.0	6	32	−.43165	−.43166	33	2:15:48	2.765
−1.0	2.0	6	32	−.78158	−.78156	34	2:20:28	2.334
−1.0	4.0	6	32	−.93018	−.93015	27	1:52:50	2.752
−1.0	1.0	4	24	−.42854	−.43166	41	55:37	2.196
−1.0	2.0	4	24	−.78043	−.78156	38	51:12	1.810
−1.0	4.0	4	24	−.93062	−.93015	29	39:06	2.128

($\mathcal{O}(h^6) = \mathcal{O}\big((1/4)^6\big) = \mathcal{O}(10^{-4})$) for $A = -1$, $m = 1.0$. The lower order spline ($k = 4$) produces the expected $\mathcal{O}(h^4)$ accuracy at the right τ for $A = -1$, $m = 4.0$, but not for $A = -1$, $m = 1.0$ because τ is too small for this case.

There are three concluding observations. First, although the theory in §2 is not directly applicable to this MHD problem, these numerical results suggest that the homotopy algorithm is more widely applicable than the theory indicates. Second, the accuracy is exactly what would be expected of a Galerkin approximation given the spline order and knot spacing. Third, all the solutions for Table 1 were obtained on the first computer run by HOMPACK with no tweaking of initial points or error tolerances whatsoever.

REFERENCES

[1] E. ALLGOWER AND K. GEORG, *Simplicial and continuation methods for approximating fixed points*, SIAM Rev., 22 (1980), pp. 28–85.

[2] S. C. BILLUPS, *An augmented Jacobian matrix algorithm for tracking homotopy zero curves*, M.S. Thesis, Dept. of Computer Sci., VPI & SU, Blacksburg, VA, Sept., 1985.

[3] P. BOGGS, *The solution of nonlinear systems of equations by A-stable integration techniques*, SIAM J. Numer. Anal., 8 (1971), pp. 767–785.

[4] F. BREZZI, J. RAPPAZ, AND P. A. RAVIART, *Finite dimensional approximation of nonlinear problems. Part I: Branches of nonsingular solutions*, Numer. Math., 36 (1980), pp. 1–25.

[5] S. N. CHOW, J. MALLET-PARET, AND J. A. YORKE, *Finding zeros of maps: homotopy methods that are constructive with probability one*, Math. Comp., 32 (1978), pp. 887–899.

[6] P. G. CIARLET, M. H. SCHULTZ, AND R. S. VARGA, *Numerical methods of high-order accuracy for nonlinear boundary value problems. I: One-dimensional problems*, Numer. Math., 9 (1967), pp. 394–430.

[7] P. G. CIARLET, M. H. SCHULTZ, AND R. S. VARGA, *Numerical methods of high-order accuracy for nonlinear boundary value problems. II: Nonlinear boundary conditions*, Numer. Math., 11 (1968), pp. 331–345.

[8] P. G. CIARLET, M. H. SCHULTZ, AND R. S. VARGA, *Numerical methods of high-order accuracy for nonlinear boundary value problems. III: Eigenvalue problems*, Numer. Math., 12 (1968), pp. 120–133.

[9] P. G. CIARLET, M. H. SCHULTZ, AND R. S. VARGA, *Numerical methods of high-order accuracy for nonlinear boundary value problems. IV: Periodic boundary conditions*, Numer. Math., 12 (1968), pp. 266–279.

[10] P. G. CIARLET, M. H. SCHULTZ, AND R. S. VARGA, *Numerical methods of high-order accuracy for nonlinear boundary value problems. V: Monotone operators*, Numer. Math., 13 (1969), pp. 51–77.

[11] D. F. DAVIDENKO, *On the approximate solution of systems of nonlinear equations*, Ukrain. Mat. Ž., 5 (1953), pp. 196–206.

[12] C. DEBOOR, *A Practical Guide to Splines*, Springer-Verlag, New York, 1978.

[13] B. C. EAVES AND R. SAIGAL, *Homotopies for computation of fixed points on unbounded regions*, Math. Programming, 3 (1972), pp. 225–237.

[14] K. GEORG, private communication, 1981.

[15] D. GOTTLIEB AND S. A. ORSZAG, *Numerical Analysis of Spectral Methods: Theory and Applications*, Society for Industrial and Applied Mathematics, Philadelphia, 1977.

[16] H. B. KELLER, *Numerical Solution of Two-point Boundary Value Problems*, Society for Industrial and Applied Mathematics, Philadelphia, 1976.

[17] ———, *Approximation methods for nonlinear problems with application to two-point boundary value problems*, Math. Comp., 29 (1975), pp. 464–474.

[18] ———, *Numerical solution of bifurcation and nonlinear eigenvalue problems*, in Applications of Bifurcation Theory, Academic Press, New York, 1977.

[19] R. W. KLOPFENSTEIN, *Zeros of nonlinear functions*, J. Assoc. Comput. Mech., 8 (1961), pp. 336–373.

[20] M. KUBICEK, *Dependence of solutions of nonlinear systems on a parameter*, ACM Trans. Math. Software, 2 (1976), pp. 98–107.

[21] S. K. KUMAR, W. I. THACKER, AND L. T. WATSON, *Magnetohydrodynamic flow and heat transfer about a rotating disk with suction and injection at the disk surface*, Comput. & Fluids, to appear.

[22] L. D. LANDAU AND E. M. LIFSHITZ, *Fluid Mechanics*, Pergamon, Oxford, 1959.

[23] R. MEJIA, *CONKUB: A conversational path-follower for systems of nonlinear equations*, J. Comput. Phys., 63 (1986), pp. 67–84.

[24] R. MENZEL AND H. SCHWETLICK, *Zur Lösung parameterabhängiger nichtlinearer Gleichungen mit singulären Jacobi-Matrizen*, Numer. Math., 30 (1978), pp. 65–79.

[25] G. MEYER, *On solving nonlinear equations with a one-parameter operator embedding*, SIAM J. Numer. Anal., 5 (1968), pp. 739–752.

[26] A. P. MORGAN, *A transformation to avoid solutions at infinity for polynomial systems*, Appl. Math. Comput., 18 (1986), pp. 77–86.

[27] ———, *A homotopy for solving polynomial systems*, Appl. Math. Comput., 18 (1986), pp. 87–92.

[28] W. C. RHEINBOLDT AND J. V. BURKARDT, *Algorithm 596: A program for a locally parameterized continuation process*, ACM Trans. Math. Software, 9 (1983), pp. 236-241.

[29] R. SAIGAL AND M. J. TODD, *Efficient acceleration techniques for fixed point algorithms*, SIAM J. Numer. Anal., 15 (1978), pp. 997–1007.
[30] M. H. SCHULTZ, *Spline Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1973.
[31] L. F. SHAMPINE AND M. K. GORDON, *Computer Solution of Ordinary Differential Equations: The Initial Value Problem*, W. H. Freeman, San Francisco, 1975.
[32] C. Y. WANG AND L. T. WATSON, *Squeezing of a viscous fluid between elliptic plates*, Appl. Sci. Res., 35 (1979), pp. 195–207.
[33] ——, *Viscous flow between rotating discs with injection on the porous disc*, Z. Angew. Math. Phys., 30 (1979), pp. 773–787.
[34] ——, *On the large deformations of C-shaped springs*, Internat. J. Mech. Sci., 22 (1980), pp. 395–400.
[35] ——, *Theory of the constant force spring*, J. Appl. Mech., 47 (1980), pp. 956–958.
[36] ——, *The fluid-filled cylindrical membrane container*, J. Engrg. Math., 15 (1981), pp. 81–88.
[37] ——, *Equilibrium of heavy elastic cylindrical shells*, J. Appl. Mech., 48 (1981), pp. 582–586.
[38] ——, *The elastic catenary*, Internat. J. Mech. Sci., 24 (1982), pp. 349–357.
[39] ——, *Free rotation of a circular ring about a diameter*, Z. Angew. Math. Phys., 34 (1983), pp. 13–24.
[40] C. Y. WANG, L. T. WATSON, AND M. P. KAMAT, *Buckling, postbuckling and the flow through a tethered elastic cylinder under external pressure*, J. Appl. Mech., 50 (1983), pp. 13–18.
[41] L. T. WATSON, T. Y. LI AND C. Y. WANG, *Fluid dynamics of the ellilptic porous slider*, J. Appl. Mech., 45 (1978), pp. 435–436.
[42] L. T. WATSON AND C. Y. WANG, *A homotopy method applied to elastica problems*, Internat. J. Solids Structures, 17 (1981), pp. 29–37.
[43] ——, *Deceleration of a rotating disc in a viscous fluid*, Phys. Fluids, 22 (1979), pp. 2267–2269.
[44] ——, *The circular leaf spring*, Acta Mech., 40 (1981), pp. 25–32.
[45] ——, *Hanging an elastic ring*, Internat. J. Mech. Sci., 23 (1981), pp. 161–168.
[46] ——, *Overhang of a heavy elastic sheet*, Z. Angew. Math. Phys., 33 (1982), pp. 17–23.
[47] ——, *Periodically supported heavy elastic sheet*, J. Engrg. Mech., 109 (1983), pp. 811–820.
[48] ——, *The equilibrium states of a heavy rotating column*, Internat. J. Solids Structures, 19 (1983), pp. 653–658.
[49] L. T. WATSON AND W. H. YANG, *Optimal design by a homotopy method*, Applicable Anal., 10 (1980), pp. 275–284.
[50] ——, *Methods for optimal engineering design problems based on globally convergent methods*, Comput. & Structures, 13 (1981), pp. 115–119.
[51] L. T. WATSON AND D. FENNER, *Chow-Yorke algorithm for fixed points or zeros of C^2 maps*, ACM Trans. Math. Software, 6 (1980), pp. 252–260.
[52] L. T. WATSON, *A globally convergent algorithm for computing fixed points of C^2 maps*, Appl. Math. Comput., 5 (1979), pp. 297–311.
[53] ——, *Computational experience with the Chow-Yorke algorithm*, Math. Programming, 19 (1980), pp. 92–101.
[54] ——, *Solving the nonlinear complementarity problem by a homotopy method*, SIAM J. Control Optim., 17 (1979), pp. 36–46.
[55] ——, *An algorithm that is globally convergent with probability one for a class of nonlinear two-point boundary value problems*, SIAM J. Numer. Anal., 16 (1979), pp. 394–401.
[56] ——, *Solving finite difference approximations to nonlinear two-point boundary value problems by a homotopy method*, this Journal, 1 (1980), pp. 467–480.
[57] ——, *Engineering applications of the Chow-Yorke algorithm*, Appl. Math. Comput., 9 (1981), pp. 111–133.
[58] L. T. WATSON AND M. R. SCOTT, *Solving spline collocation approximations to nonlinear two-point boundary value problems by a homotopy method*, Tech. Rep. CS84015-R, Dept. of Computer Sci., VPI&SU, Blacksburg, VA, 1984.
[59] L. T. WATSON, S. C. BILLUPS, AND A. P. MORGAN, *HOMPACK: A suite of codes for globally convergent homotopy algorithms*, Tech. Rep. 85–34, Dept. of Industrial and Operations Eng., Univ. of Michigan, Ann Arbor, MI, 1985.
[60] L. T. WATSON, *Numerical linear algebra aspects of globally convergent homotopy methods*, Tech. Report TR–85–14, Dept. of Computer Sci., VPI&SU, Blacksburg, VA, 1985.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 5, September 1987

© 1987 Society for Industrial and Applied Mathematics
009

A HIGH-ORDER METHOD FOR STIFF BOUNDARY VALUE PROBLEMS WITH TURNING POINTS*

DAVID L. BROWN† AND JENS LORENZ‡

Abstract. This paper describes some high-order collocation-like methods for the numerical solution of stiff boundary-value problems with turning points. The presentation concentrates on the implementation of these methods in conjunction with the implementation of the a priori mesh construction algorithm introduced by Kreiss, Nichols and Brown [SIAM J. Numer. Anal., 23 (1986), pp. 325–368] for such problems. Numerical examples are given showing the high accuracy which can be obtained in solving the boundary value problem for singularly perturbed ordinary differential equations with turning points.

Key words. stiff boundary value problems, turning points, collocation, mesh construction

AMS(MOS) subject classifications. 34A50, 34B05, 34E20, 65L10, 65L50

Introduction. In recent papers [6], [12], Kreiss, Nichols and Brown introduced a new numerical method for solving boundary value problems for stiff linear systems of ordinary differential equations (ODEs). The method they presented is unique among methods for boundary value problems in that it provides an a priori procedure for constructing a difference mesh which will be appropriate for resolving rapidly varying features of the solutions of these problems, such as those that arise in connection with "turning points." Once the mesh has been constructed, the differential equations are approximated using a combination of symmetric and unsymmetric two-point difference formulas in a way which essentially gives second order accuracy. In this paper, we show how high-order collocation methods can be used in a natural way in conjunction with the ideas presented in [6], [12] in order to increase the accuracy of the computed solutions. Numerical experiments show that extremely accurate results can be obtained in this way.

It should be emphasized that the use of collocation methods for solving stiff boundary value problems is not a new idea (see e.g. the papers of Ascher and Weiss [2], [3], and Ringhofer [13]). Symmetric collocation formulas, such as those used in [2], [3] have been shown to work well for stiff boundary value problems if the mesh is carefully constructed so as to resolve boundary layers, and if no turning points are present. (A certain eigenvalue condition must be satisfied as well (see [1], [10], [11]).) For problems with turning points, however, numerical experiments have shown that there can be difficulties with applying symmetric schemes directly (see e.g. [12]). For this reason, as suggested by the ideas in [12], we describe a combination of symmetric and unsymmetric formulas which can be used for a discretization of the problem. In order to apply these formulas the system of ODEs must be transformed to a form where growing, moderate, and decaying modes are essentially decoupled. This transformation is done automatically in our code, and in fact done simultaneously with the a priori mesh construction. For certain problems without turning points in which fast-growing, moderate, and fast-decaying modes are decoupled in the given differential

* Received by the editors January 13, 1986; accepted for publication (in revised form) September 29, 1986. This research was supported in part by National Science Foundation grant DMS-8312264 and Department of Energy grant DE/AT03/76ER/72012. Part of the computer time used for the code development described in this paper was provided by IBM on an IBM 4341 at the California Institute of Technology.

† Los Alamos National Laboratory, Center for Nonlinear Studies and Computing Division, Los Alamos, New Mexico 87545.

‡ Applied Mathematics, California Institute of Technology, Pasadena, California 91125.

system Ringhofer [13] has suggested and analyzed collocation methods similar to those we present here.

In § 1 we discuss the derivation of the collocation formulas that we use. Section 2 discusses their application to systems of ODEs. In § 3 we briefly discuss the local elimination procedure by which the interior collocation points are eliminated. In order to apply our formulas to systems of ODEs, each equation in the system is considered as a scalar equation, and depending on the size of the real part of the diagonal element in the coefficient matrix, either a symmetric formula or the appropriate unsymmetric formula is applied. The choice of which formula to apply depends on accuracy considerations; this is discussed in § 4. The a priori construction of the difference mesh is done in such a way that the resulting computed solution is guaranteed to be resolved by that mesh. A discussion of the mesh construction procedure is given in § 5. In § 6 we present numerical examples which demonstrate the accuracy of the collocation formulas when applied to a system of two second-order singularly perturbed ODEs for which the exact solution is known.

1. Derivation of the difference formulas for scalar equations. Consider the linear scalar ODE given by

$$\frac{dy(x)}{dx} = a(x)y(x)+f(x), \qquad 0 \leqq x \leqq 1. \tag{1.1}$$

This ODE is to be approximated using collocation formulas. Let $m \geqq 2$ be the number of nodes on the normalized interval $[0, 1]$: $0 = \rho_0 < \rho_1 < \cdots < \rho_m = 1$. The ρ_j are assumed to be placed symmetrically with respect to $\rho = \frac{1}{2}$; for example, the ρ_j can denote the Lobatto points on $[0, 1]$ or the Radau points on $[0, 1)$ together with the Radau points on $(0, 1]$. For the time being, we assume that mesh points $0 = x_1 < x_2 < \cdots < x_N = 1$ have been constructed already, and we consider a fixed mesh interval $x_\nu \leqq x \leqq x_{\nu+1} = x_\nu + h$. The nodes

$$0 = \rho_0 < \rho_1 < \cdots < \rho_m = 1, \qquad m \geqq 2 \text{ fixed},$$

in the normalized interval $0 \leqq \rho \leqq 1$ lead on each mesh interval $[x_\nu, x_{\nu+1}]$ to auxiliary points $x_{\nu j} = x_\nu + h\rho_j$, $j = 0, \cdots, m$, the collocation points.

Ultimately, the formulas derived here will be applied to systems of ODEs in the following way: Each differential equation in the system is considered separately, and based on information about the corresponding diagonal element of the coefficient matrix, either a symmetric or an appropriate unsymmetric formula is selected and applied to that scalar equation. Collocation formulas of the type we will describe lead to two-point difference formulas of a high order of accuracy. The high accuracy is obtained by using values of the coefficients of the ODE at auxiliary points between the actual meshpoints as well as the values at the meshpoints. Since the differential equations for each scalar component of the unknown dependent variable in the system of ODEs involves values of the other components at these auxiliary points as well, it is clear that the same set of collocation points must be used for each scalar equation even if the same difference formulas are not used. We are thus led to difference formulas similar to those studied by Ringhofer [13] for problems without turning points.

Three formulas will be described:

(a) A symmetric formula, suitable for $h|a| = O(1)$:

(b) A right-biased formula, suitable for $\operatorname{Re} a \ll -1$:

(c) A left-biased formula, suitable for $\operatorname{Re} a \gg 1$.

(a) *The symmetric formula.* For $j = 1, \cdots, m$ there are uniquely determined weights w_{jk}^0 $(k = 0, 1, \cdots, m)$ such that for all polynomials $p(s)$ of degree $\leqq m$

$$\int_0^{\rho_j} p(s)\, ds = \sum_{k=0}^{m} w_{jk}^0 p(\rho_k).$$

(These weights can be expressed as integrals of the Lagrange interpolation polynomials (see e.g. [16]).) Now let $x_{\nu+1} = x_\nu + h$, $x_{\nu j} = x_\nu + h\rho_j$, $j = 0, \cdots, m$. Integrating (1.1), we have

$$y(x_{\nu j}) - y(x_\nu) = h \int_0^{\rho_j} (ay + f)(x_\nu + hs)\, ds,$$

which motivates the difference equations

$$u_{\nu j} - u_\nu = h \sum_{k=0}^{m} w_{jk}^0 [a(x_{\nu k}) u_{\nu k} + f(x_{\nu k})], \qquad j = 1, \cdots, m. \tag{1.2}$$

This is a system of m linear equations for the $m+1$ unknowns $u_{\nu j}$, $j = 0, \cdots, m$.

(b) *The right-biased formula.* For $j = 1, \cdots, m$ there are uniquely determined weights w_{jk}^R $(k = 1, \cdots, m)$ such that for all polynomials $p(s)$ of degree $\leqq m-1$

$$\int_0^{\rho_j} p(s)\, ds = \sum_{k=1}^{m} w_{jk}^R p(\rho_k). \tag{1.3}$$

In contrast to the symmetric formula, the node $\rho_0 = 0$ is not allowed on the right-hand side of (1.3). As above, this motivates the difference equations

$$u_{\nu j} - u_\nu = h \sum_{k=1}^{m} w_{jk}^R [a(x_{\nu k}) u_{\nu k} + f(x_{\nu k})], \qquad j = 1, \cdots, m.$$

Again we have a system of m equations for $m+1$ unknowns.

(c) *The left-biased formula.* For $j = 1, \cdots, m$ there are uniquely determined weights w_{jk}^L $(k = 0, \cdots, m-1)$ such that for all polynomials $p(s)$ of degree $\leqq m-1$

$$\int_{\rho_{j-1}}^{1} p(s)\, ds = \sum_{k=0}^{m-1} w_{jk}^L p(\rho_k).$$

Integrating (1.1), we obtain

$$y(x_{\nu+1}) - y(x_{\nu j-1}) = h \int_{\rho_{j-1}}^{1} (ay + f)(x_\nu + hs)\, ds$$

which motivates the difference equations

$$u_{\nu+1} - u_{\nu j-1} = h \sum_{k=0}^{m-1} w_{jk}^L [a(x_{\nu k}) u_{\nu k} + f(x_{\nu k})], \qquad j = 1, \cdots, m. \tag{1.4}$$

Summary. With suitable definition for the coefficients e_{jk}^* and w_{jk}^*, each of these three sets of differences formulas can be written in the form

$$\sum_{k=0}^{m} e_{jk}^* u_{\nu k} = h \sum_{k=0}^{m} w_{jk}^* [a(x_{\nu k}) u_{\nu k} + f(x_{\nu k})], \qquad j = 1, \cdots, m. \tag{1.5}$$

These difference formulas are equivalent to collocation or implicit Runge-Kutta type formulas (cf. [16]).

Remark. The local truncation error of the symmetric formula is $O(h^{m+1} \| y^{(m+2)} \|)$ whereas the one-sided formulas lead to a local truncation error of the order

$O(h^m \|y^{(m+1)}\|)$. This holds for all choices of nodes ρ_j. In particular, we have studied two choices for the set of nodes $\{\rho_j\}$, namely the Lobatto points, and the Radau points used twice (as explained below). As is well known, the special choice of Lobatto points leads to an error $O(h^{2m})$ at the mesh points for the symmetric formula. This favorable "superconvergence" result is lost for the one-sided formulas, and indeed we were not able to make use of superconvergence results for the mixed schemes which we describe in the next section. If one uses the Radau points on $(0, 1]$ as nodes $\rho_1, \cdots, \rho_m$, then superconvergence will be obtained for the right-biased formula. For symmetry reasons, the Radau points on $[0, 1)$ should be included as nodes also if both fast-growing and fast-decaying modes are present in the differential equation. However, for the examples tested these schemes with "Radau points twice" do not compare favorably with the schemes using Lobatto points. (See Table 3 in § 7 for an example.) It should also be emphasized that the concept "order of convergence" must be used with care in the present context of stiff equations since one is not studying the case $h \to 0$. The estimates of the local truncation error are relevant, however, since by the mesh construction employed the solution will be smooth with respect to the underlying grid.

2. Application of the collocation formulas to systems of equations. Consider now the system of ODEs given by

$$\frac{dy(x)}{dx} = A(x)y(x) + f(x), \qquad 0 \leqq x \leqq 1 \tag{2.1}$$

where $A(x) \in \mathbf{C}^{n,n}$, $y(x), f(x) \in \mathbf{C}^n$. It follows from the results of [12] that if the matrix $A(x)$ is essentially diagonally dominated, the pth row of (2.1) can be discretized using one of the three formulas introduced in § 1, and the choice of which formula to use for each row is determined by looking at the size and sign of the real part of the diagonal element of that row, Re $a_{pp}(x)$. Specifically, denoting by $z^{(i)}$ the ith component of a vector $z \in \mathbf{C}^n$, we can look at the scalar equation

$$\frac{dy^{(p)}(x)}{dx} - a_{pp}(x)y^{(p)}(x) = \sum_{i \neq p} a_{pi}(x)y^{(i)}(x) + f^{(p)}(x) \tag{2.2}$$

as if the variables $y^{(i)}$, $i \neq p$ are known functions when we make the choice of which formula to use. Applying (1.5) to (2.2) leads to the system of equations

$$\sum_{k=0}^{m} e_{jk}^{(p)} u_{\nu k}^{(p)} = h \sum_{k=0}^{m} w_{jk}^{(p)} \left[a_{pp}(x_{\nu k}) u_{\nu k}^{(p)} + \sum_{i \neq p} a_{pi}(x_{\nu k}) u_{\nu k}^{(i)} + f^{(p)}(x_{\nu k}) \right], \tag{2.3}$$

$$p = 1, \cdots, n, \quad j = 1, \cdots, m.$$

Here the coefficients $w_{jk}^{(p)}$, $j = 1, \cdots, m$, $k = 0, \cdots, m$ can take on the values w_{jk}^0, w_{jk}^R or w_{jk}^L and similarly for the $e_{jk}^{(p)}$. With the notation $\mathbf{u}_\nu := (u_{\nu 0}^T, \ldots, u_{\nu m}^T)^T$ (2.3) can be written as a system of mn linear equations for $\mathbf{u}_\nu$ of the form

$$M\mathbf{u}_\nu = \mathbf{r}_\nu \tag{2.4}$$

where $M \in \mathbf{C}^{(m+1)n, mn}$, $\mathbf{r}_\nu \in \mathbf{C}^{mn}$. Identifying $u_{\nu m} = u_{(\nu+1)0}$, and including n boundary conditions for (2.1) we arrive at a system of mnN linear equations for a total of mnN unknowns, where N is the number of meshpoints x_ν to be used. As is well known, by a process of local elimination, this can be reduced to a banded system of Nn equations for the approximate solution of (2.1) at the meshpoints x_ν, $\nu = 1, \cdots, N$. Our implementation of this reduction is discussed in the next section.

As discussed in [12], the assumption that the system (2.1) is in essentially diagonally dominant form initially is usually too restrictive. As in that paper, it is generally

required that the system be transformed to a form in which the diagonal elements essentially dominate the rows of the matrix $A(x)$. In general, then, it is required that the system (2.1) be discretized in the transformed form.

Define $A_\nu = A(x_\nu)$ and assume that T_ν, $T_{\nu+1}$ are determined such that $T_\nu A_\nu T_\nu^{-1} = D_\nu$, $T_{\nu+1}A_{\nu+1}T_{\nu+1}^{-1} = D_{\nu+1}$ are essentially diagonally dominated. (It is assumed that $|T_{\nu+1} - T_\nu|$ is not large.) Set

$$T(x) = T_\nu + \frac{(x - x_\nu)}{h}(T_{\nu+1} - T_\nu), \qquad x_\nu \leqq x \leqq x_{\nu+1}$$

and introduce the transformed variable

$$\tilde{y}(x) = T(x)y(x).$$

From (2.1) we obtain

$$\tilde{y}' = Ty' + T'y = (TAT^{-1} + T'T^{-1})\tilde{y} + Tf. \tag{2.5}$$

By our assumptions, the matrix

$$TAT^{-1} + T'T^{-1}$$

is essentially diagonally dominated: thus the equation for $\tilde{y}$ can be treated as described above. For technical reasons, we prefer to discretize directly in terms of y instead of $\tilde{y}$ and use the following equation:

$$(Ty)' = (TA + T')y + Tf.$$

With the notation

$$B(x) = T(x)A(x) + T'(x), \qquad g(x) = T(x)f(x),$$

we have

$$(Ty)' = By + g. \tag{2.6}$$

For each component p, $1 \leqq p \leqq n$, of (2.6), the appropriate collocation formula (symmetric, right-biased or left-biased) can be determined by looking at the diagonal element of the pth row of the matrix TAT^{-1} in (2.5). Once this choice has been made, the scalar equation

$$\frac{d(Ty)^{(p)}(x)}{dx} = (By)^{(p)}(x) + g^{(p)}(x) \tag{2.7}$$

can be discretized, and equations analogous to (2.4) can be obtained. With appropriate definitions of the matrices E_{jk}, W_{jk}, they are given by

$$\sum_{k=0}^{m} E_{jk}T(x_{\nu k})u_{\nu k} = h \sum_{k=0}^{m} W_{jk}[B(x_{\nu k})u_{\nu k} + g(x_{\nu k})], \qquad j = 1, \cdots, m. \tag{2.8}$$

This is again a system of mn equations for $(m+1)n$ unknowns of the form (2.4).

3. Local elimination. For efficient numerical computation using these formulas, it is desirable to reduce the system (2.4) to one in which only $u_{\nu 0}$ and $u_{\nu m}$ appear:

$$Bu_{\nu m} + Cu_{\nu 0} = r \tag{3.1}$$

with B, $C \in \mathbf{C}^{n,n}$, $r \in \mathbf{C}^n$. To do this, we eliminate the unknowns $u_{\nu 1}, \cdots, u_{\nu,m-1}$ locally.

After an interchange of columns in the matrix M, we write the system (2.4) as

$$(P, M_0, M_m)\begin{pmatrix} \hat{\mathbf{u}}_\nu \\ u_{\nu 0} \\ u_{\nu m} \end{pmatrix} = \tilde{\mathbf{r}} \tag{3.2}$$

where $\hat{\mathbf{u}}_\nu = (u_{\nu 1}^T, \cdots, u_{\nu,m-1}^T)^T$, P is $mn \times (m-1)n$, M_0 and M_m are $mn \times n$. We assume now that at least $(m-1)n$ rows of the mn rows of P are linearly independent, i.e., that P has rank $(m-1)n$. Applying a Gaussian elimination process with column pivoting we transform the system (3.2) to an equivalent system

$$\left[\begin{array}{c|c|c} U & M_{12} & M_{13} \\ O & C & B \end{array}\right]\begin{bmatrix} \hat{\mathbf{u}} \\ u_{\nu 0} \\ u_{\nu m} \end{bmatrix} = \hat{\mathbf{r}}. \tag{3.3}$$

The desired matrices B, C and the n-vector r in (3.1) can now be read off from the last n equations of (3.3).

4. The choice of collocation formula to apply. So far, we have introduced the collocation formulas that we use, but we have not discussed how the choice is made of which formula to apply to each component of the ODE system. Consider again the scalar equation (1.1) with $a(x) \equiv a$ a constant and the forcing function $f(x)$ set to zero. Assume for the moment that $\lambda := \operatorname{Re} a \leqq 0$, so that either the centered or right-biased formulas will be appropriate to apply. After elimination of the auxiliary variables $u_{\nu j}$, $j = 1, \cdots, m-1$, each collocation formula leads to an equation of the form

$$u_{\nu+1} = G(ha)u_\nu \tag{4.1}$$

where $h = x_{\nu+1} - x_\nu$ is the local meshwidth. By definition, $G(z)$ is the growth function of the method. From consistency considerations, it is clear that $G(z)$ must be an approximation to e^z. Under reasonable assumptions on the ρ_j, it can be shown that for fixed m, and small $|z|$, the growth functions $G_0(z)$ of the symmetric formulas introduced in § 1 will be better approximations to e^z than will be the growth functions $G_R(z)$ for the right-biased formulas. However, as $z \to -\infty$, we have that

$$|G_0(z) - e^z| \to 1,$$

while

$$|G_R(z) - e^z| \to 0.$$

There is therefore a largest negative number $z = -z_C(m)$ for which

$$|G_0(z) - e^z| \leqq |G_R(z) - e^z|.$$

Table 1 lists the values of $z_C(m)$ for the Lobatto points and for the Radau points used twice (cf. § 1). Note that as the number of collocation points increases, the symmetric formula can be used for larger and larger values of h Re a. This is clear from looking at the graphical representations of the growth functions given in Figs. 1 and 2 for the Lobatto points.

Since a desirable property of a method is that its growth function approximate e^z as well as possible, it is natural to switch from the symmetric formula to the right-biased formula if h Re a is less than $-z_C$. Correspondingly, for the case h Re $a > 0$, the switch should be made to the left-biased formula when h Re a exceeds z_C.

For the variable coefficient case, the situation is slightly more complicated, since $a(x_\nu)$ and $a(x_{\nu+1})$ may not have the same properties. Similarly to [9], we choose the

TABLE 1
Values of z_C for different numbers of collocation points. ($ncol = m+1$ is the number of collocation points per mesh interval.)

Lobatto points	
ncol	z_C
2	1.00
3	2.00
4	3.60
5	3.77
6	5.29
7	5.56
8	7.05
9	7.35
Radau points twice	
4	3.74
6	5.03
8	6.27

appropriate method for the interval $[x_\nu, x_{\nu+1}]$ as follows. Let $\alpha_\nu := h \operatorname{Re} a(x_\nu)$, $\alpha_{\nu+1} := h \operatorname{Re} a(x_{\nu+1})$; then there are four cases to consider:

(1) (Moderate case) If $|\alpha_\nu| \leqq z_C$ and $|\alpha_{\nu+1}| \leqq z_C$, then use the symmetric formula.

(2) (Decaying case) If $\alpha_\nu < -z_C$, and $\alpha_{\nu+1} \leqq 0$, or $\alpha_{\nu+1} < -z_C$ and $\alpha_\nu \leqq 0$, then use the right-biased formula.

(3) (Growing case) If $\alpha_\nu > z_C$ and $\alpha_{\nu+1} \geqq 0$, or $\alpha_{\nu+1} > z_C$ and $\alpha_\nu \geqq 0$, take the left-biased formula.

(4) If α_ν and $\alpha_{\nu+1}$ have opposite signs, and either $|\alpha_\nu|$ or $|\alpha_{\nu+1}|$ exceeds z_C there is no natural choice of formula. If this case occurs, Kreiss and Kreiss [9] require that

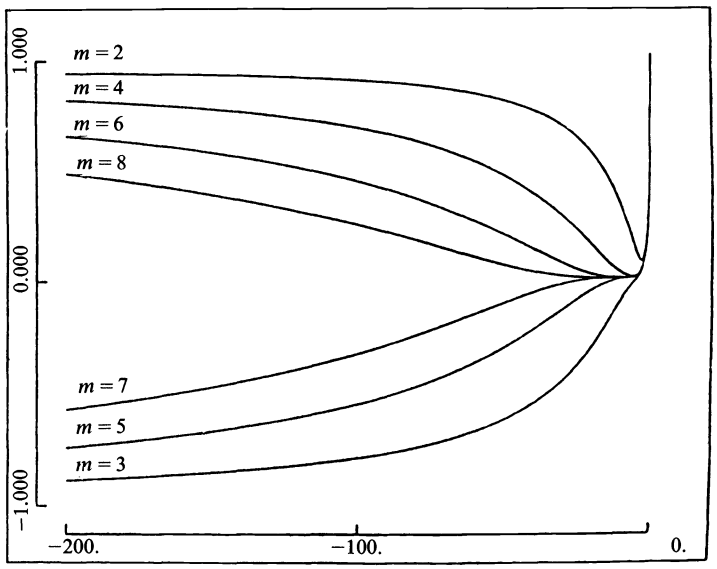


FIG. 1. *Growth functions for Lobatto points, symmetric formulas.*

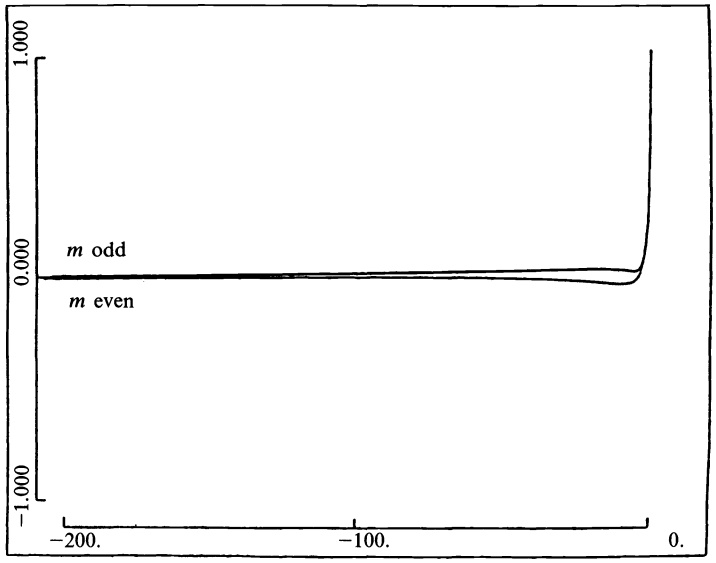


FIG. 2. *Growth functions for Lobatto points, right-biased formulas.*

a meshpoint be added within the interval $[x_\nu, x_{\nu+1}]$ such that on each of the resulting two intervals, one of the above 3 cases will apply. In practice, the mesh construction algorithm of Kreiss, Nichols and Brown [12] always leads to a mesh in which this fourth case never arises. This is basically because this situation would mean that the coefficient $a(x)$ was not logarithmically smooth with respect to the constructed mesh, which is not allowed (cf. § 5 and [6], [12]).

5. Basic principles of a priori mesh construction. An important feature of the method described in [12] for solving stiff two-point boundary value problems was the automatic construction of a mesh which will resolve the solution to be computed. In the code described here we have implemented a similar automatic mesh construction. For the convenience of the reader we describe the basic underlying principles in this section. Consider a change of the independent variable from x to $\tilde{x}$ given by $x = g(\tilde{x})$. Then a function $y(x)$ is said to be *resolved* in the new variable $\tilde{x}$ if in the neighborhood of any point $\tilde{x} = \tilde{x}_0$, a number of derivatives of y with respect to $\tilde{x}$ are bounded by a moderate constant times the function itself, more precisely,

$$\left\| \frac{d^\nu y(g(\tilde{x}))}{d\tilde{x}^\nu} \right\|_{\tilde{x}_0-\varepsilon, \tilde{x}_0+\varepsilon} \leqq K(\|y\|_{\tilde{x}_0-\varepsilon, \tilde{x}_0+\varepsilon} + 1), \qquad \nu = 1, \cdots, p. \tag{5.1}$$

(The size of the "smoothness constant" K, the size of the neighborhood 2ε, and the degree of smoothness p can very with the application.) Similarly, the function is said to be *resolved by a mesh* if (5.1) holds with derivatives replaced by divided differences with respect to the meshpoints. The importance of constructing a mesh that will resolve the computed solution follows from standard truncation error analysis for finite difference methods: If a number of derivatives of the solution are of moderate size, then the truncation error for a high order method will be small. It then follows that for a stable difference approximation, the error in the computed solution will be small also as long as the problem is not too ill posed. We conclude that (5.1) is the essential condition for adjusting the mesh. If (5.1) holds, we take a uniform mesh with respect

to the variable $\tilde{x}$; this will then lead in general to a nonuniform mesh with respect to the variable x.

In [12] it is shown how the stretching function g can be found using only information about the coefficients $A(x)$ and $f(x)$ of the system of ODEs (1.1). Since this information is available a priori for linear systems, a corresponding mesh can be constructed *before* the difference method is applied to the system of ODEs. This is an alternative approach to an adaptive mesh construction strategy.

The analysis in [12] begins with a study of systems of ODEs that are in essentially diagonally dominant form. Ultimately it will be required to make a transformation of a given system of ODEs to this form. However, for the purposes of presenting the following definition and theorem, we consider for the moment a system (2.1) for which such a transformation has already been made.

DEFINITION (Kreiss, Nichols and Brown [12]). A matrix function

$$A(x) := \begin{pmatrix} A_{11}(x) & A_{12}(x) \\ A_{21}(x) & A_{22}(x) \end{pmatrix} \in \mathbf{C}^{n,n}$$

(where $A_{11} \in \mathbf{C}^{n_+ + n_-, n_+ + n_-}$, $A_{22} \in C^{n_0, n_0}$, A_{12}, A_{21} of appropriate dimensions, $n_+ + n_- + n_0 =: n$) is said to be *essentially diagonally dominated* if the elements a_{ij} of A_{11} satisfy

$$\text{(5.2a)} \qquad \operatorname{Re} a_{ii} < 0, \quad i = 1, \cdots, n_-, \quad \operatorname{Re} a_{ii} > 0, \quad i = n_- + 1, \cdots, n_- + n_+,$$

$$\text{(5.2b)} \qquad |\Lambda_R^{-1} B_{11}| \leqq 1 - \delta, \qquad B_{11} := A_{11} - \Lambda_{11}$$

for some $0 < \delta < 1$

$$\text{(5.2c)} \qquad |\Lambda_R^{-1} \Lambda_I| \leqq \rho$$

for some constant $\rho = O(1)$, and

$$\text{(5.2d)} \qquad |\Lambda_R^{-1} A_{12}| < 1, \quad |A_{2j}| \leqq K_0, \quad j = 1, 2,$$

where $\Lambda_R := \operatorname{diag}\{\operatorname{Re} a_{ii}\}_{i=1}^{n_+ + n_-}$, $\Lambda_I := \operatorname{diag}\{\operatorname{Im} a_{ii}\}_{i=1}^{n_+ + n_-}$, $\Lambda_{11} := \Lambda_R + i\Lambda_I$ and K_0 is a constant of moderate size (more about this later).[1] (Note that in this definition there is no restriction on the size of the elements of A_{11}; this is the stiff part of the matrix.)

If $A(x)$ is essentially diagonally dominated, then the system of ODEs (2.1) is said to be in *essentially diagonally dominant (EDD) form.* The relevant theorem given in [12] that leads to a mesh construction algorithm is repeated here.

THEOREM. *Consider the system of ODEs* (2.1) *on* $0 < x < c$ *and partition the forcing function* $f(x)$ *in a corresponding way to the matrix A, i.e.,*

$$f(x) = \begin{pmatrix} f^I(x) \\ f^{II}(x) \end{pmatrix}, \quad f^I \in \mathbf{C}^{n_+ + n_-}, \quad f^{II} \in \mathbf{C}^{n_0}.$$

If (2.1) *is in EDD form and there are constants* K_1 *and* K_2 *of moderate size such that*

$$\text{(5.3a)} \qquad |\Lambda_{11}^{-1} d^\nu \Lambda_{11} / dx^\nu| \leqq K_1 \quad (\textit{eigenvalue smoothness}),$$

$$\text{(5.3b)} \qquad |\Lambda_{11}^{-1} d^\nu (B_{11} | A_{12}) / dx^\nu| \leqq K_1 \quad (\textit{off-diagonal element smoothness}),$$

$$\text{(5.3c)} \qquad |d^\nu A_{2j} / dx^\nu| \leqq K_1, \quad j = 1, 2 \quad (\textit{smoothness of the } O(1) \textit{ part}),$$

$$\text{(5.3d)} \qquad \begin{aligned} &|(|f^I| + 1)^{-1} d^\nu f^I / dx^\nu| \leqq K_1 \\ &|d^\nu f^{II} / dx^\nu| \leqq K_1 \end{aligned} \qquad (\textit{smoothness of RHS}),$$

[1] If $y \in C^n$, then $|y|$ denotes its maximum norm. If $A \in C^{n,n}$, then $|A| := \max_{y \neq 0} (|Ay|/|y|)$.

$\nu = 1, \cdots, p$, *and*

$$(5.4)\qquad \begin{aligned} &|a_{ii}(0)| \leqq K_2, \quad i = 1, \cdots, n_- \\ &|a_{ii}(c)| \leqq K_2, \quad i = n_- + 1, \cdots, n_+ + n_- \end{aligned} \qquad (\textit{boundary layer resolution}),$$

then there is a constant $K = K(K_0, K_1, K_2, \rho, \delta)$ *of moderate size such that the derivatives of the solution can be estimated by* (5.1) *with* $x \equiv \tilde{x}$.

In a few places above, we have introduced "constants of moderate size." Let us explain here what this means in terms of stiff boundary value problems. The systems of ODEs (2.1) that are of interest here are ones where $|A_{11}| \gg |A_{22}|$ and hence can have solutions which vary on two (or more) scales. It is typical for these problems that the solution will be quite smooth everywhere except for isolated narrow regions where the variation is quite rapid; the latter regions are called internal or boundary layers. An example of a nonuniform mesh that would resolve such a solution is one which is quite fine in the layers, but relatively coarse in the regions where the solution is smoothly varying. Suppose h is the maximum meshsize used in the regions of smooth variation. Then typically in these regions $h|A_{11}| \gg 1$ (which is why such a problem is called "stiff") while $h|A_{22}| \ll 1$. With these ideas in mind, a *constant of moderate size*, $C > 0$, can be defined as one for which $hC \ll 1$.

6. Construction of the mesh and transformation. In this section we will discuss the practical application of the theorem of § 5. In general, of course, we cannot restrict our consideration to problems which are in EDD form. An important feature of the stiff BVP method is the ability to automatically construct a (piecewise smooth) transformation matrix $T(x)$ that transforms the differential equation to EDD form. To be precise, in our code we transform to essentially *block* diagonally dominant form with at most three diagonal blocks. This transformation is somewhat easier to implement than a transformation to EDD form and seems to be sufficient for most applications. (More refined implementations are currently under investigation.)

We use two different procedures for the construction of $T(x)$. The first one, which we call **transform-from-scratch** in the pseudocode below, transforms a given fixed matrix A to a certain block form. The second procedure, which we call **update-transformation**, is used to obtain a transformation matrix at neighboring meshpoints and can only be used as long as the appropriate block structure does not change as a function of x. The way in which these two routines are used within the mesh construction process is described below.

First consider the problem of transforming a given fixed matrix $A = A(x)$. (At this point and below we assume that the matrix A is real.) Bavely and Stewart [5] have described a method for doing this which is similar to the one we will describe here. Using the QR method, the matrix A is first transformed to quasi-upper triangular form. (Quasi-upper triangular form is a real block upper triangular form with diagonal blocksize at most two. If two-by-two blocks occur, they contain complex conjugate eigenvalues of the matrix.) The QR method is applied in such a way that the eigenvalues occur in the diagonal blocks ordered according to the size of their *real parts*. In [15] Stewart gives a code that arranges the eigenvalues in order of decreasing absolute value along the diagonal. We have modified this code to order the eigenvalues strictly in terms of their real parts. After this first transformation we group the eigenvalues into three groups: Let $\tilde{h}$ denote the local meshsize. $\mathbf{G}_+$ contains those eigenvalues λ of $A = A(x)$ with $\tilde{h}\,\mathrm{Re}\,\lambda > z_C$. $\mathbf{G}_-$ contains those with $\tilde{h}\,\mathrm{Re}\,\lambda < -z_C$, and $\mathbf{G}_0$ contains

those with $\tilde{h}|\operatorname{Re}\lambda| \leqq z_C$. The transformed matrix can be partitioned accordingly:

$$\begin{pmatrix} A_+ & A_{12} & A_{13} \\ 0 & A_0 & A_{23} \\ 0 & 0 & A_- \end{pmatrix},$$

i.e., A_+ has the elements $\lambda \in G_+$ as eigenvalues, etc. As a second transformation, the nonzero off-diagonal blocks A_{12}, A_{13}, A_{23} are transformed to zero by elimination (or equivalently, by solving *linear* matrix Riccati equations [4], [12]) leaving a matrix of the form

$$\tilde{A} = TAT^{-1} = \begin{pmatrix} A_+ & 0 & 0 \\ 0 & A_0 & 0 \\ 0 & 0 & A_- \end{pmatrix}.$$

This procedure is summarized in the following segment of pseudocode:

PROCEDURE **Transform-from-scratch**
(**input**: A; **output** A_b, T_b, *blockstructure*, *eigenvalues*)
{*comments*: This procedure is used to transform a general matrix A to block-diagonal form A_b via a transformation with T_b. The eigenvalues in the blocks are also ordered appropriately}
Upper-triangularize-and-order-eigenvalues (input: A; **output**: A_u, T_u, *eigenvalues*);
{*comments*: $A_u = T_u A T_u^{-1}$ is now in quasi-upper-triangular form}
Choose-diagonal-blocks (**input**: *eigenvalues*, h; **output**: *blockstructure*);
Eliminate-off-diagonal-blocks (**input**: A_u, *blockstructure*, T_u; **output**: A_b, T_b, *success*); If (*success* = *false*) Stop;
{*comments*: When the matrix is quasi-upper-triangular, the diagonal block elimination can only fail if the blockstructure is incorrectly chosen}
END PROCEDURE;

Remarks. For most applications, the block matrix $\tilde{A}$ constructed above seems to be close enough to EDD form though the occurrence of large outer diagonal entries is possible. By an additional transformation using a diagonal matrix D these large outer diagonal entries could be scaled down. In general, $|D|+|D^{-1}|$ would be large, which would increase $|T|+|T^{-1}|$. In practice, we have not used such diagonal scalings.

In principle, the procedure described above could be used repeatedly at different locations x to construct a transformation function $T(x)$. However, this has not only the disadvantage of being somewhat expensive, but also the smoothness of such a transformation as a function of x is not guaranteed. We thus use an updating procedure to compute a transformation $T(x+h)$ from the given transformation $T(x)$ whenever possible. In most cases, the matrix $\hat{A} = T(x)A(x+h)T(x)^{-1}$ is a perturbation of the matrix $\tilde{A} = T(x)A(x)T(x)^{-1}$, and so $T(x+h)$ can be determined from $\hat{A}$ without first transforming to upper triangular form. The details involve the solution of several quadratic matrix Riccati equations which is done using a simple iterative technique discussed in [12] and also mentioned by Stewart [14]. The technique has the feature that as a byproduct, the eigenvalues of the blocks are computed, so the eigenstructure of the matrix can be monitored. The updating procedure is summarized in the following piece of pseudocode:

PROCEDURE **Update-transformation**
(**input**: A, T_b, *blockstructure*; **output**: A_{b_new}, S_b, S_b, *success*)
{*comments*: This procedure first pretransforms the matrix A with the similarity

transformation T_b and then by operating on this pretransformed matrix, computes a transformation S_b which blocks the (original) matrix A.}
$A_{b_old} = T_b A T_b^{-1}$;
Eliminate-off-diagonal-blocks (**input**: A_{b_old}, *blockstructure*; **output**: T_e, *success*);
If (*success* = *true*) Then
$S_b = T_e T_b$;
$A_{b_new} = T_e A_{b_old} T_e^{-1}$;
End;
END PROCEDURE;

Remarks. The variable *success* in this procedure is included to indicate that the iterative technique for eliminating the off-diagonal blocks can fail to converge. From practical experience, it is always possible to remedy this situation, by decreasing h, the distance to the next point at which the transformation is to be constructed. Since these are effects that are relevant to the overall method, the nonconvergence of this iteration is used as another way to monitor these changes. Note that, in general, the three diagonal blocks of the transformed matrix resulting from the updating procedure are full. This is in constrast to the diagonal blocks of the transformed matrix resulting from the procedure **transform-from-scratch** which are quasi-upper-triangular. This difference is another reason for not using the full transformation procedure at every meshpoint. If the blocked matrix is forced to be upper-triangular, then necessarily at least one eigenvector of the matrix is being computed. The smoothness properties of an eigenvector of a matrix are in general worse than the smoothness properties of the corresponding eigenvalue. On the other hand, one can show that a transformation that simply decouples blocks containing well-separated groups of eigenvalues can be chosen to depend smoothly on the matrix elements (cf. Kato [8]). Since information about eigenvectors is not needed for our purposes, it is therefore more reasonable to construct a transformation function $T(x)$ that does not attempt to upper-triangularize the diagonal blocks of the matrix. The only reason for first making a full transformation of the matrix to upper triangular form is to make sure that the eigenvalues appear in the correct order and end up in the correct blocks of the matrix. The method described in "**transform-from-scratch**" is therefore used only at a few isolated points $x = x_j$, while in the intervals between these points, the method described in "**update-transformation**" is used.

Having described the two procedures for blocking a matrix, we now describe some details of the mesh construction. In order to use the theorem of § 5, the coefficient matrix $A(x)$ must be blocked in the manner described above. Then the conditions that are checked are the condition that the transformation matrix be resolved by the mesh, the smoothness of the transformed right-hand side (5.3d), the boundary layer resolution conditions (5.4) and the condition that the imaginary part of the large eigenvalues not be too large (5.2c). If the latter condition is violated, the mesh is refined until these eigenvalues become $O(1)$ with respect to the mesh, and then become part of the block A_0. The constants K_0, K_1 and K_2 are taken as $O(1)$ constants in the numerical code. Typical values are $K_0 = \frac{1}{2}$, $K_1 = \frac{1}{2}$, and $K_2 = 0.4$. In the code we have implemented, the option is also given to check the smoothness of the off-diagonal elements of the transformed matrix (conditions (5.3b) and (5.3c)), but in practice we have never found it necessary to do this.

Suppose the interval on which the problem (2.1) is to be solved is given by $0 < x < 1$. Then the mesh construction procedure starts by first setting up a preliminary mesh which is uniform except near the boundaries where an exponentially stretched mesh

is put in if either of the conditions (5.4) would be violated otherwise. This preliminary mesh is used as a guide to the maximum meshsize that will be permitted at any location on the final mesh that will be constructed. Then beginning at $x = 0$ and proceeding towards $x = 1$, the transformation matrix T is constructed at every fourth meshpoint; the condition that the mesh resolve the piecewise linear interpolation of T at the meshpoints, and the remaining conditions (5.2c) and (5.3), are checked. If none of the conditions is ever violated, then the final computational mesh will be the same as the guide mesh. If any of the conditions are violated, then meshpoints will be spaced closer together until the conditions are locally not violated any more. The mesh construction procedure will, however, always try to use the maximum allowable meshsize which does not result in the violation of any of the conditions (5.2c) and (5.3) and which does not have any local meshsize larger than that indicated by the guide mesh. This somewhat complicated procedure is outlined in the diagram of Fig. 3.

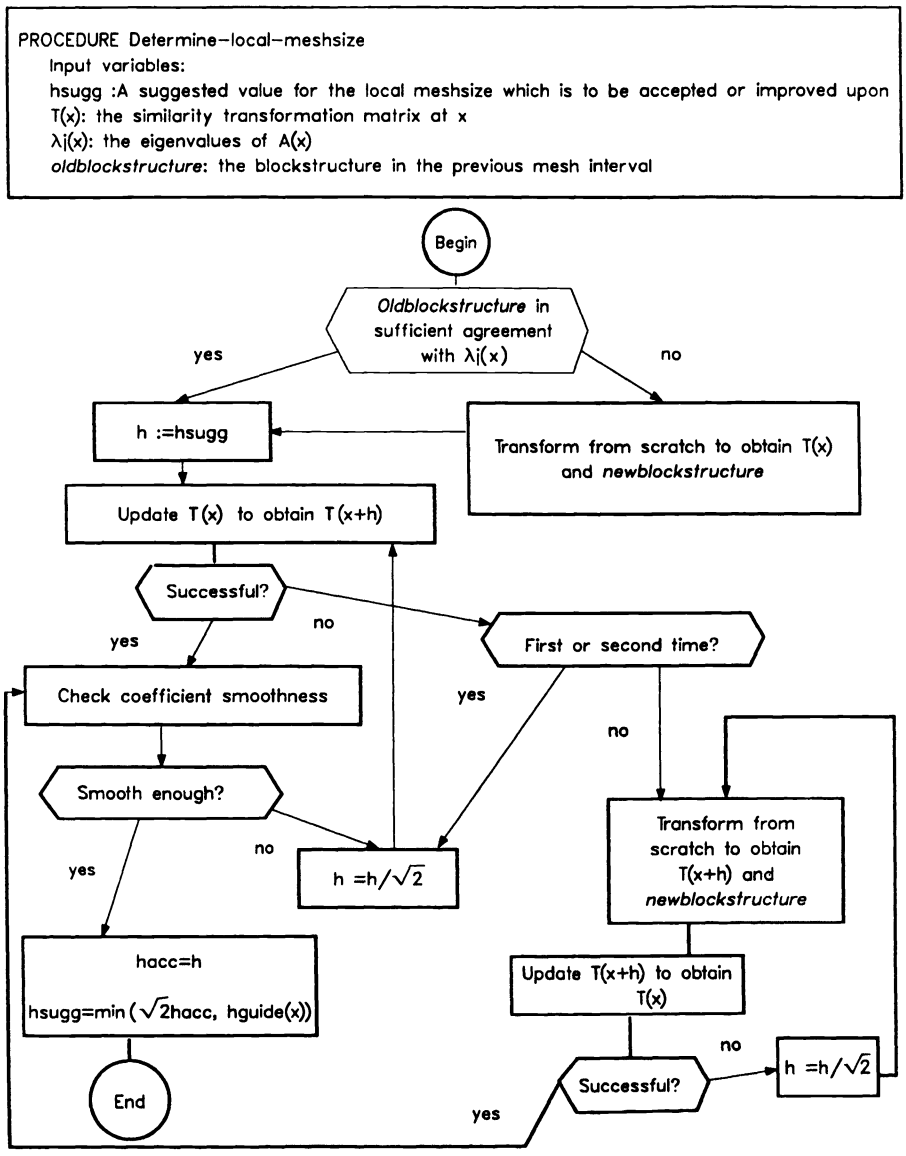


FIG. 3

As a final remark about the mesh construction, let us note that the procedure outlined so far might result in an abrupt decrease of the local mesh size. The theory presented in [12] requires a "smooth" mesh, however. Thus, it is necessary to provide the possibility of disregarding meshpoints already constructed and to approach a region again starting with a smaller mesh size at an appropriate point to the left of the "problem zone." This is implemented in our code.

7. Numerical examples. In this section we present the results of the computation of solutions to some stiff boundary value problems with turning points for which the exact solution is known. The first example is a scalar second order ODE given by [7]

$$-\varepsilon\frac{d^2y}{dx^2}-x\frac{dy}{dx}=\varepsilon\pi^2\cos(\pi x)+\pi x\sin(\pi x)=:f(x),\qquad -1\leqq x\leqq 1, \tag{7.1}$$

with boundary conditions $y(-1)=-2$, $y(1)=0$. The exact solution is given by

$$y(x)=\cos(\pi x)+\frac{\operatorname{erf}(x/(2\varepsilon)^{1/2})}{\operatorname{erf}(2\varepsilon)^{-1/2}} \tag{7.2}$$

where $\operatorname{erf}(x)$ is the error function. The equation is written as a first-order equation by introducing a second variable by the equation $dv/dx=-y+f(x)$, and then solved using our code. The maximum errors in the computed values for y for different values of the parameter ε and using different numbers of collocation points are tabulated in Table 2.

TABLE 2

Maximum error in the first component for the model second-order system (7.1). *The values given in brackets [ ] are the number of meshpoints used in each computation.* $ncol=m+1$ *is the number of collocation points per mesh interval.*

Lobatto points: $\varepsilon=$	1.E−2	1.E−4	1.E−6
ncol			
2	1.2E−2 [53]	9.8E−3 [100]	9.8E−3 [164]
3	1.6E−4 [43]	1.4E−4 [92]	8.2E−5 [156]
4	9.9E−6 [43]	2.3E−6 [88]	1.4E−6 [148]
5	1.9E−7 [43]	9.2E−8 [88]	6.0E−8 [148]
6	2.7E−9 [40]	9.1E−9 [88]	2.6E−9 [148]
7	1.5E−10 [40]	4.2E−10 [88]	5.4E−11 [148]
8	6.6E−12 [40]	5.2E−12 [88]	1.2E−12 [140]
Radau points twice:			
ncol			
4	2.0E−5 [43]	6.1E−6 [88]	4.2E−6 [148]
6	7.2E−9 [40]	3.5E−8 [88]	7.2E−9 [148]
8	2.0E−11 [40]	2.4E−11 [88]	4.2E−12 [148]

The second example presented here is a system of two singularly perturbed second-order ODEs given by

$$\begin{aligned}&-\varepsilon\frac{d^2y}{dx^2}-\frac{x}{2}\frac{dy}{dx}+\frac{x}{2}\frac{du}{dx}+u=\varepsilon\pi^2\cos(\pi x)+\frac{1}{2}\pi x\sin(\pi x)=:g(x),\\ &-\varepsilon\frac{d^2u}{dx^2}+u=0,\qquad -1\leqq x\leqq 1,\end{aligned} \tag{7.3}$$

TABLE 3

Maximum error in the first component for the model fourth-order system (7.3). The values given in brackets [ ] are the number of meshpoints used in each computation. ncol = m + 1 is the number of collocation points per mesh interval.

Lobatto points: $\varepsilon =$	1.E−4	1.E−6	1.E−8
ncol			
2	6.5E−2 [143]	6.3E−2 [250]	6.3E−2 [332]
3	1.2E−4 [122]	7.0E−5 [231]	4.1E−5 [315]
4	3.4E−7 [122]	3.5E−7 [223]	9.0E−6 [315]
5	1.1E−8 [122]	1.2E−8 [223]	1.4E−8 [308]
6	9.3E−10 [122]	1.0E−9 [223]	4.1E−10 [308]
7	5.4E−11 [122]	5.7E−11 [223]	2·5E−11 [308]
8	2.4E−12 [122]	3.0E−12 [122]	1.5E−12 [308]
Radau points twice:			
ncol			
4	3.5E−6 [122]	3.6E−6 [223]	2.0E−6 [315]
6	5.0E−9 [122]	5.1E−9 [223]	4.8E−9 [308]
8	1.2E−11 [122]	1.2E−11 [223]	1.2E−11 [308]

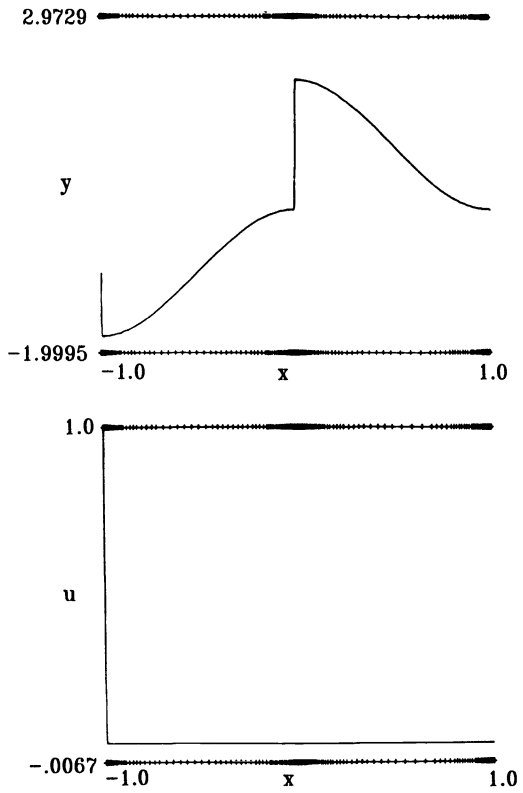


FIG. 4

with boundary conditions $y(-1)=-1$, $u(-1)=1$, $y(1)=u(1)=e^{-2/\sqrt{\varepsilon}}$. The exact solution for this problem is given by

$$y(x)=\frac{\operatorname{erf}(x/2\sqrt{\varepsilon})}{\operatorname{erf}(1/2\sqrt{\varepsilon})}+u(x)+\cos(\pi x), \qquad u(x)=e^{-(x+1)/\sqrt{\varepsilon}}. \tag{7.4}$$

Equation (7.3) is written as a first-order system by introducing additional variables w and v by $\varepsilon du/dx=v$ and $dw/dx=\frac{1}{2}y+\frac{1}{2}xv+u-g(x)$. The maximum computed errors in the variable y are shown in Table 3 for three values of ε and for various numbers of collocation points. A plot of the solution is included in Fig. 4. It is clear from the tables of errors that the collocation formulas we have discussed here can compute very accurate solutions to singular perturbation problems with turning points.

Acknowledgment. We would like to thank Heinz Kreiss for many valuable discussions.

REFERENCES

[1] U. ASCHER, *On some difference schemes for singular singularly-perturbed boundary value problems*, Numer. Math., 46 (1985), pp. 1-30.

[2] 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.

[3] ———, *Collocation for singular perturbation problems* II: *Linear first order systems without turning points*, Math. Comp., 43 (1984), pp. 157-187.

[4] R. H. BARTELS AND G. W. STEWART, *Algorithm* 432: *Solution of the matrix equation* $AX+XB=C$, Comm. ACM, 15 (1972), pp. 820-826.

[5] C. A. BAVELY AND G. W. STEWART, *An algorithm for computing reducing subspaces by block diagonalization*, SIAM J. Numer. Anal., 16 (1979), pp. 359-367.

[6] D. L. BROWN, *A numerical method for singular perturbation problems with turning points*, in Numerical Boundary Value ODES, U. Ascher and R. Weiss, eds., Birkhäuser-Verlag, Basel, Switzerland, 1985.

[7] P. W. HEMKER, *A numerical study of stiff two-point boundary problems*, Ph.D. thesis, Mathematisch Centrum, Amsterdam, 1977.

[8] T. KATO, *A short introduction to perturbation theory for linear operators*, Springer-Verlag, Berlin, 1982.

[9] B. KREISS AND H.-O. KREISS, *Numerical methods for singular perturbation problems*, SIAM J. Numer. Anal., 10 (1981), pp. 262-276.

[10] H.-O. KREISS, *Centered difference approximation for singular systems of ordinary differential equations*, Istituto Nazional di Alta Matematica, Symposia Matematica, 10 (1972), pp. 454-465.

[11] ———, *Central difference schemes and stiff boundary value problems*, BIT, 24 (1984), pp. 560-567.

[12] H.-O. KREISS, N. NICHOLS AND D. L. BROWN, *Numerical methods for stiff two-point boundary value problems*, SIAM J. Numer. Anal., 23 (1986), pp. 325-368. (An earlier version appeared as Univ. of Wisconsin Report MRC TSR 2599 (1983).)

[13] R. RINGHOFER, *On collocation schemes for quasilinear singularly perturbed boundary value problems*, SIAM J. Numer. Anal., 21 (1984), pp. 864-882.

[14] G. W. STEWART, *Error and perturbation bounds for subspaces associated with certain eigenvalue problems*, SIAM Rev., 15 (1973), pp. 727-764.

[15] ———, HQR3 and EXCHNG: FORTRAN *subroutines for calculating and ordering the eigenvalues of a real upper Hessenberg matrix*, ACM Trans. Math. Software, 2 (1976), pp. 275-280.

[16] R. WEISS, *The application of implicit Runge-Kutta and collocation methods to boundary-value problems*, Math. Comp., 28 (1974), pp. 449-464.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 5, September 1987
© 1987 Society for Industrial and Applied Mathematics
010

THE NUMERICAL SOLUTION OF NONLINEAR VOLTERRA INTEGRAL EQUATIONS OF THE SECOND KIND BY COLLOCATION AND ITERATED COLLOCATION METHODS*

J. G. BLOM† AND H. BRUNNER‡

Abstract. The subject of this paper is a variable stepsize one-step method of collocation type for solving general nonlinear second kind Volterra integral equations. We extend the iterated collocation method corresponding to polynomial spline collocation to nonlinear Volterra integral equations of the second kind. The resulting superconvergence properties of either the collocation approximation or the iterated collocation approximation are used to obtain (local and global) error estimates which in turn form the basis of a variable stepsize code. The performance of this code is illustrated by means of numerous test problems.

Key words. nonlinear Volterra integral equation, polynomial splines, collocation, iterated collocation, superconvergence, error estimates, variable stepsize

AMS(MOS) subject classifications. Primary 65R20; secondary 45D05, 45L10

0. Introduction. Most of the work on the numerical solution of Volterra integral equations of the second kind done to date deals with fixed-step methods. Recently, some automatic codes have been published. Some of these employ fixed stepsizes and make use of extrapolation tableaux to control the global error (VOLCON [13], IVRKX(C) [19]). Others use variable stepsizes based on local error controlling techniques (VOLTEX [14], RKVIEP [12], INTSOL [16], ORION [3], [17]).

The variable stepsize method we propose offers the user the choice to control the global or the local error, with the possibility to control also the uniform error, as suggested by Arndt [2].

It is believed by the authors that iterated collocation is a very robust and not too expensive method to control the global error. Since there exists a class of problems to which this method cannot be applied, some other combinations of collocation methods are investigated. The performance of these methods was considerably improved by using uniform error control.

The variable stepsize collocation methods described below have been implemented in a FORTRAN code COLVI2 (see [4]).

1. Polynomial spline collocation.

1.1. Exact collocation equations. Consider the nonlinear second kind Volterra integral equation,

$$y(t) = g(t) + \int_0^t k(t, s, y(s))\, ds, \qquad t \in I := [0, T], \tag{1.1.1}$$

where g and k are given continuous functions on I and on $S \times \mathbb{R}$ (with $S := \{(t, s): 0 \leqq s \leqq t \leqq T\}$), respectively, and where k is such that (1.1.1) possesses a unique solution $y \in C(I)$ (compare, e.g., Miller [18], or Brunner and van der Houwen [10]).

In order to discretize (1.1.1) let

$$\Pi_N: 0 = t_0 < t_1 < \cdots < t_N = T \quad (N \geqq 1)$$

* Received by the editors October 30, 1985; accepted for publication (in revised form) May 29, 1986.
† Centre for Mathematics and Computer Science, Kruislaan 413, 1098 SJ Amsterdam, the Netherlands.
‡ Department of Mathematics and Statistics, Memorial University of Newfoundland, St. John's, Newfoundland, Canada ALC 557.

denote a partition (or: mesh) for the given interval I, and set

$$h_n := t_{n+1} - t_n \qquad (n = 0, \cdots, N-1),$$
$$\sigma_0 := [t_0, t_1], \qquad \sigma_n := (t_n, t_{n+1}] \qquad (n = 1, \cdots, N-1),$$
$$Z_N := \{t_n : n = 1, \cdots, N-1\} \qquad \text{(the set of interior mesh points)},$$

and

$$\bar{Z}_N := Z_N \cup \{T\}.$$

In the following analysis it will always be assumed that the mesh sequence $\{\Pi_N\}$ $(N \in \mathbb{N})$ is quasi-uniform; i.e.,

$$\frac{\max_{(n)}(h_n)}{\min_{(n)}(h_n)} \leqq \gamma < \infty$$

uniformly for $N \in \mathbb{N}$ (for ease of notation we have suppressed the superscript N in $t_n^{(N)}$, $h_n^{(N)}$, indicating the dependence of these quantities on N). In the following $h := \max_{(n)}(h_n)$ denotes the diameter of the mesh Π_N.

The exact solution y of (1.1.1) will be approximated in the polynomial spline space

$$S_{m-1}^{(-1)}(Z_N) := \{u : u|_{\sigma_n} =: u_n \in \pi_{m-1},\ n = 0, \cdots, N-1\ (m \geqq 1)\}, \tag{1.1.2}$$

whose elements reduce, on each subinterval σ_n, to (real) polynomials of degree not exceeding $m-1$ and which, in general, possess (finite) discontinuities at their knots Z_N. Here, the value $m = 1$ yields, of course, the space of step functions having knots Z_N.

Let $\{c_j\}$, with $0 \leqq c_1 < \cdots < c_m \leqq 1$, be a given set of parameters, and define the sets

$$X_n := \{t_{n,j} := t_n + c_j h_n : j = 1, \cdots, m\} \qquad (n = 0, \cdots, N-1), \tag{1.1.3a}$$

and

$$X(N) := \bigcup_{n=0}^{N-1} X_n. \tag{1.1.3b}$$

The desired approximation $u \in S_{m-1}^{(-1)}(Z_N)$ will be determined by requiring that u satisfy the integral equation (1.1.1) on the finite set $X(N)$ ("collocation" on $X(N)$):

$$u(t) = g(t) + \int_0^t k(t, s, u(s))\, ds \quad \text{for all } t \in X(N).$$

This *collocation equation* may also be written as

$$u_n(t) = g(t) + \int_{t_n}^{t} k(t, s, u_n(s))\, ds + \sum_{i=0}^{n-1} \int_{t_i}^{t_{i+1}} k(t, s, u_i(s))\, ds, \tag{1.1.4}$$
$$t \in X_n \quad (n = 0, \cdots, N-1),$$

thus explicitly exhibiting its recursive character. Note that if the collocation parameters $\{c_j\}$ are such that $c_1 = 0$ and $c_m = 1$, then the approximation defined by (1.1.4) will be continuous on I; i.e., we then have

$$u \in S_{m-1}^{(0)}(Z_N) := S_{m-1}^{(-1)}(Z_N) \cap C(I).$$

For the subsequent analysis we introduce the integrals

$$\Phi_{n,i}^{(j)}[u_i] := \begin{cases} \displaystyle\int_0^1 k(t_{n,j}, t_i + \tau h_i, u_i(t_i + \tau h_i))\, d\tau & \text{if } 0 \leqq i \leqq n-1, \\ \displaystyle\int_0^{c_j} k(t_{n,j}, t_n + \tau h_n, u_n(t_n + \tau h_n))\, d\tau & \text{if } i = n \quad (j = 1, \cdots, m). \end{cases} \tag{1.1.5}$$

With this notation the collocation equation (1.1.4) becomes

$$u_n(t_{n,j}) = g(t_{n,j}) + h_n \Phi_{n,n}^{(j)}[u_n] + \sum_{i=0}^{n-1} h_i \Phi_{n,i}^{(j)}[u_i], \qquad j = 1, \cdots, m \quad (n = 0, \cdots, N-1). \tag{1.1.6}$$

Setting

$$u_n(t_n + \tau h_n) = \sum_{l=1}^{m} L_l(\tau)\, Y_{n,l}, \qquad t_n + \tau h_n \in \sigma_n, \tag{1.1.7}$$

with $Y_{n,l} := u_n(t_{n,l})$, and where the polynomials

$$L_l(\tau) := \prod_{\substack{\kappa=1 \\ \kappa \neq l}}^{m} \frac{(\tau - c_\kappa)}{(c_l - c_\kappa)}$$

represent the Lagrange canonical polynomials for the collocation parameters $\{c_j\}$, we see that (1.1.6) represents, for each $n = 0, \cdots, N-1$, a nonlinear system in $\mathbb{R}^m$ for the vector $Y_n := (Y_{n,1}, \cdots, Y_{n,m})^T$. Once Y_n has been determined, the approximation u on the subinterval σ_n is given by (1.1.7).

1.2. Discretization of the collocation equations. In most applications the integrals (1.1.5) occurring in the collocation equation (1.1.6) cannot be evaluated analytically, and one is forced to resort to employing suitable quadrature formulas for their approximation. In the following we shall use *m-point interpolatory quadrature formulas* of the form

$$\hat{\Phi}_{n,i}^{(j)}[u_i] := \sum_{l=1}^{m} w_l k(t_{n,j}, t_{i,l}, u_i(t_{i,l})) \quad \text{if } 0 \leqq i \leqq n-1; \tag{1.2.1}$$

for the approximation of $\Phi_{n,n}^{(j)}[u_n]$ we shall employ either

$$\tilde{\Phi}_{n,n}^{(j)}[u_n] := \sum_{l=1}^{m} w_l k(t_{n,j}, t_{n,l}, u_n(t_{n,l})), \tag{1.2.2a}$$

or

$$\hat{\Phi}_{n,n}^{(j)}[u_n] := \sum_{l=1}^{m} w_{j,l} k(t_{n,j}, t_n + c_j c_l h_n, u_n(t_n + c_j c_l h_n)). \tag{1.2.2b}$$

Here, the quadrature weights are given by

$$w_l := \int_0^1 L_l(\tau)\, d\tau,$$

and by

$$w_{j,l} := c_j w_l \qquad (j, l = 1, \cdots, m).$$

Note that when using the quadrature approximation (1.2.2a) we need kernel values $k(t, s, \cdot)$ at points (t, s) no longer contained in S (since $t_{n,l} > t_{n,j}$ for $l > j$), while (1.2.2b) is based on the abscissas $t_n + c_j c_l h_n \leqq t_{n,j}$ (i.e., the affine images of $t_{n,l}$ in $[t_n, t_{n,j}]$).

The resulting fully discretized collocation equations are thus given by, respectively,

$$\tilde{u}_n(t_{n,j}) = g(t_{n,j}) + h_n \tilde{\Phi}_{n,n}^{(j)}[\tilde{u}_n] + \sum_{i=0}^{n-1} h_i \hat{\Phi}_{n,i}^{(j)}[\tilde{u}_i], \qquad j = 1, \cdots, m \qquad (n = 0, \cdots, N-1); \tag{1.2.3a}$$

and

$$\hat{u}_n(t_{n,j}) = g(t_{n,j}) + h_n \hat{\Phi}^{(j)}_{n,n}[\hat{u}_n] + \sum_{i=0}^{n-1} h_i \hat{\Phi}^{(j)}_{n,i}[\hat{u}_i], \qquad j = 1, \cdots, m \qquad (n = 0, \cdots, N-1); \tag{1.2.3b}$$

they yield approximations $\tilde{u}$ and $\hat{u}$ in $S^{(-1)}_{m-1}(Z_N)$ (whose restrictions to the subinterval σ_n are given by interpolation formulas analogous to (1.1.7)). These approximations will, in general, be different from the "exact" collocation approximation u defined by (1.1.6); due to the differing quadrature errors associated with (1.2.2a) and (1.2.2b) we have also $\tilde{u} \neq \hat{u}$. However, the approximations u, $\tilde{u}$, and $\hat{u}$ all exhibit the same order of (global and local) convergence; in particular, we have the following results (see, e.g., Brunner and van der Houwen [10]; compare also de Hoog and Weiss [15], Brunner [7] and Brunner and Nørsett [11]).

THEOREM 1.1. *Let g and k be smooth functions, such that* (1.1.1) *has a unique solution y which is sufficiently many times continuously differentiable on I, and let u, $\tilde{u}$, $\hat{u} \in S^{(-1)}_{m-1}(Z_N)$ denote, respectively, the collocation approximations determined by* (1.1.6), (1.2.3a), *and* (1.2.3b).

Moreover, let p, $\tilde{p}$ and $\hat{p}$ be the largest integers satisfying

$$\max_{t_n \in \bar{Z}_N} |y(t_n) - u(t_n)| = O(N^{-p}),$$

$$\max_{t_n \in \bar{Z}_N} |y(t_n) - \tilde{u}(t_n)| = O(N^{-\tilde{p}})$$

and

$$\max_{t_n \in \bar{Z}_N} |y(t_n) - \hat{u}(t_n)| = O(N^{-\hat{p}}) \qquad (\text{as } N \to \infty, \text{ with } Nh \leqq \gamma T \text{ fixed}).$$

(a) *If the collocation parameters $\{c_j\}$ are given by the zeros of the Legendre polynomials $P_m(2s-1)$ (i.e., the Gauss points for $(0, 1)$), then*

$$p = \tilde{p} = \hat{p} = m. \tag{1.2.4a}$$

(b) *If the $\{c_j\}$ are the zeros of $P_{m-1}(2s-1) - P_m(2s-1)$ (i.e., the Radau* II *points for $(0, 1]$), then*

$$p = \tilde{p} = \hat{p} = 2m - 1. \tag{1.2.4b}$$

(c) *If the $\{c_j\}$ are the zeros of $s(s-1)P'_{m-1}(2s-1)$ (i.e., the Lobatto points for $[0, 1]$), then*

$$p = \tilde{p} = \hat{p} = 2m - 2. \tag{1.2.4c}$$

Note that $\tilde{p}$ attains the indicated values only if the kernel $k(t, s, \cdot)$ is smooth not only for $(t, s) \in S$ but also for $(t, s) \in S' := \{(t, s): 0 \leqq s \leqq t + \delta\} \cap I \times I$, for some $\delta > 0$. This is, of course, a consequence of the special form of the quadrature approximation (1.2.2a).

We also recall that the order of global convergence on I of these three spline approximations equals m. Hence, while collocation at the Radau II points or at the Lobatto points leads to a higher order of convergence on $\bar{Z}_N$ ("local superconvergence"), collocation using the Gauss points does not have this local superconvergence property (in contrast to polynomial spline collocation applied to initial-value problems for ordinary differential equations).

In order to prepare the ground for the convergence analysis for iterated collocation (§ 2) we consider a somewhat different discretization of the collocation equation (1.1.6). To be precise, let $z \in S_m^{(-1)}(Z_N)$ be determined by the (exact) collocation equation

$$z_n(t_{n,j}) = g(t_{n,j}) + h_n \Phi_{n,n}^{(j)}[z_n] + \sum_{i=0}^{n-1} h_i \Phi_{n,i}^{(j)}[z_i], \tag{1.2.5}$$

where now $j = 1, \cdots, m+1$, $t_{n,j} := t_n + c_j h_n$, with $0 \leqq c_1 < \cdots < c_m < c_{m+1} = 1$. Here, the integrals $\Phi_{n,i}^{(j)}$ are as in (1.1.5) (except that now j varies between 1 and $m+1$). These integrals are approximated by *m-point* (instead of $(m+1)$-point!) *interpolatory quadrature formulas* using abscissas based on the first m of the $m+1$ collocation parameters; i.e., we employ again the quadrature approximations (1.2.1), (1.2.2a), (1.2.2b), with z replacing u, and with j ranging from 1 to $m+1$. Accordingly, the two possible discretizations of (1.2.5) are given by

$$\tilde{z}_n(t_{n,j}) = g(t_{n,j}) + h_n \tilde{\Phi}_{n,n}^{(j)}[\tilde{z}_n] + \sum_{i=0}^{n-1} h_i \hat{\Phi}_{n,i}^{(j)}[\tilde{z}_i], \tag{1.2.6a}$$

$$j = 1, \cdots, m+1 \qquad (n = 0, \cdots, N-1),$$

and

$$\hat{z}_n(t_{n,j}) = g(t_{n,j}) + h_n \hat{\Phi}_{n,n}^{(j)}[\hat{z}_n] + \sum_{i=0}^{n-1} h_i \hat{\Phi}_{n,i}^{(j)}[\hat{z}_i], \tag{1.2.6b}$$

$$j = 1, \cdots, m+1 \qquad (n = 0, \cdots, N-1);$$

these equations yield, respectively, approximations $\tilde{z}$ and $\hat{z}$ in the space $S_m^{(-1)}(Z_N)$. In analogy to (1.1.7) we set, e.g.,

$$\tilde{z}_n(t_n + \tau h_n) = \sum_{l=1}^{m+1} L_l^*(\tau) \tilde{Z}_{n,l}, \qquad t_n + \tau h_n \in \sigma_n, \tag{1.2.7}$$

with $\tilde{Z}_{n,l} := \tilde{z}_n(t_{n,l})$, and with $L_l^*(\tau)$ denoting the lth Lagrange canonical polynomial (of degree m) with respect to $c_1, \cdots, c_{m+1}$.

Consider first (1.2.6a): it follows from the quadrature approximation (1.2.2a) and from (1.2.7) that

$$\tilde{\Phi}_{n,n}^{(j)}[\tilde{z}_n] = \sum_{l=1}^{m} w_l k(t_{n,j}, t_{n,l}, \tilde{Z}_{n,l}), \qquad j = 1, \cdots, m+1$$

(note that $L_{m+1}^*(c_l) = 0$ for $l = 1, \cdots, m$). Hence, the first m equations of (1.2.6a) (corresponding to $j = 1, \cdots, m$) constitute a nonlinear system in $\mathbb{R}^m$ for $\tilde{Z}_{n,1}, \cdots, \tilde{Z}_{n,m}$; the last equation ($j = m+1$) then determines, in an explicit way, the approximation $\tilde{Z}_{n,m+1}$ which, by $c_{m+1} = 1$, equals $\tilde{z}_n(t_{n+1})$ and thus approximates $y(t_{n+1})$.

Turning to the discretization (1.2.6b), we observe that now, by (1.2.2b),

$$\hat{\Phi}_{n,n}^{(j)}[\hat{z}_n] = \sum_{l=1}^{m} w_{j,l} k\left(t_{n,j}, t_n + c_j c_l h_n, \sum_{s=1}^{m+1} L_s^*(c_j c_l) \hat{Z}_{n,s}\right), \qquad j = 1, \cdots, m+1,$$

depends not only on $\hat{Z}_{n,1}, \cdots, \hat{Z}_{n,m}$, but also on $\hat{Z}_{n,m+1}$; hence, (1.2.6b) represents a nonlinear system in $\mathbb{R}^{m+1}$, with $\hat{Z}_{n,m+1}$ approximating $y(t_{n+1})$.

The following theorem shows that the approximations generated by the discretizations (1.2.6a) and (1.2.6b) exhibit the same order of convergence.

THEOREM 1.2. *Let g and k be smooth functions, such that the integral equation* (1.1.1) *possesses a unique solution y which is sufficiently many times continuously differentiable on I, and let z, $\tilde{z}$, $\hat{z} \in S_m^{(-1)}(Z_N)$ denote, respectively, the collocation approximations determined by* (1.2.5), (1.2.6a), *and* (1.2.6b). *If p, $\tilde{p}$ and $\hat{p}$ are the largest integers satisfying*

$$\max_{t_n \in \bar{Z}_N} |y(t_n) - z(t_n)| = O(N^{-p}),$$

$$\max_{t_n \in \bar{Z}_N} |y(t_n) - \tilde{z}(t_n)| = O(N^{-\tilde{p}})$$

and

$$\max_{t_n \in \bar{Z}_N} |y(t_n) - \hat{z}(t_n)| = O(N^{-\hat{p}}),$$

and if the first m collocation parameters $c_1, \cdots, c_m$ are the zeros of $P_m(2s-1)$ (Gauss points for $(0, 1)$), while $c_{m+1} = 1$, then

$$p = \tilde{p} = \hat{p} = 2m. \tag{1.2.8}$$

The assertion $\tilde{p} = 2m$ will hold only if the kernel $k(t, s, \cdot)$ can be extended smoothly to the domain $S' := \{(t, s): 0 \leqq s \leqq t + \delta\} \cap I \times I$, where $\delta > 0$ is suitably chosen.

A proof of this result may be found in Brunner [7], or in Brunner and van der Houwen [10].

We mention in passing that the discretized collocation equation (1.2.6a) represents an *m-stage implicit Volterra-Runge-Kutta method of Pouzet type* (cf. Brunner, Hairer and Nørsett [9]). It will be shown in the next section (Theorem 2.1) that this method is equivalent to a certain discretized version of an *iterated collocation method*, in the sense that they both produce identical approximations to y on $\bar{Z}_N$.

2. Iterated collocation approximations.

2.1. Exact iterated collocation. Suppose that the collocation approximation $u \in S_{m-1}^{(-1)}(Z_N)$ (defined by (1.1.6)) has been computed. The (exact) *iterated collocation approximation* u^I corresponding to u is then defined by

$$u^I(t) := g(t) + \int_0^t k(t, s, u(s))\, ds, \qquad t \in I. \tag{2.1.1}$$

(Compare also Brunner [8] and the references cited there for some background and history of iterated approximations for (linear) Fredholm and Volterra integral equations.) It is readily verified that u^I has the following properties:

(i) $u^I \in C(I)$ (while u is, in general, not continuous on I);

(ii) $u^I(t) = u(t)$ for all $t \in X(N)$.

This second property implies, in particular, that if $c_m = 1$ (i.e., if $\bar{Z}_N$ is a subset of the set $X(N)$ of collocation points), then $u^I(t_n) = u(t_n)$ for all $t_n \in \bar{Z}_N$.

In the following we shall be interested in the values $u^I(t_n)$, $t_n \in \bar{Z}_N$, in the case when $c_m < 1$. Setting

$$\Phi_{n,i}[u_i] := \int_0^1 k(t_n, t_i + \tau h_i, u_i(t_i + \tau h_i))\, d\tau, \qquad 0 \leqq i \leqq n-1 \leqq N-1, \tag{2.1.2}$$

we may write (2.1.1) in the form

$$u^I(t_n) = g(t_n) + \sum_{i=0}^{n-1} h_i \Phi_{n,i}[u_i], \qquad t_n \in \bar{Z}_N. \tag{2.1.3}$$

2.2. Discretized iterated collocation. The integrals (2.1.2) will, in general, have to be approximated by appropriate numerical quadrature; as in § 1.2 we choose m-point interpolatory quadrature formulas of the form

$$\hat{\Phi}_{n,i}[u_i] := \sum_{l=1}^{m} w_l k(t_n, t_{i,l}, u_i(t_{i,l})), \tag{2.2.1}$$

with quadrature weights $\{w_l\}$ as in (1.2.1). Moreover, as indicated in § 1.2, we shall usually not be able to compute u itself but only either $\tilde{u} \in S_{m-1}^{(-1)}(Z_N)$, or $\hat{u} \in S_{m-1}^{(-1)}(Z_N)$ (i.e., the solutions of the discretized collocation equations (1.2.3a), (1.2.3b)). Accordingly, the *discretization* of (2.1.3) will be given either by

$$\tilde{u}^I(t_n) := g(t_n) + \sum_{i=0}^{n-1} h_i \hat{\Phi}_{n,i}[\tilde{u}_i], \qquad t_n \in \bar{Z}_N, \tag{2.2.2a}$$

with $\tilde{u} \in S_{m-1}^{(-1)}(Z_N)$ defined by (1.2.3a), or by

$$\hat{u}^I(t_n) := g(t_n) + \sum_{i=0}^{n-1} h_i \hat{\Phi}_{n,i}[\hat{u}_i], \qquad t_n \in \bar{Z}_N, \tag{2.2.2b}$$

with $\hat{u} \in S_{m-1}^{(-1)}(Z_N)$ given by (1.2.3b).

In the convergence analysis for $y(t_n) - \tilde{u}^I(t_n)$ and $y(t_n) - \hat{u}^I(t_n)$ the following result will play an important role.

LEMMA 2.1. *Let the iterated approximations $\tilde{u}^I(t_n)$ and $\hat{u}^I(t_n)$ be given by* (2.2.2a) *and* (2.2.2b), *and assume that g and k in* (1.1.1) *satisfy the customary smoothness hypotheses. If the collocation parameters $\{c_j\}$ are the zeros of $P_m(2s-1)$ (Gauss points for* $(0, 1)$), *then*

$$\max_{t_n \in \bar{Z}_N} |\tilde{u}^I(t_n) - \hat{u}^I(t_n)| = O(N^{-2m}) \quad (\text{as } N \to \infty,\ Nh \leqq \gamma T). \tag{2.2.3}$$

Proof. Assuming that $k(t, s, y)$ has a bounded partial derivative $\partial k/\partial y$, with L denoting an upper bound for its absolute value, we obtain, using (2.2.2a) and (2.2.2b) together with (2.2.1),

$$\begin{aligned} |\tilde{u}^I(t_n) - \hat{u}^I(t_n)| &\leqq \sum_{i=0}^{n-1} h_i |\hat{\Phi}_{n,i}[\tilde{u}_i] - \hat{\Phi}_{n,i}[\hat{u}_i]| \\ &\leqq h \sum_{i=0}^{n-1} \sum_{l=1}^{m} |k(t_n, t_{i,l}, \tilde{u}_i(t_{i,l})) - k(t_n, t_{i,l}, \hat{u}_i(t_{i,l}))| \\ &\leqq hL \sum_{i=0}^{n-1} \sum_{l=1}^{m} |\tilde{u}_i(t_{i,l}) - \hat{u}_i(t_{i,l})|, \qquad t_n \in \bar{Z}_N. \end{aligned}$$

Here, we have used the fact that weights $\{w_l\}$ of the Gaussian quadrature formula are all positive and hence bounded by one (since $\sum_{l=1}^{m} w_l = 1$). An argument analogous to the one used in Brunner [8, pp. 1138–1139], involving the Gauss quadrature errors and a discrete Gronwall inequality, yields, observing (1.2.3a) and (1.2.3b),

$$|\tilde{u}_i(t_{i,l}) - \hat{u}_i(t_{i,l})| \leqq Ch^{2m}, \quad l = 1, \cdots, m \quad (i = 0, \cdots, N-1),$$

with some constant C not depending on N. The estimate (2.2.3) now follows since $Nh \leqq \gamma T$ for all $N \in \mathbb{N}$. □

THEOREM 2.1. *Let $0 \leqq c_1 < \cdots < c_m < 1$. Assume that $\tilde{u} \in S_{m-1}^{(-1)}(Z_N)$ is the solution of the discretized collocation equation* (1.2.3a), *and let $\tilde{z} \in S_m^{(-1)}(Z_N)$ be determined by the discretized collocation equation* (1.2.6a) (*m-stage implicit Volterra–Runge–Kutta*

method of Pouzet type), *where the collocation parameters are* $0 \leqq c_1 < \cdots < c_m < c_{m+1} = 1$. *It then follows that*

$$\tilde{u}^I(t_n) = \tilde{z}(t_n), \qquad t_n \in \bar{Z}_N; \tag{2.2.4}$$

i.e., the values of the discretized iterated collocation approximation $\tilde{u}^I$ *determined by* (2.2.2a) *are identical to the approximations* $\tilde{z}(t_n)$ *furnished by the m-stage implicit VRK method of Pouzet type* (1.2.6a).

Proof. Using

$$\tilde{u}_i(t_i + \tau h_i) = \sum_{l=1}^{m} L_l(\tau)\, \tilde{Y}_{i,l}, \qquad t_i + \tau h_i \in \sigma_i,$$

with $\tilde{Y}_{i,l} := \tilde{u}_i(t_{i,l})$, we may write (1.2.3a) as

$$\tilde{Y}_{n,j} = g(t_{n,j}) + \sum_{i=0}^{n} h_i \sum_{l=1}^{m} w_l k(t_{n,j}, t_{i,l}, \tilde{Y}_{i,l}),$$
$$j = 1, \cdots, m \qquad (n = 0, \cdots, N-1), \tag{2.2.5a}$$

and (2.2.2a) becomes

$$\tilde{u}^I(t_n) = g(t_n) + \sum_{i=0}^{n-1} h_i \sum_{l=1}^{m} w_l k(t_n, t_{i,l}, \tilde{Y}_{i,l}), \qquad t_n \in \bar{Z}_N. \tag{2.2.5b}$$

On the other hand, it follows from (1.2.7) that the first m equations of (1.2.6a) ($j = 1, \cdots, m$) assume the form

$$\tilde{Z}_{n,j} = g(t_{n,j}) + \sum_{i=0}^{n} h_i \sum_{l=1}^{m} w_l k(t_{n,j}, t_{i,l}, \tilde{Z}_{i,l}),$$
$$j = 1, \cdots, m \qquad (n = 0, \cdots, N-1). \tag{2.2.6a}$$

This nonlinear system (2.2.6a) is identical with the nonlinear system (2.2.5a); hence, they possess, for all sufficiently small values of the mesh diameter h, the same solution: $\tilde{Y}_{n,j} = \tilde{Z}_{n,j}$, $j = 1, \cdots, m$ ($n = 0, \cdots, N-1$).

For $j = m+1$ in (1.2.6a) we find, using $c_{m+1} = 1$ (and hence $t_{n,m+1} = t_{n+1}$),

$$\tilde{z}(t_{n+1}) = \tilde{Z}_{n,m+1} = g(t_{n+1}) + \sum_{i=0}^{n} h_i \sum_{l=1}^{m} w_l k(t_{n+1}, t_{i,l}, \tilde{Z}_{i,l}) \qquad (n = 0, \cdots, N-1). \tag{2.2.6b}$$

Since $\tilde{Z}_{i,l} = \tilde{Y}_{i,l}$ ($l = 1, \cdots, m$; $i = 0, \cdots, N-1$), (2.2.6b) coincides with (2.2.5b) (with n replaced by $n+1$), thus verifying (2.2.4). □

COROLLARY 2.1. *Let* g *and* k *in* (1.1.1) *be subject to the smoothness hypotheses of Theorem* 1.2, *and let* $c_1, \cdots, c_m$ *be the zeros of* $P_m(2s-1)$. *If* $\tilde{u}^I(t_n)$ *is defined by* (2.2.2a), *with* $\tilde{u} \in S_{m-1}^{(-1)}(Z_N)$ *denoting the solution of the discretized collocation equation* (1.2.3a), *then we have*

$$\max_{t_n \in \bar{Z}_N} |y(t_n) - \tilde{u}^I(t_n)| = O(N^{-2m}) \tag{2.2.7}$$

(*as* $N \to \infty$, *with* $Nh \leqq \gamma T$).

Proof. The above local superconvergence result follows readily from Theorem 1.2 and from (2.2.4). □

In most applications one will not work with the discretized collocation equation (1.2.3a) and the corresponding discretized iterated collocation approximation (2.2.2a), since it may not be possible to extend the given kernel smoothly to the domain S'. Instead, the pair of discretized equations (1.2.3b), (2.2.2b) will be used: as (1.2.3b)

employs only kernel values $k(t, s, \cdot)$ with $(t, s) \in S$, (1.2.3b) and (2.2.2b) represent the natural discretizations of exact collocation and iterated collocation for (1.1.1). The following local superconvergence result holds.

THEOREM 2.2. *Let g and k in* (1.1.1) *be subject to the smoothness hypotheses of Theorem* 1.2, *and let $c_1, \cdots, c_m$ be the zeros of $P_m(2s-1)$. If $\hat{u}^I(t_n)$ is determined by* (2.2.2b), *where $\hat{u} \in S_{m-1}^{(-1)}(Z_N)$ is the solution of the discretized collocation equation* (1.2.3b), *then we have*

$$\max_{t_n \in \bar{Z}_N} |y(t_n) - \hat{u}^I(t_n)| = O(N^{-2m}) \tag{2.2.8}$$

(*as* $N \to \infty$, $Nh \leqq \gamma T$).

Proof. The result of Theorem 2.2 can be established in complete analogy to (2.2.7) and (2.2.4), using (1.2.6b), (1.2.3b), and (2.2.2b).

An alternative way to verify (2.2.8) consists of writing

$$|y(t_n) - \hat{u}^I(t_n)| \leqq |y(t_n) - \tilde{u}^I(t_n)| + |\tilde{u}^I(t_n) - \hat{u}^I(t_n)|, \qquad t_n \in \bar{Z}_N.$$

Since the collocation parameters $c_1, \cdots, c_m$ are the Gauss points for $(0, 1)$, (2.2.8) follows by Corollary 2.1 and by Lemma 2.1. □

We note in passing that the "exact" iterated collocation approximation u^I defined by (2.1.1), with $u \in S_{m-1}^{(-1)}(Z_N)$ being the solution of the exact collocation equation (1.1.6), satisfies also

$$\max_{t_n \in \bar{Z}_N} |y(t_n) - u^I(t_n)| = O(N^{-2m}) \tag{2.2.9}$$

(as $N \to \infty$, $Nh \leqq \gamma T$).

This is an immediate consequence of

$$|y(t_n) - u^I(t_n)| \leqq |y(t_n) - \hat{u}^I(t_n)| + |\hat{u}^I(t_n) - u^I(t_n)|$$

and of the form of the Gauss quadrature errors determining the order of $\hat{u}^I(t_n) - u^I(t_n)$. Hence, (2.2.9) and (2.2.8) generalize the results of Brunner [8] to nonlinear Volterra integral equations.

As an illustration for the above local superconvergence results, we mention the case of *linear* polynomial spline collocation ($m = 2$): if $\hat{u} \in S_1^{(-1)}(Z_N)$ is determined by (1.2.3b), with $c_1 = (3-\sqrt{3})/6$, $c_2 = (3+\sqrt{3})/6$, then the corresponding values $\hat{u}^I(t_n)$ given by (2.2.2b) are convergent of *order* $\hat{p} = 4$: $\max_{t_n \in \bar{Z}_N} |y(t_n) - \hat{u}^I(t_n)| = O(N^{-4})$.

2.3. Polynomial solutions to (1.1.1). Suppose that the exact solution to (1.1.1) is a polynomial, $y \in \pi_{m-1}$. It is then clear that the (exact) collocation approximation $u \in S_{m-1}^{(-1)}(Z_N)$ defined by (1.1.6) satisfies $u(t) = y(t)$ for all $t \in I$, since π_{m-1} is a subspace of $S_{m-1}^{(-1)}(Z_N)$. This no longer holds, however, for the collocation approximation $\hat{u} \in S_{m-1}^{(-1)}(Z_N)$ determined by the *discretized* collocation equation (1.2.3b). To see this, note first that

$$\begin{aligned} y(t) - \hat{u}(t) &= (y(t) - u(t)) + (u(t) - \hat{u}(t)) \\ &= u(t) - \hat{u}(t) = \sum_{l=1}^{m} L_l(\tau)(Y_{n,l} - \hat{Y}_{n,l}), \qquad t = t_n + \tau h_n \in \sigma_n. \end{aligned}$$

Let

$$E_{n,i}^{(j)}[u_i] := \Phi_{n,i}^{(j)}[u_i] - \hat{\Phi}_{n,i}^{(j)}[u_i] \qquad (i \leqq n)$$

denote the quadrature errors induced by the approximations (1.2.1), (1.2.2b) to the integrals (1.1.5). It follows from (1.1.6) and (1.2.3b), invoking a standard argument

based on a discrete Gronwall inequality, that the order of the differences $Y_{n,l} - \hat{Y}_{n,l}$ is given by the order of the quadrature errors $E_{n,i}^{(j)}[u_i]$. According to the above we thus obtain, e.g.,

$$\max_{t_n \in \bar{Z}_N} |y(t_n) - \hat{u}(t_n)| = \begin{cases} O(N^{-2m}) & \text{if } \{c_j\} \text{ are the Gauss points;} \\ O(N^{-(2m-1)}) & \text{if } \{c_j\} \text{ are the Radau II points;} \\ O(N^{-(2m-2)}) & \text{if } \{c_j\} \text{ are the Lobatto points.} \end{cases}$$

Note in particular that if the collocation parameters $\{c_j\}$ are the zeros of $P_m(2s-1)$ (Gauss points), then the errors $y(t_n) - \hat{u}(t_n)$ and $y(t_n) - \hat{u}^I(t_n)$ (cf. (2.2.8)) exhibit the same order of local superconvergence:

$$\max_{t_n \in \bar{Z}_N} |y(t_n) - \hat{u}(t_n)| = O(N^{-2m}), \qquad \max_{t_n \in \bar{Z}_N} |y(t_n) - \hat{u}^I(t_n)| = O(N^{-2m})$$

(as $N \to \infty$, $Nh \leqq \gamma T$).

This result will have certain implications in connection with global error estimation based on collocation at the Gauss points and corresponding iterated collocation (see § 3.2.1).

3. Variable stepsize method. A one-step method with variable stepsize can be separated into three major components:

—a basic fixed-step method

—an error estimator

—a stepsize strategy.

In this chapter these three components will be discussed.

3.1. Fixed-step method. The underlying fixed-step method will be a collocation method as described in the previous chapters. The integrals will be approximated by interpolatory quadrature formulas. For the sake of convenience we give a summary of the formulas used to approximate the solution at the point t_{n+1} (cf. 1.2.3b):

$$\begin{aligned} &\hat{Y}_{n,j} = g(t_{n,j}) + \hat{F}_n(t_{n,j}) + h_n c_j \sum_{l=1}^{s} w_l k\Big(t_{n,j},\, t_n + c_j c_l h_n,\, \sum_{\kappa=1}^{m} L_\kappa(c_j c_l)\, \hat{Y}_{n,\kappa} \Big), \\ &\hat{F}_n(t) := \sum_{i=0}^{n-1} h_i \sum_{l=1}^{s} w_l k(t, t_{i,l}, \hat{Y}_{i,l}), \quad j = 1, \cdots, m; \quad s = m \text{ or } m-1. \end{aligned} \tag{3.1.1}$$

If $c_m = 1$, the approximate solution in t_{n+1}, $\hat{u}_{n+1}$ equals $\hat{Y}_{n,m}$. Otherwise, $\hat{u}_{n+1}$ is either computed by Lagrange interpolation:

$$\hat{u}_{n+1} = \sum_{\kappa=1}^{m} L_\kappa(1)\, \hat{Y}_{n,\kappa}, \tag{3.1.2}$$

which is the discrete analogue of (1.1.7), or by iterated collocation (cf. 2.2.2b):

$$\hat{u}_{n+1}^I = g(t_n + h_n) + \hat{F}_n(t_n + h_n) + h_n \sum_{l=1}^{s} w_l k(t_n + h_n, t_{n,l}, \hat{Y}_{n,l}). \tag{3.1.3}$$

The free collocation parameters can be used to get a method that combines high accuracy with a minimum amount of work involved. The order p of the method is assumed to give an indication of the accuracy of the solution. The amount of computational work involved is normally expressed by the number of kernel evaluations needed.

For various choices of collocation parameters these values are (see Theorem 1.1, Theorem 1.2 and Theorem 2.2):

(a) Gauss-Legendre points $c_1, \cdots, c_m$; $s = m$,

$$p = m,$$

$$n_{\text{kev}} = \frac{N(N-1)}{2} m^2 + N n_I m^2;$$

(b) Gauss-Legendre points $c_1, \cdots, c_{m-1}, c_m = 1$; $s = m-1$,

$$p = 2m-2,$$

$$n_{\text{kev}} = \frac{N(N-1)}{2}(m^2-m) + N n_I (m^2-m);$$

(c) Gauss-Legendre points $c_1, \cdots, c_m$ + Iterated collocation; $s = m$,

$$p = 2m,$$

$$n_{\text{kev}} = \frac{N(N-1)}{2}(m^2+m) + Nm + N n_I m^2;$$

(d) Lobatto points $c_1, \cdots, c_m$; $s = m$,

$$p = 2m-2,$$

$$n_{\text{kev}} = \frac{N(N-1)}{2}(m-1)^2 + 2N(m-1) + N n_I (m-1)^2;$$

(e) Radau II points $c_1, \cdots, c_m$; $s = m$,

$$p = 2m-1,$$

$$n_{\text{kev}} = \frac{N(N-1)}{2} m^2 + N n_I m^2.$$

Here, N denotes the number of steps taken and n_I the number of iterations needed to solve the (non)linear system for $\hat{Y}_{n,j}$; $n_I = 1$ if the integral equation has a linear kernel, and n_I is bounded by a constant for nonlinear kernels.

Notice the low number of kernel evaluations in the case of Lobatto points as compared to the amount needed when using (the same number of) Radau II points. This decrease in the number of kernel evaluations is caused by the fact that one can make use of the coincidence of the points $t_{n-1} + c_m h_{n-1}$ and $t_n + c_1 h_n$ and reformulate (3.1.1) into:

$$\begin{aligned}
&\hat{Y}_{n,1} = \hat{u}_n,\\
&\hat{Y}_{n,j} = g(t_{n,j}) + \hat{F}_n(t_{n,j}) + h_n c_j w_1 k(t_{n,j}, t_n, \hat{u}_n)\\
&\qquad + h_n c_j \sum_{l=2}^{m} w_l k\Big(t_{n,j}, t_n + c_j c_l h_n, \sum_{\kappa=1}^{m} L_\kappa(c_j c_l)\, \hat{Y}_{n,\kappa}\Big), \qquad j = 2, \cdots, m,\\
&\hat{F}_n(t) := h_0 w_1 k(t, t_0, y(t_0))\\
&\qquad + \sum_{i=0}^{n-2}\Big[h_i \sum_{l=2}^{m-1} w_l k(t, t_{i,l}, \hat{Y}_{i,l}) + (h_i + h_{i+1}) w_m k(t, t_i + h_i, \hat{Y}_{i,m})\Big]\\
&\qquad + h_{n-1} \sum_{l=2}^{m} w_l k(t, t_{n-1,l}, \hat{Y}_{n-1,l}).
\end{aligned} \tag{3.1.1'}$$

From the above, it seems that the method with the best accuracy/computational work ratio will be obtained by using the Lobatto points.

3.2. Error estimation. To estimate the error in the approximate solution $\hat{u}_{n+1}$ in the point t_{n+1} one has the choice between global and local error estimation. Both will be treated below.

3.2.1. Global error estimation. The global error in the point t_{n+1} is defined by $|y(t_{n+1}) - \hat{u}_{n+1}|$. It is normally approximated by the difference between $\hat{u}_{n+1}$ and a reference solution that is assumed to have a higher accuracy. We will compute the reference solution with a higher-order method using the same stepsize.

If we consider the possible combinations of methods to approximate the solution and to compute the reference solution two of these appear to have the preference with respect to reliability and efficiency: Gauss-Legendre for the computation of $\hat{u}_{n+1}$ combined with iterated collocation, using the same collocation points, for the reference solution, and Lobatto combined with a Lobatto method of the next higher order.

If the error in $\hat{u}_{n+1}$ is $O(N^{-2\kappa})$, then the number of kernel evaluations for these two variants is approximately:

$$\text{Gauss}_{2\kappa} + \text{iterated collocation:} \quad (2\kappa^2+\kappa)N^2,$$

$$\text{Lobatto}_{\kappa+1} + \text{Lobatto}_{\kappa+2}: \quad \frac{2\kappa^2+2\kappa+1}{2} N^2.$$

For each method the index denotes the number of collocation points needed in order to obtain the desired order.

Note, that Gauss $+ (c_m = 1)$, respectively Radau II combinations, need roughly the same number of kernel evaluations as the Lobatto method, and can be expected to be equally robust.

Although the Gauss method seems obviously less efficient, it has some additional advantages. Its implementation is straightforward and it needs considerably less overhead. Besides, the order of accuracy not only of the reference solution but also of the quadrature part of the approximation doubles the order of $\hat{u}(t)$ in the steppoints, which makes it more robust for a variable stepsize method (see also § 3.3). However, as mentioned in § 2.3, it has one drawback. If the solution of the integral equation is a polynomial of degree $< m$, $\hat{u}(t)$ has an error of $O(N^{-2m})$ not only in the collocation points but also in the steppoints. Since the reference solution is equally accurate this results in a very unreliable error estimate. It is however possible to detect this failure automatically and switch to another method whenever this undesired superconvergence in the steppoints occurs (see § 4.2).

3.2.2. Local error estimation. In analogy to what is common practice in the theory of initial-value problems for ordinary differential equations, we define the local error as the error resulting from a single step of the approximating method, not taking into account the errors inherited from previous steps. So, the local error in the point t_{n+1} is defined by:

$$LE_{n+1} := \left| \int_{t_n}^{t_n+1} k(t_{n+1}, s, y_n(s))\, ds - h_n \sum_{l=1}^{s} w_l k(t_{n+1}, t_{n,l}, \hat{Y}_{n,l}) \right|, \tag{3.2.2.1}$$

with

$$y_n(t) := g(t) + \hat{F}_n(t) + \int_{t_n}^{t} k(t, s, y_n(s))\, ds \tag{3.2.2.2}$$

(cf. Arndt [1]).

An estimation of the local error is obtained by approximating the integral by a higher-order method using the same stepsize, i.e.,

$$\int_{t_n}^{t_{n+1}} k(t_{n+1}, s, y_n(s))\, ds \approx h_n \sum_{l=1}^{\tilde{s}} \tilde{w}_l k(t_{n+1}, t_n + \tilde{c}_l h_n, \tilde{Y}_{n,l}), \tag{3.2.2.3}$$

where

$$\begin{aligned} \tilde{Y}_{n,j} &= g(t_n + \tilde{c}_j h_n) + \hat{F}_n(t_n + \tilde{c}_j h_n) \\ &\quad + h_n \tilde{c}_j \sum_{l=1}^{\tilde{s}} \tilde{w}_l k\left(t_n + \tilde{c}_j h_n,\ t_n + \tilde{c}_j \tilde{c}_l h_n,\ \sum_{\kappa=1}^{\tilde{m}} \tilde{L}_\kappa(\tilde{c}_j \tilde{c}_l)\, \tilde{Y}_{n,\kappa} \right). \end{aligned} \tag{3.2.2.4}$$

Note that it is assumed that $\tilde{c}_{\tilde{m}} = 1$.

3.3. Stepsize strategy. Having an estimation of the error, we will now discuss how to determine the stepsize to limit this error to a user specified tolerance.

The stepsize strategy consists of advancing the solution with a trial stepsize, then accept or reject the result and repeating the process with as trial stepsize a modification of the previous stepsize. This modification should account for the (lack of) accuracy in the accepted or rejected trial solution. It should result in as large a stepsize as possible while still providing an accuracy that satisfies the specified tolerance.

The control of the global error in a steppoint, say t_{n+1}, is a straightforward process if a global error estimation is available. A trial solution will then be accepted if the global error estimation is less than the specified error tolerance, i.e.,

$$GEE_{n+1} \leqq TOL. \tag{3.3.1}$$

The modification of the stepsize reads:

$$h_{new} = h_n \left(\frac{TOL}{GEE_{n+1}} \right)^{1/p}. \tag{3.3.2}$$

Here, p is the order of the underlying method, and h_{new} stands for h_{n+1} if the trial solution has been accepted, and otherwise for h_n.

Note that, once a certain point t_{n+1} has been reached, we cannot control the error in the lag term part any more, since the approximating solution for that part of the interval has already been accepted. This means that it is defensible to control only the local error and assume that the global error in the endpoint will be more or less equal to the sum of the local errors.

In this case one accepts a trial solution only if

$$LEE_{n+1} \leqq \frac{TOL \cdot h_n}{t_{n+1}}, \tag{3.3.3}$$

and modifies the stepsize by:

$$h_{new} = h_n \left(\frac{TOL \cdot h_n}{t_{n+1} \cdot LEE_{n+1}} \right)^{1/q}, \tag{3.3.4}$$

where q is the order of the quadrature formula to approximate the integral.

So far, the stepsize strategy is completely analogous to the strategy used in variable stepsize methods for ordinary differential equations. However, the intention is to limit the global error over the whole interval to the user-defined tolerance, i.e., to have

$$|y(t) - \hat{u}(t)| \leqq TOL \quad \text{for } 0 \leqq t \leqq T.$$

The disadvantage of proceeding in the same way as in the case of ODEs is, as Arndt ([1], [2]) points out, that the consequence of accepting the approximated solution to future quadrature computations is not taken into account. This means that the assumption that the sum of the local errors equals the global error in the endpoint is not necessarily valid.

To obviate the problem of the possibly large errors in future lag term computations, once t_{n+1} has been reached, one has to demand that:

$$|y(t)-\hat{u}(t)| \leqq TOL \quad \text{for all } t \in [t_{n+1}, T].$$

This reads in discretized form:

$$\left|\sum_{i=0}^{l-1}\left(\int_{t_i}^{t_{i+1}} k(t_l, s, y(s))\,ds - h_i \sum_{j=1}^{s} w_j k(t_l, t_{i,j}, \hat{Y}_{i,j})\right)\right| \leqq TOL, \tag{3.3.5}$$

$$\text{for } t_l = T, T-H, \cdots, t_{n+1}, \quad \text{with } H \text{ some fixed number.}$$

Let

$$EE_{i+1}^{l} \approx \int_{t_i}^{t_{i+1}} k(t_l, s, y(s))\,ds - h_i \sum_{j=1}^{s} w_j k(t_l, t_{i,j}, \hat{Y}_{i,j}). \tag{3.3.6}$$

Then (3.3.5) is satisfied when

$$\sum_{i=n}^{l-1} |EE_{i+1}^{l}| + \left|\sum_{i=0}^{n-1} EE_{i+1}^{l}\right| \leqq TOL.$$

In this way one gets, as Arndt suggests, a modified stepsize strategy that strives for a uniform control of the error, i.e., let

$$EE_{n+1} = \max_{t_{n+1} \leqq t_l \leqq T}\left((t_l - t_n)\cdot \frac{|EE_{n+1}^{l}|}{h_n} + \left|\sum_{i=0}^{n-1} EE_{i+1}^{l}\right|\right). \tag{3.3.7}$$

Accept the trial solution in t_{n+1} if

$$EE_{n+1} \leqq TOL. \tag{3.3.8}$$

The modification of the stepsize is

$$h_{new} = h_n\left(\frac{TOL}{EE_{n+1}}\right)^{1/q}, \tag{3.3.9}$$

with q the order of the quadrature formula.

Note that although global error control does not lead to an erroneous error estimate in the endpoint, it is certainly possible that by allowing too large a stepsize at the start, it will be impossible to meet the tolerance as the integration advances, due to an unacceptably large error in the lag term. Therefore, to avoid problems later on, it seems advisable to use the global error control without the uniform error control only if $p \ll q$, which is the case for the Gauss collocation method combined with iterated collocation to estimate the global error.

4. Implementation. For the miscellaneous programming matters and heuristics, e.g., stepsize constraints, the solution of nonlinear systems, etc., we refer to Blom and Brunner [4]. Here, we consider two more general topics.

4.1. Approximation of the lag term. The lion's share of the computational work in all methods is the approximation of the lag term by some quadrature rule. Moreover, for all methods except the iterated collocation method the additional storage space needed for all $\hat{Y}_{n,j}$ is substantial.

However, the number of kernel evaluations needed to compute the reference solution can be significantly reduced by approximating the lag term by Lagrangian interpolation instead of using a quadrature formula. Interpolation over m points yields

$$\hat{F}_n^{(1)}(t_n+\tau h_n)=\sum_{\kappa=1}^{m}\hat{F}_n(t_{n,\kappa}) \tag{4.1.1}$$

and adds an additional interpolation error of $O(h_n^m)$, unless the kernel satisfies $k\in\pi_{m-2}$. So, in general, this interpolation gives rise to a reference solution of $O(h_n^m)$, which is not adequate.

However, one can also interpolate over $2m$ points

$$\hat{F}_n^{(2)}(t_n+\tau h_n)=\sum_{\kappa=1}^{\bar{m}}\bar{L}_\kappa\left(\frac{h_{n-1}+\tau h_n}{h_{n-1}+h_n}\right)\hat{F}_n(t_{n-1}+\bar{c}_\kappa(h_{n-1}+h_n)), \tag{4.1.2}$$

where

$$\bar{m}=2m$$

and

$$\bar{c}_j=c_j\frac{h_{n-1}}{h_{n-1}+h_n},\qquad j=1,\cdots,m,$$

$$\bar{c}_j=\frac{h_{n-1}+c_l h_n}{h_{n-1}+h_n},\qquad j=\bar{m}-m+1,\cdots,\bar{m};\quad l=j+m-\bar{m},$$

and where the $\bar{L}_\kappa$ denote the Lagrange canonical polynomials corresponding to the $\bar{c}_j$. This interpolation introduces an error of $O((h_{n-1}+h_n)^{\bar{m}})$, unless $k\in\pi_{\bar{m}-2}$, which is small enough to preserve the legitimacy of the reference solution in case the approximating method is based on Gauss collocation.

For all other methods that are mentioned in § 3.1 this kind of approximation of the lag term cannot be used when computing the global error estimation, since the error in the $\hat{F}_n$ will always be a spoiling factor, no matter how accurate the interpolation will be. However, since local error estimation needs only the values of $\hat{F}_n$ in some intermediate points, one can use the interpolation formula (4.1.2) which is sufficiently accurate for all methods described.

Note that for Lobatto collocation the value of $\bar{m}$ is $2m-1$ instead of $2m$, due to the coincidence of $t_{n-1}+c_m h_{n-1}$ and $t_n+c_1 h_n$.

4.2. Detection of a polynomial solution when using Gauss collocation. As mentioned in § 2.3 the Gauss method yields an error of $O(N^{-2m})$ in the steppoints if the solution is an element of π_{m-1}. If the reference solution is equally or less accurate, then this results in an unreliable global error estimate. One can detect this error by computing the approximations $\hat{\hat{Y}}_{n,j}$ to $y(t_n+c_j h_n/2)$ and use Lagrangian interpolation to compute the solution at some point of the interval $[t_n, t_n+h_n/2]$, say t_n, both with these approximations and with the $\hat{Y}_{n,j}$. If $y\notin\pi_{m-1}$, then $e_1=|y(t_n)-\hat{u}_n|\approx O(h_n^m)$ and $e_2=|y(t_n)-\hat{\hat{u}}_n|\approx O((h_n/2)^m)$. So, if $e_1/e_2\approx 2^m$, the order of the Gauss method is as expected.

To avoid the computation of the quadrature of the lag term, we use Lagrange interpolation over the points 0.0, c_j, 1.0. No additional kernel evaluations are needed, since the $\hat{F}_n(t_n)$ and the $\hat{F}_n(t_{n+1})$ already have been calculated during the computation of the reference solution in t_n, respectively t_{n+1}. Now, if $e_1/e_2\ll 2^m$, then it is almost certain that $y\in\pi_{m-1}$.

Note that if both $y \in \pi_{m-1}$ and $\int_0^{t_n} k(t, s, y(s))\, ds \in \pi_{m+1}$, then it is not clear what the value of the ratio e_1/e_2 will be. But in this case the approximation of both the lag term and the solution is "exact", which means that the only problem that can arise is an unjustifiable rejection of the current combination of methods for approximating the solution and the reference solution.

5. Numerical performance. The ideas presented in the preceding sections have resulted in a FORTRAN code named COLV12; compare Blom and Brunner [4], where a description of this code and its usage is given.

To show some properties of the code, we tested it on the six problems that were used by Schlichte [19] to compare his method, IVRKX, with the codes ORION of Bader and Kunkel [3], VOLTEX of Hock [14], and VOLCON of Hairer, Lubich and Schlichte [13]. The performance data of these codes for the six problems are listed in the Appendix. We also added the results of VE1 of Bownds and Appelbaum [6], [5] and of INTSOL of Jones and McKee [16], insofar as we could find these results in the literature. (See Tables A1 and A2.)

The tests were performed on a CDC CYBER-750, in single precision (a machine precision of approximately 14 digits).

The problems are:

(1) (problem 4 from Schlichte)

$$y(t) = \frac{1}{2}t^2 e^{-t} + \int_0^t \frac{(t-s)^2}{2} e^{s-t} \cdot y(s)\, ds \quad \text{for } t \in [0, 5]$$

with solution:

$$y(t) = \frac{1}{3}\left(1 - e^{-\frac{3}{2}t}\left[\cos\left(\frac{1}{2}t\sqrt{3}\right) + \sqrt{3}\sin\left(\frac{1}{2}t\sqrt{3}\right)\right]\right);$$

(2) (problem 15 from Schlichte)

$$y(t) = 1 + (\sin(t))^2 - \int_0^t 3\sin(t-s) \cdot y^2(s)\, ds \quad \text{for } t \in [0, 5]$$

with solution:

$$y(t) = \cos(t);$$

(3) (problem 19 from Schlichte)

$$y(t) = \cos(t) - \int_0^t \frac{2}{(t-s+2)^2} \cdot (y(s) + y^3(s))\, ds \quad \text{for } t \in [0, 40]$$

with solution:

$$y(40) \approx -0.65013110133344;$$

(4) (problem 20 from Schlichte)

$$y(t) = 1 + \int_0^t (t-s)^3(4-t+s)\, e^{s-t} \cdot \frac{y^4(s)}{1+2y^2(s)+2y^4(s)}\, ds \quad \text{for } t \in [0, 10]$$

with solution:

$$y(10) = 1.2599558233723;$$

(5) (problem 16 from Schlichte)

$$y(t) = e^{-t} + \int_0^t e^{s-t}[y(s) + e^{-y(s)}]\, ds \quad \text{for } t \in [0, 40]$$

with solution:

$$y(t) = \ln(t+e);$$

(6) (problem 21 from Schlichte)

$$y(t) = t - 1 + (1+t^2)\, e^{-t^2} + \int_0^t t^2\, e^{-ts} \cdot y(s)\, ds \quad \text{for } t \in [0, 5]$$

with solution:

$$y(t) = t.$$

The following combinations of methods for the approximation of the solution, respectively the reference solution, have been used:

Gm+IG	m points Gauss+iterated collocation,
1Gm+1G(m+1)	$m-1$ Gauss points+($c_m = 1$), together with m Gauss points+($c_{m+1} = 1$),
Lm+L(m+1)	m, combined with $m+1$ points Lobatto collocation,
Rm+R(m+1)	m, together with $m+1$ points Radau collocation.

As stepsize strategies we used global error estimate/global error control (G), and, if applicable, local error estimate/local error control (L). In the latter the lag term needed for the reference solution has been approximated by interpolation (cf. 4.1.2).

The meaning of the headers in the tables is:

sd	number of significant digits: $-^{10}\log \lvert y(T) - \hat{u}(T)\rvert / \max(1.0, \lvert y(T)\rvert)$;
EE/GE	ratio between the global error estimate in T and the real global error. Note: if the local error is uniformly controlled, the global error in T is estimated by: $\sum_{i=0}^{N-1} EE_{i+1}^{l}$, with $t_l = T$ (cf. 3.3.6);
N	number of intervals used;
#F tot	number of failed steps in total;
#F 0	number of failed steps at the start in $t = 0.0$;
#kev	total number of kernel evaluations needed.

Table 5.1 gives the results that are obtained when no special care has been taken. For example, for the (iterated) Gauss method, no testing on polynomial solutions has been done, and for the other methods, the error is not uniformly controlled.

For all experiments the initial stepsize was 1.0; the maximal stepsize allowed was 5.0, which was actually used in problem 5.

All failures in Table 5.1 were due to a rejection of a step with the minimum allowed stepsize, which was 5 E-3.

As was expected, problem 6, with solution $y = t$, resulted in a bad performance for the Iterated Gauss method; but problem 5, with solution $y = \ln(t+e)$, also seemed to suffer from a bad error estimate. Both problems were rerun with the polynomial test on and with an automatic escape to a local error estimate and uniform local error control Gauss$_m$, Gauss$_{m+1}$+($c_{m+2} = 1$) combination. The escape was made after two consecutive occurrences of polynomial behavior, and the last three steps were discarded. As can be seen in Table 5.2, in most cases an escape was needed.

In Table 5.3 the results are listed that were obtained by using a uniform error control, or rather an error control in discrete points spaced 1.0. Here, if a step was rejected which already had a minimum size, the tolerance was relaxed by a factor 4 and the computation was resumed.

Recall that when using iterated Gauss collocation to compute the reference solution there is no need to employ a uniform error control (see § 3.3).

TABLE 5.1
Results for method combinations, without special features.

Prob. 1		Requested tolerance 1 E-4						Requested tolerance 1 E-7					
Method		sd	EE/GE	N	#F tot	#F 0	#kev	sd	EE/GE	N	#F tot	#F 0	#kev
G	G4+IG	5.2	1	8	2	2	760	9.3	1	33	5	3	12100
G	1G4+1G5	4.6	1	4	0	0	320	7.1	1	12	2	1	2592
G	L4+L5	4.5	1	4	0	0	278	7.1	1	12	2	1	2123
G	R3+R4	4.5	1	5	1	0	425	---	-	--	-	-	----
G	G8+IG	7.4	1	3	0	0	432	9.8	1	4	0	0	720
G	1G6+1G7	8.2	1	3	0	0	432	8.2	1	3	0	0	432
G	L6+L7	8.5	1	3	0	0	399	8.1	1	4	0	0	654
G	R5+R6	6.8	1	3	0	0	366	8.0	1	4	0	0	610
L	1G4+1G5	4.6		4	0	0	276	7.1		11	2	1	1311
L	L4+L5	4.5		4	0	0	282	7.1		11	2	1	1213
L	R3+R4	5.6		6	2	0	472	4.9		20	2	0	2612
L	1G6+1G7	8.2		3	0	0	429	8.2		3	0	0	429
L	L6+L7	8.5		3	0	0	438	8.1		4	0	0	634
L	R5+R6	6.8		3	0	0	366	8.0		4	0	0	538
Prob. 2		**Requested tolerance 1 E-4**						**Requested tolerance 1 E-7**					
Method		sd	EE/GE	N	#F tot	#F 0	#kev	sd	EE/GE	N	#F tot	#F 0	#kev
G	G4+IG	6.0	5E-1	9	3	1	1596	8.4	1	43	5	2	21988
G	1G4+1G5	---	-	--	-	-	----	---	-	--	-	-	----
G	L4+L5	---	-	--	-	-	----	---	-	--	-	-	----
G	R3+R4	---	-	--	-	-	----	---	-	--	-	-	----
G	G8+IG	6.2	1	3	0	0	1136	8.4	1E-1	4	1	0	2152
G	1G6+1G7	5.2	1E-1	3	0	0	1224	---	-	--	-	-	----
G	L6+L7	4.1	1	3	0	0	1070	---	-	--	-	-	----
G	R5+R6	---	-	--	-	-	----	---	-	--	-	-	----
L	1G4+1G5	1.5		7	1	0	1396	4.9		22	5	1	6429
L	L4+L5	1.3		8	2	0	1535	4.6		23	4	1	5425
L	R3+R4	2.8		12	4	1	2251	5.6		45	8	2	14730
L	1G6+1G7	5.2		3	0	0	1221	6.3		5	1	0	2490
L	L6+L7	4.1		3	0	0	1109	5.9		5	1	0	2246
L	R5+R6	2.9		4	1	0	1672	5.4		7	1	1	2740
Prob. 3		**Requested tolerance 1 E-4**						**Requested tolerance 1 E-7**					
Method		sd	EE/GE	N	#F tot	#F 0	#kev	sd	EE/GE	N	#F tot	#F 0	#kev
G	G4+IG	4.2	1	70	31	2	75696	8.7	1	366	72	4	1615392
G	1G4+1G5	5.6	2	34	14	0	31552	7.9	1	98	41	2	231828
G	L4+L5	4.7	1	43	20	0	38350	8.6	1	104	50	2	214553
G	R3+R4	4.5	1	44	18	0	39759	7.3	1	175	52	2	513900
G	G8+IG	6.2	1	27	13	0	48824	7.1	1	54	19	2	154616
G	1G6+1G7	5.3	1	15	2	0	14832	7.2	1	33	9	0	61242
G	L6+L7	5.3	1	17	1	0	13679	7.9	1	39	12	1	73395
G	R5+R6	5.0	1	28	7	0	37235	7.8	1	45	10	1	88511
L	1G4+1G5	6.1		54	21	0	34093	9.5		176	45	2	262811
L	L4+L5	7.2		57	13	0	26104	9.9		190	47	2	226241
L	R3+R4	6.3		90	30	0	58567	9.2		358	68	2	714682
L	1G6+1G7	7.3		26	5	0	21167	9.2		50	18	0	72508
L	L6+L7	6.8		28	5	0	21113	9.3		50	11	1	55681
L	R5+R6	6.8		28	3	0	18805	9.8		61	24	1	87408

TABLE 5.1. (Continued)

Prob. 4		Requested tolerance 1 E-4						Requested tolerance 1 E-7					
Method		sd	EE/GE	N	#F tot	#F 0	#kev	sd	EE/GE	N	#F tot	#F 0	#kev
G	G4+IG	4.5	1	11	5	1	2224	7.7	1	50	9	3	29588
G	1G4+1G5	4.9	1	10	0	0	2368	7.5	1	32	3	2	19168
G	L4+L5	4.8	1	10	0	0	1920	7.4	1	33	3	2	16102
G	R3+R4	5.2	1	15	2	1	3841	7.3	1	60	2	2	48541
G	G8+IG	7.5	1	4	0	0	1360	8.8	1	7	1	1	3240
G	1G6+1G7	5.7	1	5	0	0	1872	7.5	1	7	0	0	3240
G	L6+L7	5.6	1	5	0	0	1641	7.1	1	7	0	0	2761
G	R5+R6	6.0	1	5	0	0	1525	8.0	1	9	0	0	3843
L	1G4+1G5	6.1		12	1	0	2255	8.1		42	3	2	15303
L	L4+L5	6.3		13	1	0	2168	8.1		44	3	2	13338
L	R3+R4	4.7		21	2	1	3749	7.6		90	3	2	42831
L	1G6+1G7	5.7		5	0	0	1657	8.3		8	0	0	2896
L	L6+L7	5.6		5	0	0	1526	8.4		8	1	0	3001
L	R5+R6	5.5		6	0	0	1689	7.8		10	1	0	3709
Prob. 5		**Requested tolerance 1 E-4**						**Requested tolerance 1 E-7**					
Method		sd	EE/GE	N	#F tot	#F 0	#kev	sd	EE/GE	N	#F tot	#F 0	#kev
G	G4+IG	3.2	-5E-3	11	0	0	1672	5.8	-6E-3	39	2	2	16984
G	1G4+1G5	4.3	1	29	0	0	15776	7.3	1	93	1	1	146048
G	L4+L5	4.3	1	30	0	0	13335	7.3	1	98	1	1	127043
G	R3+R4	4.2	1	51	1	0	35325	7.3	1	227	3	2	658650
G	G8+IG	9.2	5E-5	10	0	0	5432	9.3	4E-4	11	0	0	6672
G	1G6+1G7	5.0	1	10	0	0	5616	7.5	1	17	0	0	13968
G	L6+L7	4.9	1	10	0	0	4868	7.5	1	18	0	0	13252
G	R5+R6	4.3	1	11	0	0	5551	7.5	1	24	0	0	21594
L	1G4+1G5	4.2		27	0	0	7349	7.3		92	1	1	61043
L	L4+L5	4.6		33	1	0	8367	7.4		99	1	1	54443
L	R3+R4	4.1		47	0	0	13581	7.2		208	2	2	212869
L	1G6+1G7	5.3		11	0	0	4693	7.0		15	3	0	8856
L	L6+L7	5.5		11	1	0	4538	8.0		19	0	0	9258
L	R5+R6	4.9		12	1	0	4791	7.0		20	1	0	9826
Prob. 6		**Requested tolerance 1 E-4**						**Requested tolerance 1 E-7**					
Method		sd	EE/GE	N	#F tot	#F 0	#kev	sd	EE/GE	N	#F tot	#F 0	#kev
G	G4+IG	3.9	-7E-3	3	0	0	120	3.9	-3E-5	5	2	0	400
G	1G4+1G5	---	-	--	-	-	----	---	-	--	-	-	----
G	L4+L5	---	-	--	-	-	----	---	-	--	-	-	----
G	R3+R4	---	-	--	-	-	----	---	-	--	-	-	----
G	G8+IG	10.4	-4E-4	3	0	0	432	10.4	-4E-4	3	0	0	432
G	1G6+1G7	5.5	1	3	0	0	432	---	-	--	-	-	----
G	L6+L7	5.4	1	3	0	0	399	---	-	--	-	-	----
G	R5+R6	4.7	1	3	0	0	366	---	-	--	-	-	----
L	1G4+1G5	2.5		4	1	0	327	3.8		9	3	1	1044
L	L4+L5	2.4		4	1	0	332	3.8		9	3	1	987
L	R3+R4	1.9		5	1	0	336	3.3		14	5	1	1679
L	1G6+1G7	5.5		3	0	0	429	5.5		4	1	0	745
L	L6+L7	5.4		3	0	0	438	5.5		4	1	0	744
L	R5+R6	4.7		3	0	0	366	4.7		4	1	0	635

TABLE 5.2
Results for Iterated Gauss; esc. to $Gauss_m + Gauss_{m+1} + (c_{m+2} = 1)$, *local error control.*

Prob. 5		Requested tolerance 1 E-4						Requested tolerance 1 E-7					
Method		sd	EE/GE	N	#F tot	#F 0	#kev	sd	EE/GE	N	#F tot	#F 0	#kev
G	G4+IG		escape	at	t	=	2.3		escape	at	t	=	3.2
L	G4+1G6	4.5	6E-2	14	6	0	8604	6.2	1E-3	33	9	2	21541
G	G8+IG		escape	at	t	=	21.0		escape	at	t	=	2.7
L	G8+1G10	9.2	-1E-1	10	3	0	11537	9.3	-3E-2	11	2	0	16360
Prob. 6		**Requested tolerance 1 E-4**						**Requested tolerance 1 E-7**					
Method		sd	EE/GE	N	#F tot	#F 0	#kev	sd	EE/GE	N	#F tot	#F 0	#kev
G	G4+IG		escape	at	t	=	2.8		escape	at	t	=	1.8
L	G4+1G6	7.3	1	7	4	2	1417	9.6	1	11	5	2	2436
G	G8+IG	10.4	-4E-4	3	0	0	624	10.4	-4E-4	3	0	0	624
L	G8+1G10												

The failures in Table 5.3 were caused by a lack of working storage, which occurred, depending on the method used, after about 200–350 successful steps.

Furthermore, we tried for one problem what the influence is if only the stepsize is adjusted according to the estimated error, but if no steps are rejected. Although in most cases this results in fewer kernel evaluations (if the code reaches the endpoint), it is by far not as reliable.

To present a survey of the performance of the code on these six problems a weighted sum has been computed over all (available) "sd"- and "#kev"-values. The weights are inversely proportional to the square of the length of the interval on which the solution has been computed.

Table 5.4 contains the figures of the average performance of the different methods. For the iterated collocation method the results have been taken from Table 5.1 and 5.2 for problems 1–4, respectively 5 and 6. For the other methods the "sd"- and "#kev"-values of Table 5.3 have been used. The results in the first column are the average results computed over all problems. The second contains the average values when the "bad ones" are left out, i.e., for the Gauss+Iterated collocation method problems 5 and 6, and for all other methods problem 2.

To illustrate how the code performs in extreme cases we tried an example with a discontinuous solution. We chose for this test the problem:

$$y(t) = g(t) + \int_0^t y(s)\, ds \quad \text{for } t \in [0, 5]$$

with:

$$y(t) = \begin{cases} t & t \leqq 2.5 \\ 1/t & t > 2.5 \end{cases} \quad \text{and} \quad g(t) = \begin{cases} t - t^2/2, & t \leqq 2.5, \\ 1/t - 3.125 - \ln(t) + \ln(2.5), & t > 2.5. \end{cases}$$

In Table 5.5 the results are listed that were obtained using the same features as in Table 5.4; i.e., in case of the Gauss method the solution was tested on polynomial behavior and in all other methods the error was controlled uniformly in discrete points spaced 1.0.

In all tests it was allowed to relax the tolerance. This has been done repeatedly in the last point before the discontinuity for all entries in the table where the number of failures in 2.5 is greater than 10; for the G8+IG combination until the tolerance reached a value greater than 0.4 and for all other methods until it was greater than 1.6.

TABLE 5.3
Results with uniform error control and tolerance relaxing.

Prob. 1		Requested tolerance 1 E-4						Requested tolerance 1 E-7					
Method		sd	EE/GE	N	#F tot	#F 0	#kev	sd	EE/GE	N	#F tot	#F 0	#kev
G	1G4+1G5	4.8	1	4	0	0	390	7.2	1	10	1	1	2016
G	L4+L5	4.6	1	4	0	0	368	7.2	1	11	1	1	2056
G	R3+R4	4.6	1	5	1	0	548	7.2	1	18	2	2	4752
G	1G6+1G7	8.2	1	3	0	0	531	9.0	1	4	0	0	830
G	L6+L7	8.5	1	3	0	0	516	8.9	1	4	0	0	784
G	R5+R6	6.8	1	3	0	0	465	8.3	1	4	0	0	731
L	1G4+1G5	4.8	5E-1	4	0	0	346	7.3	3E-1	11	1	1	1498
L	L4+L5	4.7	1	4	0	0	372	7.3	4E-1	11	1	1	1478
L	R3+R4	5.7	-1	6	1	0	536	8.2	2E-1	22	3	2	3631
L	1G6+1G7	8.2	1	3	0	0	528	9.0	1	4	0	0	742
L	L6+L7	8.5	-2	3	0	0	555	8.9	1	4	0	0	764
L	R5+R6	6.8	2	3	0	0	465	8.3	1	4	0	0	659
Prob. 2[1]		Requested tolerance 1 E-4						Requested tolerance 1 E-7					
Method		sd	EE/GE	N	#F tot	#F 0	#kev	sd	EE/GE	N	#F tot	#F 0	#kev
G	1G4+1G5	3.2	1	53	14	1	55521	6.6	1	24	16	2	20762
G	L4+L5	3.1	1	19	16	1	9849	7.4	1	24	16	2	16953
G	R3+R4	4.3	1	70	4	2	65979	6.0	1	54	17	3	50337
G	1G6+1G7	4.8	1	4	0	0	1766	6.4	1	115	13	1	506569
G	L6+L7	4.1	1	4	0	0	1577	---	-	--	-	-	----
G	R5+R6	---	-	--	-	-	----	6.1	1	8	18	1	12305
L	1G4+1G5	3.5	-1E-1	9	2	1	2204	6.3	-5E-2	26	3	2	8058
L	L4+L5	3.8	-1E-1	9	2	1	1943	6.3	-5E-2	27	3	2	7345
L	R3+R4	3.1	2E-2	15	5	2	3446	5.7	8E-3	52	4	3	18608
L	1G6+1G7	4.8	2E-1	4	0	0	1678	6.4	-4E-2	6	1	1	3026
L	L6+L7	4.1	5E-2	4	0	0	1557	6.3	-3E-2	6	1	1	2776
L	R5+R6	3.0	-5E-3	4	0	0	1452	6.3	-1E-2	8	1	1	3481
Prob. 3		Requested tolerance 1 E-4						Requested tolerance 1 E-7					
Method		sd	EE/GE	N	#F tot	#F 0	#kev	sd	EE/GE	N	#F tot	#F 0	#kev
G	1G4+1G5	5.0	1	33	6	0	30327	7.4	1	105	26	2	252649
G	L4+L5	4.5	1	38	12	0	37036	8.3	1	113	32	2	242699
G	R3+R4	4.6	1	51	24	0	63912	7.5	1	186	23	2	526699
G	1G6+1G7	4.8	1	16	0	0	17587	8.3	1	37	8	0	82764
G	L6+L7	6.1	1	18	2	0	21046	8.9	1	40	12	1	90095
G	R5+R6	5.3	1	27	5	0	40615	7.8	1	44	7	1	91051
L	1G4+1G5	6.1	1	55	25	0	47213	9.6	1	177	51	2	297461
L	L4+L5	6.5	3E-1	60	20	0	44197	9.9	2	189	44	2	263067
L	R3+R4	6.3	1	89	30	0	73244	9.2	1	359	68	2	769421
L	1G6+1G7	7.3	-1	26	5	0	28064	9.1	1	48	16	0	80400
L	L6+L7	7.7	6	28	4	0	28346	9.2	1	50	12	1	71930
L	R5+R6	6.9	-6	28	3	0	25616	10.0	2	63	22	1	106765

[1] The tolerance has been relaxed two times in all global methods except for the R3+R4, the 1G6+1G7, and the L6+L7 combinations with tolerance 1 E-4.

The "bad" error estimates for the Gauss method can be explained by the fact that the approximation is "exact" until the discontinuity has been reached and therefore the code gives no warning that the solution is a polynomial. The step that covers the discontinuity results in one warning, but since in the next step the solution is no longer a polynomial no escape will be made.

TABLE 5.3. (Continued)

Prob. 4		Requested tolerance 1 E-4						Requested tolerance 1 E-7					
Method		sd	EE/GE	N	#F tot	#F 0	#kev	sd	EE/GE	N	#F tot	#F 0	#kev
G	1G4+1G5	4.9	1	10	0	0	2697	7.4	1	32	2	2	20422
G	L4+L5	4.8	1	10	0	0	2370	7.5	1	34	2	2	18776
G	R3+R4	5.3	1	15	1	1	4320	7.3	1	60	2	2	51008
G	1G6+1G7	5.8	1	5	0	0	2158	8.8	1	8	1	0	4904
G	L6+L7	5.7	1	5	0	0	1979	8.6	1	8	1	0	4401
G	R5+R6	6.0	1	5	0	0	1822	8.8	1	10	1	0	5818
L	1G4+1G5	6.1	-1	12	1	0	2724	8.1	-3E-1	42	2	2	16896
L	L4+L5	6.3	-5	13	1	0	2816	8.1	-2E-1	44	2	2	15506
L	R3+R4	4.8	-8E-2	21	2	1	4603	7.8	-2E-1	90	2	2	46162
L	1G6+1G7	5.7	-2E+1	5	0	0	1901	8.5	2	8	1	0	3837
L	L6+L7	5.6	-5	5	1	0	2124	8.2	-1	8	0	0	3177
L	R5+R6	6.7	-2E+1	6	1	0	2310	7.9	1	11	1	0	4816

Prob. 5		Requested tolerance 1 E-4						Requested tolerance 1 E-7					
Method		sd	EE/GE	N	#F tot	#F 0	#kev	sd	EE/GE	N	#F tot	#F 0	#kev
G	1G4+1G5	4.3	1	29	0	0	19535	7.3	1	92	1	1	155629
G	L4+L5	4.3	1	30	0	0	18375	7.3	1	97	1	1	141564
G	R3+R4	4.2	1	51	1	0	42108	7.3	1	225	3	2	678032
G	1G6+1G7	5.0	1	10	0	0	7948	7.5	1	17	0	0	17543
G	L6+L7	4.9	1	10	0	0	7624	7.5	1	18	0	0	17555
G	R5+R6	4.3	1	11	0	0	7960	7.5	1	24	0	0	26478
L	1G4+1G5	4.2	3E-2	27	0	0	10898	7.3	2E-2	92	1	1	73391
L	L4+L5	4.6	3E-2	33	1	0	14145	7.4	3E-2	100	1	1	72694
L	R3+R4	4.1	2E-2	47	0	0	19580	7.2	2E-2	208	2	2	240743
L	1G6+1G7	5.3	-1E-2	11	0	0	7102	7.0	-6E-4	15	3	0	13036
L	L6+L7	5.5	1	11	1	0	7853	8.0	6E-2	19	0	0	14016
L	R5+R6	4.9	-3E-1	12	1	0	7838	7.0	-6E-3	20	1	0	14468

Prob. 6		Requested tolerance 1 E-4						Requested tolerance 1 E-7					
Method		sd	EE/GE	N	#F tot	#F 0	#kev	sd	EE/GE	N	#F tot	#F 0	#kev
G	1G4+1G5	7.7	1	9	2	2	1805	8.7	1	21	3	3	8174
G	L4+L5	7.1	1	9	2	2	1648	8.6	1	21	3	3	6918
G	R3+R4	6.3	1	12	2	2	2392	8.4	1	38	4	4	19892
G	1G6+1G7	5.6	1	5	0	0	1223	10.1	1	8	2	2	3154
G	L6+L7	5.5	1	5	0	0	1152	10.1	1	8	2	2	2922
G	R5+R6	5.7	1	6	1	1	1573	10.2	1	9	2	2	3340
L	1G4+1G5	6.8	1	9	2	2	1277	9.2	1	21	3	3	4401
L	L4+L5	6.9	1	9	2	2	1315	9.1	1	22	3	3	4383
L	R3+R4	6.9	1	12	2	2	1549	8.7	1	40	4	4	10090
L	1G6+1G7	5.6	1	5	0	0	1019	10.0	1	8	2	2	2339
L	L6+L7	5.5	1	5	0	0	1050	9.9	1	8	2	2	2345
L	R5+R6	5.7	1	6	1	1	1260	9.7	1	9	2	2	2401

6. Conclusions. It is clear that, in general, higher order results in a better performance.

In most cases local error estimation used in the higher-order methods does not seem to yield a significant decrease in the total number of kernel evaluations. Since it is less reliable than global error estimate/control, which moreover supplies a global error estimate in the endpoint for free, it seems advisable to use global error control in the stepsize strategy.

TABLE 5.4
Average performance.

Method		All problems				Selected problems			
		Tol. 1 E-4		Tol. 1 E-7		Tol. 1 E-4		Tol. 1 E-7	
		sd	#kev	sd	#kev	sd	#kev	sd	#kev
G	G4+IG	5.3	941	8.3	11583	5.0	1023	8.5	16681
G	G8+IG	7.8	579	9.0	1162	6.8	667	8.5	1524
G	1G4+1G5	5.0	9861	7.4	7072	5.3	729	7.6	4334
G	L4+L5	4.7	2220	7.7	6104	5.1	694	7.8	3934
G	R3+R4	4.9	11942	7.3	17759	5.0	1135	7.5	11243
G	1G6+1G7	5.7	743	8.4	85557	5.9	538	8.7	1355
G	L6+L7	5.8	697	8.8	1297	6.1	522	8.8	1297
G	R5+R6	5.6	650	8.1	3277	5.6	650	8.5	1472
L	1G4+1G5	5.3	902	8.0	3995	5.6	642	8.3	3183
L	L4+L5	5.5	874	8.0	3721	5.8	660	8.4	2996
L	R3+R4	5.2	1355	7.8	9942	5.6	937	8.2	8209
L	1G6+1G7	6.2	708	8.3	1421	6.4	514	8.7	1100
L	L6+L7	6.2	709	8.4	1337	6.6	540	8.8	1049
L	R5+R6	5.7	712	8.2	1606	6.2	565	8.6	1231

TABLE 5.5
Results for a problem with a discontinuous solution.

Method		Requested tolerance 1 E-4						Requested tolerance 1 E-7					
		sd	EE/GE	N	#F		#kev	sd	EE/GE	N	#F		#kev
					tot	2.5					tot	2.5	
G	G4+IG	0.1	-6E-5	6	3	2	716	1.8	-1E-3	17	22	22	5912
G	G8+IG	2.2	5E-12	18	16	16	19728	2.2	5E-12	18	21	21	22248
G	1G4+1G5	0.1	-8E-5	5	2	2	780	1.9	2	18	22	22	10405
G	L4+L5	1.8	1	18	17	17	7770	1.8	1	18	22	22	8815
G	R3+R4	0.1	2E-4	7	12	3	2506	3.0	2E+1	18	22	22	8305
G	1G6+1G7	0.1	5E-9	5	2	2	1660	0.1	5E-9	5	2	2	1660
G	L6+L7	1.8	1	20	17	17	20303	1.8	1	20	22	22	22688
G	R5+R6	0.1	2E-7	5	2	2	1440	0.1	1E-7	6	2	2	1817
L	1G4+1G5	0.1	9E-7	6	3	2	841	1.9	2E-1	18	22	22	5701
L	L4+L5	1.8	1E-1	18	17	17	4693	1.8	1E-1	18	22	22	5303
L	R3+R4	0.1	4E-5	7	3	3	801	3.0	2	18	22	22	4587
L	1G6+1G7	0.1	-5E-10	5	2	2	1341	0.1	-5E-10	5	2	2	1341
L	L6+L7	1.8	8E-2	18	17	17	10673	1.8	8E-2	18	22	22	12043
L	R5+R6	0.1	9E-8	5	2	2	1174	0.1	3E-8	6	2	2	1407

If the solution is not polynomial, then the Gauss+Iterated collocation method is the most efficient when both computational time and storage are taken into account, especially for tolerances that are not too tight.

The uniform error control was of great benefit to all other methods. The approximations were more dependable with in most cases only a slight increase in computational work.

Appendix. The results of INTSOL were derived from Kunkel [17], all other results, except those of VEl, from Schlichte [19]. VElca gives the results of VEl when the kernel has been approximated by a Chebyshev series. The number of terms, m, has been chosen as the minimal number such that the estimated error, as given by VEl, is less than the tolerance. VElex gives the results when an exact decomposition was provided. In this case no global error estimate can be calculated.

Note that for VOLCON and for IVRKXC, "#kev" stands for the total number of K- plus f-evaluations where $k(t, s, y)$ is written as $K(t-s) \cdot f(s, y)$. For VElex, "#kev" represents the number of evaluations of the right-hand side of the system of ODE's, "#Fev", times the dimension of the system, "DIM", times two. In the case of VElca, #kev = #Kev + (DIM + 2) · #Fev, where $DIM = 2m + 1$.

TABLE A.1
Results of other published codes.

Prob	Method	Tol. 1 E-4		Tol. 1 E-7		Prob	Method	Tol. 1 E-4		Tol. 1 E-7	
		sd	#kev	sd	#kev			sd	#kev	sd	#kev
1	IVRKX	7.2	510	9.0	1712	2	IVRKX	5.4	2124	7.7	5208
	ORION	3.5	923	6.5	1624		ORION	2.0	1442	4.4	2998
	VOLTEX	5.6	1124	8.7	4110		VOLTEX	2.0	3562	5.6	15092
	INTSOL	4.0	2722	4.8	127964		INTSOL	1.1	64808		
	VElca	4.1	1537	7.6	5684		VElca	5.2	2527	7.6	7550
	VElex	5.4	414	8.1	630		VElex	1.8	300	4.1	488
	IVRKXC	7.2	359	9.0	868		IVRKXC	5.4	1041	7.7	1980
	VOLCON	4.7	238	9.6	762		VOLCON	4.5	1262	8.3	2662
3	IVRKX	5.0	23763	7.6	54320	4	IVRKX	4.9	1044	7.4	2704
	ORION	6.3	42325	7.8	182344		ORION	4.7	1805	6.1	6411
	VOLTEX	5.4	76882	7.6	417014		VOLTEX	5.3	5626	8.2	22918
	VElca	2.4	19943	---	----		VElca	4.5	4157	8.5	12111
							VElex	4.9	830	6.8	1450
	IVRKXC	5.0	4237	7.6	6358		IVRKXC	4.9	582	7.4	1052
	VOLCON	6.6	2230	8.5	5370		VOLCON	4.8	366	7.3	1382
5	IVRKX	5.2	8625	8.9	21528	6	IVRKX	4.4	1029	9.7	4512
	ORION	5.6	2647	8.1	7114		ORION	6.0	402	8.7	675
	VOLTEX	5.2	120752	8.1	160910		VOLTEX	2.5	634	6.3	2648
	INTSOL	3.6	337332				INTSOL	1.7	4312	2.0	9749
	VElca	3.7	9225	---	----						
	VElex	4.3	342	6.8	730						
	IVRKXC	5.2	2167	8.9	3864						
	VOLCON	6.5	2578	8.2	5170						

TABLE A.2
Average performance.

Method	Tol. 1 E-4		Tol. 1 E-7	
	sd	#kev	sd	#kev
IVRKX	5.4	738	8.4	2216
ORION	4.7	653	6.9	1643
VOLTEX	4.3	1636	7.4	6101
INTSOL	2.6	19278	3.4	68857
VElca	4.0	1112	7.9	5420
VElex	4.1	232	6.5	373
IVRKXC	5.5	329	8.1	654
VOLCON	5.4	333	8.4	787

REFERENCES

[1] H. ARNDT (1983), *On step size control for Volterra integral equations*, in Numerical Methods of Approximation Theory, ISNM, 67, L. Collatz, G. Meinardus and H. Werner, eds., Birkhaüser Verlag, Basel-Boston-Stuttgart, pp. 9-17.

[2] H. ARNDT (1985), *An adaptive step size control for Volterra integral equations*, in Constructive Methods for the Practical Treatment of Integral Equations, ISNM, 73, G. Hammerlin and K.-H. Hoffmann, eds., Birkhaüser Verlag, Basel-Boston-Stuttgart, pp. 44–52.

[3] G. BADER AND P. KUNKEL, *An adaptive multistep method for the solution of second kind Volterra integral equations*, in preparation.

[4] J. G. BLOM AND H. BRUNNER, *Algorithm* XXX: *Discretized collocation and iterated collocation for nonlinear Volterra integral equations of the second kind*, ACM Trans. Math. Software, submitted.

[5] J. M. BOWNDS (1982), *Theory and performance of a subroutine for solving Volterra integral equations*, Computing, 28, pp. 317–332.

[6] J. M. BOWNDS AND L. APPELBAUM (1985), *Algorithm* 627: *A* FORTRAN *subroutine for solving Volterra integral equations*, ACM Trans. Math. Software, 11, pp. 58–65.

[7] H. BRUNNER (1980), *Superconvergence in collocation and implicit Runge-Kutta methods for Volterra-type integral equations of the second kind*, in Numerical Treatment of Integral Equations, ISNM, 53, J. Albrecht and L. Collatz, eds., Birkhaüser Verlag, Basel-Boston-Stuttgart, pp. 54–72.

[8] ——— (1984), *Iterated collocation methods and their discretization for Volterra integral equations*, SIAM J. Numer. Anal., 21, pp. 1132–1145.

[9] H. BRUNNER, E. HAIRER AND S. P. NØRSETT (1982), *Runge-Kutta theory for Volterra integral equations of the second kind*, Math. Comp., 39, pp. 147–163.

[10] H. BRUNNER AND P. J. VAN DER HOUWEN (1986), *The Numerical Solution of Volterra Equations*, North-Holland, Amsterdam.

[11] H. BRUNNER AND S. P. NØRSETT (1981), *Superconvergence of collocation methods for Volterra and Abel integral equations of the second kind*, Numer. Math., 36, pp. 347–358.

[12] R. P. DUNCAN (1982), *A Runge-Kutta method using variable stepsizes for Volterra integral equations of the 2nd kind*, Master's thesis, Technical Report 157/82, Department of Computer Science, University of Toronto.

[13] E. HAIRER, CH. LUBICH AND M. SCHLICHTE (1985), *Fast numerical solution of nonlinear Volterra convolution equations*, this Journal, 6, pp. 532–541.

[14] W. HOCK (1981), *An extrapolation method with step size control for nonlinear Volterra integral equations*, Numer. Math., 38, pp. 155–178.

[15] F. DE HOOG AND R. WEISS (1975), *Implicit Runge-Kutta methods for second kind Volterra integral equations*, Numer. Math., 23, pp. 199–213.

[16] H. M. JONES AND S. MCKEE (1985), *Variable step-size predictor-corrector schemes for second kind Volterra integral equations*, Math. Comp., 44, pp. 391–404.

[17] P. KUNKEL (1982), *Ein adaptives Verfahren zur Lösung von Volterra'schen Integralgleichungen zweiter Art*, Diplomarbeit, Institut für Angewandte Mathematik, Universität Heidelberg.

[18] R. K. MILLER (1971), *Nonlinear Volterra Integral Equations*, W. A. Benjamin, Menlo Park, CA.

[19] M. SCHLICHTE (1984), *Anwendung eines impliziten Runge-Kutta-Verfahrens auf Volterra'sche Integralgleichungen zweiter Art mit Faltungskern*, Diplomarbeit, Institut für Angewandte Mathematik, Universität Heidelberg.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 5, September 1987
© 1987 Society for Industrial and Applied Mathematics
011

ADAPTIVE, SELF-VALIDATING NUMERICAL QUADRATURE*

GEORGE F. CORLISS† AND L. B. RALL‡

Abstract. Integrals of a function of a single variable can be expressed as the sum of a numerical quadrature rule and a remainder term. The quadrature rule is a linear combination of function values and weights, or the integral of a Taylor polynomial, while the remainder term depends on some derivative of the integrand evaluated at an unknown point in the interval of integration. Numerical quadrature is made self-validating by using interval computations to capture both the roundoff and truncation errors made when using a given formula. Necessary derivatives can be generated automatically by using well-known recurrence relations for Taylor coefficients. In order for quadrature methods of this type to be accurate (in the sense that small intervals are produced) and efficient (to obtain results of given accuracy in a reasonably short time), an accurate scalar product and an adaptive strategy are required. The necessary scalar product and support for interval arithmetic are provided in Pascal-SC (for microcomputers) and ACRITH (for IBM 370 computers). The adaptive strategy chooses the subintervals of integration and the order of the quadrature formula in each subinterval on the basis of guaranteed, rather than estimated, information about the error of numerical integration in each subinterval. The program described here implements standard Newton–Cotes, Gaussian and Taylor series formulas for numerical integration. The handling of singularities is discussed, and comparisons are given with a standard numerical integration method.

Key words. numerical quadrature, adaptive quadrature, guaranteed error bounds, automatic differentiation, interval computation, self-validation, ACRITH

AMS(MOS) subject classifications. 65D30, 65G10

1. Requirements for automatic integration algorithms.

In [3], de Boor formulates fundamental requirements for an automatic algorithm for numerical approximation of the integral

$$If = I_{[a,b]}f = \int_a^b f(x)\,dx \tag{1.1}$$

of a function of a single real variable. Such an algorithm requires (i) finite limits of integration a, b, (ii) access to a procedure for the evaluation of $f(x)$ for x in the interval of integration, (iii) tolerances α, ρ on the desired absolute and relative error, respectively, and (iv) a limit M on the number of function evaluations allowed.

As output, the algorithm should produce an estimate I^* for the value of If which satisfies

$$|If - I^*| \leqq \max\{\alpha, \rho \cdot |If|\}. \tag{1.2}$$

The algorithm should be *efficient*, computing as few function values as possible. It should also be *reliable*, in the sense that either the desired accuracy (1.2) is guaranteed, or a message to the contrary is returned to the user with additional information about the cause of failure. As pointed out by de Boor [3], algorithms which use only values of f at a finite number of points cannot meet the above requirements in general; nevertheless, efficient and reliable automatic integration algorithms can be formulated for wide classes of integrands [4], [22], [28].

*Received by the editors May 13, 1985; accepted for publication (in revised form) April 28, 1986. This research was supported by the IBM Germany Research and Development Laboratory, Boeblingen, West Germany.

† Department of Mathematics, Statistics and Computer Science, Marquette University, Milwaukee, Wisconsin 53233.

‡ Mathematics Research Center, University of Wisconsin, Madison, Wisconsin 53705.

This paper presents automatic quadrature algorithms which attain the goals of reliability and efficiency by use of automatic differentiation and interval computation. They make use of information about the integrand on entire subintervals of integration, rather than at a discrete set of points. The results combine the self-validating algorithms of Gray and Rall [12]-[14] and the notion of adaptive quadrature [3], [4], [28]. Adaptation is carried out on the basis of guaranteed, rather than estimated, bounds for the error of the approximate integration over each subinterval. Furthermore, the given algorithm has the ability to detect and handle certain types of singularities in the integrand, and even to verify nonexistence of the integral in some cases. In the terminology of Rice [28], the method described here has the following features:

Interval Processor Component:
  Variable order rule with interval remainder to give guaranteed bounds.
Bound Estimator Component:
  Direct analysis.
Special Behavior Components:
  Polynomials.
  Roundoff level.
  Singularities in derivatives.
  Jump discontinuities.
  Removable singularities.
  x^{α}-type singularities.
  All are strictly validated.
Interval Collection Management Component:
  Ordered list.
  None discarded.

The method of this paper does not belong to the large family of 10^6 or so algorithms considered by Rice because it uses interval computation and automatic differentiation. The algorithms presented here have been implemented in an environment which supports interval computation and an accurate scalar product. The implementation is discussed in § 7. The next few sections describe the underlying methodology.

2. Self-validating evaluation of quadrature formulas. Self-validation of numerical computations is one of the basic motivations of interval analysis [2], [20], [21]. The goal is to obtain an interval which contains the desired result. In the case of the integration problem (1.1), a self-validating interval method produces an interval which is guaranteed to contain the value If of the integral. The width of this interval inclusion will depend on uncertainties in the values of the integrand and the limits of integration, the roundoff error in the actual computation, and the truncation error appropriate to the method used. All of these quantities can be estimated in a tedious way by the techniques of classical error analysis, an effort which is unnecessary in the computational environment described below. However, once an interval $J=[c, d]$ containing If is found by whatever method, one has the following approximations to If and corresponding error bounds [26]:

$$I^* = \tfrac{1}{2}(c+d), \qquad |If - I^*| \leqq \tfrac{1}{2}(d-c), \tag{2.1}$$

for absolute error, or

$$I^* = \frac{2cd}{c+d}, \qquad \left|\frac{If - I^*}{If}\right| \leqq \left|\frac{d-c}{c+d}\right| \tag{2.2}$$

for relative error, with $cd > 0$ in this case. It follows that (1.2) will be satisfied if the width $w(J) = d - c$ is small enough so that $w(J) \leqq 2\alpha$ and $w(J) \leqq \rho \cdot |c+d|$.

First, the problem of finding an interval inclusion J of If will be considered. The basic method for interval integration by use of standard formulas for numerical quadrature or Taylor series was first described by Moore [19]. To illustrate Moore's idea, consider a standard interpolatory integration *formula* of the form

$$\int_a^b f(x)\,dx = \sum_{i=1}^{n} w_i f(x_i) + c_n h \cdot \frac{f^{(p)}(\xi) h^p}{p!}, \tag{2.3}$$

where $h = b - a$, and $a < \xi < b$. A formula of type (2.3) will be called a quadrature formula of *order p on n points.* The ordinary Gauss and Newton–Cotes integration formulas follow this pattern [11].

Formula (2.3) gives the *exact* value of the integral If of functions which are differentiable p times. The only difficulty is that the value of ξ is unknown. For practical computation, it is thus customary to express (2.3) as the sum of a *rule*

$$r_n f = \sum_{i=1}^{n} w_i f(x_i) \tag{2.4}$$

of numerical integration, and a (truncation) *error term*

$$e_n f = c_n h \cdot f_p(\xi, h), \tag{2.5}$$

where

$$f_p(\xi, h) = \frac{f^{(p)}(\xi) h^p}{p!} \tag{2.6}$$

denotes the Taylor coefficient of order p in the expansion of $f(\xi + h)$. It is usual to compute $I^* = r_n f$ to approximate the value of the integral, and to estimate $e_n f$ somehow.

The algorithms discussed in this paper provide highly accurate, self-validated results. The achievement of high accuracy requires an environment which provides an accurate scalar product; to achieve self-validation requires an environment which provides interval arithmetic and monotone interval inclusions of the library functions used in the evaluation of f. We have implemented the algorithms of this paper in the environment provided by ACRITH [1] for IBM 370 computers using only the scalar product and the interval arithmetic capabilities of ACRITH, not its more controversial features [15]. The environment necessary for these algorithms is also provided by Pascal-SC for microcomputers [27], and it could be provided in a general Fortran environment using tools provided by AUGMENT [10].

Let $\mathbf{S}$ denote the set of floating-point numbers available on a particular computer, and let $\mathbf{IS}$ denote the corresponding set of closed intervals $[u, v]$, $u, v \in \mathbf{S}$. Let ∇ and Δ denote the monotone downward and upward roundings, respectively, from the real numbers to $\mathbf{S}$ [16]. If f is evaluated on an interval $X \in \mathbf{IS}$ using interval arithmetic and interval library functions, then the result is the *natural interval inclusion* $F(X)$ of f on X such that

$$f(X) = \{f(x) \mid x \in X\} \subseteq F(X) \tag{2.7}$$

[20], [21]. Let $W_i = [\nabla w_i, \Delta w_i]$, and $X_i = [\nabla x_i, \Delta x_i]$. Then the inclusion

$$r_n f \in R_n f = \sum_{i=1}^{n} W_i F(X_i) \tag{2.8}$$

is guaranteed.

An automatic, self-validating computation of the error term (2.5) requires subroutines for the generation of the Taylor coefficients $f_p(x, h)$ of f. These use well-known

recurrence relations for the arithmetic operations and library functions used to evaluate f [9], [20], [21], [24]. The computer time required to calculate $f_0(x, h)$, $f_1(x, h), \cdots, f_p(x, h)$ is about

$$t = \alpha p^2 + \beta p + \gamma \tag{2.9}$$

units of time [7], where α depends on the number of multiplications, divisions, and calls to library functions in the computation of f, and β depends on the number of additions and subtractions required. Let $X = [a, b]$, $H = [\nabla h, \Delta h]$ and $C_n = [\nabla c_n, \Delta c_n] \in$ **IS**. (It is assumed that $a \leqq b$; the contrary case can be handled easily.) The interval evaluation of exactly the same recurrence relations as for $f_k(x, h)$ yields the corresponding interval inclusions $F_0(X, H), \cdots, F_p(X, H)$ such that

$$f_k(x, h) \in F_k(X, H), \qquad k = 0, 1, \cdots, p, \tag{2.10}$$

for all $x \in X$ and $h \in H$ [20], [21]. Thus, the desired interval inclusion

$$e_n f \in E_n f = C_n H \cdot F_p(X, H) \tag{2.11}$$

can be computed automatically. It follows from the above that

$$If \in R_n f + E_n f = [c, d], \tag{2.12}$$

a formula for automatic, self-validating numerical quadrature. If f is a polynomial of degree $p-1$ or less, then $F_p(X, H) \equiv [0, 0]$, so that $If \in R_n f$, and the width of $[c, d]$ depends only on the roundoff error in the calculation of $r_n f$.

Instead of splitting the numerical integration formula (2.3) into a rule of numerical integration (2.4) and an error term (2.5), it is helpful to represent it as the scalar product of the augmented *weight vector*

$$\mathbf{W} = (W_1, \cdots, W_n, C_n H), \tag{2.13}$$

which is independent of the integrand f and depends only on the specific formula (2.3) used, and the augmented *function-value vector*

$$\mathbf{F} = (F(X_1), \cdots, F(X_n), F_p(X, H)). \tag{2.14}$$

Thus,

$$If \in \mathbf{W} \cdot \mathbf{F} = J. \tag{2.15}$$

This allows the computation to use recently developed methods for highly accurate calculation of real and interval scalar products [17]. This results in a considerable decrease in width due to roundoff error in the computed value of J, especially for large n.

The integration formula (2.3) can be interpreted as a *single-panel* rule, or as a *multipanel* rule, meaning that a simpler formula on m points is applied k times to the corresponding number of subintervals of $X = [a, b]$, with $n \leqq km$. As the width ν of the longest subinterval approaches zero, the width of the error term is reduced by a factor of at least $(\nu / w(X))^p$, and by sharper bounds for $f^{(p)}$ on each subinterval. The result is a narrowing of the interval J. Roundoff error in adding a number of interval inclusions of one-panel rules is reduced considerably by expressing the result as a single scalar product.

3. Taylor series methods. The seminal paper by Moore [19] provides the basis for self-validating numerical integration by the use of Taylor series, although the techniques presented there are directed toward the solution of the initial-value problem for ordinary differential equations. For numerical integration, Taylor series are more appropriate than fixed quadrature formulas for interval-valued endpoints of intervals of integration,

as will be discussed in § 6. Furthermore, Taylor series support the rigorous approach to automatic recognition and treatment of singularities described in § 5.

We use automatic differentiation to generate a Taylor polynomial and its interval remainder term. This provides a self-validating calculation of the desired integral. Assume that the integrand f has $p \geqq 0$ derivatives on the interval of integration $X = [a, b]$. Then expand f at the midpoint $c = (a+b)/2$ of X. For $n \leqq p$, and ξ between c and x,

$$\begin{aligned} f(x) &= f(c)+f'(c)(x-c)+\cdots+f^{(n-1)}(c)\frac{(x-c)^{n-1}}{(n-1)!}+f^{(n)}(\xi)\frac{(x-c)^n}{n!} \\ &\in f(c)+f'(c)(x-c)+\cdots+f^{(n-1)}(c)\frac{(x-c)^{(n-1)}}{(n-1)!}+F^{(n)}(X)\frac{(x-c)^n}{n!}, \end{aligned} \tag{3.1}$$

where $F^{(n)}$ is an interval inclusion of $f^{(n)}$. Let

$$g(x) = f(c)(x-c)+f'(c)\frac{(x-c)^2}{2!}+\cdots+f^{(n-1)}(c)\frac{(x-c)^n}{n!} \tag{3.2}$$

be an indefinite integral of the Taylor polynomial of degree $n-1$ of f. Let $H =$ interval hull $(\nabla(c-a), \nabla(b-c), \Delta(c-a), \Delta(b-c))$. Then,

$$\int_a^b f(x)\,dx \in g(x)\Big|_a^b + F^{(n)}(X)\frac{(x-c)^{n+1}}{(n+1)!}\Big|_a^b \subseteq J_n, \tag{3.3}$$

where

$$J_n = 2\sum_{\substack{i=0 \\ i \text{ even}}}^{n-1} F^{(i)}(c)\frac{H^{i+1}}{(i+1)!}+\begin{cases} F^{(n)}(X)\dfrac{H^{n+1}}{(n+1)!}-F^{(n)}(X)\dfrac{H^{n+1}}{(n+1)!} & \text{for } n \text{ odd}, \\ 2F^{(n)}(X)\dfrac{H^{n+1}}{(n+1)!} & \text{for } n \text{ even}. \end{cases} \tag{3.4}$$

J_n can be computed as a single scalar product directly from the automatically generated interval Taylor coefficients $F_k(C, H)$ and $F_k(X, H)$ of f, $k = 0, 1, \cdots, n \leqq p$.

Since

$$If = \int_a^b f(x)\,dx \in J_n, \qquad n = 0, 1, \cdots, p, \tag{3.5}$$

the *intersection principle* [12], [13] can be used. One has

$$If \in \bigcap_{n=0}^{p} J_n. \tag{3.6}$$

This intersection can be calculated as the corresponding interval Taylor coefficients of the integrand are generated:

$$\begin{aligned} I_0 &= J_0 \qquad \text{(a Riemann sum)}, \\ I_n &= I_{n-1} \cap J_n, \qquad n = 1, 2, \cdots, p. \end{aligned} \tag{3.7}$$

This provides a means to determine the highest useful term of the Taylor expansion, since the calculation can be terminated when effective decrease in the widths of the intervals I_n ceases, or when the desired tolerance is met.

4. Adaptive strategies. The computer program INTE [13] implemented automatic, self-validating numerical quadrature using Newton–Cotes and Gaussian integration formulas as discussed in § 2. However, there was no attempt to address the problem of efficiency. In order to satisfy the accuracy criterion (1.2), if possible, INTE simply

divided the interval of integration into a sufficient number of equal subintervals [13], [23]. By contrast, popular numerical integration packages such as CADRE [4] and QUADPACK [22] use information obtained about the behavior of the integrand to attempt to reduce the number of function evaluations to a minimum. The method given here uses similar strategies, except that estimates of the error based on evaluation of the integrand at a finite set of points are replaced by guaranteed bounds based on interval computations.

Adaptive strategies fall into the categories of *order* adaptation strategies which select the formula used in each subinterval, and *subinterval* adaptation strategies which determine how the original interval of integration is broken up into subintervals. Order adaptation is somewhat simpler than subinterval adaptation, and will be considered first. For methods based either on standard quadrature formulas or Taylor series, an *order zero* formula is the interval Riemann sum $F(X) \cdot w(X)$, which always contains $\int_X f(x)\,dx$ [5].

For the remainder of this section, assume that $X \in \mathbf{IS}$. The case of endpoints which are not machine numbers is discussed in § 6. Consider a suite of numerical integration formulas of the form (2.3). Specifically, suppose that the given rules are of order i using n_i points, for $i = 0, 1, \cdots, k$. Several strategies could be employed to select the order to be used. For example, once the interval Taylor coefficients $F_i(X, H)$ have been formed, then the order $p \leqq k$ of the most accurate rule can be chosen to be the value of i for which the width of the error term

$$E_{n_i} f = C_{n_i} H \cdot F_i(X, H) \tag{4.1}$$

is minimized.

Alternatively, the actual approximate integrals J_i could be examined. However, this requires more computation than use of the error terms alone. Using more nodes usually yields a narrower interval at the "cost" of more function evaluations. To study the trade-off, suppose that μ denotes the maximum value of the width $w(F(C))$ of the interval evaluation of f at a node C of the integration formula being used. The total cost of the formula of order i is proportional to $n_i \cdot w(J_i)$. The width $w(J_i)$ can be estimated by $\mu + w(E_{n_i} f)$, and thus i can be chosen as the minimizer of

$$m(i) = \min \{w(J_0), n_i \cdot (\mu + w(E_{n_i} f))\}. \tag{4.2}$$

Thus, more nodes are used in a given subinterval only if a significant reduction in width of the approximate integral results. It should also be noted that a given integration rule can be used in several integration formulas having remainder terms of different orders. For example, Stroud and Secrest [29] give error terms for Gaussian integration rules on n nodes which have orders $1 \leqq p \leqq 2n$. In certain cases, the error terms corresponding to $p < 2n$ can be narrower (for example, for highly oscillatory integrands).

In the case of Taylor series, the intersection procedure (3.6) can result in approximations I_n which are considerably better than J_n given by (3.4). The intervals I_n are monotonically decreasing in width, so the increase in accuracy has to be balanced against the cost in time (2.9) of generating more terms of the series. The constants α, β and γ in (2.9) can always be determined for a given integrand, so the strategy corresponding to (4.2) is based on the function

$$\theta(n) = w(I_n) \cdot (\alpha n^2 + \beta n + \gamma). \tag{4.3}$$

No additional terms of the Taylor series are generated when

(i) $I_{n-2} = I_{n-1} = I_n$ (bounds are not sharpened),

(ii) $\theta(n-2) \leqq \theta(n)$ (cost-effective order has been reached), or
(iii) $n = p$ (maximum number of derivatives has been used).

Because of the difference between the remainder terms in (3.4) for odd and even n, it is prudent to calculate at least one extra value of J_n. However, the conditions above guarantee that no more than two series terms will be computed beyond the one which gives the narrowest I_n.

Another important strategy for order adaptation involves the case that the integrand has singularities in a certain derivative in $[a, b]$. This will be discussed more fully in the next section. If a certain derivative cannot be evaluated, this is detected, and the method is restricted to rules which use only the derivatives which can be evaluated.

The strategy for subinterval adaptation retains all subintervals. At each step, the subinterval which makes the largest contribution to the width of J is processed by breaking it into further subintervals. This processing continues until

(i) $w(J)$ is small enough to satisfy the accuracy requirement (1.2),
(ii) the noise inherent in function evaluation limits further reduction of $w(J)$, or
(iii) more than the maximum number M of function evaluations have been performed.

The second termination criterion in this list is particularly important. If the noise in the function evaluation is large relative to the accuracy requested, eventually the width of the truncation error $E_n f$ is made so small that it adds nothing to the width of the rule $R_n f$ alone. At this point, further increase in accuracy is not possible. Many standard methods cannot recognize when this point has been reached because they lack the guaranteed bounds provided by the interval computation. Consequently, they continue to work after the calculation should have been terminated. Malcolm and Simpson [18] observe that the strategy of processing the worst subinterval results in local errors of roughly equal magnitude. Rice [28] calls this an ordered list interval collection management component and lists some advantages and disadvantages.

5. Treatment of singularities. Among the quadrature routines which provide estimates for If, the more successful ones have special provisions to handle and perhaps recognize certain types of singularities. The QUADPACK package of programs [22], for example, can handle integrals of the form

$$\int_a^b (x-a)^\alpha (b-x)^\beta v(x) f(x)\, dx, \tag{5.1}$$

where $\alpha, \beta > -1$ and $v(x) = 1, \log(x-a), \log(b-x)$, or $\log(x-a)\log(b-x)$, *provided the user* supplies α, β, and the form of v. CADRE [4] attempts to detect and verify the presence of jump discontinuities or x^α-type singularities in the integrand. The algorithms described here can be used to recognize and handle automatically

(1) singularities in derivatives of f,
(2) some removable singularities,
(3) jump discontinuities, and
(4) some algebraic singularities at endpoints.

Since guaranteed bounds are computed for If, the task of such a program is more difficult than for methods which yield only an estimate. For example, Gaussian quadrature can be used in the neighborhood of an endpoint singularity, because the integrand need not be evaluated at the endpoint. However, the integrand or one of its derivatives must be evaluated on the entire interval in order to give guaranteed bounds. Hence, the price of the guarantee is a restriction on the applicability of the program.

The program discussed here either verifies that the integrand is in its domain of applicability, or it notifies the user if it cannot deliver guaranteed bounds.

5.1. Singularities in derivatives of f. The integrand f may not have enough derivatives for some rules which the integration program could apply. In the process of automatic generation of interval Taylor coefficients on an interval X, the nonexistence of a derivative of f beyond a certain order is detected. As long as f itself can be evaluated on the entire interval of integration, the order adaptation strategy handles singularities in the derivatives of f. For example, the only formula which can be applied to $\int_0^4 \sqrt{x}\,dx$ is the Riemann sum

$$\int_0^4 \sqrt{x}\,dx \in [\min_{[0,4]} \sqrt{x}, \max_{[0,4]} \sqrt{x}] \cdot 4 = [0, 8]. \tag{5.2}$$

During the subinterval adaptation, the subinterval containing zero requires a Riemann sum, and thus is frequently selected for further processing. All other subintervals can be processed using higher-order formulas. This strategy applies to singularities in f' which occur anywhere in the interval of integration, and not just at endpoints. It also works for singularities in higher derivatives, whose presence might not even be known to the user. Once a singularity of this type has been confined to a sufficiently small subinterval, a highly accurate, self-validated result is possible.

5.2. Jump discontinuities. Jump discontinuities in the integrand appear to the program as singularities in f'. The Riemann sum formula is to be applied on any subinterval which contains a jump. The width of the result leads to frequent selection of this subinterval for further processing. In effect, the jump is located by a bisection algorithm. This behavior is similar to CADRE [4]. Upon recognizing a jump, CADRE subdivides the interval and uses a low-order rule.

5.3. Some removable singularities. The Taylor series method permits handling of some removable singularities. Consider the problem

$$If = \int_0^\pi \frac{\sin x}{x}\,dx. \tag{5.3}$$

The integrand cannot be evaluated directly on any interval containing zero because it has a removable singularity at $x = 0$. However, f can be expanded at $x = 0$ using l'Hôpital's rule, which can be applied automatically [9]. A short Taylor series with remainder for f at 0 is sufficient to bound If in a neighborhood of zero, and the rest of the interval of integration is processed in the normal manner.

5.4. Some algebraic singularities at endpoints. Suppose that the integrand has the form

$$f(x) = (b-x)^{-s}\phi(x), \tag{5.4}$$

where $\phi(x)$ is analytic at $x = b$. A singularity at a can be handled similarly. If c is chosen closer to b than any other singularity, then the series for f expanded at $x = c$ is asymptotic to the series for $v(x) = (b-x)^{-s}$, also expanded at c. The Taylor coefficients $v_i = v_i(b, h)$ of v satisfy the recurrence relation

$$\begin{aligned} v_{i+1} &= v_i\left(1 + \frac{s-1}{i}\right)\frac{h}{R_c}, \\ s &= i\left(\frac{v_{i+1}}{v_i}\,\frac{h}{R_c} - 1\right) + 1, \end{aligned} \tag{5.5}$$

where $h = b - c$, and $R_c = b - c$ is the radius of convergence of the series. If f cannot be evaluated on $[a, b]$, then one attempts to find constants $\underline{K}$, $\bar{K}$, $\underline{s}$, and $\bar{s}$ such that

$$\underline{K}(b-x)^{-\underline{s}} \leqq f(x) \leqq \bar{K}(b-x)^{-\bar{s}} \tag{5.6}$$

for x near b. If such constants can be found and the bounds of (5.6) validated, then we proceed. If $\underline{s} > 1$, then the algorithm can guarantee that *If* does not exist. We know of no other numerical quadrature routine which can *validate* nonexistence of an integral. If $\bar{s} < 1$, then

$$\frac{\underline{K}}{1-\underline{s}}(b-b')^{1-\underline{s}} \leqq \int_{b'}^{b} f(x)\,dx \leqq \frac{\bar{K}}{1-\bar{s}}(b-b')^{1-\bar{s}}. \tag{5.7}$$

These bounds can be made as tight as desired by taking b' close enough to b. The interval $[b', b]$ is placed on the list of subintervals, and processing continues on the subinterval $[a, b']$. If $\underline{K}$, $\bar{K}$, $\underline{s}$, and $\bar{s}$ cannot be found, or if $\underline{s} < 1 \leqq \bar{s}$, then the integrand cannot be handled by this method, and a message to that effect is issued.

These methods for recognizing and handling certain singularities extend the domain of applicability of the program to include many integrands which arise in applications. However, their usefulness is somewhat limited. For one thing, the location of the singularity must be known in advance, or guessed. For popular sets of test problems, guessing one or both endpoints usually works. For real problems, locations of singularities may be unknown. However, the method given here validates correct guesses.

The endpoints to be investigated for the presence of singularities must be machine numbers, not intervals. If the width of an interval-valued endpoint is even the same as the distance between two adjacent machine numbers, then the problem contains integrands which are unbounded on a set of positive measure. For example, the problem

$$\int_0^{\pi} \sqrt{\pi - x}\,dx = \tfrac{2}{3}\pi^{3/2} \tag{5.8}$$

could be presented to the computer as

$$\int_0^{[\nabla\pi, \Delta\pi]} \sqrt{[\nabla\pi, \Delta\pi] - x}\,dx. \tag{5.9}$$

Although the original problem (5.8) is well behaved, the interval integral in (5.9) contains integrals such as

$$\int_0^{\Delta\pi} \sqrt{\nabla\pi - x}\,dx, \tag{5.10}$$

for which the integrand is not defined. This example can be handled by a special technique which recognizes π or by a special technique for evaluating the square root. However, it illustrates a difficulty which cannot be avoided in general.

If a singularity is in the interior of the interval of integration, then the subinterval adaptive strategy determines its location to within one or two machine numbers using a bisection-like algorithm. From this, its contribution to *If* could be estimated, but not guaranteed. The possibility that the integrand is unbounded on a set of positive measure cannot be eliminated, and a validated answer cannot be computed. If the user knows that the singularity occurs at a machine number, then the integral should be calculated as the sum of two integrals, with the singularity at one endpoint of each.

6. Extension to interval values. The definition of the integral (1.1) has been extended to arbitrary interval-valued integrands f [5], [25]. For smooth, real-valued functions such as those considered in § 2, the interval integral and the Riemann integral coincide. The concept of interval integration is useful in connection with integrands which have jump discontinuities or singularities of various kinds.

The definition given in [5] can be extended to the case that the limits of integration are interval-valued:

$$\int_A^B f(x)\,dx = \left\{ \int_a^b f(x)\,dx \;\middle|\; a \in A,\ b \in B \right\} \tag{6.1}$$

for A and $B \in \mathbf{IR}$, the set of all finite intervals with real endpoints. Interval-valued endpoints of integration are used for real numbers which cannot be represented exactly in **S** or for limits of integration whose exact values are uncertain.

Similar considerations apply to interval-valued integrands. The case which usually arises in practice is that the integrand f is a function not only of the independent variable x, but also several parameters $c_1, c_2, \cdots, c_m$ whose values are uncertain.

The meaning of tolerance for problems with interval-valued endpoints requires some clarification. Consider the problem

$$\int_0^{[0,1]} \frac{dx}{1+x^2} = \left[\int_0^0 \frac{dx}{1+x^2}, \int_0^1 \frac{dx}{1+x^2} \right] = \left[0, \frac{\pi}{4} \right]. \tag{6.2}$$

If we compute $J = [-0.004, 0.79]$, for example, then $w(J) = 0.794$, although the estimate for each endpoint is in error by less than 0.005. Hence, a requested tolerance must be large enough to accommodate the uncertainties which are inherent in the problem being solved.

The cases that one or both endpoints of integration are nondegenerate intervals in **IS** will now be considered. One strategy for handling interval-valued endpoints is to allow the uncertainties in the locations of the endpoints to be carried over into uncertainties in the locations of the nodes, as in INTE [13]. For example, using a one-panel Simpson's rule on 4-decimal digit machine gives

$$\begin{aligned} \int_{[0,0.1]}^{[3.1,3.2]} f(x)\,dx \in{}& \frac{(B-A)}{6}\left(F([0,0.1]) + 4F([1.55,1.65]) + F([3.1,3.2])\right) \\ & - \frac{(B-A)^5}{2880} F^{iv}([0,3.2]), \end{aligned} \tag{6.3}$$

where $A = [0, 0.1]$, $B = [3.1, 3.2]$. For $f(x) = \sin x$, (6.3) yields [1.880, 2.214] using the accurate scalar product, while the correct answer is [1.994, 2.000].

One could also rewrite

$$\int_A^B f(x)\,dx = \int_A^0 f(x)\,dx + \int_0^B f(x)\,dx = A\int_1^0 f(At)\,dt + B\int_0^1 f(Bt)\,dt \tag{6.4}$$

to replace a problem with interval-valued endpoints with a problem with an interval-valued integrand. This decomposition gives [1.936, 2.069] for the example of (6.3).

A better strategy is to concentrate the uncertainty in the computations in the same place as the uncertainty in the problem: at the ends.

$$\begin{aligned} \int_{[0,0.1]}^{[3.1,3.2]} f(x)\,dx = {}& \int_{[0,0.1]}^{0.1} f(x)\,dx + \int_{0.1}^{3.1} f(x)\,dx + \int_{3.1}^{[3.1,3.2]} f(x)\,dx \\ & \in [1.993, 2.002]. \end{aligned} \tag{6.5}$$

As the widths of the endpoints increase, the advantage of this strategy becomes more pronounced.

The general case (6.1) can be expressed in terms of integrals in one of six cases, according to whether A and B are (i) disjoint, (ii) overlapping, or (iii) one is contained in the other. Each of the six cases can be computed as a sum of integrals of four types which are distinguished by the nature of their endpoints.

Using the same notation as in § 3, $g(x)$ denotes an indefinite integral (3.2) of the Taylor polynomial of degree $n-1$ of f. The one-panel form of the various integration formulas are then:

Type RR:

$$\int_a^b f(x)\,dx$$

is handled by Gaussian, Newton-Cotes, or Taylor polynomial quadrature as described in §§ 3-4.

Type RI:

$$\int_a^{[a,b]} f(x)\,dx \subseteq g(x)\Big|_a^{[a,b]} + F^{(n)}([a,b])\frac{(x-c)^{n+1}}{(n+1)!}\Big|_a^{[a,b]}. \tag{6.6}$$

Type IR:

$$\int_{[a,b]}^b f(x)\,dx \subseteq g(x)\Big|_{[a,b]}^b + F^{(n)}([a,b])\frac{(x-c)^{n+1}}{(n+1)!}\Big|_{[a,b]}^b. \tag{6.7}$$

Type II:

$$\int_{[a,b]}^{[a,b]} f(x)\,dx \subseteq g(x)\Big|_{[a,b]}^{[a,b]} + F^{(n)}([a,b])\frac{(x-c)^{n+1}}{(n+1)!}\Big|_{[a,b]}^{[a,b]}. \tag{6.8}$$

Integrals of types RI, IR, and II can be handled by Gauss or Newton-Cotes formulas with interval-valued nodes, but this leads to relatively wide interval bounds. The method of Taylor series outlined in § 4 applies as easily to the cases of one or both endpoints interval-valued as to RR type integration. Hence, the program uses Taylor polynomials for integrals of types RI, IR, and II, even if the user has chosen Gauss or Newton-Cotes formulas for use on type RR subintervals. Each of the quadrature formulas given by (6.6), (6.7) and (6.8) uses intersections of subsequent estimates and order adaptation as in the RR algorithm.

Formulation of the multipanel forms for the above types requires some care. A detailed discussion can be found in [6] or [8].

In the general case of the integral (6.1), the allowable tolerances and the maximum number of function evaluations are apportioned automatically by the program among three distinct, independent integrations in proportion to the width of the respective subintervals. If the integral on the first subinterval is narrower than its share of the total tolerance requires, then the tolerances on the other subintervals are relaxed so that the total tolerance can be met more efficiently. On the other hand, if the integral on the first subinterval is a little too wide, then the integrals on the remaining subintervals can sometimes be computed accurately enough that the total tolerance can still be satisfied.

7. Implementation details. It follows from the above discussion that realization of adaptive, self-validating quadrature routines on a computer requires that the following features be supported:

(i) Interval computation (arithmetic operations and standard library functions).

(ii) Subroutines for automatic generation of Taylor coefficients.
(iii) Accurate scalar product of interval vectors (to minimize width due to roundoff error).

The program INTE [13] was written in FORTRAN for the Sperry 1100, on which only (i) [30] and (ii) were supported. As mentioned above, INTE performs self-validating numerical integration without adaptation. The microcomputer language Pascal-SC [27] and the ACRITH package for the IBM 370 series of computers support (i) and (iii), to which the routines (ii) have been added. The implementation described in [9] for Pascal-SC is immediately adaptable to ACRITH. The program described here is written in FORTRAN for use with ACRITH. In particular, the following operators and library functions are supported:

+	SIN	SINH	EXP
−	COS	COSH	LOG = LN
*	TAN	TANH	LOG10
/	COTAN = COT	COTANH = COTH	CHAR
**	ASIN	ASINH	
ABS	ACOS	ACOSH	
SQR	ATAN	ATANH	
SQRT	ACOTAN = ACOT	ACOTNH = ACOTH	

The Taylor series terms $F_n(C, H)$ and $F_n(X, H)$ are calculated by using the well-known recurrence relations [20], [21], [24] for the operators and functions given above. In this application, the "point" of expansion and the stepsize are interval-valued to give the desired inclusions, but the recurrence relations remain the same.

Given an integrand of the type considered, for example,

$$f(x) = \frac{4}{1+x^2}, \tag{7.1}$$

the program first parses it into a *code list* [24]:

Operator	Operand 1	Operand 2	Result
SQR	x (=Temp1)		Temp4
+	1 (=Temp2)	Temp4	Temp5
/	4 (=Temp3)	Temp5	F

In order to compute the series for f expanded at C with stepsize H, this code list is interpreted to obtain the sequence of calls

```
Temp1 := (C, H)   {The series for x.}
Temp2 := (1)      {Constant series.}
Temp3 := (4)      {Constant series.}
Call ITSQR(Temp1, Temp4)
Call ITADD(Temp2, Temp4, Temp5)
Call ITDIV(Temp3, Temp5, F).
```

The result F is an array which contains the interval-valued series for f and an indication of how many terms were computed. For example, the subroutine ITSQR(U, V) computes the series for $V = U^2$ by means of the recurrence relations

$$V_i = \begin{cases} 2 * \sum_{j=1}^{(i-1)/2} U_j * U_{i-j+1} + \mathrm{ISQR}(U_{(i+1)/2}), & i \text{ odd}, \\ 2 * \sum_{j=1}^{i/2} U_j * U_{i-j+1}, & i \text{ even}, \end{cases} \tag{7.2}$$

where $V_i = V_i(C, H)$ and $U_i = U_i(C, H)$ denote the ith Taylor coefficients of U and V, respectively [24]. ISQR is the interval square operator. The parsing and interpretation at runtime described above is unnecessary in Pascal-SC, because the compiler generates the required code [9].

The algorithms used depend on whether the endpoints of the interval X of integration are machine numbers or intervals. The basic algorithm is for the case $X = [a, b] \in \mathbf{IS}$, and is called the RR (Real-Real) algorithm:

1. Compute the integral on $[a, b]$;
2. Add $[a, b]$ to the list of subintervals;
3. Loop
4. Find the subinterval on which the width of the integral is largest;
5. Bisect it;
6. Compute the integral on the left subinterval;
7. Add the left subinterval to the list;
8. Compute the integral on the right subinterval;
9. Add the right subinterval to the list;
10. Compute the integral on $[a, b]$ by summing the integrals on all the subintervals;
11. Exit when accuracy tolerance is met;
12. Exit with warning when
13. no further improvement in accuracy is possible,
14. or M function evaluations are exceeded;
15. End loop.

The algorithms for integrals of types RI, IR, and II are similar, except that they use the inclusions given by (6.6)-(6.8). They are described in more detail in [6].

8. Numerical examples. We give eight examples to illustrate the accuracy, reliability, and speed of the program. All computations were done in double precision on an IBM 4341 computer, using calls to ACRITH routines for all necessary interval calculations and scalar products.

These examples present a preliminary comparison with the routine QAGS from QUADPACK [22]. They are intended to give an indication of the relative performance of the program reported here; a more comprehensive test is in progress. It must be remembered that QAGS provides an *estimate* for *If*, while the self-validating quadrature provides an interval on which *If* is *guaranteed* to lie. The intervals shown are those computed by self-validating quadrature using adaptive Gaussian quadrature with at most 9 points per subinterval. The absolute and relative error tolerances requested were 0.0.

Example 1.

$$\int_{[0,0.1]}^{[3.1,3.2]} \sin x \, dx \in [1.9932, 2.0013].$$

This tests the handling of interval-valued endpoints as shown in (6.5). The interval computed agrees with the one given there. QAGS was not intended to handle approximate endpoints. For comparison, QAGS was asked to compute the easier problem $\int_{0.05}^{3.15} \sin x \, dx$.

Example 2.

$$\int_0^1 B \sin(Bx) - A \sin(Ax)\, dx \in [1.9140, 2.0811],$$

where $A = [0, 0.1]$ and $B = [3.1, 3.2]$. This tests the handling of the problem with interval-valued endpoints in Example 1 as a problem with an interval-valued integrand as shown in (6.4). These numerical results show that a narrower inclusion is obtained using (6.5). QAGS was not intended to handle problems with approximate integrands, so it was asked to compute the corresponding problem using the midpoints of A and B, a problem which it found to be very hard.

Example 3.

$$I = \int_{0.6}^{0.7} \frac{1}{1-x}\, dx \in 0.28768\ 20724\ 5178^{11}_{08}.$$

Interval bounds for the answer I can be computed in three ways:

$$I = \log(1-0.6) - \log(1-0.7), \tag{8.1}$$

$$I = \log\left(\tfrac{4}{3}\right), \tag{8.2}$$

or by adaptive, self-validating quadrature.

Neither 0.6 nor 0.7 are machine numbers, so they are converted to intervals whose endpoints are adjacent machine numbers. All three methods give the decimal interval shown, but the resulting hexadecimal intervals contain 20, 14, and 18 machine numbers, respectively. That is, the program is capable of accuracies comparable to evaluation of the analytic expression for the answer.

Example 4.

$$\int_0^4 \sqrt{x}\, dx = \tfrac{16}{3} \in 5.33333\ 3333^{45}_{11}.$$

This example illustrates the order adaptation required to handle the nonexistence of $f'(0)$ as discussed in § 5.1. The program was stopped when it had used 1090 effective function evaluations on 24 subintervals. Most of the subintervals were clustered near the origin. Away from 0, as many as 19 series terms were used. This illustrates the need for the improved technique for handling endpoint singularities discussed in § 5.4. QAGS was able to compute a much more accurate estimate because it was not concerned by the singularity at 0.

Example 5.

$$\int_0^1 f(x)\, dx \quad \text{where } f(x) = \begin{cases} 0, & x < 0.3, \\ 1, & x \geqq 0.3. \end{cases}$$

This example illustrates integration of a simple discontinuous function.

Example 6.

$$\int_{0.3}^1 1\, dx.$$

This is the same as Example 5, except that the integrator need not locate the singularity. The performance of both programs is improved by a factor of about 10 when user recognizes the presence of a discontinuity. The self-validating quadrature spends most of its time processing a very small interval containing 0.3.

Example 7.

$$\int_0^1 (x-2)\,dx = [-1.5, -1.5] \quad \text{(exactly)}.$$

This example tests the handling of a low degree polynomial which can be integrated exactly.

Example 8.

$$\int_0^1 \frac{dx}{1+x^4} \in 0.86697\,29873\,39910\,46_{09}^{12}.$$

This is a smooth function which self-validating quadrature handles quickly. Further, the estimate computed by QAGS is 0.86697 29873 39910 46 which lies *outside* the self-validated interval. (See Table 1.)

Error codes for self-validating quadrature:

0 Normal return. No errors detected.

63 Requested tolerance relaxed. Noise in function evaluation prevents reaching the requested tolerance.

64 Requested tolerance relaxed. Reached the maximum number of function evaluations.

Table 2 shows the performance of QAGS on the same test problems. The ratio of times shown is the time required by self-validated quadrature divided by the time required by QAGS.

TABLE 1
Performance of self-validating quadrature.

Test	Error code	Absolute error	Function evaluations	Subintervals	CPU seconds
1	64	8.0E−03	652	35	2.025
2	63	1.7E−01	55	1	0.648
3	63	2.5E−16	69	5	0.298
4	64	3.3E−10	1069	24	3.271
5	63	4.2E−17	90	59	1.965
6	63	4.2E−17	6	3	0.132
7	0	0.0	3	1	0.072
8	63	2.6E−16	263	6	1.488

TABLE 2
Performance of QUADPACK's QAGS.

Test	Error code	Estimated error	Function evaluations	CPU seconds	Ratio of times
1	2	2.2E−14	21	0.035	58.0
2	2	1.1E−09	1785	17.816	0.03
3	2	3.2E−15	21	0.055	5.37
4	4	6.5E−14	651	1.295	2.53
5	4	8.4E−15	819	0.493	3.99
6	2	7.8E−15	21	0.004	30.7
7	2	1.7E−14	21	0.021	3.44
8	2	9.6E−16	483	2.106	0.71

Error codes for QAGS:

0 Normal return. No errors detected.

2 Tolerance relaxed. Roundoff error detected.

4 Algorithm does not converge.

REFERENCES

[1] ACRITH *High Accuracy Subroutine Library: General Information Manual*, IBM publications, GC33-6163, Boeblingen, West Germany, 1985.

[2] G. ALEFELD AND J. HERZBERGER, *Introduction to Interval Computation*, translated by J. Rokne, Academic Press, New York, 1983.

[3] CARL DE BOOR, *On writing an automatic integration algorithm*, in Mathematical Software, John R. Rice, ed., Academic Press, New York, 1971, pp. 201-209.

[4] ———, CADRE: *An algorithm for numerical quadrature*, in Mathematical Software, John R. Rice, ed., Academic Press, New York, 1971, pp. 417-449.

[5] OLE CAPRANI, KAJ MADSEN AND L. B. RALL, *Integration of interval functions*, SIAM J. Math. Anal., 12 (1981), pp. 321-341.

[6] G. F. CORLISS, *Computing narrow inclusions for definite integrals*, Proc. Conference on Computer Arithmetic, Scientific Computation, and Programming Languages, Karlsruhe, 1985.

[7] G. F. CORLISS AND Y. F. CHANG, *Solving ordinary differential equations using Taylor series*, ACM Trans. Math. Software, 8 (1982), pp. 114-144.

[8] G. F. CORLISS AND L. B. RALL, *Adaptive, self-validating numerical quadrature*, MRC Technical Summary Report No. 2815, University of Wisconsin-Madison, 1985.

[9] ———, *Automatic generation of Taylor coefficients in* Pascal-SC: *Basic applications to ordinary differential equations*, Trans. First Army Conference on Applied Mathematics and Computing, U.S. Army Research Office, Research Triangle Park, NC, 1984, pp. 177-209.

[10] F. D. CRARY, *A versatile precompiler for nonstandard arithmetics*, ACM Trans. Math. Software, 5 (1979), pp. 204-217.

[11] P. J. DAVIS AND P. RABINOWITZ, *Methods of Numerical Integration*, 2nd edition, Academic Press, New York, 1984.

[12] JULIA H. GRAY AND L. B. RALL, *A computational system for numerical integration with rigorous error estimation*, Proc. 1974 Army Numerical Analysis Conference, U.S. Army Research Office, Research Triangle Park, NC, 1974, pp. 341-355.

[13] ———, INTE: A UNIVAC 1108/1110 *program for numerical integration with rigorous error estimation*, MRC Technical Summary Report No. 1428, University of Wisconsin-Madison, 1975.

[14] ———, *Automatic Euler-Maclaurin integration*, Proc. 1976 Army Numerical Analysis and Computers Conference, U.S. Army Research Office, Research Triangle Park, NC, 1976, pp. 431-444.

[15] W. KAHAN AND E. LEBLANC, *Anomalies in the* IBM ACRITH *package*, IEEE Proc. 7th Symposium on Computer Arithmetic, 1985.

[16] U. W. KULISCH AND W. L. MIRANKER, *Computer Arithmetic in Theory and Practice*, Academic Press, New York, 1981.

[17] U. W. KULISCH AND W. L. MIRANKER, EDS., *A New Approach to Scientific Computation*, Academic Press, New York, 1983.

[18] M. A. MALCOLM AND R. B. SIMPSON, *Local vs. global strategies for adaptive quadrature*, ACM Trans. Math. Software, 1 (1975), pp. 127-146.

[19] R. E. MOORE, *The automatic analysis and control of error in digital computation based on the use of interval numbers*, in Error in Digital Computation, Vol. 1, L. B. Rall, ed., John Wiley, New York, 1965, pp. 61-130.

[20] R. E. MOORE, *Interval Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1966.

[21] ———, *Techniques and Applications of Interval Analysis*, SIAM Stud. Appl. Math., 2, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

[22] R. PIESSENS, E. DE DONCKER-KAPENGA, C. W. ÜBERHUBER AND D. K. KAHANER, QUADPACK: *A Subroutine Package for Automatic Integration*, Springer Ser. Comput. Math., 1, Springer, Berlin, New York, 1983.

[23] L. B. RALL, *Optimization of interval computation*, in Interval Mathematics 1980, K. L. E. Nickel, ed., Academic Press, New York, 1980, pp. 489-498.

[24] ———, *Automatic Differentiation: Techniques and Applications*, Lecture Notes in Comput. Sci., 120, Springer, Berlin, New York, 1981.

[25] L. B. RALL, *Integration of interval functions* II. *The finite case*, SIAM J. Math. Anal., 13 (1982), pp. 690-697.

[26] ———, *Representations of intervals and optimal error bounds*, Math. Comp., 41 (1983), pp. 219-227.

[27] ———, *An introduction to the scientific computing language* Pascal-SC, Trans. Second Army Conference on Applied Mathematics and Computing, U.S. Army Research Office, Research Triangle Park, NC, 1985, pp. 117-148.

[28] J. R. RICE, *A metalgorithm for adaptive quadrature*, J. Assoc. Comput. Mach., 22 (1975), pp. 61-82.

[29] A. H. STROUD AND D. SECREST, *Gaussian Quadrature Formulas*, Prentice-Hall, Englewood Cliffs, NJ, 1966.

[30] J. M. YOHE, *The interval arithmetic package*, MRC Technical Summary Report No. 1755, University of Wisconsin-Madison, 1977.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 5, September 1987
© 1987 Society for Industrial and Applied Mathematics
012

CORRELATION STRUCTURING AND THE STATISTICAL ANALYSIS OF STEADY-STATE GROUNDWATER FLOW*

FAI MA†, MEIN SIENG WEI‡ AND WENDELL H. MILLS‡

Abstract. In this paper a general formulation is given for the synthesis of random fields in which groundwater flows. Both homogeneous and inhomogeneous random fields are considered, and an algorithm is devised to effectively capture any specified spatial correlation structure. Monte Carlo simulations of the flow systems are employed to explore the effect of input uncertainty on output variability.

Key words. covariance matrices, Monte Carlo simulation, groundwater flow, uncertainty analysis, random processes, statistical correlation

AMS(MOS) subject classifications. 65C05, 65U05

1. Introduction. It has long been recognized that the innate variability of a geophysical system is a complex phenomenological process that can only be described statistically. The existence of spatial randomness in various structural parameters of geologic formations has been admitted [1], but the extent and significance of such uncertainty are always under dispute. As a result, this awareness has tended to urge overdesign and conservatism in estimates of system performance rather than an organized study of the random effects. That, coincidentally, is also a reflection of the lack of a general formulation for the construction of stochastic mathematical models with inherent colored random variations. The heavy machinery of Ito calculus and the diffusional approach of Kolmogorov are both developed for white noise [2], [3], which often serves as the background excitation of many electronic and mechanical systems. The study of stochastic systems with colored noise parameters is far less extensive. Analytical techniques for colored noise are both scant and restrictive, and they are usually applicable to models with either small input uncertainties or unbounded extent [4], [5]. These limitations render most analytical methods inappropriate for the analysis of real-life systems. Simulation is the common procedure to be employed on colored stochastic systems.

Monte Carlo techniques can be easily applied to systems with large input uncertainties or defined over bounded domains. These techniques involve the repetitive evaluations of various system properties, and as such there is always the question of computational feasibility. The advent of modern computers, however, has now made the brute-force simulation study of many stochastic problems acceptable. In fact, the design and survivability analysis of many important military and civilian systems are currently performed by Monte Carlo methods [6], [7]. Coupled with the effective utilization of parallel processing, Monte Carlo analysis of stochastic problems will become increasingly attractive in the next decade as computing costs decrease. At the present time, one important trend of research in stochastic systems is the design of realistic models that are easily amenable to computational procedures.

This paper sets out to explore the synthesis of random fields in which groundwater flows. The statistical variations within the finite flow field are represented by a spatial stochastic process. An algorithm is constructed to effectively capture the spatial correlation structure of the flow field. Monte Carlo simulation is employed for the evaluation

* Received by the editors April 8, 1985; accepted for publication (in revised form) November 18, 1986.
† Department of Mechanical Engineering, University of California, Berkeley, California 94720.
‡ The Standard Oil Company, Cleveland, Ohio 44128.

of various system parameters, and one basic objective is to determine the effect of input uncertainty on the output. It is decided that emphasis should be placed on algorithmic development. For this reason, one-dimensional simulations will be discussed at length, although limited two-dimensional results will also be presented. As is usual, this paper builds upon several earlier studies [8]–[12]. However, it is believed that the methodology adopted herein has not been previously employed in similar work. Techniques expounded in this paper may be extended and applied to many diverse areas of applied sciences, especially when the synthesis of homogeneous and inhomogeneous random fields is involved. The design of networks for two-dimensional flow and other practical applications of the results will appear elsewhere.

2. Problem formulation. The equation governing steady-state saturated groundwater flow through a heterogeneous medium may be written as [13]

$$\nabla \cdot (K \nabla \phi) = 0 \tag{1}$$

where ϕ is the hydraulic head, ∇ is the gradient operator, and K is the hydraulic conductivity tensor which is a function of the space coordinates. The hydraulic conductivity is characterized by the material properties of the porous medium. On account of the innate variability of geologic formations, the function K is realistically regarded as a spatial stochastic process. With K serving as the random input to the system (1), the question then is how the statistical properties of the output variable ϕ may be determined.

To make subsequent conceptualization more transparent, let us concentrate on the one-dimensional form of system (1), where the flow is described by

$$\frac{\partial}{\partial x}\left(K(x) \frac{\partial \phi}{\partial x} \right) = 0. \tag{2}$$

The boundary conditions are set by

$$\phi(0) = \phi_0, \qquad \phi(L) = \phi_1. \tag{3}$$

Suppose that $\{x_0, x_1, \cdots, x_n\}$ is a uniform partition of the finite flow domain $[0, L]$ under consideration, for which $x_0 = 0$ and $x_n = L$. Let

$$K_i = K(\tfrac{1}{2}(x_{i-1} + x_i))$$

denote the hydraulic conductivity value over the entire subinterval (x_{i-1}, x_i). Since the hydraulic conductivity $K(x)$ is modeled as a spatial stochastic process, the column vector of random elements

$$V = (K_1, K_2, \cdots, K_n)^T \tag{4}$$

may be considered as a representation of $K(x)$ over the flow domain. It is then straightforward to develop an exact difference solution of (2) over the discretized domain, as illustrated in Fig. 1. If (2) is integrated once, the following expression is obtained:

$$K(x) \frac{\partial \phi}{\partial x} = -q \tag{5}$$



FIG. 1. *Discretized domain for linear flow.*

where q, which may be identified as the specific discharge, is constant over the partition for each realization of the stochastic process $K(x)$. This implies that for every $x \in [x_{i-1}, x_1]$ and for $1 \leqq i \leqq n$

$$\phi(x) = -\frac{q}{K_i}(x - x_{i-1}) + \phi(x_{i-1}) \tag{6}$$

which is a piecewise linear approximation of the hydraulic head ϕ over $[0, L]$. Let us define the point hydraulic head ϕ_i by

$$\phi_i = \phi(x_i) \tag{7}$$

for $0 \leqq i \leqq n$. It follows that

$$\phi_i = -\frac{q}{K_i}\left(\frac{L}{n}\right) + \phi_{i-1} \tag{8}$$

and therefore by solving inductively, we find that

$$\phi_n = -\frac{Lq}{n}\sum_{i=1}^{n}\frac{1}{K_i} + \phi_0. \tag{9}$$

Applying the boundary conditions (3), we have

$$q = -\frac{1}{L}(\phi_1 - \phi_0)\left(\frac{1}{n}\sum_{i=1}^{n}\frac{1}{K_i}\right)^{-1}. \tag{10}$$

This completes the difference solution for system (2). Once a definite value of V in (4) has been selected, expressions (6) and(10) provide a convenient way for numerically evaluating the hydraulic head $\phi(x)$ over the given partition. Naturally, how the elements of V may be selected is our next question.

There is now a large body of literature indicating that the hydraulic conductivity values K_i are log-normally distributed [9], [14]. That means the variables $Y_i = \ln K_i$ would follow a normal distribution. For an arbitrarily fixed i, the expected values and variances of Y_i and K_i are related to each other by [15]

$$E(K_i) = \exp\left(E(Y_i) + \tfrac{1}{2}\operatorname{Var}(Y_i)\right), \tag{11}$$

$$\operatorname{Var}(K_i) = E^2(K_i)\{\exp(\operatorname{Var}(Y_i)) - 1\}. \tag{12}$$

In fact, it can be shown that

$$E(K_i^j) = \exp\left(jE(Y_i) + \tfrac{1}{2}j^2 \operatorname{Var}(Y_i)\right). \tag{13}$$

From the above relations, it becomes permissible to talk about $\{Y_i\}$ or $\{K_i\}$ interchangeably. Most software libraries contain a standard normal generator, which produces normal random variates with zero mean and unit variance. For this reason, $\{Y_i\}$ can be readily generated, and it will be more convenient for us to concentrate on the Y_i values. Let us denote the point means and variances by

$$\mu_i = E(Y_i) \tag{14}$$

and

$$\sigma_i^2 = \operatorname{Var}(Y_i). \tag{15}$$

With K_i given in centimeters per second, it is reported that μ_i generally lies in the range -21 to 5, and σ_i between 0.2 and 3 [9], [16]. As a digression, it is interesting to remark that the log-normal distribution is often used to describe positive stochastic perturbations [6] in other areas.

Suppose the uncertainty in $K(x)$ is prescribed over the flow domain $[0, L]$. Or, equivalently, the values μ_i and σ_i are given over the partition. Then, assuming that the partition affords a sufficiently accurate discretization of the flow domain, the statistical properties of the output hydraulic head ϕ may be evaluated by the following Monte Carlo procedure.

(I) Select a sample of n values R_i by using a standard normal generator. These normal random numbers would have zero mean and unit variance.

(II) For $1 \leqq i \leqq n$, take

$$Y_i = \mu_i + \sigma_i R_i. \tag{16}$$

This operation produces normal random numbers Y_i with means μ_i and standard deviations σ_i.

(III) Set $K_i = \exp(Y_i)$ to evaluate the vector V of (4). Invoke expressions (6) and (10) to calculate the hydraulic head ϕ.

(IV) Repeat operations I-III for a total of m times and interpret output data statistically.

Among other things, the above procedure may be employed to determine the standard deviation $s(x)$ of the hydraulic head $\phi(x)$ over the flow field $[0, L]$. This is performed in [8], [9] with a procedure similar to the one described above, but with the additional simplification that the hydraulic conductivity $K(x)$ has the same expected value and variance for all $x \in [0, L]$. In that case it is found that $s(x)$ varies with n, the number of blocks composing the partition. In general, $s(x)$ diminishes as n increases, or as the partition gets finer. It is explained in [10] that the anomaly is due to the lack of a spatial correlation structure in the model. Since a random number generator produces independent random variates [17], the hydraulic conductivity values of the previous procedure are uncorrelated. That would mean hydraulic conductivity values in neighboring blocks of the partition can vary independently of each other, contrary to observations.

To incorporate the autocorrelation of hydraulic conductivity, the pioneering papers of Smith and Freeze [10], [18] assume that $K(x)$ constitutes a homogeneous random field. A sequence $\{R_i\}$ of uncorrelated normal variates is first selected as in stage (I) of the previous Monte Carlo procedure. This sequence is converted into an internally correlated sequence $\{Z_i\}$ by

$$Z_i = \alpha(Z_{i-1} + Z_{i+1}) + R_i \tag{17}$$

where α is an autoregressive parameter expressing the degree of spatial dependence of Z_i on its two neighbors Z_{i-1} and Z_{i+1}. The values Y_i are then derived by taking

$$Y_i = \mu_i + \sigma_i Z_i \tag{18}$$

for $1 \leqq i \leqq n$. This method, appropriately called the nearest-neighbor approach [19]-[21], is an extension of the traditional time series techniques and requires that the random field generated by $K(x)$ be statistically homogeneous. Since $\ln(K(x))$ is Gaussian and since [22]

$$E(K_i K_j) = \exp[\mu_i + \mu_j - \mu_i\mu_j + \tfrac{1}{2}(\sigma_i^2 + \sigma_j^2) + E(Y_i Y_j)] \tag{19}$$

where μ_i, σ_i are defined in (14) and (15), statistical homogeneity dictates that for all i, j

$$\mu_i = \mu_j \tag{20}$$

and that the covariance function $E((Y_i - \mu_i)(Y_j - \mu_j))$ must depend only upon the magnitude of $i - j$ and not individually on i or j. If merely (20) is not satisfied, then

it is pointed out in [18] that occasionally the variation in μ_i may be considered as a conditional trend of the constant mean of a homogeneous field. However, the assumption of statistical homogeneity generally renders the nearest-neighbor approach inappropriate for random fields for which the mean and covariance are dependent on the spatial coordinate, as is the usual case for field situations.

A second limitation of the above approach lies in the fact that the correlation of $\{Y_i\}$, which is an output completely determined by the parameter α, is a characteristically smooth function. According to Mandelbrot, models of correlation occurring in nature often fail to conform to simple patterns, for they invariably contain ripples and slow trends that characterize the presence of frequencies with lower and higher scales [23]. For this reason, many correlation models calculated from field data [14] would not be accurately described by (17) for any value of α. It therefore appears feasible to derive an algorithm that not only can handle inhomogeneous random fields, but also would read in a specified correlation relationship as input. This task will be pursued in the next section. Meanwhile, it should be remarked that with the incorporation of autocorrelation in the homogeneous random fields generated by $K(x)$, it has been found that the output uncertainty in hydraulic head $\phi(x)$ no longer depends upon the partition [10], provided, of course, that the discretization is fine enough to capture all the significant features of the continuous system.

3. Correlation structuring. Suppose a covariance matrix G of order n is given over a sufficiently fine uniform partition $\{x_0, x_1, \cdots, x_n\}$ of the flow domain $[0, L]$. In this case, the manner in which the internal blocks of the partition are correlated is fully prescribed by G. Let

$$R = (R_1, R_2, \cdots, R_n)^T \tag{21}$$

be a random vector whose elements R_i are independent standard normal variates obtained from stage I of the Monte Carlo procedure of the last section. We would like to find a means to convert R into

$$Y = (Y_1, Y_2, \cdots, Y_n)^T \tag{22}$$

whose entries $Y_i = \ln K_i$ are internally correlated in such a way that

$$G = E((Y - E(Y))(Y - E(Y))^T). \tag{23}$$

In mathematical terms, this amounts to devising a transformation

$$f: \mathbb{R}^n \to \mathbb{R}^n \tag{24}$$

such that $f(R) = Y$. Since the elements of both R and Y are normal, we need only consider linear transformations [24]. Thus, we may take

$$f(R) = AR + B \tag{25}$$

where A is a square matrix of order n and $B \in \mathbb{R}^n$. It is obvious that

$$B = E(Y). \tag{26}$$

Furthermore, condition (23) implies that

$$G = A(E(RR^T))A^T. \tag{27}$$

But the elements of R are uncorrelated with unit variance, so that

$$E(RR^T) = I \tag{28}$$

where I represents the identity matrix, and (27) simplifies to

$$G = AA^T. \tag{29}$$

Hence transformation f of (24) can be determined if and only if (29) can be solved.

We now assert that (29) is solvable if and only if G is positive semidefinite. To see that, take any nonzero column vector $x \in \mathbb{R}^n$. If G can be factorized in the form (29), then it must be symmetric, and in addition

$$\begin{aligned} x^T G x &= x^T A A^T x \\ &= (A^T x)^T (A^T x) \geqq 0, \end{aligned}$$

proving positive semidefiniteness. On the other hand, the assumption that G is positive semidefinite implies that it can be diagonalized by an orthogonal transformation P such that [25]

$$G = PDP^T \tag{30}$$

where the diagonal matrix D is written as

$$D = \begin{pmatrix} \lambda_1 & & & 0 \\ & \lambda_2 & & \\ & & \ddots & \\ 0 & & & \lambda_n \end{pmatrix}.$$

Since all the eigenvalues λ_i of G are nonnegative, D admits a square root C such that

$$C = \begin{pmatrix} \lambda_1^{1/2} & & & 0 \\ & \lambda_2^{1/2} & & \\ & & \ddots & \\ 0 & & & \lambda_n^{1/2} \end{pmatrix}.$$

It follows that

$$\begin{aligned} G &= PDP^T \\ &= P(CC)P^T \\ &= (PC)(PC)^T \end{aligned}$$

whereby G is cast into the form (29). This completes the proof of the assertion.

By definition, a covariance matrix is always positive semidefinite. Hence (29) can always be solved for any given covariance matrix G, although the solution matrix A is in general not unique. In most practical applications, the covariance matrix G is strictly positive definite. Otherwise, some of the eigenvalues of G would have zero magnitude, and the corresponding n-dimensional ellipsoids of constant probability density would have vanishing lengths along some axes. Under such circumstances, the problem may be reduced to one of lower dimensions [24], [26], [27]. Henceforth, we assume without loss of generality that G is a positive definite matrix. This makes the application of the Cholesky Decomposition [28], [29] possible, and we would have an efficient algorithm for the factorization of G into the form (29). The Cholesky Decomposition gives a unique solution matrix A which is lower triangular.

We have thus determined a filter (25) by which a specified covariance structure can be built into the input random variables of system (2). Let us proceed to revise the Monte Carlo procedure of the last section. Given any positive definite covariance matrix G, a unique lower triangular matrix A satisfying (29) is first computed. This

matrix A serves as the coefficient matrix of transformation (25). Recalling definitions (14) and (22), we note that

$$E(Y)=(\mu_1, \mu_2, \cdots, \mu_n)^T. \tag{31}$$

Then the Monte Carlo algorithm of the previous section may be modified as follows.

(I'), (III'), (IV') These stages are identical respectively to (I), (III) and (IV) of the previous section.

(II') Evaluate and replace

$$Y=AR+E(Y). \tag{32}$$

This operation produces the random vector Y, whose elements are normal variates satisfying (23).

The modified Monte Carlo algorithm can be readily applied to the uncertainty analysis of groundwater flow described by (2). The covariance filter (32) would provide a realistic representation of the homogeneous or inhomogeneous random field generated by the hydraulic conductivity and in which the flow takes place. To demonstrate the effect of the filter (32), a total of 500 realizations of the random vector $R \in \mathbb{R}^{50}$ is selected by the standard normal generator. Each vector R of (21) is laid on the unit interval by representing the elements R_i as vertical coordinates of 50 equidistant points on the interval. The lack of internal correlation of the elements of R manifests itself as erratic fluctuations as shown in Fig. 2, where two of the random vectors are plotted. Suppose the elements of R are to be homogeneously correlated in accordance with

$$\rho(x)=\exp\left(-\frac{1}{\lambda}|x|\right) \tag{33}$$

where x denotes the separation between the elements and λ is a given parameter. The corresponding covariance matrix G of order 50 is

$$G=\left(\exp\left(-\frac{1}{50\lambda}|i-j|\right)\right) \tag{34}$$

for $1 \leqq i, j \leqq 50$. This positive definite matrix G can be decomposed in the form (29), whereby the linear filter

$$Y=AR \tag{35}$$

is defined, which transforms R into the vector Y whose elements are internally correlated standard normal variates. Since

$$\int_0^\infty \rho(x)\,dx=\lambda, \tag{36}$$

the parameter λ may be thought of as the average correlation length [5]. Set $\lambda=0.1$. The effect of passing the two sample vectors of Fig. 2 through the filter (35) is illustrated in Fig. 3. For the two filtered sample vectors of Fig. 3, there is a tendency for large elements to be surrounded by relatively large elements, and abrupt fluctuations have been greatly smoothed down, even though the average correlation length is 0.1. It becomes very plain that correlation structuring should be a very important step in stochastic simulations. The one-sided correlation function corresponding to (33) is illustrated in Fig. 4, where the resulting correlation of the 500 realizations of the vectors R and Y are also shown. Due to the finite sample size as well as the fact that computers can only generate pseudorandom numbers [17], the correlation of the elements of R

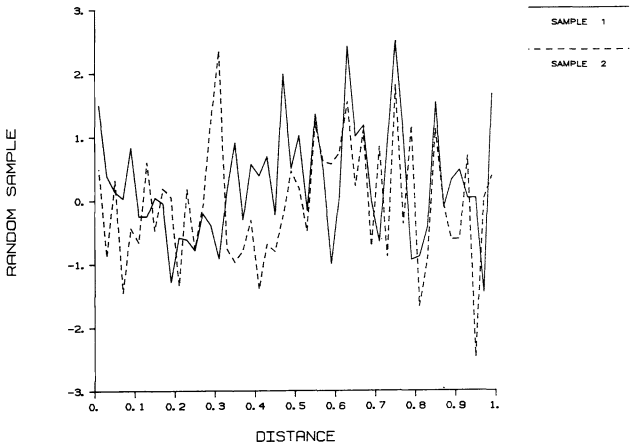


FIG. 2. *The uncorrelated sample elements of random vectors.*

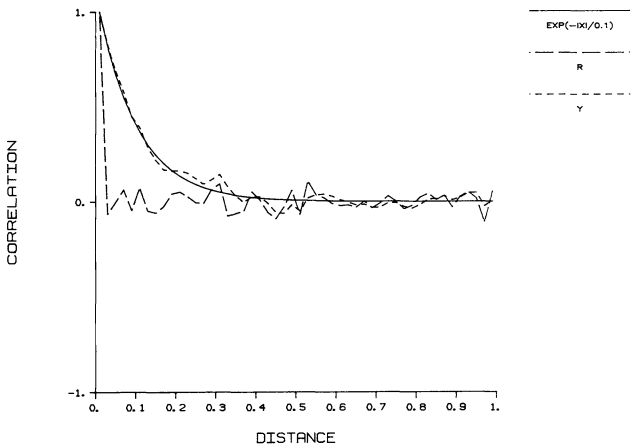


FIG. 4. *Correlation functions from* 500 *realizations of R and Y.*

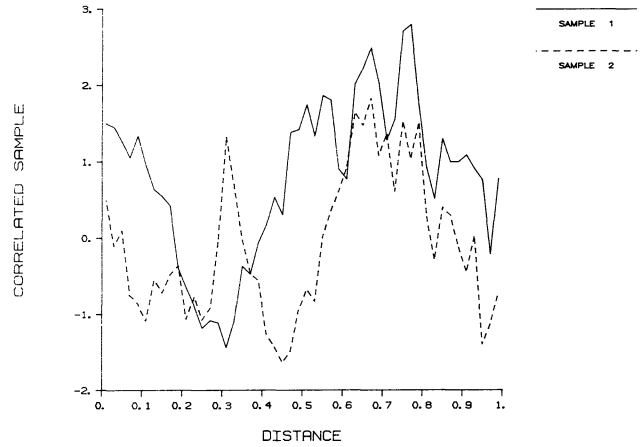


FIG. 3. *The internally correlated sample elements of filtered random vectors, with correlation length of* 0.1.

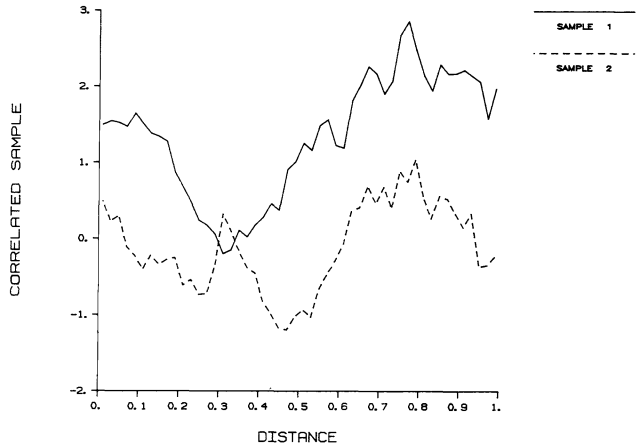


FIG. 5. *The internally correlated sample elements of filtered random vectors, with correlation length of* 0.5.

is not exactly an impulse function. For the same reasons, the internal correlation of the vector Y is nearly, but not exactly, equal to the input correlation (33). Transforming the two samples of Fig. 2 by a stronger filter with $\lambda = 0.5$ results in more highly structured output sample vectors, as shown in Fig. 5. Similar experiments with covariance matrices of other forms lead to the identical observation that correlation plays a significant role in structuring random variates.

4. Simulations of linear flow. The results of several numerical experiments on system (2) will be reported in this section. It is assumed that the first two moments of the output hydraulic head ϕ are the parameters of primary interest. A uniform partition with $n = 25$ blocks is used in the simulations. The value of n is varied slightly in each experiment, and it is checked that the differences in the output data are negligible. This convinces us that the partition of 25 blocks is fine enough to capture all the essential characteristics of the problems on hand. For each case study, a total of $m = 500$ Monte Carlo passes is employed, which in every case is found to be greater than the number of cycles required for convergence of the output data within the 95% confidence limits [6], [30]. In plotting the results, we have decided to resort to dimensionless quantities by the conversion

$$x' = \frac{x}{L}, \tag{37}$$

$$\eta(x) = E(\phi(x))|(\phi_1 - \phi_0)^{-1}|, \tag{38}$$

$$s(x) = [E(\phi(x) - \eta(x))^2]^{1/2}|(\phi_1 - \phi_0)^{-1}| \tag{39}$$

for $x \in [0, L]$, so that the flow field is always of unit length and the mean $\eta(x)$ and standard deviation $s(x)$ of the hydraulic head are expressed as fractions of the total pressure difference across the flow domain. We set for convenience $L = 1$, $\phi_0 = 1$, and $\phi_1 = 0$ in all experiments. On the average, each case study consumes about $50s$ in CPU time on the VAX 11-780.

For the first experiment, a homogeneous random field is synthesized by taking, in definition (31)

$$\mu_i = -2 \tag{40}$$

for all i and setting

$$G = \sigma^2\left(\exp\left(-\frac{L}{\lambda n}|i-j|\right)\right) \tag{41}$$

for $1 \leqq i, j \leqq n$, where σ^2 is the variance of each element of Y and λ is the average correlation length. To proceed, fix $\lambda/L = 0.1$. It is observed that as σ increases, corresponding to increasing input uncertainty, so does the degree of uncertainty associated with the predicted hydraulic head values, as illustrated in Fig. 6. In addition, the standard deviation $s(x)$ of the hydraulic head can attain relatively large values even for fairly moderate input uncertainty. The mean hydraulic head $\eta(x)$ in Fig. 7 decreases linearly over the flow domain for all σ, as is expected. Also displayed in Fig. 7 is the 125th realization of the hydraulic head ϕ over the flow domain corresponding to $\sigma = 1$. The next step is to freeze σ and examine the influence of correlation strength on the standard deviation $s(x)$ of the hydraulic head. An average correlation length of $\lambda = 2$ in a linear field of dimension $L = 10$ is significant, but the effect of this same correlation length can hardly be felt in a field of dimension $L = 100{,}000$. Therefore, the parameter λ/L is used to represent correlation strength. Put $\sigma = 1$. As shown in

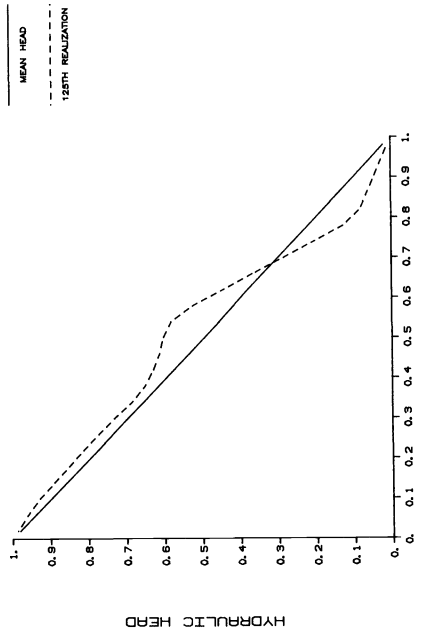


FIG. 7. *Spatial variation of the hydraulic head, with* $\mu_i = -2$.

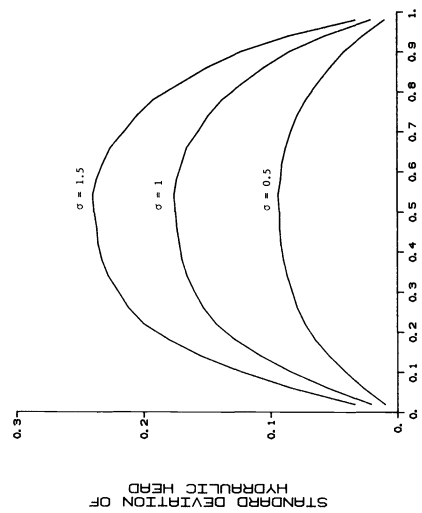


FIG. 6. *Spatial variation of the standard deviation of hydraulic head for different* σ, *with* $\mu_i = -2$.

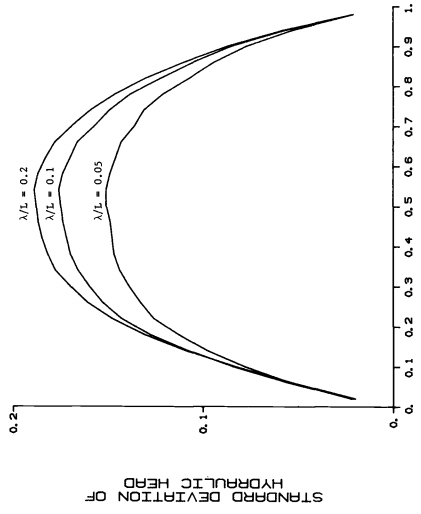


FIG. 8. *Spatial variation of the standard deviation of hydraulic head for different* λ/L, *with* $\mu_i = -2$.

Fig. 8, the output uncertainty is magnified when the strength of correlation increases. The explanation is fairly straightforward [10]. Within a highly correlated sequence, there is a strong tendency for large values to be surrounded by values that are relatively large. This implies that if the hydraulic head $\phi(x)$ at a given point is above its mean, then it is probable that neighboring head values will also be higher than their respective means. On averaging over a series of realizations, this correlation effect will tend to expand the standard deviation $s(x)$ of the hydraulic head at any point. Similar observations on the variation of $s(x)$ with λ/L are made when σ is frozen at other values.

In an applied context it is useful to examine output uncertainty in terms of some overall measure of performance such as the flow rate. The flux q through the system (2) is defined by (5), and over the discretized flow domain it is given by (10). For a partitioned system of the type under consideration, it is well known that the equivalent system conductivity K_e is related to the component values K_i by [13], [16].

$$K_e = \left(\frac{1}{n} \sum_{i=1}^{n} \frac{1}{K_i} \right)^{-1}. \tag{42}$$

Therefore, (10) can be written in the form

$$q = -\frac{1}{L}(\phi_1 - \phi_0) K_e,$$

from which

$$E(q) = -\frac{1}{L}(\phi_1 - \phi_0) E(K_e), \tag{43}$$

$$\operatorname{Var}(q) = \frac{1}{L^2}(\phi_1 - \phi_0)^2 \operatorname{Var}(K_e). \tag{44}$$

Thus the coefficient of variation Ω of the flux is given by

$$\Omega = \left| \frac{\operatorname{Var}(K_e)^{1/2}}{E(K_e)} \right|. \tag{45}$$

In Fig. 9, the influence of the input uncertainty σ on the coefficient of variation Ω of the flux is exhibited, and it can be seen that the two increase or decrease together. Furthermore, Ω generally increases more rapidly than σ for any given λ/L, so that the resulting graphs are convex. All figures shown above are representative of the overall system behavior of (2) within reasonable operating ranges of σ and λ/L. The next task would be to relax the condition (40) to allow for trending mean.

For the second experiment, it is assumed that the elements of $E(Y)$ in (31) are defined in accordance to

$$E(\ln(K(x))) = -1 - \frac{2x}{L} \tag{46}$$

for $x \in [0, L]$. All other parameters are otherwise identical to those of the previous experiment. The dependence of the output uncertainty $s(x)$ on σ is displayed in Fig. 10, when $\lambda/L = 0.1$. The corresponding nonlinear decay of the mean hydraulic head $\eta(x)$ is shown in Fig. 11. The dependence of $s(x)$ on the correlation strength λ/L is presented in Fig. 12, with $\sigma = 1$. Finally, the effect of σ on the coefficient of variation Ω of the flux is exhibited in Fig. 13, and the curvature of the resulting graphs changes as λ/L increases.

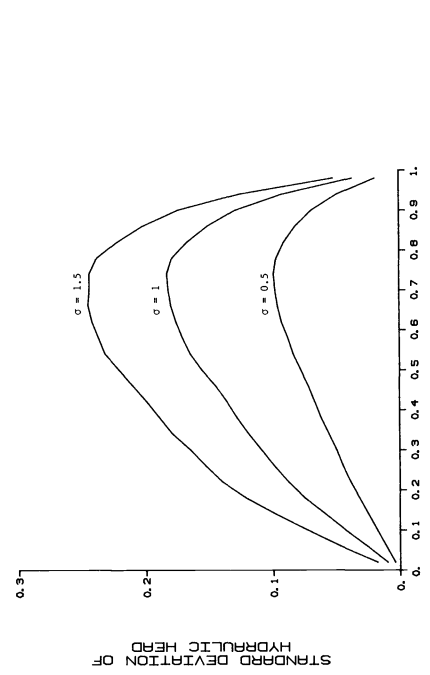


FIG. 10. *Spatial variation of the standard deviation of hydraulic head for different* σ, *when* (46) *is satisfied.*

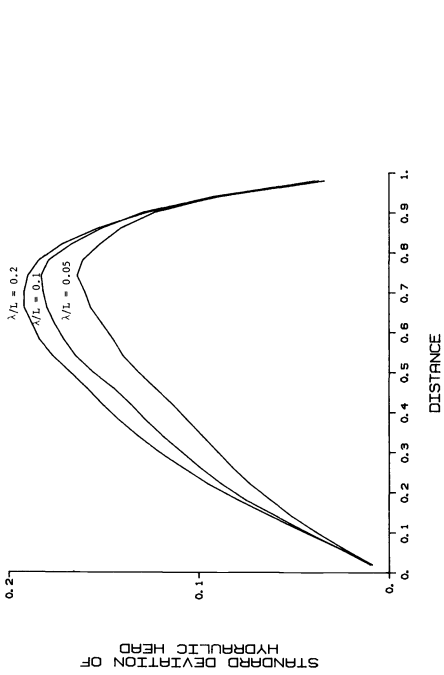


FIG. 12. *Spatial variation of the standard deviation of hydraulic head for different* λ/L, *when* (46) *is satisfied.*

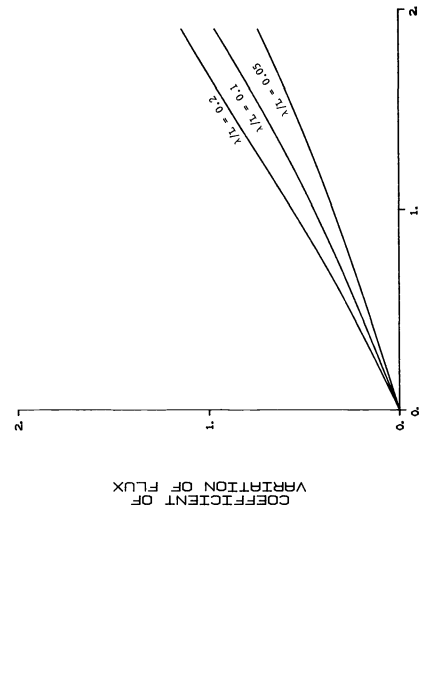


FIG. 9. *Dependence of the coefficient of variation of flux on the input standard deviation* σ, *with* $\mu_i = -2$.

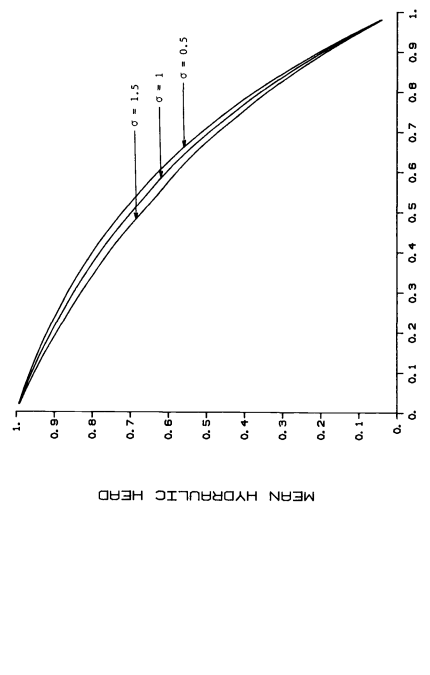


FIG. 11. *Spatial variation of the mean hydraulic head for different* σ, *when* (46) *is satisfied.*

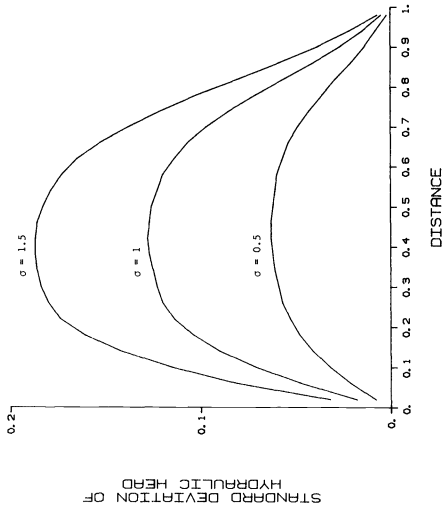


FIG. 14. *Spatial variation of the standard deviation of hydraulic head for different σ, when* (47) *and* (48) *are satisfied.*

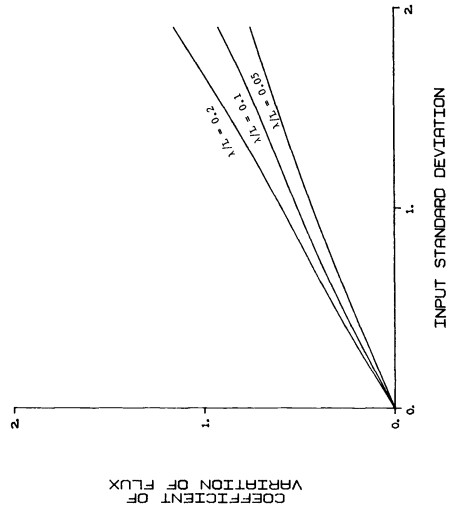


FIG. 13. *Dependence of the coefficient of variation of flux on the input standard deviation σ, when* (46) *is satisfied.*

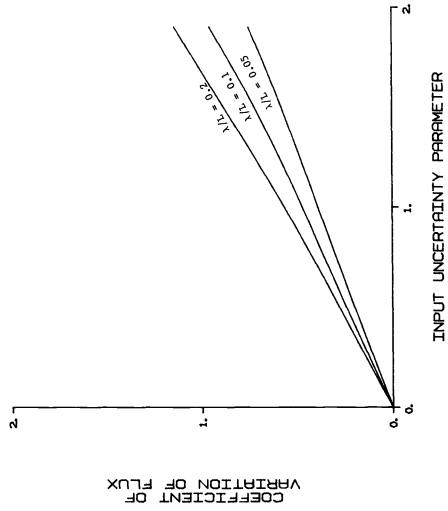


FIG. 16. *Dependence of the coefficient of variation of flux on the input uncertainty parameter σ, when* (47) *and* (48) *are satisfied.*

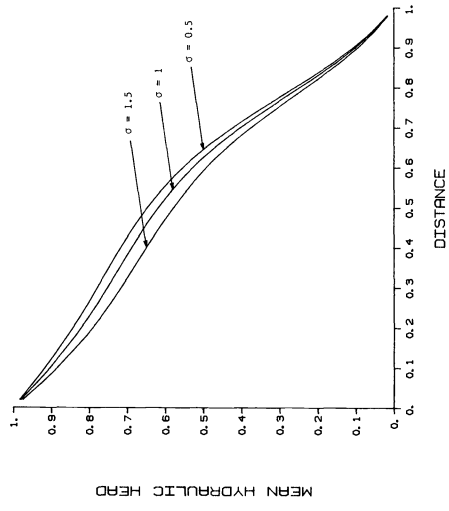


FIG. 15. *Spatial variation of the mean hydraulic head for different σ, when* (47) *and* (48) *are satisfied.*

The third experiment allows $E(\ln(K(x)))$ to oscillate slightly over the flow field $[0, L]$ by setting

$$E(\ln(K(x))) = \frac{x}{L}\sin\left(\frac{2\pi x}{L}\right) - 1. \tag{47}$$

The covariance matrix G of order 25 is defined by

$$G = (\sigma_i \sigma_j \rho_{ij}), \tag{48}$$

$$\sigma_i = \sigma\left(1 - \frac{i-1}{25}\right), \tag{49}$$

$$\rho_{ij} = \exp\left(-\frac{L}{25\lambda}|i-j|\right) \tag{50}$$

for $1 \leqq i, j \leqq 25$, where σ is a constant parameter. There is a decrease of covariance strength towards the boundary $x = L$. The dependence of the output uncertainty $s(x)$ and the mean hydraulic head $\eta(x)$ on σ are shown respectively in Fig. 14 and Fig. 15, with $\lambda/L = 0.1$. The coefficient of variation Ω of the flux is evaluated for several values of the input uncertainty σ, and the results are plotted in Fig. 16. The flow of fluid in an inhomogeneous random field appears to be governed by complex factors such as the extreme values. The isolation and identification of some of the dominant factors in play will be deferred to a subsequent paper.

5. Comparisons with planar flow. As we have pointed out, the aim of this paper is to develop methodology for the synthesis of porous random media. Consequential issues that arise from the applications of the results to practical problems will be addressed elsewhere. For this reason, we shall present only a very brief account of two-dimensional simulations to serve one single purpose: comparisons of output variability for linear and planar flow models subjected to similar boundary conditions.

It is well known that one-dimensional studies tend to overestimate the predictive uncertainties in hydraulic head fields [5], [10]. The reason is that in one dimension the flow cannot be routed around low conductivity inclusions, and their positioning in the flow field will, to a large extent, control the hydraulic head values everywhere over the flow field. In two dimensions the effect of the low conductivity inclusions is localized, and their influence on the hydraulic head is reduced. To demonstrate this, let us consider a special two-dimensional version of system (1) by assuming that the hydraulic conductivity tensor K is diagonal. The equation governing planar flow in this case simplifies to

$$\frac{\partial}{\partial x}\left(K_{xx}\frac{\partial \phi}{\partial x}\right) + \frac{\partial}{\partial y}\left(K_{yy}\frac{\partial \phi}{\partial y}\right) = 0 \tag{51}$$

where K_{xx} is the hydraulic conductivity in the x-direction and K_{yy} the conductivity in y-direction. For the purpose of making comparisons with linear flows, we assume that the flow field is the rectangle $[0, L] \times [0, W]$ whose length L in the x-direction is identical to that used in the previous section. To make any contrast more transparent, we further assume $K_{xx} = K_{yy}$. Constant head conditions

$$\phi(0, y) = \phi_0, \qquad \phi(L, y) = \phi_1 \tag{52}$$

for $y \in [0, W]$ are imposed on the boundaries $x = 0$ and $x = L$ respectively. Along the boundaries $y = 0$ and $y = W$, the condition for no-flow is taken, so that

$$\frac{\partial \phi}{\partial y}(x, 0) = 0, \qquad \frac{\partial \phi}{\partial y}(x, W) = 0 \tag{53}$$

for $x \in [0, L]$. The flow domain is partitioned by a uniform grid into nr rectangular cells, with n blocks in the x-direction and r blocks in the y-direction. The hydraulic conductivity value at the center of each cell is used to represent the conductivity of the entire cell. Suppose K_{ij} denotes the random hydraulic conductivity of the (i, j)th cell. Then, in place of expression (4), there is the $n \times r$ matrix of random elements

$$S = (K_{ij}) \tag{54}$$

for $1 \leqq i \leqq n$ and $1 \leqq j \leqq r$. Let $V_p \in \mathbb{R}^n$ denote the pth column of S, so that

$$S = (V_1, V_2, \cdots, V_r).$$

Define the column string $\text{cs}(S) \in \mathbb{R}^{nr}$ of S by

$$\text{cs}(S) = \begin{pmatrix} V_1 \\ V_2 \\ \vdots \\ V_r \end{pmatrix}.$$

Let $Y \in \mathbb{R}^{nr}$ be a vector whose elements are the natural logarithms of the corresponding elements of cs (S). Then the elements of Y are normally distributed, and their correlation structuring can be accomplished as previously discussed. For the numerical experiments reported in this section, it is assumed that the covariance function is statistically isotropic, so that the correlation strength at any point is not dependent on direction. Clearly, the planar flow models of this section may be regarded as a parallel system of linear uni-directional flows in the x-direction. The covariance structure is the only mechanism regulating flow properties in the y-direction. In fact, by taking the column strings of certain matrices [11], [31] and utilizing the concepts of Kronecker algebra, the current planar flow problem can be discretized and cast in a system form identical to that of model (2). This will not be pursued here. Rather, a finite element scheme with a linear basis function [32], [33] is used to solve for the hydraulic head values. Although not needed for the present investigation, finite element methods possess the ability to approximate accurately bounded domains with irregular boundaries. Statistical measures such as the mean $\eta(x, y)$ and standard deviation $s(x, y)$ of the hydraulic head are obtained by repeated evaluations of the system model.

For the two experiments reported here, a uniform partition, with $n = 25$ blocks in the x-direction and $r = 15$ blocks in the y-direction, is used. That such a partition is accurate enough is verified by slightly changing the values of n and r to check that the differences in output data are negligible. As before, a total of $m = 500$ Monte Carlo passes is employed, which would be sufficient for convergence of the output to within 95% confidence limits. In plotting the results, distances and standard deviations in hydraulic head are expressed as dimensionless quantities, defined by relation (37) and

$$y' = \frac{y}{W}, \tag{55}$$

$$\eta(x, y) = E(\phi(x, y))|(\phi_1 - \phi_0|^{-1}|, \tag{56}$$

$$s(x, y) = [E(\phi(x, y) - \eta(x, y))^2]^{1/2}|(\phi_1 - \phi_0)^{-1}| \tag{57}$$

for $x \in [0, L]$ and $y \in [0, W]$. We set for convenience $L = 1$, $W = 0.6$, $\phi_0 = 1$, and $\phi_1 = 0$ in all cases. In general, each experiment consumes about 25 min in CPU time on the CYBER 205.

Suppose μ_{ij} denotes the mean of $\ln(K(x, y))$ in the (i, j)th cell. In the first experiment, condition (40) is extended to

$$\mu_{ij} = -2 \tag{58}$$

for all i, j. The covariance structure (41) is generalized to the following isotropic function

$$\rho(\xi) = \sigma^2 \exp\left(-\frac{L_1}{\lambda}\xi\right) \tag{59}$$

where the separation ξ is defined by

$$\xi = (x^2 + y^2)^{1/2} \tag{60}$$

for $x \in [0, L]$ and $y \in [0, W]$. Set $L_1 = L$. In this case, however, proper caution is needed in indexing the covariance matrix. If we fix λ/L_1 at 0.1, the standard deviation $s(x, y)$ of hydraulic head is evaluated for different values of σ. Since $s(x, y)$ does not seem to vary along the y-direction, the values of $s(x, y)$ for different σ may be displayed together in Fig. 17. The $s(x, y)$ field is, however, individually and separately evaluated for each value of σ. In comparison to Fig. 6, it can be seen that the standard deviations for planar flow are approximately 70% of those for the corresponding linear flow models. The spatial profile of $s(x, y)$ for constant y is similar to that of one-dimensional flow. The mean hydraulic head $\eta(x, y)$ over the flow field is practically the same for all σ, and is shown in Fig. 18. Finally, fixing σ at 1 and varying λ/L_1 the influence of correlation strength on the standard deviation $s(x, y)$ is examined in Fig. 19, which shows the combined results of three different sets of simulations. When compared to Fig. 8, the two-dimensional effect on the reduction of uncertainty once again becomes evident.

An inhomogeneous field is synthesized in the second experiment by generalizing (46) to

$$E(\ln(K(x, y))) = -1 - \frac{2x}{L} \tag{61}$$

for $x \in [0, L]$ and $y \in [0, W]$. The covariance structure is specified by (59) and (60), and λ/L_1 is initially fixed at 0.1. The dependence of output uncertainty on σ is exhibited in Fig. 20, which is to be compared with Fig. 10 to appreciate the two-dimensional effect on reduction of output variability. The decay of the mean hydraulic head $\eta(x, y)$ for different values of σ is illustrated in Fig. 21. By varying λ/L_1 and fixing σ at 1, the influence of correlation strength on output uncertainty is shown in Fig. 22. All figures presented in this section are representative of the overall system behavior, when λ/L_1 or σ take on other values. The two numerical experiments of this section show that the output uncertainty in two-dimensional systems is, in general, about 70% of the corresponding linear models. Clearly, simulations of steady-state flow for other two-dimensional systems with more general conductivity tensors can be similarly conducted.

6. Conclusion. A general formulation is given for the synthesis of porous random fields in which groundwater flows. The stochastic approach to flow simulation sets out by assuming that the hydraulic conductivity at each point in the flow domain is a random variable, which is interrelated to neighboring values by means of a set of joint probability density functions. Conceived in this manner, hydraulic conductivity constitutes a spatial stochastic process, or, in modern language, a random field [22], [34]. When the underlying spatial stochastic process is stationary, the field is statistically

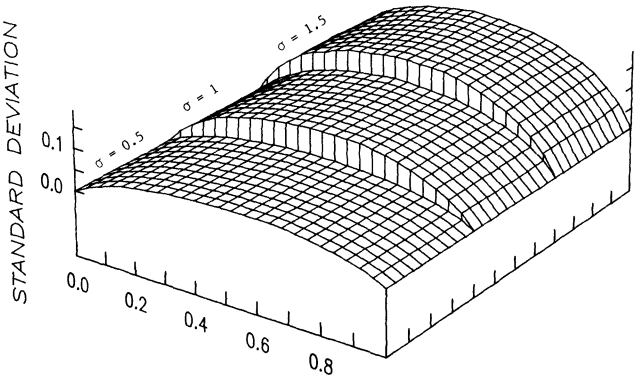


FIG. 17. *Spatial variation of the standard deviation of hydraulic head for different* σ, *with* $\mu_{ij} = -2$.

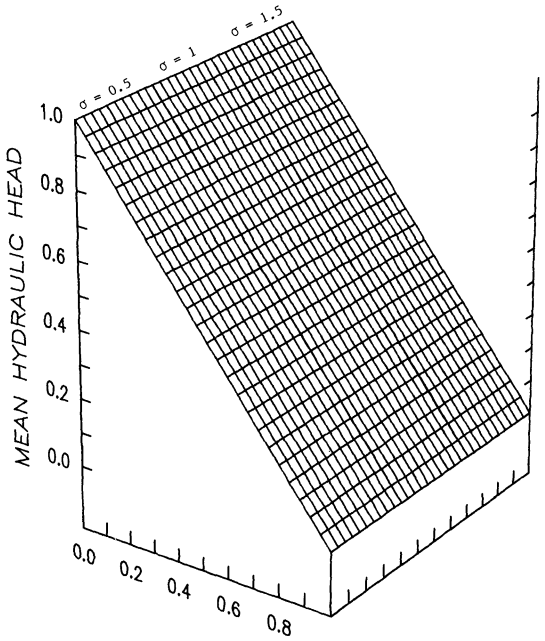


FIG. 18. *Spatial variation of the mean hydraulic head for different* σ, *with* $\mu_{ij} = -2$.

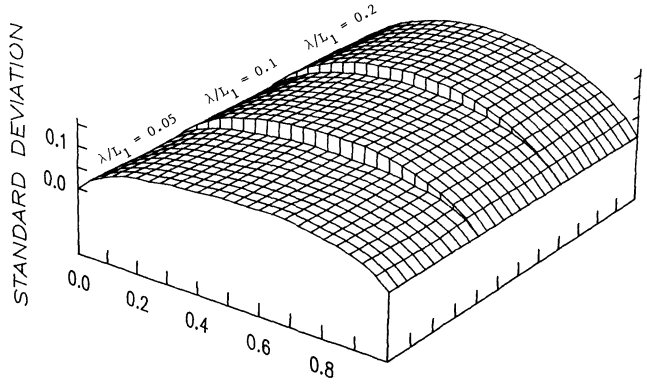


FIG. 19. *Spatial variation of the standard deviation of hydraulic head for different* λ / L_1, *with* $\mu_{ij} = -2$.

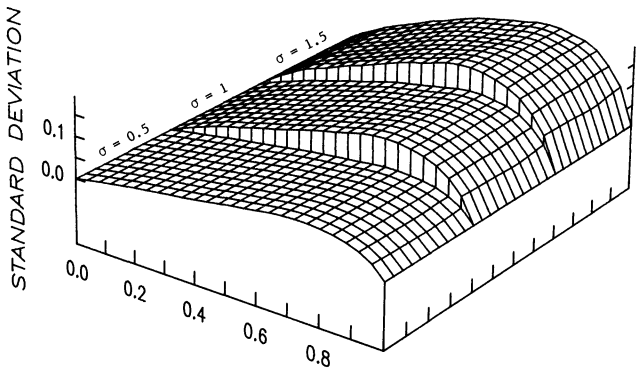


FIG. 20. *Spatial variation of the standard deviation of hydraulic head for different* σ, *when* (61) *is satisfied.*

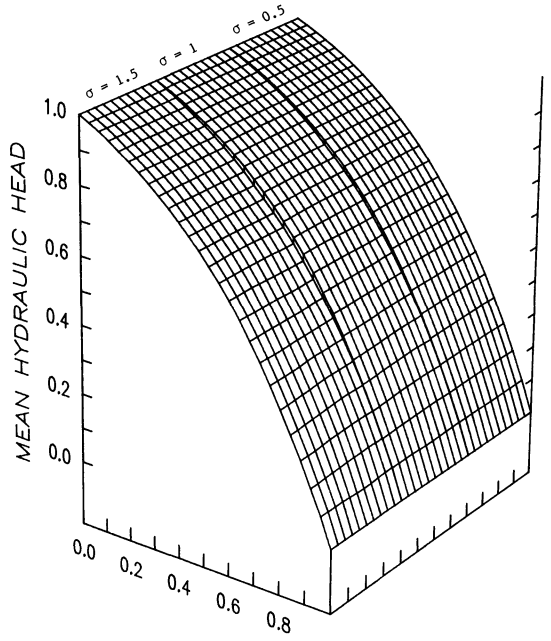


FIG. 21. *Spatial variation of the mean hydraulic head for different* σ, *when* (61) *is satisfied.*

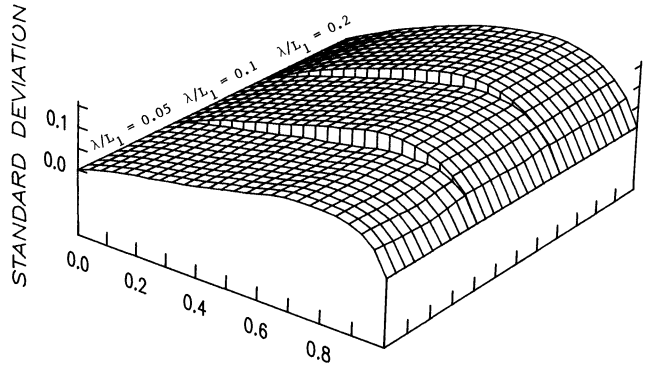


FIG. 22. *Spatial variation of the standard deviation of hydraulic head for different* λ / L_1, *when* (61) *is satisfied.*

homogeneous, the manipulation of which may be greatly simplified. In many practical problems, however, the random fields generated by the hydraulic conductivity are not necessarily homogeneous. In addition, definite correlation structures, which may not conform to elementary patterns, preexist in the fields. An algorithm has therefore been devised in this paper to synthesize homogeneous and inhomogeneous porous random fields with specified covariance structures. The design of the scheme draws upon classical techniques in multivariate statistics and matrix algebra. The Monte Carlo algorithm thus constructed has been applied to the study of steady-state groundwater flow, and for linear flow it is shown how output variability is affected by factors such as correlation strength and standard deviation of hydraulic conductivity. Moreover, the reduction in output uncertainties that takes place when linear models are extended to two dimensions has been demonstrated. The approach proposed in this paper can be readily generalized and applied to transient and multiphase flow problems, as well as to many other diverse areas of science and engineering.

Acknowledgments. We would like to thank Dr. James W. Jennings for assistance with the graphics. We also wish to thank an anonymous reviewer for many useful comments.

REFERENCES

[1] M. MUSKAT, *Physical Principles of Oil Production*, McGraw-Hill, New York, 1949.
[2] T. K. CAUGHEY, *Nonlinear theory of random vibrations*, Adv. in Appl. Mech., 11 (1971), pp. 209–253.
[3] D. LUDWIG, *Persistence of dynamical systems under random perturbations*, SIAM Rev., 17 (1975), pp. 605–640.
[4] J. L. WILLEMS, *Stability criteria for stochastic systems with colored multiplicative noise*, Acta Mech., 23 (1975), pp. 171–178.
[5] A. A. BAKR, L. W. GELHAR, A. L. GUTJAHR AND J. R. MACMILLAN, *Stochastic analysis of spatial variability in subsurface flows*, 1. *Comparison of one- and three-dimensional flows*, Water Resource Res., 14 (1978), pp. 263–271.
[6] W. H. ROWAN, N. LIPNER, J. J. FARRELL, J. D. OLIVER AND A. S. NII, *Failure analysis by statistical techniques*, Vol. I, DNA 3336F-1, TRW Systems Group, Redondo Beach, CA, 1974.
[7] J. D. COLLINS AND J. M. HUDSON, *Applications of risk analysis to nuclear structures*, in Probabilistic Methods in Structural Engineering, M. Shinozuka and J. T. P. Yao, eds., ASCE, 1981, pp. 157–178.
[8] J. E. WARREN AND H. S. PRICE, *Flow in heterogeneous porous media*, Soc. Petrol. Engrg. J., 1 (1961), pp. 153–169.
[9] R. A. FREEZE, *A stochastic conceptual analysis of one-dimensional groundwater flow in nonuniform homogeneous media*, Water Resource Res., 11 (1975), pp. 725–741.
[10] L. SMITH AND R. A. FREEZE, *Stochastic analysis of steady state groundwater flow in a bounded domain*, 1. *One-dimensional simulations*, Water Resource Res., 15 (1979), pp. 521–528.
[11] F. MA, *Approximate analysis of a class of linear stochastic systems with colored noise parameters*, Internat. J. Engrg. Sci., 24 (1986), pp. 19–34.
[12] ———, *Extension of second moment analysis to vector-valued and matrix-valued functions*, Internat. J. Non-Linear Mech., accepted for publication.
[13] R. A. GREENKORN, *Flow Phenomena in Porous Media*, Marcel Dekker, New York, 1983.
[14] J. LAW, *A statistical approach to the interstitial heterogeneity of sand reservoirs*, Trans. American Institute of Mechanical Engineering, 155 (1944), pp. 202–222.
[15] A. H-S. ANG AND W. H. TANG, *Probability Concepts in Engineering Planning and Design, Vol.* I, *Basic Principles*, John Wiley, New York, 1975.
[16] R. A. FREEZE AND J. A. CHERRY, *Groundwater*, Prentice-Hall, Englewood Cliffs, NJ, 1979.
[17] D. E. KNUTH, *The Art of Computer Programming, Vol.* 2, *Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 1969.
[18] L. SMITH AND R. A. FREEZE, *Stochastic analysis of steady state groundwater flow in a bounded domain*, 2. *Two-dimensional simulations*, Water Resource Res., 15 (1979), pp. 1543–1559.
[19] P. WHITTLE, *On stationary processes in the plane*, Biometrika, 41 (1954), pp. 434–449.
[20] M. S. BARTLETT, *The Statistical Analysis of Spatial Pattern*, Chapman and Hall, London, 1975.

[21] R. L. MARTIN, *On spatial dependence, bias and the use of first spatial differences in regression analysis*, Area, 6 (1974), pp. 185–194.
[22] E. H. VANMARCKE, *Random Fields: Analysis and Synthesis*, MIT Press, Cambridge, MA, 1983.
[23] B. B. MANDELBROT AND R. VAN NESS, *Fractional Brownian motions, fractional noises and applications*, SIAM Rev., 10 (1968), pp. 422–437.
[24] E. PARZEN, *Stochastic Processes*, Holden-Day, San Francisco, 1962.
[25] R. BELLMAN, *Introduction to Matrix Analysis*, McGraw-Hill, New York, 1960.
[26] G. AUGUSTI, A. BARATTA AND F. CASCIATI, *Probabilistic Methods in Structural Engineering*, Chapman and Hall, London, 1984.
[27] G. MARSAGLIA AND I. OLKIN, *Generating correlation matrices*, this Journal, 5 (1984), pp. 470–475.
[28] R. S. MARTIN, G. PETERS AND J. H. WILKINSON, *Symmetric decomposition of a positive definite matrix*, Numer. Math., 7 (1965), pp. 362–383.
[29] J. STOER AND R. BULIRSCH, *Introduction to Numerical Analysis*, Springer-Verlag, New York, 1980.
[30] J. S. BENDAT AND A. G. PIERSOL, *Random Data: Analysis and Measurement Procedures*, Wiley-Interscience, New York, 1971.
[31] F. MA, *Stability theory of stochastic difference systems*, in Probabilistic Analysis and Related Topics, Vol. 3, A. T. Bharucha-Reid, ed., Academic Press, New York, 1983, pp. 127–160.
[32] J. F. BOTHA AND G. F. PINDER, *Fundamental Concepts in the Numerical Solution of Differential Equations*, Wiley-Interscience, New York, 1983.
[33] Y. K. CHEUNG AND M. F. YEO, *A Practical Introduction to Finite Element Analysis*, Pitman, London, 1979.
[34] P. WHITTLE, *Stochastic processes in several dimensions*, Bull. Inst. Internat. Statist., 40 (1963), pp. 979–994.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 5, September 1987
© 1987 Society for Industrial and Applied Mathematics
013

ALGORITHMS TO COMPUTER ORDER STATISTIC DISTRIBUTIONS*

CHARLES G. BONCELET, JR.†

Abstract. We present a series of fast and easily implemented algorithms for the computation of single and multivariate distributions of the order statistics for sequences of random variables which may be either independent or Markov. We discuss the computation of moments of L-statistics, principally means and variances, and present algorithms which do not require that the distributions first be "differenced" to yield densities.

Key words. order statistics, recursive computation, distribution, moments

AMS(MOS) subject classifications. 62G30, 62E15, 68Q25

1. Introduction. Order statistics are widely used in statistical inference, with recent interest arising in the area of robust estimation. However, the properties of order statistics are not well understood and researchers must often resort to Monte Carlo methods to analyze their procedures. We address this question and present a series of algorithms to compute the distributions of single and joint order statistics. These algorithms are fast, make no restriction that the random variables be identically distributed and allow the random variables to be either independent or Markov. We also consider the specific question of calculating moments.

Order statistics are defined simply (see David [1] or Kendall and Stuart [2]): Take the original data sequence, x_m, and sort into increasing order. The kth order statistic is the kth largest. In particular, the first order statistic is the minimum of the data sequence and the last is the maximum. The kth order statistic is usually denoted by $x_{(k)}$; we will modify this slightly and denote the kth order statistic out of n samples by $x_{(k;n)}$.

An important class of applications arises in robust statistical procedures. The class of robust estimators called L-estimates is based on order statistics. These are the estimators which can be written as linear combinations of order statistics. For example, a common class of L-estimates for a location parameter are the α-trimmed means, defined by excluding or "trimming off" the lowest and highest αn order statistics and taking the average of the rest. A common L-estimate of scale is the "interquartile distance" defined as $d_I = x_{(3n/4;n)} - x_{(n/4;n)}$. Many L-estimates of location and a few of scale are analyzed in the Monte Carlo study of Andrews et al. [3].

The theory of order statistics is well established for independent, identically distributed random variables. In the continuous case, the density of the kth order statistic is formed easily. Decompose the event that $s \leqq x_{(k;n)} \leqq s + ds$ into 3 parts: that $k-1$ of the x_m are less than or equal to s, that one is between s and $s+ds$, and that $n-k$ are greater than or equal to s. The probability that $k-1$ are less than or equal to s is $F^{k-1}(s)$, that one is between s and $s+ds$ is $f(s)\,ds$, and that $n-k$ are greater than or equal to s is $(1-F(s))^{n-k}$. Now, the number of ways n items can be divided into three groups with a, b, and c in each is

$$\frac{n!}{a!b!c!}. \tag{1}$$

* Received by the editors January 20, 1986; accepted for publication (in revised form) August 26, 1986. This work was supported by the University of Delaware Research Foundation.

† Department of Electrical Engineering, University of Delaware, Newark, Delaware 19716.

In this case, $a = k-1$, $b = 1$, $c = n-k$. Putting these together, one obtains:

$$(2)\quad f_{(k;n)}(s)\,ds = P(s \leqq x_{(k;n)} \leqq s+ds) = \frac{n!}{(k-1)!(n-k)!} F^{k-1}(s)(1-F(s))^{n-k} f(s)\,ds.$$

Unfortunately, the above argument does not extend to discrete random variables. In this case, David and Mishriky [4] write the distribution function of $x_{(k;n)}$ as

$$(3)\quad P(x_{(k;n)} \leqq s) = P \quad (\text{at least } k \text{ of } n \text{ RV's are} \leqq s)$$

$$(4)\quad = \sum_{j=k}^{n} P \quad (\text{exactly } j \text{ RV's are} \leqq s)$$

$$(5)\quad = \sum_{j=k}^{n} \binom{n}{j} F^{j}(s)(1-F(s))^{n-j}.$$

The theory of order statistics for nonidentically distributed and dependent random variables is not as well understood as that for the independent, identically distributed case. Vaughan and Venables [5] provide a permanent expansion of the density for continuous, independent, nonidentically distributed random variables. For the application to signal filtering Ataman et al. [6] studied the properties of the median for the $n = 3$ case. Maurer and Margolin [7], using the inclusion-exclusion formula, present an expression for the distribution of order statistics for dependent random variables. Recently, Nodes and Gallagher [8] considered the median filtering problem and presented expressions for the distributions of order statistics with various overlapping windows.

The objection to these generalizations is that they are all *exponential* algorithms, that is, the number of operations needed is an exponential function of the number of variables, n. For example, consider extending David and Mishriky's formula to the case of independent, nonidentically distributed random variables. The step in going from (4) to (5) is now incorrect. The probability that exactly j random variables are less than or equal to s is equal to the sum over all combinations with exactly j random variables less than or equal to s of the probability of each combination. To calculate all the probabilities for $0 \leqq j \leqq n$, one must examine 2^n different combinations. For example, with $n = 21$, $2^n = 2{,}097{,}152$. Computing the probability of each combination requires $n-1 = 20$ arithmetic operations, yielding a total computation count of approximately 40 million operations. In contrast, the algorithms presented here require less than $n^2 = 441$ arithmetic operations for the same task.

In this paper, we consider the problem of calculating order statistic distributions for arbitrary random variables. We make the assumption that the random variables are either independent or Markov, but make no other restrictions on them. In particular, we do not assume that the random variables are identically distributed. In § 2, we present the main result of this paper: a polynomial time, recursive algorithm for computing order statistic distributions with independent random variables. In § 3, we extend this result to Markov random variables. In § 4, we consider the specific problem of computing moments and show how some well-known, but little used, results can be put to our advantage.

2. Independent random variables. We make the following definitions:

1) As above, denote the kth order statistic out of n random variables by $x_{(k;n)}$.

2) Denote the mth random variable by x_m.

3) Let $(s_1, s_2, \cdots, s_l)$ be a set of real numbers with the property that $s_j < s_{j+1}$ for all j. Also, let $s_0 = -\infty$ and $s_{l+1} = +\infty$.

4) For each random variable, say x_m, partition the real line into $l+1$ disjoint events, $D(j, m)$ with $j=1, \cdots, l+1$, as follows:

$$D(j, m)=\{s_{j-1}<x_m \leqq s_j\}. \tag{6}$$

5) Define $E(k_1, k_2, \cdots, k_l; n)(s_1, s_2, \cdots, s_l)$ as the event that the chosen l order statistics are less than or equal to some values, $(s_1, s_2, \cdots, s_l)$, i.e.,

$$E(k_1, k_2, \cdots, k_l; n)(s_1, s_2, \cdots, s_l)=\{x_{(k_1;n)} \leqq s_1, x_{(k_2;n)} \leqq s_2, \cdots, x_{(k_l;n)} \leqq s_l\}. \tag{7}$$

6) Let

$$F(k_1, k_2, \cdots, k_l; n)(s_1, s_2, \cdots, s_l)=P(E(k_1, k_2, \cdots, k_l; n)(s_1, s_2, \cdots, s_l)).$$

The mnemonics are that "D" refers to *disjoint* events, that "E" is the *event* of interest, and that "F" is the probability of that event. The algorithms to follow vary $(k_1, k_2, \cdots, k_l; n)$ for fixed $(s_1, s_2, \cdots, s_l)$. Often in cases where the dependence on it is not crucial, we will drop $(s_1, s_2, \cdots, s_l)$ from E and F and write simply $E(k_1, k_2, \cdots, k_l; n)$ and $F(k_1, k_2, \cdots, k_l; n)$, respectively.

The development of the main result is done in two parts. We first derive a preliminary result on events of order statistics and then apply probabilities. With the definitions above, the fundamental separation theorem is the following.

THEOREM 1.

$$\begin{aligned} &E(k_1, k_2, \cdots, k_l; n+1) \\ &\quad=\cup_{j=1}^{l+1} E(k_1, k_2, \cdots, k_{j-1}, k_j-1, \cdots, k_l-1; n) \cap D(j, n+1). \end{aligned} \tag{8}$$

Proof. Since the $D(j, n+1)$ for fixed n are mutually exclusive,

$$E(k_1, k_2, \cdots, k_l; n+1)=\cup_{j=1}^{l+1} E(k_1, k_2, \cdots, k_l; n+1) \cap D(j, n+1).$$

Consider the generic term, $E(k_1, k_2, \cdots, k_l; n+1) \cap D(j, n+1)$. This is the event that at least k_1 of the $n+1$ random variables are less than or equal to s_1, that at least k_2 are less than or equal to s_2, and so on through at least k_l of the random variables are less than or equal to s_l intersected with the event that the $(n+1)$st random variable is in the set $D(j, n+1)$, i.e., that the $(n+1)$st random variable is greater than s_{j-1} and less than or equal to s_j. Necessarily, it must be true that at least k_1 of the first n are less than or equal to s_1 through k_{j-1} of the first n are less than or equal to s_{j-1} and that at least k_j-1 or the first n are less than or equal to s_j through k_l-1 of first n are less than or equal to s_l. Thus,

$$\begin{aligned} &E(k_1, k_2, \cdots, k_l; n+1) \cap D(j, n+1) \\ &\quad=E(k_1, k_2, \cdots, k_{j-1}, k_j-1, \cdots, k_l-1; n) \cap D(j, n+1). \end{aligned} \tag{9}$$

Summing up all the terms yields the stated result. □

An easy corollary of this result is a simple recursive formula to calculate the distributions when the random variables are independent.

THEOREM 2. *For independent random variables,*

$$\begin{aligned} &F(k_1, k_2, \cdots, k_l; n+1) \\ &\quad=\sum_{j=1}^{l+1} F(k_1, k_2, \cdots, k_{j-1}, k_j-1, \cdots, k_l-1; n) P(D(j, n+1)). \end{aligned} \tag{10}$$

Proof. Since the $D(j, n+1)$ are disjoint, the probability of the union is equal to the sum of the individual probabilities. Thus,

$$F(k_1, k_2, \cdots, k_l; n+1) = \sum_{j=1}^{l+1} P(E(k_1, k_2, \cdots, k_{j-1}, k_j - 1, \cdots, k_l - 1; n) \cap D(j, n+1)). \tag{11}$$

Now, the event $E(k_1, k_2, \cdots, k_l; n)$ is determined by the first n random variables and is not a function of the last one and, hence is independent of $D(j, n+1)$. Therefore, $P(E \cap D) = P(E)P(D)$ and the result follows. □

We now offer some comments concerning the algorithm and discuss some of the implementation issues.

1) Theorem 2 does not require that the random variables be identically distributed. In fact, it makes no assumptions about the random variables except that they are independent.

2) Individual points of the distributions are calculated independently of all others. This results in considerable savings in problems where only a few points are needed; for example, in hypothesis testing often the only probability needed is whether a given statistic exceeds a threshold. This property also makes the algorithm ideally suited for a recursive implementation. In § 4, we exploit this property in computing moments of L-estimates.

3) Initially the k_j will be sorted into increasing order (otherwise, the problem can be reduced in size by eliminating any k_i that are out of order). However, as the computation progresses, this order will be disturbed. Increasing order can easily be maintained by using the following obvious boundary conditions:

(a) $F(\cdots, k_j, k_{j+1}, \cdots; n) = F(\cdots, k_j, k_j, \cdots; n)$ for $k_{j+1} < k_j$.

(b) $F(\cdots, k_l; n) = 0$ for $k_l > n$.

(c) $F(\cdots, k_j, \cdots; n) = F(\cdots, 0, \cdots; n)$ for $k_j < 0$.

(d) $F(0, 0, \cdots, 0; n) = 1$.

4) The computational requirements are easily determined. Each application of Theorem 2 requires $O(l)$ arithmetic operations. Let S be the number of points on which the random variables are defined and N be the total number of random variables. Then there are $O(S^l)$ combinations of s_j. (Of course, many of these points will have negligible probability in practice and can be ignored.) If one is calculating all $\binom{N}{l} = O(N^l)$ combinations of l-variate order statistics with N random variables for each combination of s_j, then the computation required is $O(l)$ operations for each of $O(s^l N^{l+1})$ quantities yielding a total of $O(S^l N^{l+1} l)$ operations. In particular, computing all the first order distributions requires approximately SN^2 operations.

As an example of the use of these formulas, consider the calculation of the distribution function of the minimum and the maximum. In this case, $l = 1$, $P(D(1, m)) = F_m(s_1)$, and $P(D(2, m)) = 1 - F_m(s_1)$ where $F_m(s_1)$ is the cumulative distribution function of the mth random variable. Thus,

$$F(n+1; n+1)(s_1) = F(n; n)(s_1)F_{n+1}(s_1) + F(n+1; n)(s_1)(1 - F_{n+1}(s_1)) \tag{12}$$

$$= F(n; n)(s_1)F_{n+1}(s_1) \tag{13}$$

$$= \prod_{i=1}^{n+1} F_i(s_1) \tag{14}$$

and

$$F(1; n+1)(s_1) = F(0; n)(s_1)F_{n+1}(s_1) + F(1; n)(s_1)(1 - F_{n+1}(s_1)) \tag{15}$$

$$= 1 \cdot F_{n+1}(s_1) + F(1; n)(s_1)(1 - F_{n+1}(s_1)) \tag{16}$$

$$= 1 - (1 - F_{n+1}(s_1))(1 - F(1; n)(s_1)) \tag{17}$$

$$= 1 - \prod_{i=1}^{n+1} (1 - F_i(s_1)). \tag{18}$$

Equation (13) uses boundary condition 3(b); (16) uses boundary condition 3(d).

3. Markov random variables. For dependent random variables, Theorem 2 is invalid. The step in going from $P(ED)$ to $P(E)P(D)$ requires independence. The easiest way to counter this difficulty and extend the theorem is to condition on some event so that events E and D are conditionally independent and then integrate out the event conditioned on. In this section, we illustrate this technique for the special case of discrete, first order, Markov random variables. Extensions for other cases are certainly possible, but will not be considered here. The resulting formulas are slightly more complicated than that in Theorem 2 and the amount of computation needed will increase somewhat.

In order to clarify the development, we will simplify the notation, sacrificing precision somewhat for simplicity. Let

$$P(t_m) \equiv P(\{x_m = t_m\}), \tag{19}$$

$$P(\cdot \mid t_m) \equiv P(\cdot \mid \{x_m = t_m\}), \tag{20}$$

$$D_m^j \equiv D(j, m), \tag{21}$$

$$E_m^j \equiv E(k_1, k_2, \cdots, k_{j-1}, k_j - 1, \cdots, k_l - 1; m), \tag{22}$$

$$E_m^{l+1} \equiv E(k_1, k_2, \cdots, k_l; m), \tag{23}$$

$$I_m^j \equiv \begin{cases} 1 & s_{j-1} < t_m \leqq s_j, \\ 0 & \text{otherwise.} \end{cases} \tag{24}$$

In all these definitions, the subscript refers to "time" and the superscript to "ordering" or "partitioning." In this notation, Theorem 1 can be written:

$$E_{n+1}^{l+1} = \prod_{j=1}^{l+1} E_n^j D_{n+1}^j. \tag{25}$$

First order Markov processes, as they are usually defined, satisfy the following basic property (see Karlin and Taylor [9] or Feller [10]):

$$P(t_n \mid t_{n-1}, \cdots, t_0) = P(t_n \mid t_{n-1}). \tag{26}$$

It will be convenient to reverse the time index and think of the process as propagating backward in time, thus

$$P(t_n \mid t_{n+1}, t_{n+2}, \cdots) = P(t_n \mid t_{n+1}). \tag{27}$$

The probabilities, $P(t_n \mid t_{n+1})$, are the *transition probabilities.* The only property of Markov processes needed here is that the past and future are conditionally independent given the present, i.e., for $k \geqq 0$ and $j \geqq 0$,

$$P(t_{n-k}, t_{n+j} \mid t_n) = P(t_{n-k} \mid t_n) P(t_{n+j} \mid t_n). \tag{28}$$

The development will proceed in two parts, the first of which is easy:

$$F(k_1, k_2, \cdots, k_l; n+1) \equiv P(E_{n+1}^{(l+1)}) \tag{29}$$

$$= \sum_{t_{n+1}} P(E_{n+1}^{l+1} \mid t_{n+1}) P(t_{n+1}). \tag{30}$$

The second part is more detailed. Consider one term in the sum:

$$P(E_{n+1}^{l+1}|t_{n+1})P(t_{n+1}) = P(E_{n+1}^{l+1}t_{n+1}) \tag{31}$$

$$= P\left(\prod_{j=1}^{l+1} E_n^j D_{n+1}^j t_{n+1}\right) \tag{32}$$

$$= \sum_{j=1}^{l+1} P(E_n^j D_{n+1}^j t_{n+1}) \tag{33}$$

$$= \sum_{t_n} \sum_{j=1}^{l+1} P(E_n^j D_{n+1}^j t_{n+1}|t_n)P(t_n) \tag{34}$$

$$= \sum_{t_n} \sum_{j=1}^{l+1} P(E_n^j|t_n)P(D_{n+1}^j t_{n+1}|t_n)P(t_n). \tag{35}$$

Equation (32) used Theorem 1, (33) follows because the D_{n+1}^j are disjoint and (35) used the conditional independence of past and future for Markov random variables. It is easy to see that

$$P(D_{n+1}^j t_{n+1}|t_n) = I_{n+1}^j P(t_{n+1}|t_n) \tag{36}$$

because if $s_{j-1} < t_{n+1} \leqq s_j$ then D_{n+1}^j is redundant; otherwise, the intersection of D_{n+1}^j and $\{x_{n+1} = t_{n+1}\}$ is null. Thus,

$$P(E_{n+1}^{l+1}|t_{n+1}) = \sum_{t_n} \frac{P(t_{n+1}|t_n)P(t_n)}{P(t_{n+1})} \sum_{j=1}^{l+1} I_{n+1}^j P(E_n^j|t_n) \tag{37}$$

$$= \sum_{t_n} P(t_n|t_{n+1}) \sum_{j+1}^{l+1} I_{n+1}^j P(E_n^j|t_n). \tag{38}$$

We have proved the following theorem.

THEOREM 3. *The distribution functions of order statistics taken from discrete, reverse time, Markov random variables may be propagated as follows*:

$$F(k_1, k_2, \cdots, k_l; n+1) = \sum_{t_{n+1}} F(k_1, k_2, \cdots, k_l; n+1|t_{n+1})P(t_{n+1}), \tag{39}$$

$$F(k_1, k_2, \cdots, k_l; n+1|t_{n+1}) = \sum_{t_n} P(t_n|t_{n+1}) \sum_{j=1}^{l+1} I_{n+1}^j F(k_1, k_2, \cdots, k_{j-1}, k_j - 1, \cdots, k_l - 1; n|t_n). \tag{40}$$

As is the case with independent random variables, there is no restriction that the random variables be identically distributed. The I_{n+1}^j term acts as a switch; given t_{n+1}, only one value of j causes it to be nonzero. The sum over j does not represent any computation. At time $n+1$ for each of $O(N^l)$ combinations of order statistics, we propagate $O(S^{l+1})$ quantities with each requiring $O(S)$ operations. Thus, the total computation required is $O(S^{l+2}N^{l+1})$ as compared with $O(S^l N^{l+1} l)$ for the independent case.

4. Calculation of moments. In this section, we consider the calculation of moments of L-estimates. Often it is not necessary that one know the distribution of a statistic, but only certain functions of it. For example, in many applications knowledge of a statistic's mean and variance is sufficient to characterize its performance. We will restrict attention to discrete random variables for the pragmatic reason that the distribution formulas presented in this paper calculate individual points of the distribution.

Continuous random variables must be quantized at some point in the computation. We will also restrict attention to first and second moments as these are generally the most useful.

An L-estimate, say y, can be written as a linear combination of the order statistics:

$$y = \sum_{k=1}^{N} \alpha_k x_{(k;N)}. \tag{41}$$

The first moment is

$$Ey = \sum_{k=1}^{N} \alpha_k E_{x(k;N)} \tag{42}$$

and the second,

$$Ey^2 = \sum_{k_1=1}^{N} \alpha_{k_1} Ez_{k_1}^2 + 2 \sum_{k_2=2}^{N} \sum_{k_1=1}^{k_2-1} \alpha_{k_1}\alpha_{k_2} Ex_{(k_1;N)}x_{(k_2;N)}. \tag{43}$$

Thus, as is well known, the first and second moments of the L-estimate can be computed if one knows the univariate and bivariate distributions of the order statistics.

Since the algorithm given above calculates distributions and not densities, it will be convenient to express these expected values in terms of distributions rather than having to "difference" the distributions to obtain the densities. Furthermore, this will significantly reduce computation time and storage requirements because the recursive algorithm calculates *the same point*, $(s_1, s_2, \cdots, s_l)$, for many different combinations of order statistics. The loops involving $(s_1, s_2, \cdots, s_l)$ can be brought furthest outside rather than furthest in as is customary.

Formulas which express the moments in terms of the distribution instead of the density are well known, but are usually ignored in terms of usefulness. (See, for example, Woodroofe [11].) These formulas can be derived from the "summation by parts" formula [12]:

$$\sum_{i=1}^{S} a_i(C_i - C_{i-1}) = a_S C_S - \sum_{i=1}^{S-1} (a_{i+1} - a_i)C_i - a_1 C_0. \tag{44}$$

Let the order statistics be defined on S points denoted s_i for $i = 1, 2, \cdots, S$. Further, let $a_i \equiv A(s_i)$, $p_i \equiv P(\{x_{(k;N)} = s_i\})$, and $C_i \equiv P(\{x_{(k;N)} \leqq s_i\})$. C satisfies the boundary conditions, $C_0 = 0$ and $C_S = 1$. Then, one can write

$$EA(x_{(k;N)}) = \sum_{i=1}^{S} a_i p_i \tag{45}$$

$$= \sum_{i=1}^{S} a_i(C_i - C_{i-1}) \tag{46}$$

$$= a_S - \sum_{i=1}^{S-1} (a_{i+1} - a_i)C_i. \tag{47}$$

When $s_i = \gamma i + \delta$, particularly simple formulas result for the first and second moments:

$$\begin{aligned} Ex_{(k;N)} &= \gamma S + \delta - \gamma \sum_{i=1}^{S-1} C_i, \\ Ex_{(k;N)}^2 &= (\gamma S + \delta)^2 - \gamma \sum_{i=1}^{S} (2\gamma i + 2\delta + \gamma)C_i. \end{aligned} \tag{48}$$

Second order moments of order statistics are a little more difficult due to the dependence of the two order statistics. In particular, for $k_2 > k_1$, $x_{(k_2;N)} \geqq x_{(k_1;N)}$. As above, let

$$a_{i,j} \equiv A(s_i, s_j), \tag{49}$$

$$p_{i,j} \equiv P(\{x_{(k_1;N)} = s_i\}, \{x_{(k_2;N)} = s_j\}), \tag{50}$$

$$C_{i,j} \equiv F(k_1, k_2; N)(s_i, s_j). \tag{51}$$

Then $p_{i,j} = C_{i,j} - C_{i-1,j} - C_{i,j-1} + C_{i-1,j-1}$. The $C_{i,j}$ satisfy the following boundary conditions: $C_{0,j} = C_{i,0} = C_{0,0} = 0$, $C_{S,S} = 1$, and $C_{i,j} = C_{j,j}$ for $i > j$.

We will summarize the two-dimensional formula in terms of a lemma.

LEMMA 1.

$$\begin{aligned} EA(x_{(k_1;N)}, x_{(k_2;N)}) = a_{S,S} &+ \sum_{j=2}^{S-1} \sum_{i=1}^{j-1} (a_{i+1,j+1} - a_{i,j+1} - a_{i+1,j} + a_{i,j}) C_{i,j} \\ &- \sum_{i=1}^{S-1} (a_{i+1,S} - a_{i,S}) C_{i,S} - \sum_{j=1}^{S-1} (a_{j,j+1} - a_{j,j}) C_{j,j}. \end{aligned} \tag{52}$$

Proof.

$$EA(x_{(k_1;N)}, x_{(k_2;N)}) = \sum_{j=1}^{S} \sum_{i=1}^{j} a_{i,j} p_{i,j} \tag{53}$$

$$= \sum_{j=1}^{S} \sum_{i=1}^{j} a_{i,j} (C_{i,j} - C_{i,j-1} - C_{i-1,j} + C_{i-1,j-1}) \tag{54}$$

$$\begin{aligned} &= \sum_{j=1}^{S} \sum_{i=1}^{j} a_{i,j} C_{i,j} - \sum_{j=0}^{S-1} \sum_{i=1}^{j+1} a_{i,j+1} C_{i,j} \\ &\quad - \sum_{j=1}^{S} \sum_{i=0}^{j-1} a_{i+1,j} C_{i,j} + \sum_{j=0}^{S-1} \sum_{i=0}^{j} a_{i+1,j+1} C_{i,j}. \end{aligned} \tag{55}$$

The rest of the proof simply amounts to grouping the terms as above and imposing the boundary conditions and will be omitted. □

For the special case of $s_i = \gamma i + \delta$, an easy formula for the correlation results:

$$\begin{aligned} Ex_{(k_1;N)} x_{(k_2;N)} = (\gamma S + \delta)^2 &+ \gamma^2 \sum_{j=2}^{S-1} \sum_{i=1}^{j-1} C_{i,j} \\ &- \gamma(\gamma S + \delta) \sum_{i=1}^{S-1} C_{i,S} - \gamma \sum_{j=1}^{S-1} (\gamma j + \delta) C_{j,j}. \end{aligned} \tag{56}$$

Consider the task of calculating the expected value of an L-statistic. The usual way this is done is

$$E \sum_{k=1}^{l} \alpha_k x_{(k;n)} = \sum_{k=1}^{l} \alpha_k E x_{(k;n)} \tag{57}$$

where each $Ex_{(k;n)}$ is calculated separately. It would be much faster to use (47) as

$$E \sum_{k=1}^{l} \alpha_k x_{(k;n)} = s_S \sum_{k=1}^{l} \alpha_k - \sum_{i=1}^{S-1} (s_{i+1} - s_i) \sum_{k=1}^{l} \alpha_k F(k; n)(s_i). \tag{58}$$

5. Conclusion. In this paper, we have presented an algorithm to compute order statistic distributions for independent, arbitrarily distributed, random variables and a

slightly more complicated one for Markov random variables. These algorithms are very fast and should facilitate the study and use of order statistic based estimators for a wide variety of applications. Since the algorithms calculate distributions and not densities, we have presented formulas to calculate first and second moments of discrete order statistics directly from the distributions.

REFERENCES

[1] H. A. DAVID, *Order Statistics*, in Probability and Mathematical Statistics, John Wiley, New York, 1981.
[2] M. KENDALL AND A. STUART, *The Advanced Theory of Statistics*, 1, Charles Griffin, London, 1977, pp. 347-371.
[3] D. F. ANDREWS, F. R. HAMPEL, W. H. ROGERS, P. J. BICKEL, J. W. TUKEY AND P. J. HUBER, *Robust Estimates of Location*, Princeton University Press, Princeton, NJ, 1972.
[4] H. A. DAVID AND R. S. MISHRIKY, *Order statistics for discrete populations and for grouped samples*, J. Amer. Statist. Assoc., 63 (1968), p. 1390.
[5] R. J. VAUGHAN AND W. N. VENABLES, *Permanent expressions for order statistic densities*, J. Roy. Statist. Soc., 34 (1972), pp. 308-310.
[6] E. ATAMAN, V. K. AATRE AND K. M. WONG, *Some statistical properties of median filters*, IEEE Trans. Acoust., Speech, Signal Process., ASSP-29 (Oct. 1981), pp. 1073-1075.
[7] W. MAURER AND B. H. MARGOLIN, *The multivariate inclusion-exclusion formula and order statistics from dependent variates*, Ann. Statist., 4 (1976), pp. 1190-1199.
[8] T. A. NODES AND N. C. GALLAGHER, JR., *The output distribution of median type filters*, IEEE Trans. Comm., COM-32 (May 1984), pp. 532-541.
[9] S. KARLIN AND H. M. TAYLOR, *A First Course in Stochastic Processes*, Academic Press, New York, 1975.
[10] W. FELLER, *An Introduction to Probability Theory and its Applications*, in Probability and Mathematical Statistics, I, John Wiley, New York, 1970.
[11] M. WOODROOFE, *Probability with Applications*, McGraw-Hill, New York, 1975.
[12] D. E. KNUTH, *Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1975.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 6, November 1987
© 1987 Society for Industrial and Applied Mathematics

SYMBOLIC FACTORIZATION FOR SPARSE GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING*

ALAN GEORGE† AND ESMOND NG‡

Abstract. Let $Ax = b$ be a large sparse nonsingular system of linear equations to be solved using Gaussian elimination with partial pivoting. The factorization obtained can be expressed in the form $A = P_1M_1P_2M_2 \cdots P_{n-1}M_{n-1}U$, where P_k is an elementary permutation matrix reflecting the row interchange that occurs at step k during the factorization, M_k is a unit lower triangular matrix whose kth column contains the multipliers, and U is an upper triangular matrix.

Consider the kth step of the elimination. Suppose we replace the structure of row k of the partially reduced matrix by the union of the structures of those rows which are candidates for the pivot row and then perform symbolically Gaussian elimination without partial pivoting. Assume that this is done at each step k, and let $\bar{L}$ and $\bar{U}$ denote the resulting lower and upper triangular matrices, respectively. Then the structures of $\bar{L}$ and $\bar{U}$, respectively, contain the structures of $\sum_{k=1}^{n-1} M_k$ and U. This paper describes an algorithm which determines the structures of $\bar{L}$ and $\bar{U}$, and sets up an efficient data structure for them. Since the algorithm depends only on the structure of A, the data structure can be created in advance of the actual numerical computation, which can then be performed very efficiently using the fixed storage scheme. Although the data structure is more generous than it needs to be for any *specific* sequence $P_1, P_2, \cdots, P_{n-1}$, experiments indicate that the approach is competitive with conventional methods. Another important point is that the storage scheme is large enough to accommodate the QR factorization of A, so it is also useful in the context of computing a sparse orthogonal decomposition of A. The algorithm is shown to execute in time bounded by $|\bar{L}|+|\bar{U}|$, where $|M|$ denotes the number of nonzeros in the matrix M.

Key words. sparse matrix algorithms, Gaussian elimination, partial pivoting, symbolic factorization

AMS(MOS) subject classifications. 65F05, 65F50, 68R10

1. Introduction.

Let A be an $n \times n$ nonsingular matrix and b be an n-vector, and consider the problem of finding an n-vector x such that

$$Ax = b.$$

A standard approach for solving the problem involves reducing A to upper triangular form using elementary row eliminations (i.e., Gaussian elimination). In order to maintain numerical stability, one may have to interchange rows at each step of the elimination process (i.e., use partial pivoting). Thus, we may express the elimination as follows:

$$A = P_1M_1P_2M_2 \cdots P_{n-1}M_{n-1}U$$

where P_k is an $n \times n$ elementary permutation matrix corresponding to the row interchange at step k, M_k is an $n \times n$ unit lower triangular matrix whose kth column contains the multipliers used at the kth step, and U is an $n \times n$ upper triangular matrix. When

* Received by the editors March 22, 1985; accepted for publication (in revised form) January 26, 1987. This research was sponsored by the Natural Sciences and Engineering Research Council of Canada under grant A8111, and was also sponsored by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy under contract DE-AC05-84OR21400 with the Martin Marietta Energy Systems Inc., and by the U.S. Air Force Office of Scientific Research under contract AFOSR-ISS A-84-00056.

† Department of Computer Science and Mathematics, University of Tennessee, Knoxville, Tennessee 37996. This author is also associated with the Oak Ridge National Laboratory through the Distinguished Scientist program, jointly sponsored by the University of Tennessee, and the Oak Ridge National Laboratory.

‡ Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831.

A is sparse, it is well known that fill-in occurs during the triangular decomposition. That is, there may be more nonzeros in $\sum_{k=1}^{n-1} M_k$ and U than in A. Our objective is to develop a scheme that can create from the *structure* of A and *prior* to the numerical decomposition a data structure which can accommodate all the nonzeros in $\sum_{k=1}^{n-1} M_k$ and U. This process is known as *symbolic factorization.* Note that for sparse Gaussian elimination with partial pivoting, the positions of nonzeros in the triangular factors M_k and U depend not only on the structure of the matrix A, but also depend on the row interchanges. Since the row interchanges are not known until the numerical decomposition is performed, the symbolic factorization process must be able to generate a data structure that is large enough to accommodate the nonzeros for *any* valid sequence of elementary row permutations $P_1, P_2, \cdots, P_{n-1}$. Note that some sequences will never occur because of the zeros in the matrix. That is. there will be far fewer than $n!$ possible sequences.

There are important reasons why it is desirable to perform such a symbolic factorization. First, since a symbolic factorization produces a data structure that exploits the sparsity of the triangular factors, the numerical decomposition can be performed using a *static* storage scheme. There is no need to perform storage allocations for the fill-in during the numerical computation. This reduces both storage and execution time overhead for the numerical phase. Second, we obtain from the symbolic factorization a bound on the amount of space we need in order to solve the linear system. This immediately tells us if the numerical computation is feasible. (Of course, this is important only if the symbolic factorization can be performed cheaply both in terms of storage and execution time and if the bound on space is reasonably tight.)

In [10] we have proposed an implementation of Gaussian elimination with partial pivoting for sparse matrices using a static storage scheme. The approach is based on the following results. Assume the diagonal elements of A are nonzero. Then it can be shown that the *structure* of the Cholesky factor L_C of A^TA contains the structure of U^T. That is, if U_{ij} is (structurally) nonzero, then $(L_C)_{ji}$ will also be (structurally) nonzero. Furthermore, the structure of the kth column of L_C contains the structure of the kth column of M_k. These results are *independent* of the choice of the pivot row at each step. Thus we can use the structures of L_C and L_C^T to *bound* the structures of $\sum_{k=1}^{n-1} M_k$ and U, respectively. The advantage of this approach is that if A^TA is sparse (and this is the case in some classes of problems), the structure of the Cholesky factor L_C of A^TA can be determined from that of A^TA efficiently [9] and hence a data structure which exploits the sparsity of L_C and L_C^T can be set up. Thus this provides an efficient symbolic factorization scheme for sparse Gaussian elimination with partial pivoting. Notice that permuting the columns of A corresponds to ordering the rows and columns of A^TA symmetrically. It is well known that for sparse A^TA, the sparsity of the Cholesky factor L_C depends very much on the choice of this symmetric ordering. Thus we may choose a column ordering for A so as to reduce the amount of fill-in in L_C. Numerical experiments have shown that an implementation using the static storage scheme described above, together with a good column ordering for A, is competitive with existing methods for solving certain classes of sparse systems of linear equations [10].

However, in the approach described above, it is often true that the structure of L_C substantially *overestimates* the structure of $\sum_{k=1}^{n-1} M_k$ or U^T. The objective of this paper is to propose a better approach for predicting where nonzeros will appear in sparse Gaussian elimination with partial pivoting, and to present a symbolic factorization algorithm that will generate a tighter data structure which exploits the sparsity of the triangular factors. The basic idea is to allocate space for nonzeros that would

be introduced by all possible pivotal sequences that could occur when Gaussian elimination with partial pivoting is applied to a matrix having the structure of A.

An outline of the paper is as follows. In § 2, the new approach for predicting fill-in and a symbolic factorization algorithm are described. The complexity of the algorithm is also considered. Some enhancements to the algorithm are presented in § 3, and in § 4 some numerical experiments and comparisons are provided. Finally, concluding remarks and open problems are given in § 5.

2. A symbolic factorization algorithm. In the following discussion, it is convenient to assume that the diagonal elements of the $n \times n$ matrix A are nonzero (in which case the matrix A is said to have a *zero-free diagonal*). If A is nonsingular and does not have a zero-free diagonal, it is always possible to permute the rows (or columns) of A so that the permuted matrix has a zero-free diagonal [3]. We will use the notation Nonz (B) to denote the structure of a matrix B; that is,

$$\text{Nonz}(B) = \{(i,j) \mid B_{ij} \neq 0\}.$$

When M is a matrix, $|M|$ denotes the number of nonzeros in M, and when M is a set, $|M|$ denotes the number of elements in M.

Now partition A into

$$A = \begin{pmatrix} \hat{\alpha} & \hat{u}^T \\ \hat{v} & \hat{B} \end{pmatrix},$$

and consider applying one step of Gaussian elimination to A with partial pivoting.

$$\begin{aligned} P_1^T A = P_1^T \begin{pmatrix} \hat{\alpha} & \hat{u}^T \\ \hat{v} & \hat{B} \end{pmatrix} &= \begin{pmatrix} \alpha & u^T \\ v & B \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ v/\alpha & I \end{pmatrix} \begin{pmatrix} \alpha & u^T \\ 0 & B - vu^T/\alpha \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ l & I \end{pmatrix} \begin{pmatrix} \alpha & u^T \\ 0 & A_1 \end{pmatrix}. \end{aligned}$$

Our goal is to be able to allocate space for the nonzeros in l, u and A_1, *regardless of the choice of the pivot row* (that is, the choice of P_1). Our approach is based on the observation that only the first row of A and rows j of A, where $\hat{v}_j \neq 0$, are involved in the elimination. For convenience, row 1 and rows j of A, where $\hat{v}_j \neq 0$, are referred to as *candidate pivot rows.* Thus only the structures of the candidate pivot rows will be affected after the first step of Gaussian elimination with partial pivoting. Moreover, the new structures of these rows depend only on their original structures. In fact, assuming structural and numerical cancellations do not occur, the new structure of each of these rows *must* be contained in the *union* of the structures of all the candidate pivot rows.

The discussion above also explains why we prefer $\hat{\alpha}$ to be nonzero. Suppose $\hat{\alpha}$ is zero. Then we know that the structure of $\hat{u}^T$ is not affected during the elimination except that it is moved to another row in the matrix. However, in order to have sufficient space to store the nonzeros, we have to treat the row $(\hat{\alpha} \quad \hat{u}^T)$ as a candidate pivot row as well. Thus the prediction of fill-in may be too generous. This problem is eliminated if we arrange the rows of A so that $\hat{\alpha}$ is nonzero.

Assuming structural and numerical cancellations do not occur, the union of the structures of the candidate pivot rows is identical to the structure of

$$\bar{u}^T = \hat{u}^T + \hat{v}^T \hat{B}.$$

Hence the discussion above can be summarized as follows.

$$\text{Nonz}(l) = \text{Nonz}(\hat{v}),$$

$$\text{Nonz}(u^T) \subseteq \text{Nonz}(\hat{u}^T + \hat{v}^T\hat{B}),$$

$$\text{Nonz}(A_1) \subseteq \text{Nonz}(\hat{B} - \hat{v}(\hat{u}^T + \hat{v}^T\hat{B})) = \text{Nonz}(\hat{B} - \hat{v}(\hat{u}^T + \hat{v}^T\hat{B})/\hat{\alpha}).$$

It is convenient to denote $\hat{v}$ by $\bar{l}$, and $\hat{B} - \hat{v}(\hat{u}^T + \hat{v}^T\hat{B})/\hat{\alpha}$ by $\bar{A}_1$. We should emphasize that the right-hand sides, which can be computed easily from the structure of A, are *independent* of the choice of the actual pivot row (i.e., the choice of P_1) even though the left-hand sides do depend on P_1. Hence we can use the structures of $\bar{l}$, $\bar{u}$ and $\bar{A}_1$ to *bound* those of l, u and A_1, respectively, *regardless of the choice of* P_1.

It is interesting to note that the decomposition

$$\begin{pmatrix} 1 & 0 \\ \bar{l} & I \end{pmatrix}\begin{pmatrix} \hat{\alpha} & \bar{u}^T \\ 0 & \bar{A}_1 \end{pmatrix}$$

can be obtained by applying one step of Gaussian elimination without row interchanges to the $n \times n$ matrix $\bar{A}$ below:

$$\bar{A} = \begin{pmatrix} \hat{\alpha} & \hat{u}^T + \hat{v}^T\hat{B} \\ \hat{v} & \hat{B} \end{pmatrix}.$$

In terms of structures, what we have done is to replace the first row of A by a row whose structure is the union of the structures of the candidate pivot rows and then to apply one step of Gaussian elimination without partial pivoting.

Since $\hat{B}$ has a zero-free diagonal, $\bar{A}_1 = \hat{B} - \hat{v}(\hat{u}^T + \hat{v}^T\hat{B})/\hat{\alpha}$ must also have a zero-free diagonal under the assumption that exact cancellation does not occur. Thus the argument and idea above can be applied recursively to A_1 and $\bar{A}_1$, yielding a procedure for generating a sequence of vectors whose structures can be used to bound the structures of the triangular factors (M_k and U). As we shall see later, this approach often generates a storage scheme which is much tighter than that provided by the method in [10].

An important advantage of this new approach is that at each step of the symbolic factorization, all we need is the ability to identify the candidate pivot rows and their structures, and the ability to update the structures of these candidate pivot rows by the union of their original structures. Notice that we do not have to be concerned with the actual numerical values or the actual row interchanges that may occur during the numerical factorization. Thus the symbolic factorization procedure can be used to set up a static data structure in advance of the numerical computation, which can then be performed using the fixed storage scheme.

The success of the symbolic factorization procedure described above relies heavily on the efficiency of identifying the candidate pivot rows and updating their structures at each step. Fortunately, these can be done very easily and efficiently. In particular, the symbolic factorization procedure can be implemented with no modification to the structure of A!

An important fact about the structure of the pivot row and column helps us achieve high efficiency. Specifically, it is straightforward to verify that the index of the first nonzero in the pivot column is always at least as large as the index of the first nonzero in the pivot row (after computing the union). This fact is illustrated in Fig. 2.1.

Consider the first step of the symbolic factorization. The structure of row j ($j > 1$) of $\bar{A}_1$ is either unchanged (if the row is not a candidate pivot row and is not involved in the elimination process), or the same as that of $\bar{u}^T$ (if the row is a candidate pivot row and is involved in the elimination process). Thus, all we need is an indicator

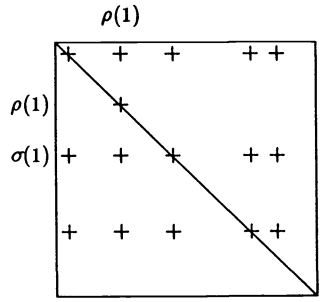


FIG. 2.1. *Positions of nonzeros after the first step of symbolic factorization.*

mask (j) for each row j that tells us whether row j is involved in step 1. In subsequent steps, if *mask* (j) is set, then the structure of row j has been used and its new structure should be obtained from (in this case) $\bar{u}^T$.

We must also maintain information that tells us when the new structure of the modified rows will be used again and where the new structure can be found. Suppose $\bar{l}_{\sigma(1)}, \bar{l}_{\sigma(2)}, \cdots, \bar{l}_{\sigma(p)}$ are nonzero, and $\bar{u}_{\rho(1)}, \bar{u}_{\rho(2)}, \cdots, \bar{u}_{\rho(q)}$ are nonzero. Since the diagonal elements of A (and hence of $\bar{A}$) are assumed to be nonzero, $A_{\sigma(1),\sigma(1)} \neq 0$, $A_{\rho(1),\rho(1)} \neq 0$, and $\rho(1) \leqq \sigma(1)$. Note that A_{ij} must be zero for $2 \leqq j < \rho(1)$, $i = \sigma(1), \sigma(2), \cdots, \sigma(p)$. See Fig. 2.1 for an illustration. Thus the new structures of the modified rows will be used again (at the earliest) at step $\rho(1)$. We should emphasize that the new structures are identical to that of $\bar{u}^T$. Also, at step $\rho(1)$, the candidate pivot rows will include those (whose row indices are greater than $\rho(1)$) of step 1, but the candidate pivot rows of step 1 could be conveniently found by examining the structure of $\bar{l}$. To summarize, the only other information we have to record is the fact that at step $\rho(1)$, we have to look at $\bar{l}$ and $\bar{u}$ to obtain the correct structure of the modified matrix. In the algorithm described below, we have used a set $S_{\rho(1)}$ for this purpose. After the first step, we have $S_{\rho(1)} = \{1\}$.

At this point, it is appropriate to present the entire symbolic factorization algorithm, which is a generalization of what we have just described. The structure of A is represented by two sets.

(1) C_j contains the structure of column j of A:

$$C_j = \{i \mid A_{ij} \neq 0\}.$$

(2) R_i contains the structure of row i of A:

$$R_i = \{j \mid A_{ij} \neq 0\}.$$

Similarly, the structures of the sequence of vectors obtained from the symbolic factorization algorithm are represented by sets.

(1) $\hat{L}_k$ contains the structure of a column vector that bounds the structure of column k of M_k.

(2) $\hat{U}_k$ contains the structure of a row vector that bounds the structure of row k of U.

SYMBOLIC FACTORIZATION ALGORITHM

(1) Global initialization

(1.1) Set *mask* $(k) = 0$, for $k = 1, 2, \cdots, n$ to denote that no rows have been considered yet.

(1.2) Set $S_k = \emptyset$, for $k = 1, 2, \cdots, n$ to denote that step k has not yet been influenced by any previous steps.

(2) For $k = 1, 2, \cdots, n$, do the following:

(2.1) Local initialization: Set $\hat{L}_k = \emptyset$ and $\hat{U}_k = \emptyset$.

(2.2) Obtain structures of those unprocessed rows: For $i \in C_k$, if *mask* $(i) = 0$, then set $\hat{L}_k = \hat{L}_k \cup \{i\}$ and $\hat{U}_k = \hat{U}_k \cup R_i$, and set *mask* $(i) = 1$ (to indicate that row i has been looked at already).

(2.3) If $S_k = \emptyset$, then go to Step (2.5). (Step k is not influenced by any previous steps.)

(2.4) If $S_k \neq \emptyset$, then $\hat{L}_k$ and $\hat{U}_k$ should include $\hat{L}_i$ and $\hat{U}_i$, respectively, for $i \in S_k$. Hence, for $i \in S_k$, collect influence caused by step i:

$$\hat{L}_k = \hat{L}_k \cup \hat{L}_i - \{1, \cdots, k-1\},$$

$$\hat{U}_k = \hat{U}_k \cup \hat{U}_i - \{1, \cdots, k-1\}.$$

(2.5) Update S: Let $\sigma = \min\{t \mid t \in \hat{L}_k - \{k\}\}$ and $\rho = \min\{t \mid t \in \hat{U}_k - \{k\}\}$. Since $\rho \leqq \sigma$, the structure of $\hat{L}_k$ and $\hat{U}_k$ will influence the determination of the structures at step ρ (if $|\hat{L}_k| > 1$). Hence set $S_\rho = S_\rho \cup \{k\}$ if $|\hat{L}_k| > 1$.

The complexity of the algorithm is linear in the number of elements in $\cup_i \hat{L}_i$ and $\cup_i \hat{U}_i$ as the following discussion shows. Since each i, $i = 1, 2, \cdots, n$, will be included in at most one S_ρ, $S_\rho \cap S_{\rho'} = \emptyset$, for $\rho \neq \rho'$. Thus we access the elements of $\hat{L}_i$ and $\hat{U}_i$ at most once for each i (see Step (2.4)). Furthermore, we access the elements of C_i and R_i at most once in this algorithm (see Step (2.2)). Hence the complexity of the algorithm is

$$O\left(\max\left(|A|, \sum_{k=1}^{n-1} |\hat{L}_k| + |\hat{U}_k|\right)\right).$$

The overhead in storage is also very small. There are n elements in *mask* for marking the rows. Since at step k, k is included in at most one set S_ρ, $\sum_i |S_i| \leqq n$.

For convenience, we define a lower triangular matrix $\bar{L}$ so that $\bar{L}_{ij} \neq 0$ if $i \in \hat{L}_j$ and $\bar{L}_{ij} = 0$ otherwise. Similarly, let $\bar{U}$ be an upper triangular matrix such that $\bar{U}_{ij} \neq 0$ if $j \in \hat{U}_i$ and $\bar{U}_{ij} = 0$ otherwise. Hence,

$$\text{Nonz}(U) \subseteq \text{Nonz}(\bar{U})$$

and

$$\text{Nonz}\left(\sum_{k=1}^{n-1} M_k\right) \subseteq \text{Nonz}(\bar{L}).$$

In other words, a static storage scheme that exploits the sparsity of $\bar{L}$ and $\bar{U}$ is always large enough to accommodate the fill-in in M_k and U, regardless of the choice of pivot rows.

It is interesting to note that the static data structure obtained from the symbolic factorization algorithm also has enough space to accommodate fill-in produced in computing an orthogonal decomposition of A using Householder transformations. Partition A as before into

$$A = \begin{pmatrix} \hat{\alpha} & \hat{u}^T \\ \hat{v} & \hat{B} \end{pmatrix}.$$

Suppose H is an $n \times n$ Householder transformation that reduces

$$\begin{pmatrix} \hat{\alpha} \\ \hat{\nu} \end{pmatrix} \quad \text{to} \quad \begin{pmatrix} \bar{\alpha} \\ 0 \end{pmatrix}.$$

The matrix H has the form

$$H = I - \frac{1}{\pi}\begin{pmatrix} \beta \\ \nu \end{pmatrix}(\beta \quad \nu^T),$$

where $\text{Nonz}(\nu) = \text{Nonz}(\hat{\nu})$. Thus,

$$HA = \left(I - \frac{1}{\pi}\begin{pmatrix} \beta \\ \nu \end{pmatrix}(\beta \quad \nu^T)\right)\begin{pmatrix} \hat{\alpha} & \hat{u}^T \\ \hat{\nu} & \hat{B} \end{pmatrix}$$

$$= \begin{pmatrix} \bar{\alpha} & \hat{u}^T - \frac{1}{\pi}(\beta^2 \hat{u}^T + \beta \nu^T \hat{B}) \\ 0 & \hat{B} - \frac{1}{\pi}(\beta \nu \hat{u}^T + \nu \nu^T \hat{B}) \end{pmatrix}.$$

Hence

$$\text{Nonz}(\nu) = \text{Nonz}(\hat{\nu}) = \text{Nonz}(\bar{l}),$$

$$\text{Nonz}\left(\hat{u}^T - \frac{1}{\pi}(\beta^2 \hat{u}^T + \beta \nu^T \hat{B})\right) = \text{Nonz}(\hat{u}^T + \hat{\nu}^T \hat{B}) = \text{Nonz}(\bar{u}^T),$$

and

$$\text{Nonz}\left(\hat{B} - \frac{1}{\pi}(\beta \nu \hat{u}^T + \nu \nu^T \hat{B})\right) = \text{Nonz}(\hat{B} - \hat{\nu}(\hat{u}^T + \hat{\nu}^T \hat{\beta})/\hat{\alpha}) = \text{Nonz}(\bar{A}_1).$$

The argument can again be applied recursively to $\bar{A}_1$ and to $\hat{B} - (1/\pi)(\beta \nu \hat{u}^T + \nu \nu^T \hat{B})$. Thus, the symbolic factorization algorithm described above can be used to generate a static storage scheme for sparse orthogonal decomposition using Householder transformations. Note that both the upper triangular matrix and the Householder transformations can be saved in the data structure.

Although the symbolic factorization algorithm has been described in terms of $n \times n$ matrices, with some slight modifications it can also be applied to rectangular matrices. The lower trapezoidal portion of $\bar{L}$ and the upper trapezoidal portion of $\bar{U}$ are stored, respectively, by columns and rows. If the matrix A is $m \times n$, then the *mask* vector will be of size m and the number of elements in the sets S_i will be at most $\min\{m, n\}$.

3. Enhancements.

3.1. Subscript compression. From the sets $\{\hat{L}_i\}$ and $\{\hat{U}_i\}$, we can generate a storage scheme for storing the nonzeros of $\bar{L}$ and $\bar{U}$ (and hence the nonzeros of M_k and U). For example, consider the upper triangular matrix $\bar{U}$. One way to represent $\bar{U}$ is to store the nonzeros row by row in a floating-point array UNZ, since the structure of $\bar{U}$ is generated row by row. Thus we need an integer array XUNZ to store the beginning positions in UNZ of the nonzeros in the rows of $\bar{U}$. More specifically, the nonzeros of row i of $\bar{U}$ are found in $\text{UNZ}(p)$, for $p = \text{XUNZ}(i), \text{XUNZ}(i)+1, \cdots, \text{XUNZ}(i+1)-1$. (It is assumed that $\text{XUNZ}(n+1) = 1 + \sum_i |\hat{U}_i|$.) It is also

necessary to store the column subscripts of the numbers in UNZ. Hence we need an additional integer array SUBU where SUBU (p) contains the column index of the nonzero stored in UNZ (p). The lower triangular matrix $\bar{L}$ is represented column by column using a similar storage scheme. We refer to this storage scheme as the *uncompressed storage scheme.* If we use this scheme, we need $\sum_i (|\hat{L}_i|+|\hat{U}_i|)$ floating-point locations for the numerical values, $\sum_i (|\hat{L}_i|+|\hat{U}_i|)$ integer locations for the subscripts, and $2n+2$ integer locations for the pointers. If an integer location and a floating-point location have the same number of bits, then the total number of locations required by the uncompressed storage scheme is approximately

$$2\sum_i (|\hat{L}_i|+|\hat{U}_i|).$$

It has been observed from experiments that very often the set of subscripts of a row (say row i) of $\bar{U}$ is a subsequence of the subscripts of a previous row (in particular, row $i-1$). This is also true for the columns of the lower triangular matrix $\bar{L}$. Thus, it makes sense to remove these redundancies using a process referred to as *subscript compression,* so that the total number of subscripts that have to be stored is smaller than $\sum_i (|\hat{L}_i|+|\hat{U}_i|)$. The use of subscript compression is common in storage schemes for solving sparse symmetric positive definite systems, and is due to Sherman [12].

On the other hand, if the subscripts are compressed, there is no longer a one-to-one correspondence between the numerical values in, say UNZ, and the subscripts in SUBU. We need additional information in order to be able to retrieve the subscripts. For example, consider the upper triangular matrix $\bar{U}$ again. We now need an extra integer array XSUBU of size n with XSUBU (i) indicating where we can find the begining of the subscript sequence for the nonzeros in row i. That is, the nonzeros of row i are stored in UNZ (p), for $p = \text{XUNZ}(i), \text{XUNZ}(i)+1, \cdots, \text{XUNZ}(i+1)-1$. The corresponding column subscripts are stored in SUBU (q), for $q = \text{XSUBU}(i)$, $\text{XSUBU}(i)+1, \cdots$. Of course, the length of the subscript sequence is simply given by $\text{XUNZ}(i+1)-\text{XUNZ}(i)$. This storage scheme is widely known as the *compressed storage scheme,* and was first employed by Sherman [12]. As we will see in the numerical experiments presented in the next section, the additional storage required by XSUBL and XSUBU is usually more than offset by the saving in storage for the subscripts. However, since additional indexing is needed in order to retrieve the nonzeros of a row of $\bar{U}$ or a column of $\bar{L}$, the numerical computation phase using the compressed storage scheme is expected to be slower than that using the uncompressed storage scheme.

There are two techniques employed in subscript compression. First, after we have determined the structure of the kth row of $\bar{U}$ (or kth column of $\bar{L}$), we can compare it with that of the $(k-1)$st row of $\bar{U}$ (or $(k-1)$st column of $\bar{L}$) to find out if the "tail" of the latter sequence is the same as the "head" of the former one. If they are the same, then we do not have to store the "head" of the subscript sequence for the current row (or column).

Another observation not only allows us to quickly compress the subscripts, but also improves the execution time of the algorithm. Often, particularly during later stages of the algorithm, there remain no unprocessed rows at step i, and the set S_i (defined in § 2) has only one element. (In particular, it is often true that after the symbolic factorization algorithm has gone through far fewer than n steps, all n rows of A may have been processed.) Suppose $S_i = \{l\}$. This simply means that the structures of $\hat{L}_i$ and $\hat{U}_i$ are determined entirely by $\hat{L}_l - \{i\}$ and $\hat{U}_l - \{i\}$, respectively. Hence, it is unnecessary to access the elements of $\hat{L}_l$ and $\hat{U}_l$. This allows us to very quickly and

inexpensively compress the subscripts, which in turn usually reduces the execution time of the algorithm.

The techniques described above have been incorporated into our implementation, and numerical experiments have indicated that there is indeed a significant reduction in both the execution time of the symbolic factorization algorithm and the total number of subscripts that have to be stored.

3.2. Column ordering. At each step of the symbolic factorization algorithm, we replace the structures of the candidate pivot rows by the union of their original structures. This operation is independent of the initial row ordering. That is, the number of nonzeros ($|\bar{L}|$ and $|\bar{U}|$) determined by the symbolic factorization algorithm is constant, regardless of the choice of the initial row ordering, as long as this row ordering preserves the zero-free diagonal. (However, the choice of row ordering may have an effect on the extent to which subscript compression can be achieved.)

Column orderings, on the other hand, may affect the sparsity of the triangular matrices $\bar{L}$ and $\bar{U}$, which is indirectly related to the sparsity of the triangular factors $\sum_{k=1}^{n-1} M_k$ and U. (Again, the column ordering has to preserve the zero-free diagonal.) Consider the two matrices A and B below, where B is obtained by interchanging the first and the last columns of A.

$$A = \begin{pmatrix} \times & \times & \times & \times & \times \\ & \times & & & \times \\ & & \times & & \times \\ & & & \times & \times \\ \times & & & & \times \end{pmatrix}, \qquad B = \begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & & & \\ \times & & \times & & \\ \times & & & \times & \\ \times & & & & \times \end{pmatrix}.$$

The union of the structures of the matrices $\bar{L}$ and $\bar{U}$ in each case is displayed below.

$$\bar{L}_A + \bar{U}_A: \begin{pmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix}, \qquad \bar{L}_B + \bar{U}_B: \begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix}.$$

Hence it is important to choose a column ordering for A so that the triangular matrices $\bar{L}$ and $\bar{U}$ are sparse.

In our experiments, since we want to compare the storage scheme generated by the new symbolic factorization algorithm with that created using the approach described in [10], we have used a column ordering that is in general effective for that method, which uses the structures of the Cholesky factors of A^TA to bound the structures of the triangular factors $\sum_{k=1}^{n-1} M_k$ and U of A. Hence it is the sparsity of the Cholesky factors of A^TA that has to be preserved. It is well known that when A^TA is sparse, a minimum degree ordering is an effective ordering in reducing fill-in in its Cholesky factors [11]. Hence we use that ordering to permute the columns of A before we apply our symbolic factorization algorithm to A.

A natural alternative strategy for ordering the columns of A is as follows. Consider the first step of the symbolic factorization algorithm. Since we have to replace the structures of the candidate pivot rows by the union of their original structures, a $p \times q$ dense submatrix is introduced into the matrix, where p is the number of candidate pivot rows and q is the number of nonzeros in the union of these candidate pivot rows. By rearranging, the columns of A, p and q will be changed. Thus we could

permute the columns (and rows) before we carry out this step of symbolic factorization so that p, q or even the product pq is small (or minimized). Efficient implementation of these ideas is under investigation.

3.3. Dense rows and dense columns. In some applications, the matrix A may have a few rows or columns that are relatively dense. Since the method proposed in [10] works with A^TA, dense rows of A form large dense submatrices in A^TA. In our present approach, the effect of having some dense rows in A may be less severe, so long as when one of these dense rows becomes a candidate pivot row, the number of candidate pivot rows at that time is small. However, since we order the rows and columns of A^TA to obtain a column ordering for A, these dense rows may affect the quality of the column ordering because the sparsity of the remaining rows of A may be lost when we form A^TA. One technique we have found useful is to remove dense rows from A when we order the columns of A and then to put them back into A when we carry out symbolic factorization. Some of the numerical experiments presented in the next section will illustrate the improvement that may be achieved.

The effect of dense columns is not as clear as that of dense rows. Suppose the last column of A is dense. It should be clear that such a dense column should have no effect on the sparsity of $\bar{L}$ and $\bar{U}$. On the other hand, if the first column of A is dense, the number of candidate pivot rows at the first step of symbolic factorization is large and the union of these candidate pivot rows is likely to be quite dense. Even if the union is sparse, more nonzeros will likely be added to this union as the symbolic factorization algorithm proceeds. Thus, it would appear to be desirable to have any dense columns appear least.

Instead of working with A^TA and A, we may work with AA^T and A^T since a triangular factorization of A^T is as useful as a triangular factorization of A. Dense rows and dense columns in A now become, respectively, dense columns and dense rows in A^T. In terms of applying the symbolic factorization algorithm to A and A^T, the effect of having dense rows and dense columns in each case is in general different.

As an example, suppose all the rows of A are sparse, but its first column is dense. Then applying the symbolic factorization algorithm to A may result in substantial fill-in in $\bar{L}$ and $\bar{U}$. However, if we work with A^T, then the first row of A^T is now dense and A^T has no dense columns. We can apply the symbolic factorization algorithm to A^T. Together with the row withholding column ordering strategy mentioned above, this latter approach may generate a sparse $\bar{L}$ and $\bar{U}$.

Clearly, there are two related problems associated with exploiting these ideas. One is to determine when a row or column should be considered dense, and the other is to decide whether to work with A or A^T. Answers to these questions are probably problem dependent, and require further study. Our experiments provide some evidence about the potential effectiveness of the techniques.

4. Numerical experiments. In this section, we present some numerical experiments to illustrate the efficiency of our symbolic factorization algorithm and to compare its performance with the method proposed in [10], and with the well-known Harwell MA28 code [2].

For the new scheme, and the method proposed in [10], the experiments involved the solution of sparse systems of linear equations

$$P^T(QA)PP^Tx = P^T(Qb)$$

where A and b are, respectively, the coefficient matrix and the right-hand side vector, Q is a permutation matrix chosen so that QA has a zero-free diagonal, and P is a

permutation matrix corresponding to a minimum degree ordering of A^TA. (We premultiply QAP by P^T since we want to preserve the zero-free diagonal.) Symbolic factorization is applied to the matrix $B = P^TQAP$ to create a data structure for the numerical decomposition of B. The triangular factors M_k and U, which are computed using Gaussian elimination with partial pivoting, are stored in the resulting data structure.

All experiments were carried out on a DEC VAX 11/780 computer. The programs were written in ANSI standard FORTRAN using single precision floating-point arithmetic, and were compiled using the Berkeley FORTRAN 77 compiler. In the discussion below, storage requirements are expressed in terms of number of storage locations required and execution times are in seconds. (Note that one floating-point location and one integer location have the same number of bits.)

Our experiments involved three sets of test problems typical of those arising in scientific and engineering applications. The first set consists of nine finite element problems having the structure of those described in [8]. These problems are unsymmetric, although they have symmetric structure. Their numerical values were generated using a uniform random number generator. The second set is a collection of problems arising from chemical engineering calculations, and the third set consists of problems arising from various applications, such as surveying and linear programming. The second and third sets of problems were kindly provided by Duff, Grimes, Lewis and Poole [5]. Numerical values were provided for most of the problems in the last two sets. For problems that do not have numerical values, we generated values using a uniform random number generator. The right-hand side vector for each problem was chosen so that the solution vector consisted of all ones. The characteristics of the problems are summarized in Table 4.1.

In presenting the results, we have employed the following notation:

(1) *store* denotes the number of storage locations required;
(2) *time* denotes the execution time (in seconds) required;
(3) *lower* and *upper* refer to the lower and upper triangular portions of a matrix, respectively;
(4) *factor time* and *solve time* are, respectively, the execution times (in seconds) required for the numerical factorization and numerical triangular solutions;
(5) *percentage utilization* is the ratio of the actual number of off-diagonal nonzeros obtained in the numerical factorization to the number of off-diagonal nonzeros accommodated by the static data structure obtained from a symbolic factorization algorithm.

Also for convenience, the approach described in [10] is referred to as "SF-A^TA," and the new symbolic factorization algorithm described in § 2 of this paper is referred to as "SF-A." The new symbolic factorization algorithm with subscript compression as discussed in § 3.1 is abbreviated as "SFC-A."

For comparison purposes, we first summarize in Table 4.2 the results of using SF-A^TA.

The first experiment we did involved implementing the symbolic factorization algorithm described in § 2, and then using the algorithm to generate a static data structure for sparse Gaussian elimination with partial pivoting. Numerical factorization and triangular solutions were performed using that fixed storage scheme. The results are presented in Table 4.3. A comparison of the results in Tables 4.2 and 4.3 leads us to the following observations.

Note that the number of nonzeros in the lower triangular matrix $|\bar{L}|$ determined by the SF-A symbolic factorization algorithm is smaller than that in the Cholesky factor L_C of A^TA predicted in SF-A^TA. This is true in all test problems, and the

TABLE 4.1
Characteristics of test problems.

Problem	Order	Number of nonzeros	Number of nonzeros per row		Number of nonzeros per column		Comment
			Min	Max	Min	Max	
1	936	6,264	4	7	4	7	small hole square (finite element mesh)
2	1,009	6,865	4	7	4	7	graded L (finite element mesh)
3	1,089	7,361	3	7	3	7	plain square (finite element mesh)
4	1,440	9,504	3	7	3	7	large hole square (finite element mesh)
5	1,180	7,750	3	7	3	7	+ shaped domain (finite element mesh)
6	1,377	8,993	3	7	3	7	H shaped domain (finite element mesh)
7	1,138	7,450	4	7	4	7	3 hole problem (finite element mesh)
8	1,141	7,465	4	7	4	7	6 hole problem (finite element mesh)
9	1,349	9,101	4	7	4	7	pinched hole problem (finite element mesh)
10	156	371	1	7	1	6	simple chemical plant model (chemical engineering problem)
11	167	507	1	9	1	19	rigorous model of a chemical stage (chemical engineering problem)
12	381	2,157	1	25	1	50	multiply fed column, 24 components (chemical engineering problem)
13	132	414	1	9	1	19	rigorous flash unit (chemical engineering problem)
14	67	294	1	6	2	10	cavett problem with 5 components (chemical engineering problem)
15	655	2,854	1	12	1	35	16 stage column section, some stages simplified (chemical engineering problem)
16	479	1,910	1	12	1	35	8 stage column section, all sections rigorous (chemical engineering problem)
17	497	1,727	1	28	1	55	rigorous flash unit with recycling (chemical engineering problem)
18	989	3,537	1	12	1	26	7 stage column section, all sections rigorous (chemical engineering problem)
19	113	655	1	20	1	27	unsymmetric pattern supplied by Morven Gentleman
20	199	701	1	6	2	9	unsymmetric pattern of order 199 given by Willoughby
21	130	1,282	1	124	1	124	unsymmetric matrix from laser problem given by A. R. Curtis
22	663	1,712	1	426	1	4	unsymmetric basis from LP problem (Shell)
23	363	3,279	1	33	1	34	unsymmetric basis from LP problem (Stair)
24	822	4,841	1	304	1	21	unsymmetric basis from LP problem (BP)
25	541	4,285	1	11	5	541	unsymmetric facsimile convergence matrix

reductions are large in some cases. As a result, the percentage utilization for the lower triangular matrix $\bar{L}$ is significantly improved using the SF-A approach. That is, in terms of accommodating the fill-in in $\sum_{k=1}^{n-1} M_k$, the structure of $\bar{L}$ is much tighter than the structure of the Cholesky factor L_C of A^TA.

Note also that the number of nonzeros in the upper triangular matrix $\bar{U}$ determined by the SF-A symbolic factorization algorithm is either identical or close to that in L_C^T in SF-A^TA. We believe the explanation for this is as follows. First, it is known that L_C^T and the R obtained from the QR factorization of A have identical structures when

TABLE 4.2
Symbolic factorization using A^TA (SF-A^TA).

	Symbolic factorization					Numerical solution			Percentage utilization	
				Number of nonzeros						
Problem	Store	Time	Number of subscripts	Lower	Upper	Store	Factor time	Solve time	Lower	Upper
1	30,620	1.083	7,713	37,554	37,554	91,247	57.183	1.617	50.36	68.82
2	34,159	1.233	8,914	46,945	46,945	111,887	78.617	1.983	50.13	67.29
3	36,244	1.333	9,227	47,484	47,484	113,998	80.383	2.033	50.50	66.65
4	45,900	1.600	11,317	49,273	49,273	122,825	62.683	2.150	50.01	65.52
5	36,751	1.283	8,588	26,570	26,570	72,350	20.650	1.233	49.28	64.03
6	42,304	1.417	9,699	28,556	28,556	79,206	20.550	1.350	49.24	64.61
7	36,507	1.283	9,460	35,291	35,291	90,286	39.783	1.550	50.43	65.14
8	37,169	1.250	10,040	38,748	38,748	97,807	49.867	1.700	50.42	66.82
9	43,842	1.517	10,465	53,161	53,161	128,930	72.250	2.300	49.34	63.55
10	2,348	0.050	417	566	566	2,955	0.150	0.033	29.68	38.52
11	3,276	0.083	559	875	875	3,814	0.217	0.050	30.51	40.11
12	23,172	1.050	6,993	18,334	18,334	47,092	14.933	0.833	32.47	36.27
13	2,645	0.100	466	742	742	3,140	0.183	0.050	27.49	40.57
14	1,746	0.067	385	843	843	2,676	0.267	0.033	34.99	42.35
15	21,058	0.800	5,143	12,906	12,906	36,852	7.600	0.633	24.14	39.12
16	13,617	0.500	3,118	7,383	7,383	22,197	4.267	0.350	31.79	38.48
17	15,416	0.550	2,223	6,363	6,363	19,424	1.783	0.300	13.30	24.96
18	23,907	0.750	4,746	8,143	8,143	29,935	2.500	0.450	23.04	37.34
19	3,702	0.133	557	1,322	1,322	4,220	0.500	0.067	22.54	64.45
20	4,364	0.117	1,143	2,194	2,194	7,324	1.017	0.117	36.46	56.61
21	16,829	2.000	260	7,763	7,763	16,958	9.817	0.350	67.51	48.78
22	188,207	59.700	964	91,080	91,080	189,093	14.567	3.733	0.55	1.54
23	15,420	0.633	4,007	7,772	7,772	22,820	4.567	0.383	24.14	49.83
24	118,904	21.250	6,809	57,817	57,817	129,843	37.467	2.417	11.79	15.02
25	24,429	0.900	6,634	16,600	16,600	44,705	12.950	0.767	40.37	46.77

A is irreducible [1]. Moreover, we have shown at the end of § 2 that the structures of $\bar{U}$ and R are identical. Thus, when A is irreducible, we would *expect* the structure of the upper triangular matrix $\bar{U}$ to be identical to that of L_C^T.

When A is reducible, the situation is quite different, as the following example illustrates:

$$A = \begin{pmatrix} \times & \times & \times & \times & \times & \times & \times \\ & \times & & & & & \times \\ & & \times & & & & \times \\ & & & \times & & & \times \\ & & & \times & \times & & \times \\ & & \times & & & \times & \times \\ & \times & & & & & \times \end{pmatrix} \rightarrow \text{symb. fact.} \rightarrow \begin{pmatrix} \times & \times & \times & \times & \times & \times & \times \\ & \times & & & & & \times \\ & & \times & & & \times & \times \\ & & & \times & \times & & \times \\ & & & \times & \times & & \times \\ & & \times & & & \times & \times \\ & \times & & & & & \times \end{pmatrix}.$$

Clearly A^TA is dense, and L_C is full lower triangular. Apparently, many of our test problems are irreducible, or are nearly so in the sense of having relatively few irreducible blocks.

TABLE 4.3
Symbolic factorization algorithm of § 2 (SF-A). (*Without subscript compression.*)

	Symbolic factorization						Numerical solution			Percentage utilization	
			Number of subscripts		Number of nonzeros						
Problem	Store	Time	Lower	Upper	Lower	Upper	Store	Factor time	Solve time	Lower	Upper
1	79,755	3.550	20,310	37,554	20,310	37,554	124,154	39.833	1.267	93.13	68.82
2	96,533	4.117	25,765	46,945	25,765	46,945	154,503	59.017	1.600	91.33	67.29
3	98,974	4.450	25,875	47,484	25,875	47,484	156,521	57.617	1.550	92.67	66.65
4	109,146	5.017	26,462	49,273	26,462	49,273	164,432	44.017	1.633	93.11	65.52
5	68,025	3.033	14,152	26,570	14,152	26,570	92,066	14.383	0.917	92.52	64.03
6	75,615	3.417	15,300	28,556	15,300	28,556	100,107	14.383	1.017	91.91	64.61
7	80,603	3.550	19,029	35,291	19,029	35,291	118,884	28.350	1.150	93.53	65.14
8	86,013	3.733	20,922	38,748	20,922	38,748	129,611	35.433	1.250	93.38	66.82
9	113,251	4.900	28,395	53,161	28,395	53,161	175,255	49.917	1.700	92.38	63.55
10	3,105	0.167	256	544	256	544	3,006	0.100	0.050	65.63	40.07
11	3,866	0.217	349	830	349	830	3,863	0.150	0.033	76.50	42.29
12	34,446	1.667	8,780	17,539	8,780	17,539	56,069	10.617	0.550	67.80	37.91
13	3,109	0.150	268	690	268	690	3,106	0.117	0.050	76.12	43.62
14	2,530	0.133	426	843	426	843	3,143	0.217	0.017	69.25	42.35
15	29,413	1.517	5,358	11,794	5,358	11,794	40,201	5.233	0.383	58.16	42.81
16	19,514	1.033	3,698	7,203	3,698	7,203	26,115	3.183	0.250	63.47	39.44
17	15,622	0.783	1,447	5,748	1,447	5,748	18,865	0.950	0.183	58.47	27.63
18	27,303	1.433	2,771	7,565	2,771	7,565	29,575	1.600	0.300	67.70	40.20
19	4,078	0.217	313	1,322	313	1,322	4,289	0.300	0.050	95.21	64.45
20	6,549	0.317	960	2,194	960	2,194	8,101	0.733	0.083	83.33	56.61
21	19,032	0.867	7,402	7,763	7,402	7,763	31,502	8.567	0.317	70.81	48.78
22	112,290	4.983	11,153	91,080	11,153	91,080	210,435	6.133	1.967	4.52	1.54
23	19,725	0.950	2,562	6,972	2,562	6,972	22,337	2.267	0.217	73.22	55.55
24	91,038	4.050	16,574	56,559	16,574	56,559	153,666	19.200	1.517	41.14	15.35
25	38,193	1.800	7,610	16,600	7,610	16,600	53,291	9.017	0.517	88.07	46.77

The SF-A symbolic factorization algorithm described in § 2 is usually slower than the SF-A^TA symbolic factorization algorithm. (There are a few instances where the SF-A algorithm runs faster than the SF-A^TA algorithm. For example, see Problems 22 and 24.) In most cases, the execution time of the SF-A symbolic factorization algorithm is about twice to three times that of the SF-A^TA symbolic factorization algorithm. In the SF-A^TA algorithm, the structures of the Cholesky factors L_C and L_C^T of A^TA are used to bound the structures of the triangular factors $\sum_{k=1}^{n-1} M_k$ and U. Only one set of subscripts is needed to describe the structures of L_C and L_C^T. However, in the SF-A symbolic factorization algorithm, the structure of $\bar{L}$ is in general not the same as the structure of $\bar{U}^T$. Thus two sets of subscripts are needed, one set for describing the structure of the lower triangular matrix and another for describing the structure of the upper triangular matrix. Consequently, extra work is done in the SF-A algorithm to generate these two sets of subscripts, so it is not surprising that the SF-A algorithm runs slower than the SF-A^TA algorithm. The discussion above also explains why the storage requirements for the symbolic factorization and numerical solution in Table 4.3 are larger than those in Table 4.2, since in SF-A, there is exactly one subscript

for every nonzero stored. This is not the case in SF-A^TA because subscript compression is employed in the symbolic factorization algorithm.

A final observation in connection with Tables 4.2 and 4.3 is that the execution times of the numerical factorization and triangular solution are smaller for SF-A than those for SF-A^TA. One reason is that there are fewer nonzeros in $\bar{L}+\bar{U}$ in SF-A than in $L_C+L_C^T$ in SF-A^TA. In addition, the uncompressed storage scheme created by the SF-A symbolic factorization algorithm is simpler than the compressed storage scheme generated by the SF-A^TA symbolic factorization algorithm. We will return to this later.

The percentage utilization for Problem 22 is extremely small for both SF-A^TA and SF-A. This is caused by a few dense rows in the coefficient matrix A, as one can see from Table 4.1. These dense rows effectively destroy the sparsity of some of the other rows when we determine a column ordering from A^TA. As we shall see later in this section, there will be substantial improvements when we withhold these dense rows from A in determining a column ordering.

The analysis in § 2 showed that the complexity of the SF-A symbolic factorization algorithm is $O(\max(|A|, \sum_k |\hat{L}_k|+|\hat{U}_k|))$. That is, for large sparse problems, the complexity of the SF-A symbolic factorization algorithm is essentially proportional to the number of subscripts generated by the algorithm. To verify this, we applied the SF-A algorithm to a sequence of finite element problems defined on graded-L meshes. The results, given in Table 4.4, indeed suggest that the complexity of the new symbolic factorization algorithm is $O(\max(|A|, \sum_k |\hat{L}_k|+|\hat{U}_k|))$.

TABLE 4.4
Applying SF-A *to a sequence of graded-L finite element problems.*

Order	Number of nonzeros in A	Time t	Number of subscripts			Ratio t/nz ($\times 10^5$)
			Lower	Upper	Total (nz)	
265	1,753	0.750	3,942	7,088	11,030	6.800
406	2,716	1.300	7,377	13,104	20,481	6.347
577	3,889	2.067	11,520	21,058	32,578	6.345
778	5,272	3.100	18,697	33,433	52,130	5.947
1,009	6,865	4.150	25,765	46,945	72,710	5.708
1,270	8,668	5.500	34,376	62,783	97,159	5.661
1,561	10,681	7.350	47,797	86,720	134,517	5.464
1,882	12,904	8.967	57,384	105,836	163,220	5.494
2,233	15,337	11.367	73,112	134,773	207,885	5.468
2,614	17,980	13.650	91,393	167,508	258,901	5.272
3,025	20,833	16.467	111,237	203,579	314,816	5.231
3,466	23,896	19.417	132,385	242,870	375,255	5.174

In terms of the number of nonzeros accommodated by the data structure, the results presented so far have shown that the symbolic factorization algorithm SF-A is considerably better than SF-A^TA. However, in terms of storage and time required, the SF-A algorithm is not as efficient as the SF-A^TA algorithm. The high storage requirement is due to the fact that there is one subscript for every nonzero determined by the SF-A algorithm. If the number of nonzeros determined by the SF-A algorithm is η, then we require at least η locations to execute the SF-A symbolic factorization algorithm, since we need η locations to store the subscripts. Consequently, we need at least 2η locations to perform the numerical solution.

In § 3, subscript compression was proposed to reduce the storage and execution time required by the SF-A symbolic factorization algorithm. Our next experiment involved incorporating subscript compression in the implementation of the new symbolic factorization algorithm (SFC-A), and was intended to illustrate the improvement that may be achieved. The results in Table 4.5 show that the new symbolic factorization algorithm with subscript compression performs extremely well. The number of subscripts that have to be stored is much smaller than that required in SF-A. Also note that even though the storage scheme in SFC-A is more complicated, the reduction in the number of subscripts is large enough to offset the additional storage required for the more sophisticated storage scheme. As a result, the storage requirements for both the symbolic factorization phase and the numerical solution phase associated with SFC-A are much smaller than those required by SF-A. Indeed, in most cases the storage required is even less than that required by SF-A^TA.

It is noteworthy that the SFC-A symbolic factorization algorithm runs faster than the SF-A symbolic factorization algorithm. This is because fewer subscripts have to be generated in the former case. As noted earlier, extra indexing is needed in using

TABLE 4.5
Symbolic factorization algorithm with subscript compression (SFC-A).

	Symbolic factorization						Numerical solution			Percentage utilization	
			Number of subscripts		Number of nonzeros						
Problem	Store	Time	Lower	Upper	Lower	Upper	Store	Factor time	Solve time	Lower	Upper
1	35,459	1.667	3,981	7,713	20,310	37,554	79,858	44.033	1.250	93.13	68.82
2	39,202	1.817	4,445	8,914	25,765	46,945	97,172	63.450	1.583	91.33	67.29
3	42,143	1.967	5,121	9,227	25,875	47,484	99,690	64.133	1.583	92.67	66.65
4	53,535	2.550	5,925	11,317	26,462	49,273	108,821	48.900	1.650	93.11	65.52
5	42,636	2.000	4,383	8,588	14,152	26,570	66,677	16.450	1.000	92.52	64.03
6	49,045	2.283	4,831	9,699	15,300	28,556	73,537	16.867	1.067	91.91	64.61
7	42,623	1.950	4,602	9,460	19,029	35,291	80,904	32.100	1.217	93.53	65.14
8	43,461	2.133	4,794	10,040	20,922	38,748	87,059	42.083	1.433	93.38	66.82
9	50,791	2.600	5,931	10,465	28,395	53,161	112,795	62.317	1.950	92.38	63.55
10	3,176	0.167	152	405	256	544	3,077	0.117	0.033	65.63	40.07
11	3,720	0.167	173	524	349	830	3,717	0.167	0.033	76.50	42.29
12	17,897	0.933	2,462	6,544	8,780	17,539	39,520	12.000	0.583	67.80	37.91
13	2,992	0.150	140	435	268	690	2,989	0.133	0.033	76.12	43.62
14	1,953	0.083	171	385	426	843	2,566	0.217	0.033	69.25	42.35
15	20,140	1.017	1,579	4,988	5,358	11,794	30,928	6.133	0.417	58.16	42.81
16	13,771	0.683	1,158	3,040	3,698	7,203	20,372	3.567	0.267	63.47	39.44
17	12,082	0.650	558	2,101	1,447	5,748	15,325	1.083	0.217	58.47	27.63
18	25,026	1.200	1,523	4,556	2,771	7,565	27,298	1.850	0.333	67.70	40.20
19	3,351	0.200	111	569	313	1,322	3,562	0.300	0.050	95.21	64.45
20	5,393	0.250	440	1,158	960	2,194	6,945	0.800	0.083	83.33	56.61
21	4,780	0.283	391	260	7,402	7,763	17,250	9.633	0.333	70.81	48.78
22	14,988	3.617	2,638	965	11,153	91,080	113,133	6.917	2.067	4.52	1.54
23	15,337	0.733	800	3,618	2,562	6,972	17,949	2.633	0.217	73.22	55.55
24	29,887	2.617	4,195	6,141	16,574	56,559	92,515	22.050	1.533	41.14	15.35
25	23,868	1.133	2,167	6,634	7,610	16,600	38,966	10.200	0.533	88.07	46.77

the storage scheme created by the SFC-A algorithm. Thus, we should expect the execution times for the numerical solution associated with SFC-A to be larger than those in SF-A. The results in Table 4.5 show that this is indeed the case, although fortunately the increase is not very significant. Furthermore, even though the execution times for the numerical solution associated with SFC-A have been increased slightly, they are still less than those associated with SF-A^TA.

For the SF-A symbolic factorization algorithm, our analysis in § 2 showed that its run time is proportional to the number of subscripts generated. We have not been able to establish that the SFC-A symbolic factorization algorithm behaves in a similar manner, although preliminary experiments contained in Table 4.6 for the graded-L problems suggest that the execution time of the SFC-A algorithm is proportional to the number of subscripts generated as well.

TABLE 4.6
Applying SFC-A *to a sequence of graded-L finite element problems.*

Order	Number of nonzeros in A	Time t	Number of subscripts			Ratio t/nz ($\times 10^5$)
			Lower	Upper	Total (nz)	
265	1,753	0.467	1,029	2,087	3,116	14.987
406	2,716	0.700	1,667	3,346	5,013	13.964
577	3,889	1.033	2,459	5,102	7,561	13.662
778	5,272	1.367	3,365	7,003	10,368	13.185
1,009	6,865	1.833	4,445	8,914	13,359	13.721
1,270	8,668	2.350	6,070	11,639	17,709	13.270
1,561	10,681	2.917	7,202	14,543	21,745	13.415
1,882	12,904	3.583	9,178	17,118	26,296	13.626
2,233	15,337	4.333	11,276	20,864	32,140	13.482
2,614	17,980	5.033	12,713	24,933	37,646	13.369
3,025	20,833	5.967	15,779	28,270	44,049	13.546
3,466	23,896	6.934	18,746	32,280	51,026	13.589

Recall that the column ordering we used for a matrix A in the experiments was an ordering obtained by applying Liu's variant of the minimum degree algorithm [11] to the matrix A^TA. As observed at the end of § 3, dense rows in the matrix A may affect the quality of the column ordering since they form dense submatrices in A^TA that may cause unnecessary fill-in in $\bar{L}$ and $\bar{U}$. (Problem 22 is an example.) A suggested solution to this problem is to withhold such rows from A when the ordering is determined from A^TA. The column ordering is then applied to the *original* matrix A when we perform the symbolic factorization algorithm (with subscript compression). This strategy has been implemented and applied to the problems in Table 4.1 that have rows with more than 50 nonzeros. If we compare the results in Table 4.7 with the corresponding results in Table 4.5, we observe that there are indeed significant improvements when dense rows are withheld in determining the column orderings. In each case, there is a large reduction in the number of nonzeros determined by the symbolic factorization algorithm. Consequently, there is also a large reduction in the storage required by the numerical solution phase. Furthermore, the factorization and solution times are much smaller than those in Table 4.5. In some cases, the run time for the symbolic factorization algorithm and the number of (compressed) subscripts generated are also smaller.

TABLE 4.7
Symbolic factorization algorithm with subscript compression (SFC-*A*).
(With withholding of dense rows in determining a column ordering.)

	Symbolic factorization						Numerical solution			Percentage utilization	
			Number of subscripts		Number of nonzeros						
Problem	Store	Time	Lower	Upper	Lower	Upper	Store	Factor time	Solve time	Lower	Upper
21	4,561	0.667	131	301	1,213	7,804	10,883	1.100	0.200	54.33	11.38
22	13,168	0.883	429	1,354	617	2,216	11,913	0.283	0.150	86.87	29.78
24	28,855	1.617	2,164	7,140	7,501	18,756	44,607	7.750	0.650	56.14	37.09

Another strategy for possibly improving the performance of the symbolic factorization algorithm is to apply the symbolic factorization algorithm to the matrix A^T and compute (numerically) a triangular factorization of it. This decomposition is equally useful in solving $Ax = b$. In Table 4.8, we have provided the results of applying SFC-*A* to the transpose of some of the matrices described in Table 4.1. Rows with more than 50 nonzeros are withheld when a column ordering for A^T is determined. (Row withholding occurs in Problems 17, 21 and 25.) Comparing the results in Table 4.8 with those in Tables 4.5 and 4.7, we see that it may be desirable to use the transpose in some instances. For example, Problem 22 has a few very dense rows but no dense columns. Applying SFC-*A* to the matrix itself results in low percentage utilization,

TABLE 4.8
Symbolic factorization algorithm with subscript compression (SFC-*A*).
(The transpose is used and dense rows in the transpose are withheld in determining a column ordering.)

	Symbolic factorization						Numerical solution			Percentage utilization	
			Number of subscripts		Number of nonzeros						
Problem	Store	Time	Lower	Upper	Lower	Upper	Store	Factor time	Solve time	Lower	Upper
10	3,074	0.150	128	327	204	444	2,823	0.100	0.033	69.61	47.07
11	3,860	0.150	224	613	507	1,083	4,268	0.250	0.050	67.85	51.89
12	17,624	0.983	2,671	6,062	11,160	20,204	44,292	13.100	0.667	55.27	27.96
13	3,123	0.133	195	511	481	993	3,636	0.250	0.050	55.51	45.72
14	1,915	0.083	171	347	381	764	2,404	0.200	0.033	74.54	39.01
15	19,926	1.033	1,696	4,657	6,942	16,269	36,773	10.817	0.533	73.93	50.45
16	13,037	0.667	871	2,593	3,129	8,035	19,901	2.783	0.283	53.95	37.06
17	11,944	0.617	480	2,041	1,304	5,405	14,701	1.100	0.183	60.58	31.51
18	26,370	1.333	1,717	5,706	4,675	13,303	36,284	3.417	0.483	53.67	30.94
19	3,475	0.200	238	566	710	1,522	4,283	0.683	0.050	96.48	75.89
20	5,393	0.233	488	1,110	1,355	2,471	7,617	0.983	0.100	80.37	51.19
21	4,767	0.517	146	492	1,325	8,058	11,455	1.250	0.183	78.49	5.85
22	12,595	0.483	134	1,076	174	1,274	9,955	0.100	0.133	89.08	78.73
23	17,050	1.000	1,235	4,896	4,358	12,341	26,827	4.917	0.417	53.85	36.58
24	26,650	1.300	1,396	5,703	5,189	16,703	38,037	6.650	0.533	71.71	43.69
25	22,560	1.333	1,957	5,536	7,614	16,484	37,546	9.750	0.533	86.77	44.86

high storage requirement and large execution time. Another interesting example is Problem 17 which has a few dense columns but no dense rows. Applying SFC-A to the transpose with withholding dense rows in determining the column ordering seems to be better than just applying SFC-A to the matrix itself.

However, the opposite can also hold. As an example, the results (including storage requirements and execution times) of applying SFC-A (with withhholding dense rows in determining the column ordering) to Problem 21 are better than the corresponding results in Table 4.8. At this point, we do not know an effective scheme for deciding whether to use A or A^T, other than simply applying the SFC-A symbolic factorization algorithm to both A and A^T, and choosing the one that requires the least storage. Since the SFC-A symbolic factorization algorithm is very efficient, this is not an unreasonable thing to do.

In [10], a number of comparisons of the SF-A^TA algorithm with various standard alternative schemes was provided. However, for completeness, we include here some experimental results obtained by applying the "industry standard" Harwell library code MA28 to our set of test problems. (The version of MA28 used in our comparisons has since been replaced by a version with pivoting options that can significantly reduce storage and execution time when the matrix being factored has symmetric structure; see [4].)

This code employs row and column permutations to maintain both numerical stability and sparsity. Since the permutations depend on the numerical values, and the relative importance attached to stability and sparsity, one cannot predict in advance how much space will be needed for the factors. Moreover, since storage is allocated during the computation, the data structures employed must be quite flexible, and require substantial execution and storage "overhead." A threshold pivoting technique is used when searching for a pivot; that is, at the kth step of the decomposition, an element in the reduced matrix, say A_{ki}, will be chosen as the pivot if it satisfies

$$|A_{ki}| \geqq u \max_j |A_{kj}|$$

where u is a user specified parameter satisfying $0 \leqq u \leqq 1$. (See [2], [6] for details.) Increasing the threshold parameter may improve the accuracy, but this may also increase both the storage requirements and execution times. In our experiments, we have set $u = 0.1$.

The amount of space reported in Table 4.9 for MA28 is the *minimum amount* required in order to solve each problem. With the minimum amount of space, MA28 has to perform numerous data compressions and consequently requires substantial execution time. Thus, in order to reduce the execution time, it is common practice to provide more space than the minimum amount. The extra space is sometimes known as *elbowroom.* See [7] for some numerical experiments. In our experiments, we provided enough elbow room so that no data compressions occurred.

The results show that for the finite element problems, our new scheme enjoys a considerable advantage over MA28. For the nonfinite element problems, the comparison is less decisive, although the new scheme is comparable to MA28 in the majority of cases. An examination of the results in Tables 4.5, 4.7 and 4.8 suggests that *reliable* identification and exploitation of dense rows, and the appropriate choice of either A or A^T would make the new scheme comparable with or better than MA28 in all the test examples.

While it is true that the structure of $\bar{U}$ often substantially overestimates the structure of the upper triangular factor U for any given pivot sequence, we contend

TABLE 4.9
Results obtained using the Harwell MA28 *code.*

Problem	Numerical solution	
	Minimal store	Total time
1	116,630	428.267
2	144,365	551.317
3	141,168	326.200
4	145,106	363.167
5	96,390	92.167
6	103,699	101.233
7	110,327	305.583
8	117,398	321.233
9	181,826	830.100
10	3,367	0.433
11	4,087	0.550
12	16,353	6.750
13	3,348	0.500
14	2,412	0.500
15	23,526	10.233
16	15,058	4.333
17	12,795	1.717
18	27,001	11.950
19	3,905	0.883
20	6,525	1.633
21	5,934	1.517
22	14,418	1.150
23	17,297	5.233
24	27,817	4.950
25	45,191	35.450

that the important number in this connection is the *total* storage requirement (which includes storage for the numbers and the overhead storage). The use of a static data structure in our scheme allows for very low overhead storage, so that even though a significant number of zeros is stored in the data structure, the total storage requirement is competitive with the more elaborate dynamic storage schemes used by codes such as MA28.

As noted earlier, an important feature of the new method is that the amount of space allocated to store the LU-decomposition is not sensitive to the numerical values in A and the row interchanges. The same data structure can be used for different coefficient matrices so long as they have the same structure. After the first system has been solved, it is not necessary to determine the data structure again when subsequent systems having the same structure are to be solved. This is not true for MA28, where the amount of space depends on the numerical values and the row interchanges (unless one wants to decompose a new matrix having the same structure using the pivotal sequence obtained during the decomposition of the old matrix).

The MA28 suite of subroutines provides a mechanism for decomposing a matrix having a structure identical to a previously decomposed matrix, and employing the previously determined pivotal sequence. In this mode, the storage requirement is in general smaller than that needed in the initial factorization, and the execution times are considerably less. Although we have not yet pursued its implementation, the same

strategy could be used in connection with our method. After the decomposition of the first matrix, we can postprocess the data structure to release any unused storage locations. We could then decompose a new matrix having the same structure using the same pivotal sequence.

The efficient implementation of this idea is complicated by the possibility of numerical cancellation occurring in the initial factorization. Since such cancellation might not occur in a subsequent factorization, one cannot simply examine the data structure containing the initial factorization, and "compress out" all the zeros. Some of them might have been caused by cancellation rather than being a consequence of the structure of the matrix. Thus, the postprocessing of the data structure mentioned above must involve a mechanism that preserves space for the factors under the condition that no numerical cancellation occurs.

5. Concluding remarks. In this paper, we have described a new symbolic factorization algorithm for Gaussian elimination with partial pivoting. Several enhancements have been proposed to improve the performance of the algorithm. Preliminary numerical experiments have shown that the data structure created by the new symbolic factorization algorithm (together with the enhancements) for sparse Gaussian elimination with partial pivoting is much tighter than that generated by the approach proposed in [10]. Storage required for the numerical solution phase is smaller in the new aproach than in [10]. There are also substantial improvements in execution times for the numerical solution phase. The method appears to be quite competitive to alternative approaches for solving general sparse systems of equations.

Several problems are still outstanding. First, preliminary results indicate that the run time of the new symbolic factorization algorithm with subscript compression is proportional to the number of (compressed) subscripts generated. We continue to work on establishing this result. Second, the column ordering used for A in the experiments presented in this paper is obtained by applying the minimum degree ordering algorithm to the matrix A^TA. As observed in § 3.2, there are other possibilities in determining "good" column orderings for A. We are currently designing efficient algorithms for generating these column orderings. Third, an important problem is to establish criteria for determining when a row should be regarded as "dense." This is a problem common to many sparse matrix computations. A related problem is to investigate when we should work with A^T instead of A. Finally, if the matrix A is reducible, then A can be permuted symmetrically so that the permuted matrix has a block triangular form. In this case, it is only necessary to decompose the diagonal blocks. The ideas described in this paper can of course be applied to these diagonal blocks. Efficient implementation of this generalization is under consideration.

REFERENCES

[1] T. F. COLEMAN, A. EDENBRANDT AND J. R. GILBERT, *Predicting fill for sparse orthogonal factorization*, J. Assoc. Comput. Mach., 33 (1986), pp. 517–532.

[2] I. S. DUFF, MA-28—*A set of* FORTRAN *subroutines for sparse unsymmetric linear equations*, Tech. Rpt. AERE R-8730, Harwell, England, 1977.

[3] ———, *On algorithms for obtaining a maximum traversal*, ACM Trans. Math. Software, 7 (1981), pp. 315–330.

[4] ———, *The solution of nearly symmetric sparse linear equations*, in Computing Methods in Applied Sciences and Engineering VI, R. Glowinski and J. L. Lions, eds., North-Holland, Amsterdam, 1984, pp. 57–74.

[5] I. S. DUFF, R. G. GRIMES, J. G. LEWIS AND W. G. POOLE, JR., *Sparse matrix test problems*, ACM SIGNUM Newsletter 17, No. 2 (1982), p. 22.

[6] I. S. DUFF AND J. K. READ, *Some design features of a sparse matrix code*, ACM Trans. Math. Software, 5 (1979), pp. 18–35.

[7] J. A. GEORGE, M. T. HEATH AND E. G. NG, *A comparison of some methods for solving sparse linear least squares problems*, this Journal, 4 (1983), pp. 177–187.

[8] J. A. GEORGE AND J. W-H. LIU, *Algorithms for matrix partitioning and the numerical solution of finite element systems*, SIAM J. Numer. Anal., 15 (1978), pp. 297–327.

[9] ———, *An optimal algorithm for symbolic factorization of symmetric matrices*, SIAM J. Comput., 9 (1980), pp. 583–593.

[10] J. A. GEORGE AND E. G-Y. NG, *An implementation of Gaussian elimination with partial pivoting for sparse systems*, this Journal, 6 (1985), pp. 390–409.

[11] J. W-H. LIU, *Modification of the minimum degree algorithm by multiple elimination*, ACM Trans. Math. Software, 11 (1986), pp. 141–153.

[12] A. H. SHERMAN, *On the efficient solution of sparse systems of linear and nonlinear equations*, Research Rpt. 46, Dept. of Computer Science, Yale University, New Haven, CT, 1975.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 6, November 1987
© 1987 Society for Industrial and Applied Mathematics

FAST PARALLEL ALGORITHMS FOR QR AND TRIANGULAR FACTORIZATION*

J. CHUN†, T. KAILATH† AND H. LEV-ARI†‡

Abstract. We present a family of new fast algorithms for QR factorization of certain structured matrices, including rectangular Toeplitz matrices and a variety of other Toeplitz-like matrices. It possesses a very regular structure, and appears to be very convenient for parallel implementation. Moreover it is shown that the same architecture can be used for either triangular factorization or QR factorization. Our approach separates the conceptual and implementational aspects of the problem. Our analysis reveals a variety of algorithmic implementations of the basic procedure, all with potentially different numerical properties that need further examination.

Our approach is based on the observations that the matrix $\mathbf{R}$ in $\mathbf{A}=\mathbf{QR}$ is the Cholesky factor of $\mathbf{A}^T\mathbf{A}$ and that fast algorithms for Cholesky factorization of positive definite Toeplitz and certain "close-to-Toeplitz" matrices are readily obtained via a so-called displacement representation of matrices. In this paper we show that if $\mathbf{A}$ is Toeplitz or is expressed in suitable displacement form then $\mathbf{A}^T\mathbf{A}$ also has such a displacement representation, so that $\mathbf{R}$ can be obtained by a suitable fast Cholesky algorithm. The matrix $\mathbf{Q}$ is then found by extending a result, also presented in this paper, that $\mathbf{R}$ and $\mathbf{R}^{-1}$ can be found simultaneously by appropriate circular and hyperbolic rotations applied to a matrix (with Toeplitz blocks) constructed from the elements of $\mathbf{A}$.

Key words. QR factorization, triangular factorization, displacement structure, Toeplitz, Schur algorithm

AMS(MOS) subject classifications. Primary 65F05, 65F30; secondary 15A06

1. Introduction.

An $m\times n$ matrix $\mathbf{T}\in\mathbf{R}^{m\times n}$ is called *Toeplitz* if it has identical elements on each diagonal, i.e., if $\mathbf{T}=(t_{i-j})$, $1\leqq i\leqq m$, $1\leqq j\leqq n$. Toeplitz systems of linear equations arise in numerous applications, and consequently several fast algorithms for solving square Toeplitz systems have been developed; we refer here only to the recent survey article by Bunch [4]. Among others, well-known $O(n^2)$ algorithms are the Levinson algorithm [7]-[9], [14], [21], which computes the triangular factorization of $\mathbf{T}^{-1}$ and the Schur algorithm [6], [15], [16], [19], [20], [24], that computes the triangular factorization of $\mathbf{T}$. Divide-and-conquer type algorithms [2], [3], [22] that need only $O(n\log^2 n)$ *flops* (multiplications, divisions or square-root computations) are also known.

The least squares solution $\mathbf{x}$ of an overdetermined system of equations $\mathbf{Tx}=\mathbf{b}$, where $\mathbf{T}$ is Toeplitz, can be obtained by solving the normal equations, $\mathbf{T}^T\mathbf{Tx}=\mathbf{T}^T\mathbf{b}$. However, this approach is often discouraged because of the accuracy problems in forming $\mathbf{T}^T\mathbf{T}$ as well as the $O(mn^2)$ flops required by the matrix multiplication. Therefore many authors have advocated the use of algorithms based on the QR factorization of $\mathbf{T}$, for which some fast algorithms for Toeplitz matrices have recently been developed. Sweet [26] was the first to develop an $O(n^2)$ QR factorization algorithm for square Toeplitz matrices. Bojanczyk, Brent and de Hoog [1] proposed a slightly faster QR factorization algorithm for possibly rectangular Toeplitz matrices. Cybenko [5] has recently presented another fast algorithm (however, for computing $\mathbf{Q}$ and $\mathbf{R}^{-1}$, rather than $\mathbf{R}$, where $\mathbf{T}=\mathbf{QR}$).

Motivated by the work of Bojanczyk et al. [1] and calling upon some of our earlier work, we have developed a fairly general fast QR factorization algorithm, and moreover

* Received by the editors March 19, 1986; accepted for publication (in revised form) January 31, 1987.
† Information Systems Laboratory, Stanford University, Stanford, California 94305.
‡ The work of this author was supported by the National Science Foundation under grant ECS-850246, by the Air Force Office of Scientific Research, Air Force Systems Command under contract AF83-0228, and by the Department of the Navy (NAVELEX) under contract N00039-84-C-0211.

one that can be readily adapted to the triangular factorization of positive definite matrices. Our approach for the QR factorization of $\mathbf{T} \in R^{m \times n}$ is based on the following observations:

(1) There is a known fast and stable algorithm due to Schur for the triangular (Cholesky) factorization of a positive definite symmetric Toeplitz matrix.

(2) Though $\mathbf{T}^T\mathbf{T}$ is not Toeplitz, it is close to Toeplitz in the sense of low *displacement rank*, as will be explained in § 2. A generalized Schur algorithm [15], [20] can be used to factorize such matrices with $O(n^2)$ flops without explicitly forming $\mathbf{T}^T\mathbf{T}$. This gives the $\mathbf{R}$ of $\mathbf{T} = \mathbf{QR}$.

(3) The so-called Schur parameters generated in obtaining $\mathbf{R}$ can also be used to compute $\mathbf{R}^{-1}$ in $O(n^2)$ flops.

(4) A simple extension of the last observation allows us to find the matrix $\mathbf{Q} = \mathbf{TR}^{-1}$ with $O(mn)$ flops.

Because our procedure is based on the Schur algorithm, it avoids inner product computations (unlike, e.g., Cybenko's algorithm in [5]), and thus is more convenient for parallel VLSI implementation.

The simplicity of the derivation is also to be noted. This arises from the introduction of the displacement structure, recognition of which allows us to significantly extend the results to encompass many non-Toeplitz matrices; in fact, the approach can be applied to *any* matrix (of course with an increased number of computations) provided it is re-expressed in a so-called displacement form. The computational complexity of our factorization procedure for a given matrix $\mathbf{A}$ is proportional to the rank of the difference $\mathbf{A} - \mathbf{ZAZ}^T$, where $\mathbf{Z}$ is the lower shift matrix, with 1's on the first subdiagonal and 0's elsewhere. Another feature that will become evident is that our approach separates the computational and implementational aspects of the problem. The form of our factorization procedure readily exhibits different possibilities of initialization and computation steps, resulting in a variety of algorithmic implementations, one of which is described in § 5.

These different implementations may have different numerical properties, an aspect we do not explore in this paper. In particular, they can involve different numbers of so-called "downdating" or "hyperbolic rotation" operations, which are known to be unstable for ill-conditioned problems (see [25]). However, as Bojanczyk has pointed out to us in a comment on a preprint of this paper, having a smaller number of downdating steps does not seem to substantially improve the numerical behavior, so that a much more detailed error analysis is necessary to evaluate the performance of the possible implementations.

We shall start in § 2 with a discussion of the concept of displacement structure and some of its properties. Section 3 contains an application of these concepts to the triangular factorization of certain structured matrices *and* of their inverses. In § 4 we show how the displacement concept allows us to extend the results of § 3 to obtain the QR factorization of such structured matrices. In § 5, we describe the detailed computational implementation of the ideas of §§ 3 and 4. Proofs of the lemmas in § 2 are given in the Appendix.

2. Displacement representations of matrices. We give a brief review (and slight extension) of ideas discussed in more detail in Kailath et al. [17] (see also [16], [18]-[20]). We shall say that a matrix $\mathbf{A} \in \mathbf{R}^{m \times n}$, where $n \leqq m$, has a *displacement representation* if it can be written as

$$\mathbf{A} = \sum_{i=1}^{\hat{\alpha}} \mathbf{L}_n(\mathbf{x}_i)\mathbf{L}_n^T(\mathbf{y}_i), \quad \mathbf{x}_i \in \mathbf{R}^m, \quad \mathbf{y}_i \in \mathbf{R}^n, \tag{1}$$

where $\mathbf{L}_n(\mathbf{s})$ denotes the so-called Krylov matrix generated by a vector $\mathbf{s} \in \mathbf{R}^k$; i.e.,

$$\mathbf{L}_n(\mathbf{s}) \equiv (\mathbf{s}, \mathbf{Z}_k\mathbf{s}, \cdots, \mathbf{Z}_k^{n-1}\mathbf{s}) \in \mathbf{R}^{k\times n},$$

and $\mathbf{Z}_k$ is the $k \times k$ *lower shift matrix*, zero everywhere except for 1's on the first subdiagonal. A *generator* of $\mathbf{A}$ is defined as a matrix pair, $G_{\hat{\alpha}}(\mathbf{A}) \equiv \{\mathbf{X}, \mathbf{Y}\}$, where $\mathbf{X} \equiv (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_{\hat{\alpha}})$ and $\mathbf{Y} \equiv (\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_{\hat{\alpha}})$, where the $\{\mathbf{x}_i, \mathbf{y}_i\}$ are the vectors that determine $\mathbf{A}$ via the displacement representation (1). The number $\hat{\alpha}$ is called the *length* of the generator $G_{\hat{\alpha}}(\mathbf{A})$. We shall suppress the subscripts in $\mathbf{L}_n(\mathbf{s})$, $\mathbf{Z}_k$ and $G_{\hat{\alpha}}(\mathbf{A})$ whenever they are clear from the context. Generators are clearly not unique and can be of different lengths. The smallest possible length will be called the *displacement rank* of $\mathbf{A}$ and denoted by $\alpha(\mathbf{A})$. The reason for the name is made clear by the following result [17].

LEMMA 1. *For an arbitrary matrix* $\mathbf{A} \in \mathbf{R}^{m\times n}$,

$$\mathbf{A} - \mathbf{Z}_m\mathbf{A}\mathbf{Z}_n^T = \sum_{i=1}^{\hat{\alpha}} \mathbf{x}_i\mathbf{y}_i^T \quad \textit{if and only if} \quad \mathbf{A} = \sum_{i=1}^{\hat{\alpha}} \mathbf{L}(\mathbf{x}_i)\mathbf{L}^T(\mathbf{y}_i). \tag{2}$$

A proof follows easily by using the fact that $\mathbf{Z}_k$ is nilpotent, so that $\mathbf{Z}_m^m\mathbf{x}_i = 0$ and $\mathbf{Z}_n^n\mathbf{y}_i = 0$. The lemma implies that $\hat{\alpha}$, the number of terms in the displacement representation (1) is bounded below by $\alpha(\mathbf{A}) = \text{rank}\,(\mathbf{A} - \mathbf{Z}_m\mathbf{A}\mathbf{Z}_n)$, i.e., by the rank of the matrix obtained by subtracting from $\mathbf{A}$ the matrix obtained by shifting $\mathbf{A}$ down the main diagonal by one entry. This remark also explains the reason for the term *displacement.*

A symmetric matrix $\mathbf{A} \in \mathbf{R}^{n\times n}$ has a *symmetric generator* $G_{\hat{\alpha}}(\mathbf{A}) = \{\mathbf{X}, \mathbf{Y}\}$ in the sense that $\mathbf{x}_i = \pm\mathbf{y}_i$, $i \leqq i \leqq \hat{\alpha}$. Hence, its displacement representation has the symmetric form

$$\mathbf{A} = \sum_{i=1}^{p} \mathbf{L}(\mathbf{x}_i)\mathbf{L}^T(\mathbf{x}_i) - \sum_{i=p+1}^{p+q} \mathbf{L}(\mathbf{x}_i)\mathbf{L}^T(\mathbf{x}_i). \tag{3}$$

If p and q are as small as possible, then p is the number of positive eigenvalues of $\mathbf{A} - \mathbf{Z}_n\mathbf{A}\mathbf{Z}_n^T$, while q is the number of negative eigenvalues of the same matrix. In that case, (p, q) are called the *displacement inertia* of $\mathbf{A}$, $\hat{\alpha} = p + q$ equals the displacement rank $\alpha(\mathbf{A})$, and $G_{\hat{\alpha}}(\mathbf{A})$ is called *minimal.* For a symmetric matrix $\mathbf{A}$, the generator has the form $G_{\hat{\alpha}}(\mathbf{A}) = \{\mathbf{X}, \mathbf{X}\mathbf{\Sigma}\}$, where $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_{\hat{\alpha}})$ and

$$\mathbf{\Sigma} = \begin{pmatrix} \mathbf{I}_p & 0 \\ 0 & -\mathbf{I}_q \end{pmatrix} \equiv \mathbf{I}_p \oplus -\mathbf{I}_q.$$

An important property [17] of displacement representations is that if $\mathbf{A}$ is nonsingular, then $\mathbf{A}$ and $\tilde{\mathbf{I}}\mathbf{A}^{-1}\tilde{\mathbf{I}}$ have similar displacement representations (3), where $\tilde{\mathbf{I}}$ is the matrix with 1's on the antidiagonal and zeros elsewhere.

Example. Symmetric Toeplitz matrices. The reader may find it instructive to check that if $\mathbf{T} = (t_{i-j})$ is a symmetric Toeplitz matrix with nonzero diagonal, then a symmetric generator $G_2(\mathbf{T}) = \{\mathbf{X}, \mathbf{X}\mathbf{\Sigma}\}$ is given by $\Sigma = 1 \oplus -1$, and $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2)$, where

(4a) $\mathbf{x}_1 =$ the first column of $\mathbf{T}$ divided by $t_0^{1/2}$,

(4b) $\mathbf{x}_2 =$ the same as $\mathbf{x}_1$ with the first entry equal to zero.

As noted before, the generator of $\mathbf{T}^{-1}$ has the same displacement inertia, a fact that can also be seen from the now well-known Gohberg–Semencul formula [11], [15].

To obtain the QR factorization of a Toeplitz matrix $\mathbf{T}$, we shall need to examine closely the generator of the symmetric matrix $\mathbf{T}^T\mathbf{T}$.

LEMMA 2. (Generator of $\mathbf{T}^T\mathbf{T}$.) *Let* $\mathbf{T}\in\mathbf{R}^{m\times n}$ *be Toeplitz. Then* $\alpha(\mathbf{T}^T\mathbf{T})\leqq 4$, *and a symmetric generator,* $G_4(\mathbf{T}^T\mathbf{T})=\{\mathbf{W},\mathbf{W}\boldsymbol{\Sigma}\}$ *is given by* $\boldsymbol{\Sigma}=\mathbf{I}_2\oplus-\mathbf{I}_2$ *and* $\mathbf{W}=(\mathbf{w}_1,\mathbf{w}_2,\mathbf{w}_3,\mathbf{w}_4)$ *with*

$$\mathbf{w}_1=\mathbf{T}^T\mathbf{t}_1/\|\mathbf{t}_1\|,\quad \mathbf{w}_2=\mathbf{t}_2,\quad \mathbf{w}_3=\mathbf{Z}_n\mathbf{Z}_n^T\mathbf{w}_1,\quad \mathbf{w}_4=\mathbf{Z}_n\mathbf{l}_1, \tag{5}$$

where

$$\mathbf{t}_1=(t_0t_1,\cdots,t_{m-1})^T,\quad \mathbf{t}_2=(0,t_{-1},\cdots,t_{1-n})^T,\quad \mathbf{l}_1=(t_{m-1},\cdots,t_{m-n})^T,$$

and $\|\mathbf{t}_1\|$ *denotes the Euclidean norm of* $\mathbf{t}_1$.

Remark 1. From Lemma 2, which is proved in the Appendix, we note that, in general, the displacement inertia of $\mathbf{T}^T\mathbf{T}$ is $(p,q)=(2,2)$. All known fast QR factorization algorithms [1], [5], [26] including ours contain certain so-called downdating operations (rank one modifications by subtraction) or their equivalents because of the fact that $q\neq 0$ (see § 5). Downdating operations are unstable [25] and that is why further error analysis of all fast QR factorization algorithms is necessary.

Remark 2. For certain Toeplitz matrices, the displacement rank of $\mathbf{T}^T\mathbf{T}$ can be less than 4. For example, for the so-called pre-windowed Toeplitz matrix $\mathbf{w}_2=0$, for the post-windowed Toeplitz matrix $\mathbf{w}_4=0$, and for the square circulant matrix $\mathbf{w}_2=\mathbf{w}_4$ (see [15] for a definition of these matrices).

Now let us consider an arbitrary matrix $\mathbf{A}\in\mathbf{R}^{m\times n}$. The important fact is that the displacement rank of $\mathbf{A}^T\mathbf{A}$ is still reasonably low for a low displacement rank matrix $\mathbf{A}$. The following lemmas are new, though analogues have been known for the integral operator case [18]. (Proofs appear in the Appendix.)

LEMMA 3. (Displacement rank of products.) *For arbitrary matrices* $\mathbf{A}\in\mathbf{R}^{m\times n}$ *and* $\mathbf{B}\in\mathbf{R}^{n\times s}$,

$$\alpha(\mathbf{AB})\leqq\alpha(\mathbf{A})+\alpha(\mathbf{B})+1. \tag{6}$$

Therefore, in particular, $\alpha(\mathbf{A}^T\mathbf{A})\leqq 2\alpha(\mathbf{A})+1$. Also, after some algebra, we can obtain a symmetric generator of $\mathbf{A}^T\mathbf{A}$, as demonstrated by the following lemma.

LEMMA 4. ($G(\mathbf{A}^T\mathbf{A})$ from $G(\mathbf{A})$.) *Let* $\mathbf{A}\in\mathbf{R}^{m\times n}$, *and a generator* $G_{\hat\alpha}(\mathbf{A})=\{\mathbf{X},\mathbf{Y}\}$ *be given. Then a symmetric generator* $G_{2\hat\alpha+1}(\mathbf{A}^T\mathbf{A})=\{\mathbf{W},\mathbf{W}\boldsymbol{\Sigma}\}$ *is given by* $\boldsymbol{\Sigma}=\mathbf{I}_{\hat\alpha}\oplus-\mathbf{I}_{\hat\alpha+1}$, *and for* $1\leqq k\leqq\hat\alpha$

$$\mathbf{w}_k=\left[\sum_{i=1}^{\hat\alpha}\mathbf{L}(\mathbf{y}_i)\mathbf{Z}_n\mathbf{Z}_n^T\mathbf{L}^T(\mathbf{x}_i)+\frac{1}{2}\sum_{i\neq k}^{\hat\alpha}\mathbf{y}_i\mathbf{x}_i^T+\mathbf{y}_k\mathbf{x}_k^T\right]\mathbf{x}_k\Big/\|\mathbf{x}_k\|, \tag{7a}$$

for $\hat\alpha+1\leqq k\leqq 2\hat\alpha$

$$\mathbf{w}_k=\left[\sum_{i=1}^{\hat\alpha}\mathbf{L}(\mathbf{y}_i)\mathbf{Z}_n\mathbf{Z}_n^T\mathbf{L}^T(\mathbf{x}_i)+\frac{1}{2}\sum_{i\neq k-\hat\alpha}^{\hat\alpha}\mathbf{y}_i\mathbf{x}_i^T\right]\mathbf{x}_{k-\hat\alpha}\Big/\|\mathbf{x}_{k-\hat\alpha}\|, \tag{7b}$$

and finally

$$\mathbf{w}_{2\hat\alpha+1}=\sum_{i=1}^{\hat\alpha}\mathbf{L}(\mathbf{y}_i)\mathbf{Z}_n\mathbf{I}_i,\ \mathbf{I}_i^T\ \textit{is the last row of}\ \mathbf{L}(\mathbf{x}_i). \tag{7c}$$

Admissible generators [20]. Now, consider a square matrix $\mathbf{A}$. Sometimes, it would be more convenient to have a fast algorithm for directly computing the triangular factorization of $\mathbf{A}^{-1}$, rather than of $\mathbf{A}$. This is possible, as shown in § 3, if $G_{\hat\alpha}(\mathbf{A})$ satisfies a certain condition called *admissibility.*

A generator $G_{\hat\alpha}(\mathbf{A})=\{\mathbf{X},\mathbf{Y}\}$ will be called *left* (*right*) *admissible with parameter* $\boldsymbol{\mu}(\boldsymbol{\sigma})$ if there exists $\boldsymbol{\mu}\in\mathbf{R}^{\hat\alpha}$ ($\boldsymbol{\sigma}\in\mathbf{R}^{\hat\alpha}$) such that $\mathbf{X}\boldsymbol{\mu}=\mathbf{e}_1$ ($\mathbf{Y}\boldsymbol{\sigma}=\mathbf{e}_1$), where $\mathbf{e}_1=(1,0,\cdots,0)^T$. A generator that is both left and right admissible is called *admissible with parameter* $(\boldsymbol{\mu},\boldsymbol{\sigma})$.

As an example, the generator $G_2(\mathbf{T}) = \{\mathbf{X}, \mathbf{X\Sigma}\}$ of a symmetric Toeplitz matrix T, as given in a previous example, is admissible with parameter $(\boldsymbol{\mu}, \boldsymbol{\Sigma\mu})$, where $\boldsymbol{\mu} = (t_0^{-1/2}, -t_0^{-1/2})^T$.

The significance of admissibility in the context of factoring $\mathbf{A}^{-1}$ is that if a generator $G_{\hat{\alpha}}(\mathbf{A}) = \{\mathbf{X}, \mathbf{Y}\}$ is admissible with parameter $(\boldsymbol{\mu}, \boldsymbol{\sigma})$, then

$$\sum_{i=1}^{\hat{\alpha}} \boldsymbol{\mu}_i \mathbf{L}(\mathbf{x}_i) = \mathbf{I}, \qquad \sum_{i=1}^{\hat{\alpha}} \boldsymbol{\sigma}_i \mathbf{L}(\mathbf{y}_i) = \mathbf{I}. \tag{8}$$

This fact is central to the derivation of our triangular factorization procedure (Theorem 1) in § 3.

If a minimal generator $G_{\hat{\alpha}}(\mathbf{A})$ is admissible, then every generator of the same length is admissible. If $G_{\hat{\alpha}}(\mathbf{A})$ is not admissible, then we can always construct an admissible generator of $\mathbf{A}$ of length $\hat{\alpha}+2$ or less [20]. It is interesting to note that $G(\mathbf{A}^T\mathbf{A})$ given in Lemma 4 is admissible if $G_{\hat{\alpha}}(\mathbf{A})$ is right admissible; more precisely, we have the following result (also proved in the Appendix).

LEMMA 5. (Admissibility of $G(\mathbf{A}^T\mathbf{A})$.) *If $G_{\hat{\alpha}}(\mathbf{A})$ is right admissible with parameter $\boldsymbol{\sigma}$, then $G_{2\hat{\alpha}+1}(\mathbf{A}^T\mathbf{A}) = \{\mathbf{W}, \mathbf{W\Sigma}\}$ in Lemma 4 is admissible with parameter $(\boldsymbol{\mu}, \boldsymbol{\Sigma\mu})$, where*

$$\boldsymbol{\mu} = (\boldsymbol{\sigma}_1/\|\mathbf{x}_1\|, \cdots, \boldsymbol{\sigma}_{\hat{\alpha}}/\|\mathbf{x}_{\hat{\alpha}}\|, -\boldsymbol{\sigma}_1/\|\mathbf{x}_1\|, \cdots, -\boldsymbol{\sigma}_{\hat{\alpha}}/\|\mathbf{x}_{\hat{\alpha}}\|, 0)^T. \tag{9}$$

To obtain the QR factorization of $\mathbf{T}$ it will be necessary to find a generator of the form $G(\mathbf{T}) = \{\mathbf{V}, \mathbf{W}\}$ when a symmetric generator $G(\mathbf{T}^T\mathbf{T}) = \{\mathbf{W}, \mathbf{W\Sigma}\}$ is given. For $G_4(\mathbf{T}^T\mathbf{T})$ as in Lemma 2, it is easy to see that the vectors

$$\mathbf{v}_1 = -\mathbf{v}_3 = \mathbf{t}_1/\|\mathbf{t}_1\|, \qquad \mathbf{v}_2 = \mathbf{e}_1, \qquad \mathbf{v}_4 = 0 \tag{10}$$

satisfy

$$\sum_{i=1}^{4} \mathbf{L}(\mathbf{v}_i)\mathbf{L}^T(\mathbf{w}_i) = \mathbf{T}.$$

This can be generalized as follows.

LEMMA 6. ($G(\mathbf{A})$ from $G(\mathbf{A}^T\mathbf{A})$.) *Let $G_{\hat{\alpha}}(\mathbf{A}) = \{\mathbf{X}, \mathbf{Y}\}$, and let $G_{2\hat{\alpha}+1}(\mathbf{A}^T\mathbf{A}) = \{\mathbf{W}, \mathbf{W\Sigma}\}$ be as given in Lemma 4. Then $G_{2\hat{\alpha}+1}(\mathbf{A}) = \{\mathbf{V}, \mathbf{W}\}$, where $\mathbf{v}_i = -\mathbf{v}_{\hat{\alpha}+i} = \mathbf{x}_i/\|\mathbf{x}_i\|$ for $1 \leqq i \leqq \hat{\alpha}$, and $\mathbf{v}_{2\hat{\alpha}+1} = 0$.*

3. Triangular factorization of a structured matrix and its inverse. Considerable insight into the QR factorization problem can be obtained by first considering the much more extensively studied problems of the triangular factorization of $n \times n$ symmetric matrices and their inverses. Later we shall consider nonsymmetric (non-Toeplitz) matrices.

For fast algorithms, we should consider matrices in their displacement representation, i.e., with a given generator. In § 5, we give detailed operation counts showing that the algorithm implementations require $O((\hat{\alpha}-1)n^2)$ flops, so that we see that the length $\hat{\alpha}$ of the generator indicates the degree of computational complexity. Recall that for Toeplitz matrices $\hat{\alpha} = 2$, while for a completely arbitrary matrix $\hat{\alpha} = n$. Finally we may note that using Lemma 4, the ideas of this section can be applied immediately to obtain the upper triangular matrices, $\mathbf{R}$ and $\mathbf{R}^{-1}$, of the QR factorization (see § 4).

For simplicity, we start with a symmetric Toeplitz matrix $\mathbf{T} \in \mathbf{R}^{n\times n}$. Later, we shall consider nonsymmetric matrices. We shall say that a matrix Θ is *J-orthogonal* if $\mathbf{J} = \boldsymbol{\Theta}\mathbf{J}\boldsymbol{\Theta}^T$, where $\mathbf{J}$ is a diagonal matrix with diagonal entries of 1 or -1. Let $G_2(\mathbf{T}) = \{\mathbf{X}, \mathbf{X\Sigma}\}$ be as in (4), and be admissible with $\boldsymbol{\mu} = (\mu_1, \mu_2)^T$. Let $\mathbf{J} \in \mathbf{R}^{2n\times 2n}$ be the Kronecker product, $\mathbf{J} = \boldsymbol{\Sigma} \otimes \mathbf{I}_n \equiv \mathbf{I}_n \oplus -\mathbf{I}_n$, and let the matrix $\boldsymbol{\Theta} \in \mathbf{R}^{2n\times 2n}$ be a J-orthogonal matrix that triangularizes the matrix $\boldsymbol{\Delta}$ (consisting of two $n \times n$ lower

triangular Toeplitz matrices above two $n \times n$ diagonal matrices):

$$\mathbf{\Delta} = \begin{pmatrix} \mathbf{L}(\mathbf{x}_1) & \mathbf{L}(\mathbf{x}_2) \\ \mu_1 \mathbf{I} & -\mu_2 \mathbf{I} \end{pmatrix}, \tag{11}$$

such that

$$\mathbf{\Delta\Theta} = \begin{pmatrix} \mathbf{L}_1 & 0 \\ \mathbf{U}_2 & \mathbf{L}_3 \end{pmatrix} \equiv \bar{\mathbf{\Delta}}, \quad \text{say.} \tag{12}$$

Then, because

$$\mathbf{\Delta J \Delta}^T = \bar{\mathbf{\Delta}} \mathbf{J} \bar{\mathbf{\Delta}}^T, \tag{13}$$

we obtain the following equalities:

$$\mathbf{L}(\mathbf{x}_1)\mathbf{L}^T(\mathbf{x}_1) - \mathbf{L}(\mathbf{x}_2)\mathbf{L}^T(\mathbf{x}_2) = \mathbf{T} = \mathbf{L}_1 \mathbf{L}_1^T, \tag{14a}$$

$$\mu_1 \mathbf{L}(\mathbf{x}_1) + \mu_2 \mathbf{L}(\mathbf{x}_2) = \mathbf{I} = \mathbf{L}_1 \mathbf{U}_2^T, \tag{14b}$$

$$\mathbf{U}_2 \mathbf{U}_2^T = \mathbf{L}_3 \mathbf{L}_3^T \tag{14c}$$

from the (1, 1), (1, 2) and (2, 2) blocks of (13), respectively. The first equality of (14b) follows from (8) and the second equality shows that $\mathbf{U}_2$ is indeed an upper triangular matrix. Therefore, triangular factorizations of $\mathbf{T}$ and $\mathbf{T}^{-1}$ are given by

$$\mathbf{T} = \mathbf{L}_1 \mathbf{L}_1^T, \qquad \mathbf{T}^{-1} = \mathbf{U}_2 \mathbf{U}_2^T = \mathbf{L}_3 \mathbf{L}_3^T.$$

This rather suprising fact does not immediately yield a convenient algorithm, because we need to say more about how the (highly nonunique) **J**-orthogonal matrix $\mathbf{\Theta}$ is to be constructed. This will be done in § 5, but here we first proceed at a more conceptual level to show how the above results can be generalized.

THEOREM 1. (Triangular factorization.) *Let* $G_{\hat{\alpha}}(\mathbf{A}) = \{\mathbf{X}, \mathbf{X\Sigma}\}$ *with* $\mathbf{\Sigma} = \mathbf{I}_p \oplus -\mathbf{I}_q$ *be an admissible generator of* $\mathbf{A}$ *with parameter* $(\boldsymbol{\mu}, \mathbf{\Sigma\mu})$. *Let* $\mathbf{J} = \mathbf{\Sigma} \otimes \mathbf{I}_n \in \mathbf{R}^{\hat{\alpha}n \times \hat{\alpha}n}$, *and let* $\mathbf{\Theta}$ *be a* **J***-orthogonal matrix that nulls out the* $(1, 2), (1, 3), \cdots, (1, p+q)$ *blocks and preserves the triangularity of the* $(1, 1)$ *block of the following matrix* $\mathbf{\Delta}$:

$$\mathbf{\Delta} = \begin{pmatrix} \mathbf{L}(\mathbf{x}_1) & \mathbf{L}(\mathbf{x}_2) & \cdots & \mathbf{L}(\mathbf{x}_p) & \mathbf{L}(\mathbf{x}_{p+1}) & \cdots & \mathbf{L}(\mathbf{x}_{p+q}) \\ \mu_1 \mathbf{I} & \mu_2 \mathbf{I} & \cdots & \mu_p \mathbf{I} & -\mu_{p+1} \mathbf{I} & \cdots & -\mu_{p+q} \mathbf{I} \end{pmatrix}. \tag{15}$$

Then we shall have

$$\bar{\mathbf{\Delta}} = \mathbf{\Delta\Theta} = \begin{pmatrix} \mathbf{L}_1 & 0 & \cdots & 0 \\ \mathbf{U}_2 & * & \cdots & * \end{pmatrix},$$

where the resulting $\mathbf{U}_2$ *is indeed upper triangular, and*

$$\mathbf{A} = \mathbf{L}_1 \mathbf{L}_1^T, \qquad \mathbf{A}^{-1} = \mathbf{U}_2 \mathbf{U}_2^T.$$

Remark 1. The proof follows the same lines as in the Toeplitz case, and will be omitted. Notice that to obtain the triangular factorization of $\mathbf{A}$ alone, only the upper part of Δ is involved, and admissibility is not required. This is a well-known fact (see [20]).

Remark 2. Let $\mathbf{A} \in \mathbf{R}^{n \times n}$ be a nonsymmetric matrix with nonsingular leading principal submatrices. Let $G_{\hat{\alpha}}(\mathbf{A}) = \{\mathbf{X}, \mathbf{Y}\}$ be an admissible generator with parameter $(\boldsymbol{\mu}, \boldsymbol{\sigma})$. Let

$$\mathbf{\Delta} = \begin{pmatrix} \mathbf{L}(\mathbf{x}_1) & \cdots & \mathbf{L}(\mathbf{x}_{\hat{\alpha}}) \\ \sigma_1 \mathbf{I} & \cdots & \sigma_{\hat{\alpha}} \mathbf{I} \end{pmatrix}, \qquad \mathbf{\Omega} = \begin{pmatrix} \mathbf{L}(\mathbf{y}_1) & \cdots & \mathbf{L}(\mathbf{y}_{\hat{\alpha}}) \\ \mu_1 \mathbf{I} & \cdots & \mu_{\hat{\alpha}} \mathbf{I} \end{pmatrix}. \tag{16}$$

Suppose we can find a matrix Φ such that

$$\bar{\Delta} = \Delta\Phi = \begin{pmatrix} L_1 & 0 & \cdots & 0 \\ U_2 & * & \cdots & * \end{pmatrix}, \qquad \bar{\Omega} = \Omega\Phi^{-T} = \begin{pmatrix} U_1^T & 0 & \cdots & 0 \\ L_2^T & * & \cdots & * \end{pmatrix}.$$

Then, because $\Delta\Omega^T = \bar{\Delta}\bar{\Omega}^T$, we conclude that

$$L_2 = L_1^{-1}, \quad U_2 = U_1^{-1}, \quad A = L_1 U_1, \quad A^{-1} = U_2 L_2.$$

4. QR factorization of a structured matrix. By using Lemma 4 of § 2 along with the ideas of § 3 we shall now show how to obtain fast QR factorizations. We first consider the Toeplitz case. Let $G_4(T^T T) = \{W, W\Sigma\}$ be as given in Lemma 2, and $G_4(T) = \{V, W\}$ be as in (10). Let $J = \Sigma \otimes I_n$, and let Θ be a J-orthogonal matrix that nulls out (1, 2), (1, 3) and (1, 4) blocks, and preserves the triangularity of the (1, 1) block of the matrix Γ consisting of the lower triangular Toeplitz matrices $L(w_i) \in R^{n \times n}$ and $L(v_i) \in R^{m \times n}$:

$$\Gamma = \begin{pmatrix} L(w_1) & L(w_2) & L(w_3) & L(w_4) \\ L(v_1) & L(v_2) & -L(v_3) & -L(v_4) \end{pmatrix}. \tag{17}$$

Then we shall have

$$\bar{\Gamma} \equiv \Gamma\Theta = \begin{pmatrix} R^T & 0 & 0 & 0 \\ Q & * & * & * \end{pmatrix},$$

and because

$$\Gamma J \Gamma^T = \bar{\Gamma} J \bar{\Gamma}^T, \tag{18}$$

we obtain the equalities

$$L(w_1)L^T(w_1) + L(w_2)L^T(w_2) - L(w_3)L^T(w_3) - L(w_4)L^T(w_4) = T^T T = R^T R,$$

$$L(v_1)L^T(w_1) + L(v_2)L^T(w_2) + L(v_3)L^T(w_3) + L(v_4)L^T(w_4) = T = QR,$$

from the (1, 1) and (2, 1) blocks of (18). Therefore $Q^T Q = R^{-T} T^T T R^{-1} = I$, and we obtain the orthogonal factorization $T = QR$. The following theorem generalizes the above considerations.

THEOREM 2. (QR factorization.) *Suppose* $G_{\hat{\alpha}}(A) = \{X, Y\}$ *is given. Let* $G_{2\hat{\alpha}+1}(A^T A) = \{W, W\Sigma\}$ *be as in Lemma* 4, *and let* $G_{2\hat{\alpha}+1}(A) = \{V, W\}$ *be as in Lemma* 6. *Let* $J = \Sigma \otimes I_n$, *and let* Θ *be a* J*-orthogonal matrix that nulls out* $(1, 2), (1, 3), \cdots, (1, 2\hat{\alpha}+1)$ *blocks and preserves the triangularity of the* (1, 1) *block of the following matrix*:

$$\Gamma = \begin{pmatrix} L(w_1) & L(w_2) & \cdots & L(w_{\hat{\alpha}}) & L(w_{\hat{\alpha}+1}) & \cdots & L(w_{2\hat{\alpha}+1}) \\ L(v_1) & L(v_2) & \cdots & L(v_{\hat{\alpha}}) & -L(v_{\hat{\alpha}+1}) & \cdots & -L(v_{2\hat{\alpha}+1}) \end{pmatrix}. \tag{19}$$

Then we shall have

$$\bar{\Gamma} \equiv \Gamma\Theta = \begin{pmatrix} R^T & 0 & \cdots & 0 \\ Q & * & \cdots & * \end{pmatrix},$$

where R *and* Q *are such that* $A = QR$.

Remark 1. If the given $G(A)$ is right admissible then, as in Theorem 1, we can also get R^{-1} by stacking another row of diagonal matrices below the matrix Γ.

The above results are all at a conceptual level. We now turn to detailed implementations.

5. Fast algorithms for implementation. We present in this section a fast triangular factorization algorithm, based on Theorem 1, for triangular factorization of positive

definite symmetric matrices, and a fast QR factorization algorithm, based on Theorem 2, for possibly rectangular full rank matrices. Later, we shall derive even faster versions of these algorithms for obtaining $\mathbf{LDL}^T$ factorizations of nonsingular *symmetric* matrices, and **QDR** factorizations, where **D** is diagonal and $\mathbf{QD}^{1/2}$ is orthogonal.

Construction of a **J***-orthogonal matrix.* First, we shall show how to construct **J**-orthogonal matrices $\mathbf{\Theta}$ to null out the $(1,2), (1,3), \cdots, (1, p+q)$ blocks of the matrix $\mathbf{\Delta}$ in Theorem 1, or of $\mathbf{\Gamma}$ in Theorem 2. Then, we shall describe the details of the factorization algorithms. Notice that $\mathbf{\Delta}$ is a special case of $\mathbf{\Gamma}$, with $\mathbf{w}_i = \mathbf{x}_i$, $\mathbf{v}_i = \pm\mu_i \mathbf{e}_1$, so that it is enough to consider the matrix $\mathbf{\Gamma}$.

Consider the block matrix $\mathbf{\Gamma}$, made up from $\mathbf{L}(\mathbf{w}_i) \in \mathbf{R}^{n\times n}$ and $\mathbf{L}(\mathbf{v}_i) \in \mathbf{R}^{m\times n}$:

$$\mathbf{\Gamma} = \begin{pmatrix} \mathbf{L}(\mathbf{w}_1) & \cdots & \mathbf{L}(\mathbf{w}_p) & \mathbf{L}(\mathbf{w}_{p+1}) & \cdots & \mathbf{L}(\mathbf{w}_{p+q}) \\ \mathbf{L}(\mathbf{v}_1) & \cdots & \mathbf{L}(\mathbf{v}_p) & -\mathbf{L}(\mathbf{v}_{p+1}) & \cdots & -\mathbf{L}(\mathbf{v}_{p+q}) \end{pmatrix}. \tag{20}$$

We need to find a **J**-orthogonal matrix $\mathbf{\Theta} \in \mathbf{R}^{n(p+q)\times n(p+q)}$, where $\mathbf{J} = \mathbf{\Sigma} \otimes \mathbf{I}_n$ and $\mathbf{\Sigma} = \mathbf{I}_p \oplus -\mathbf{I}_q$, such that

$$\mathbf{\Gamma\Theta} = \begin{pmatrix} \mathbf{R}^T & 0 & \cdots & 0 \\ \mathbf{Q} & * & \cdots & * \end{pmatrix}. \tag{21}$$

Let $\mathbf{G}_{i,j} \in \mathbf{R}^{n(p+q)\times n(p+q)}$ and $\mathbf{H}_{i,j} \in \mathbf{R}^{n(p+q)\times n(p+q)}$ be identity matrices except for the following 4 entries:

$$\mathbf{G}_{i,j}(i,i) = c, \quad \mathbf{G}_{i,j}(i,j) = -s, \quad \mathbf{G}_{i,j}(j,i) = s, \quad \mathbf{G}_{i,j}(j,j) = c,$$
$$\mathbf{H}_{i,j}(i,i) = ch, \quad \mathbf{H}_{i,j}(i,j) = -sh, \quad \mathbf{H}_{i,j}(j,i) = -sh, \quad \mathbf{H}_{i,j}(j,j) = ch,$$

where

$$c^2 + s^2 = 1, \qquad ch^2 - sh^2 = 1.$$

The matrices $\mathbf{G}_{i,j}$ and $\mathbf{H}_{i,j}$ are called *circular* and *hyperbolic rotations in the* (i,j) *plane*, respectively.

The following matrices are **J**-orthogonal:

$$\mathbf{G}_{i,j}: 0 \leqq i, j \leqq np \quad \text{or} \quad np+1 \leqq i, j \leqq n(p+q),$$
$$\mathbf{H}_{i,j}: 0 \leqq i \leqq np \quad \text{and} \quad np+1 \leqq j \leqq n(p+q).$$

Therefore, products of the above matrices are also **J**-orthogonal. For a given vector $\mathbf{a} \in \mathbf{R}^{n(p+q)}$, we can *annihilate* a_j *with respect to* a_i by choosing $\mathbf{G}_{i,j}$ or $\mathbf{H}_{i,j}$ such that the jth element of $\mathbf{a}^T\mathbf{G}_{i,j}$ or $\mathbf{a}^T\mathbf{H}_{i,j}$ is zero:

$$c = \frac{a_i}{(a_i^2 + a_j^2)^{1/2}}, \qquad s = \frac{a_j}{(a_i^2 + a_j^2)^{1/2}}$$

or

$$ch = \frac{a_i}{(a_i^2 - a_j^2)^{1/2}}, \qquad sh = \frac{a_j}{(a_i^2 - a_j^2)^{1/2}}.$$

When we try to annihilate a_j with respect to a_i, by applying $\mathbf{H}_{i,j}$ to **a**, we shall encounter the following 3 cases:

$$\begin{aligned} &\text{(i)} \quad a_i < a_j, \\ &\text{(ii)} \quad a_i = a_j, \\ &\text{(iii)} \quad a_i > a_j. \end{aligned} \tag{22}$$

Since we need to compute $(a_i^2 - a_j^2)^{-1/2}$, the hyperbolic rotation $\mathbf{H}_{i,j}$ breaks down in cases (i) and (ii). The square-root-free hyperbolic rotation, which will be presented

shortly, breaks down only in case (ii). We shall show how to avoid these difficulties in our fast factorization algorithms.

Next, notice that the elements along each diagonal of $\mathbf{L}(\mathbf{w}_i)$ or $-\mathbf{L}(\mathbf{v}_i)$ are identical. Hence, if we determine the rotation that annihilates the (1, 1) element of one block with respect to the (1, 1) element of another block, and apply the rotation to the corresponding two columns, we can, in fact, annihilate all the elements on the diagonal without further computations. Then we choose other blocks and continue this process until we annihilate all the diagonals of the $(1,2),(1,3),\cdots,(1,p+q)$ blocks. Next, we can annihilate the 1st subdiagonals of the $(1,2),(1,3),\cdots,(1,p+q)$ blocks similarly, since the Toeplitz structure is still maintained in the submatrices of the modified $\mathbf{L}(\mathbf{w}_i)$ and $-\mathbf{L}(\mathbf{v}_i)$, where the subsequent rotations are going to be applied. Therefore, after carrying out this procedure of annihilating the kth subdiagonals, which we call the kth *sweep*, n times, starting from the 0th sweep (which annihilates the diagonals), we completely null out the $(1,2),(1,3),\cdots,(1,p+q)$ blocks. Such n sweeps need $O((p+q-1)(n+m)n)$ flops, since each sweep needs $O((p+q-1)\times(n+m))$ flops. Also notice that the kth sweep generates the kth superdiagonal in the (2, 1) block, but keeps the (1, 1) block lower triangular. We encourage readers to try a simple example.

Each sweep can be performed with several different sequences or *orderings* of circular and hyperbolic rotations. However, we would like to reduce the total number of hyperbolic rotations per sweep, since the hyperbolic rotation does not preserve Euclidean norm and may be unstable. Also, we want to avoid the singular conditions (i) or (ii) of (22), when we apply hyperbolic rotations. Notice that these conditions can occur in the intermediate steps even if the matrix we factor is well conditioned.

However, if we choose the ordering of a sweep such that we rotate the first p blocks and the last q blocks separately, and only then annihilate the resulting subdiagonals of these last q blocks with respect to the resulting subdiagonals of the first p blocks, then we need only 1 hyperbolic rotation per sweep, and it is easy to see that the condition (i) or (ii) does not occur in the triangular factorization of a *positive definite* symmetric matrix or in the QR factorization of a *full rank* matrix.

The sweep procedure above can be described compactly, by introducing the matrix $\mathbf{F}\in\mathbf{R}^{(n+m)\times(p+q)}$ consisting of the column vectors that determine the matrix $\mathbf{\Gamma}$:

$$\mathbf{F}=(f_{i,j})\equiv\begin{pmatrix}\mathbf{w}_1 & \cdots & \mathbf{w}_p & \mathbf{w}_{p+1} & \cdots & \mathbf{w}_{p+q}\\ \mathbf{v}_1 & \cdots & \mathbf{v}_p & -\mathbf{v}_{p+1} & \cdots & -\mathbf{v}_{p+q}\end{pmatrix}. \tag{23}$$

```
Procedure SWEEP (k);
begin {The kth sweep.}
  for i := p − 1 downto 1 do
    begin
      Determine G_{i,i+1} to annihilate f_{k,i+1};
      F := FG_{i,i+1}
    end;
  for i := p + q − 1 downto p + 1 do
    begin
      Determine G_{i,i+1} to annihilate f_{k,i+1};
      F := FG_{i,i+1}
    end;
  Determine H_{1,p+1} to annihilate f_{k,p+1};
  F := FH_{1,p+1}
end;
```

The two algorithms based on Theorems 1 and 2 can be described in terms of the module SWEEP.

ALGORITHM 1. *Triangular factorization of a positive-definite symmetric matrix.*

begin {Assume an admissible generator $G(\mathbf{A})$ with parameter $(\boldsymbol{\mu}, \boldsymbol{\Sigma}\boldsymbol{\mu})$ is given.}
  Form the matrix $\mathbf{F}$;
  for $k := 0$ **to** $n-1$ **do**
    begin {kth sweep}
      SWEEP (k);
      {Now, the vector $\mathbf{w}_1$ gives the $(k+1)$st column of the lower triangular matrix $\mathbf{L}_1$ of $\mathbf{A} = \mathbf{L}_1\mathbf{L}_1^T$.}
      {The vector $\mathbf{v}_1$ gives the $(k+1)$st column of $\mathbf{U}_2$ of $\mathbf{A}^{-1} = \mathbf{U}_2\mathbf{U}_2^T$.}
      $\mathbf{w}_1 := \mathbf{Z}_n\mathbf{w}_1$; {Shift down the vector $\mathbf{w}_1$, by one position.}
      $\mathbf{v}_1 := \mathbf{Z}_n\mathbf{v}_1$ {Shift down the vector $\mathbf{v}_1$ by one position.}
    end
end.

ALGORITHM 2. QR *factorization of a rectangular full rank matrix.*
begin {Assume $G(\mathbf{A})$ is given.}
  Obtain $\mathbf{W}$ and $\mathbf{V}$ (by Lemma 4 and 6), and form the matrix $\mathbf{F}$;
  for $k := 0$ **to** $n-1$ **do**
    begin {kth sweep}
      SWEEP (k);
      {Now, the vector $\mathbf{w}_1$ gives the $(k+1)$st column of the lower triangular matrix $\mathbf{R}^T$ of $\mathbf{A} = \mathbf{QR}$.}
      {The vector $\mathbf{v}_1$ gives the $(k+1)$st column of $\mathbf{Q}$.}
      $\mathbf{w}_1 := \mathbf{Z}_n\mathbf{w}_1$; {Shift down the vector $\mathbf{w}_1$, by one position.}
      $\mathbf{v}_1 := \mathbf{Z}_m\mathbf{v}_1$ {Shift down the vector $\mathbf{v}_1$ by one position.}
    end
end.

Remark 1. Notice that the same basic sweep module can be used either for triangular factorizations or QR factorizations.

Remark 2. In the above algorithms, the nonuniqueness of generators allows us to choose other generators for a given matrix. For instance, we could use the nonminimal generators $G_5(\mathbf{T}^T\mathbf{T})$, $G_5(\mathbf{T})$ obtained from Lemma 4 and 6 for the QR factorization of a Toeplitz matrix, instead of the generators (5) and (10). Different generators might have different numerical behavior.

Remark 3. We can explain Bojanczyk et al.'s QR factorization algorithm of a Toeplitz matrix in our framework. They use the generator (5), but with a different ordering of SWEEP. That is, they annihilate a subdiagonal of $\mathbf{L}(\mathbf{w}_2)$ with respect to $\mathbf{L}(\mathbf{w}_1)$ with a circular rotation, and a subdiagonal of $\mathbf{L}(\mathbf{w}_4)$ with respect to $\mathbf{L}(\mathbf{w}_1)$ with a hyperbolic rotation, and then a subdiagonal of $\mathbf{L}(\mathbf{w}_3)$ with respect to $\mathbf{L}(\mathbf{w}_1)$ with a hyperbolic rotation. This ordering uses 2 hyperbolic rotations, as compared to ours which uses only 1 hyperbolic rotation; it is interesting, however, that this reduction does not seem to affect the numerical behavior of the algorithm.

Operation counts for a Toeplitz matrix. If we apply Algorithm 1 to compute the triangular factorizations of $\mathbf{T} \in \mathbf{R}^{n\times n}$, then we need $n+1$ flops to obtain the generator (4), and $2n^2 - n - 1$ flops for $\mathbf{L}_1$ only, and additional $2n(n-1)$ flops for $\mathbf{U}_2$ and $\mathbf{L}_3$.

For the QR factorization of $\mathbf{T} \in \mathbf{R}^{m \times n}$ with the generator (5), we need $2m + mn + 1$ flops to obtain the generator (5), $6n^2 - 3n - 3$ flops for $\mathbf{R}$, and additional $12m(n-1)$ flops for $\mathbf{Q}$.

If we use the fast rotations which will be explained shortly, these operation counts would be reduced approximately by half.

Fast hyperbolic rotations for $\mathbf{LDL}^T$ *and* $\mathbf{QDR}$ *factorizations.* In the above algorithms, the major part of the computation arises from the application of the rotations $\mathbf{G}$ or $\mathbf{H}$ to the matrix $\mathbf{F}$. The application of a 2×2 circular or hyperbolic rotation to a vector needs 4 multiplications. It was shown [10], [12], [13], [23] that this multiplication count can be reduced to 2 or 3 for a circular rotation. It is straightforward to derive a similar result for hyperbolic rotations. If we use these fast rotations, then we shall get $\mathbf{LDL}^T$ factorization of symmetric nonsingular matrices and $\mathbf{QDR}$ factorization of full rank rectangular matrices, where $\mathbf{D}$ is diagonal and $\mathbf{QD}^{1/2}$ is orthogonal.

Two and three multiplication formulas. Let the two columns in the plane of rotation be given in the factored form:

$$(\mathbf{a}\ \mathbf{b})\begin{pmatrix} d_1 & 0 \\ 0 & d_2 \end{pmatrix}, \quad \mathbf{a} = (a_1, \cdots, a_{n+m})^T, \quad \mathbf{b} = (b_1, \cdots, b_{n+m})^T,$$

and suppose that d_1^2 and d_2^2 (instead of d_1 and d_2) are known. Put

$$(\mathbf{a}\ \mathbf{b})\begin{pmatrix} d_1 & 0 \\ 0 & d_2 \end{pmatrix}\begin{pmatrix} ch & -sh \\ -sh & ch \end{pmatrix} = (\bar{\mathbf{a}}\ \bar{\mathbf{b}})\begin{pmatrix} \bar{d}_1 & 0 \\ 0 & \bar{d}_2 \end{pmatrix},$$

where

$$\bar{\mathbf{a}} = (\bar{a}_1, \cdots, \bar{a}_{n+m})^T, \qquad \bar{\mathbf{b}} = (\bar{b}_1, \cdots, \bar{b}_{n+m})^T.$$

Then we require

$$-shd_1 a_1 + chd_2 b_1 = 0,$$

and for all $1 \leqq i \leqq n+m$,

$$chd_1 a_i - shd_2 b_i = \bar{d}_1 \bar{a}_i, \qquad -shd_1 a_i + chd_2 b_i = \bar{d}_2 \bar{b}_i. \tag{24a}$$

The increased degrees of freedom obtained by introducing the matrix $\operatorname{diag}(d_1, d_2)$ allow us to choose $\operatorname{diag}(\bar{d}_1, \bar{d}_2)$ in several different ways, which lead to two or three multiplication formulas. For instance, if we set

$$\bar{d}_1 = shd_2, \qquad \bar{d}_2 = shd_1, \tag{24b}$$

then

$$(chd_1/shd_2)a_i - b_i = \bar{a}_i, \qquad -a_i + (chd_2/shd_1)b_i = \bar{b}_i, \tag{24c}$$

which is a two multiplication formula. Moreover, the square root computation for ch or sh is not necessary, since

$$chd_1/shd_2 = d_1^2 a_1/d_2^2 b_1, \qquad chd_2/shd_1 = a_1/b_1. \tag{24d}$$

To prepare for the next rotation, we can compute and store $\bar{d}_1^2$ and $\bar{d}_2^2$ from (24b), since

$$sh^2 = d_2^2 b_1^2/(d_1^2 a_1^2 - d_2^2 b_1^2). \tag{24e}$$

We can obtain different two or three multiplication formulas by choosing $\bar{d}_1$ and $\bar{d}_2$, differently.

Finally, we note that the above formulas are different from the fast circular rotation only by some signs in (24). The practical significance of this fact is that the same hardware can be used for both fast circular and fast hyperbolic rotations. Also notice that the magnitudes of d_1 and d_2 of the diagonal matrix do not monotonically decrease or increase with the choice of (24b) in the case of fast hyperbolic rotation (in contrast to fast circular rotations).

Triangular factorization of nonsymmetric matrices. Let $\mathbf{A} \in \mathbf{R}^{n \times n}$ be a nonsymmetric matrix with nonsingular leading principal submatrices. To obtain a triangular factorization of $\mathbf{A}$, we need to find a matrix $\mathbf{\Phi}$ such that

$$(25) \qquad \bar{\mathbf{\Delta}} \equiv \mathbf{\Delta}\mathbf{\Phi} = \begin{pmatrix} \mathbf{L}_1 & 0 & \cdots & 0 \\ \mathbf{U}_2 & * & \cdots & * \end{pmatrix}, \qquad \bar{\mathbf{\Omega}} \equiv \mathbf{\Omega}\mathbf{\Phi}^{-T} = \begin{pmatrix} \mathbf{U}_1^T & 0 & \cdots & 0 \\ \mathbf{L}_2^T & * & \cdots & * \end{pmatrix},$$

where

$$\mathbf{\Delta} = \begin{pmatrix} \mathbf{L}(\mathbf{x}_1) & \cdots & \mathbf{L}(\mathbf{x}_{\hat{\alpha}}) \\ \sigma_1 \mathbf{I} & \cdots & \sigma_{\hat{\alpha}} \mathbf{I} \end{pmatrix}, \qquad \mathbf{\Omega} = \begin{pmatrix} \mathbf{L}(\mathbf{y}_1) & \cdots & \mathbf{L}(\mathbf{y}_{\hat{\alpha}}) \\ \mu_1 \mathbf{I} & \cdots & \mu_{\hat{\alpha}} \mathbf{I} \end{pmatrix}.$$

The matrix $\mathbf{\Phi}$ can be constructed as a product of the matrices $\mathbf{M}_{i,j} \in \mathbf{R}^{n\hat{\alpha} \times n\hat{\alpha}}$, which are identity matrices except for the following 4 entries:

$$\mathbf{M}_{i,j}(i,i) = c, \quad \mathbf{M}_{i,j}(i,j) = -s_1, \quad \mathbf{M}_{i,j}(j,i) = s_2, \quad \mathbf{M}_{i,j}(j,j) = c,$$

where

$$c^2 + s_1 s_2 = 1.$$

The circular and hyperbolic rotations are special cases of $\mathbf{M}_{i,j}$.

Notice that $\mathbf{M}_{i,j}^{-T}$ is of similar form, i.e., it is an identity matrix except for

$$\mathbf{M}_{i,j}^{-T}(i,i) = c, \quad \mathbf{M}_{i,j}^{-T}(i,j) = -s_2, \quad \mathbf{M}_{i,j}^{-T}(j,i) = s_1, \quad \mathbf{M}_{i,j}^{-T}(j,j) = c.$$

We construct $\mathbf{\Phi}$ and $\mathbf{\Phi}^{-T}$ with $\mathbf{M}_{i,j}$ and $\mathbf{M}_{i,j}^{-T}$ such that c, s_1 and s_2 are chosen to annihilate an element in the matrix $\mathbf{\Delta}$ and $\mathbf{\Omega}$. We shall omit the detailed description.

6. Concluding remarks. We have shown that displacement structure is inherited by the product of two matrices. After recognizing this, we presented a fast triangular factorization algorithm and a fast QR factorization algorithm for a full rank matrix, *when its generator is given.*

Remark 1. The computational tools of our algorithms are circular and hyperbolic rotations. Once we have obtained a generator, there are no further inner product computations. The absence of inner products inside the iteration loop has definite advantage in parallel implementation. Our algorithm shows that there are many different choices of generators for the factorization problem. Different generators would lead to numerically different QR factorization algorithms.

Remark 2. Block Toeplitz matrices arise in a multichannel signal processing application. Although we can still talk about block displacement structure, it is always possible to transform any $n \times n$ block Toeplitz matrix having $m \times m$ submatrices (i.e. an $n \times n$ array of submatrices) into an $m \times m$ block matrix with $n \times n$ Toeplitz submatrices by pre- and postmultiplying a permutation matrix $\mathbf{P}$. The resulting permuted matrix has displacement rank of $2m$, and its generator can be obtained by inspection. We can apply the QR factorization algorithm in § 4 to obtain the so-called complete factorization of $\mathbf{B} = \mathbf{QRP}$.

Appendix: Proofs of lemmas.

Proof of Lemma 2. Construct $\mathbf{T}^T\mathbf{T}$ and $\mathbf{Z}_n\mathbf{T}^T\mathbf{T}\mathbf{Z}_n^T$ explicitly, then straightforward manipulation shows that $\mathbf{T}^T\mathbf{T}-\mathbf{Z}_n\mathbf{T}^T\mathbf{T}\mathbf{Z}_n^T$ can be expressed as a sum of 4 rank one matrices. □

Proof of Lemma 3. Notice that $\mathbf{Z}_n^T\mathbf{Z}_n=\mathbf{I}-\mathbf{e}_n\mathbf{e}_n^T$. Therefore,

$$\begin{aligned}\mathbf{AB}-\mathbf{Z}_m\mathbf{ABZ}_s^T &= \mathbf{AB}-\mathbf{Z}_m\mathbf{A}(\mathbf{Z}_n^T\mathbf{Z}_n-\mathbf{e}_n\mathbf{e}_n^T)\mathbf{BZ}_s^T\\ &= \mathbf{AB}-(\mathbf{Z}_m\mathbf{AZ}_n^T)(\mathbf{Z}_n\mathbf{BZ}_s^T)-(\mathbf{Z}_m\mathbf{Ae}_n)(\mathbf{e}_n^T\mathbf{BZ}_s^T)\\ &= \tfrac{1}{2}(\mathbf{A}-\mathbf{Z}_m\mathbf{AZ}_n^T)(\mathbf{B}+\mathbf{Z}_n\mathbf{BZ}_s^T)+\tfrac{1}{2}(\mathbf{A}+\mathbf{Z}_m\mathbf{AZ}_n^T)\\ &\quad\cdot(\mathbf{B}-\mathbf{Z}_n\mathbf{BZ}_s^T)-(\mathbf{Z}_m\mathbf{Ae}_n)(\mathbf{e}_n^T\mathbf{BZ}_s^T).\end{aligned}$$

Hence

$$\alpha(\mathbf{AB})\leqq\alpha(\mathbf{A})+\alpha(\mathbf{B})+1,$$

since

$$\begin{aligned}&\operatorname{rank}(\mathbf{A}_1\mathbf{A}_2)\leqq\min(\operatorname{rank}(\mathbf{A}_1),\operatorname{rank}(\mathbf{A}_2)),\\ &\operatorname{rank}(\mathbf{A}_1+\mathbf{A}_2)\leqq\operatorname{rank}(\mathbf{A}_1)+\operatorname{rank}(\mathbf{A}_2).\end{aligned}$$ □

Proof of Lemma 4. Following the proof of Lemma 3 we obtain, assuming $G_{\hat{\alpha}}(\mathbf{A})=\{\mathbf{X},\mathbf{Y}\}$, that

$$\begin{aligned}\mathbf{A}^T\mathbf{A}-\mathbf{ZA}^T\mathbf{AZ}^T &= \tfrac{1}{2}(\mathbf{A}^T-\mathbf{ZA}^T\mathbf{Z}^T)(\mathbf{A}+\mathbf{ZAZ}^T)\\ &\quad+\tfrac{1}{2}(\mathbf{A}^T+\mathbf{ZA}^T\mathbf{Z}^T)(\mathbf{A}-\mathbf{ZAZ}^T)-(\mathbf{ZA}^T\mathbf{e}_m)(\mathbf{e}_m^T\mathbf{AZ}^T)\\ &= \tfrac{1}{2}\mathbf{YX}^T(\mathbf{A}+\mathbf{ZAZ}^T)+\tfrac{1}{2}(\mathbf{A}^T+\mathbf{ZA}^T\mathbf{Z}^T)\mathbf{XY}^T-(\mathbf{ZA}^T\mathbf{e}_m)(\mathbf{e}_m^T\mathbf{AZ}^T).\end{aligned}$$

Therefore, a symmetric generator of $\mathbf{A}^T\mathbf{A}$ is $\{\mathbf{W},\mathbf{W\Sigma}\}$ where

$$\mathbf{W}=\left(\frac{(\mathbf{A}^T+\mathbf{ZA}^T\mathbf{Z}^T)\mathbf{X}+\mathbf{Y}}{2},\frac{(\mathbf{A}^T+\mathbf{ZA}^T\mathbf{Z}^T)\mathbf{X}-\mathbf{Y}}{2},\mathbf{ZA}^T\mathbf{e}_m\right)$$

and

$$\mathbf{\Sigma}=\begin{pmatrix}\mathbf{I}_{\hat{\alpha}} & 0 & 0\\ 0 & -\mathbf{I}_{\hat{\alpha}} & 0\\ 0 & 0 & -1\end{pmatrix}.$$

Since only the generator $\{\mathbf{X},\mathbf{Y}\}$ is known a priori, it is more convenient to express $\mathbf{W}$ as

$$\mathbf{W}=\left(\mathbf{ZA}^T\mathbf{Z}^T\mathbf{X}+\frac{\mathbf{YX}^T\mathbf{X}+\mathbf{Y}}{2},\mathbf{ZA}^T\mathbf{Z}^T\mathbf{X}+\frac{\mathbf{YX}^T\mathbf{X}-\mathbf{Y}}{2},\mathbf{ZA}^T\mathbf{e}_m\right).$$

The computation required to obtain $\mathbf{W}$ can be further reduced by introducing the *equivalent* generator $\{\mathbf{XU},\mathbf{YU}^{-1}\}$ where $\mathbf{U}$ is an arbitrary diagonal matrix, viz.,

$$\mathbf{U}=\operatorname{diag}(u_1,u_2,\cdots,u_{\hat{\alpha}}).$$

The corresponding generator of $\mathbf{A}^T\mathbf{A}$ now involves

$$\tilde{\mathbf{W}}\equiv\left(\mathbf{ZA}^T\mathbf{Z}^T\mathbf{XU}+\frac{\mathbf{YX}^T\mathbf{XU}+\mathbf{YU}^{-1}}{2},\mathbf{ZA}^T\mathbf{Z}^T\mathbf{XU}+\frac{\mathbf{YX}^T\mathbf{XU}-\mathbf{YU}^{-1}}{2},\mathbf{ZA}^T\mathbf{e}_m\right),$$

where the elements of $\mathbf{U}$ can be chosen to simplify $\mathbf{W}$. For instance, the kth column of $\mathbf{YX}^T\mathbf{XU}-\mathbf{YU}^{-1}$ is

$$\left(\sum_i \mathbf{y}_i\mathbf{x}_i^T\right)\mathbf{x}_k u_k - \mathbf{y}_k u_k^{-1} = \left(\sum_{i\neq k} \mathbf{y}_i\mathbf{x}_i^T\right)\mathbf{x}_k u_k + \mathbf{y}_k\|\mathbf{x}_k\|^2 u_k - \mathbf{y}_k u_k^{-1},$$

where $\mathbf{y}_k$ is the kth column of $\mathbf{Y}$. Thus, if we choose $u_k = \|\mathbf{x}_k\|^{-1}$, the above expression reduces to

$$\left(\sum_{i\neq k} \mathbf{y}_i\mathbf{x}_i^T\right)\mathbf{x}_k\|\mathbf{x}_k\|^{-1}.$$

This choice of u_k results in (7), with $\mathbf{A}^T$ replaced by $\sum_{i=1}^{\hat{\alpha}} \mathbf{L}(\mathbf{y}_i)\mathbf{L}^T(\mathbf{x}_i)$. □

Proofs of Lemma 5 and Lemma 6. Use formula (7), and the terms $\sum_{i=1}^{\hat{\alpha}} \mathbf{L}(\mathbf{y}_i)\mathbf{Z}_n\mathbf{Z}_n^T\mathbf{L}^T(\mathbf{x}_i)$ cancel out. □

REFERENCES

[1] A. W. BOJANCZYK, R. P. BRENT AND F. R. DE HOOG, QR *factorization of Toeplitz matrices*, Numer. Math., 49 (1986), pp. 81-94.

[2] R. BITMEAD AND B. ANDERSON, *Asymptotically fast solution of Toeplitz and related systems of equations*, Linear Algebra Appl., 34 (1980), pp. 103-116.

[3] R. BRENT, F. GUSTAVSON AND D. YUN, *Fast solution of Toeplitz systems of equations and computation of Padé approximations*, J. Algorithms, 1 (1980), pp. 259-295.

[4] J. BUNCH, *Stability of methods for solving Toeplitz systems of equations*, this Journal, 6 (1985), pp. 349-364.

[5] G. CYBENKO, *Fast Toeplitz orthogonalization using inner products*, this Journal, 8 (1987), pp. 734-740.

[6] J. DELOSME, *Algorithms and architectures for finite shift-rank processes*, Ph.D. dissertation, Dept. of Electrical Engineering, Stanford Univ., Stanford, CA, 1982.

[7] P. DELSARTE, Y. GENIN AND Y. KAMP, *A polynomial approach to the generalized Levinson algorithm based on the Toeplitz distance*, IEEE Trans. Inform. Theory, IT-29 (1983), pp. 268-278.

[8] J. DURBIN, *The fitting of time-series models*, Rev. Inter. Inst. Stat., 28 (1960), pp. 233-244.

[9] B. FRIEDLANDER, T. KAILATH, M. MORF AND L. LJUNG, *Extended Levinson and Chandrasekhar equations for general discrete time linear estimation problems*, IEEE Trans. Automat. Control, AC-23 (1978), pp. 653-659.

[10] M. GENTLEMAN, *Least squares computations by Givens transformations without square roots*, J. Inst. Math. Appl., 12 (1973), pp. 329-336.

[11] I. GOHBERG AND A. SEMENCUL, *On the inversion of finite Toeplitz matrices and their continuous analogs*, Mat. Issled., 2 (1972), pp. 201-233.

[12] G. H. GOLUB AND C. VAN-LOAN, *Matrix Computations*, The Johns Hopkins Univ. Press, Baltimore, 1983.

[13] S. HAMMARLING, *A note on modifications to the Givens plane rotation*, J. Inst. Math. Appl., 13 (1974), pp. 215-218.

[14] T. KAILATH, *A view of three decades of linear filtering theory*, IEEE Trans. Inform. Theory, IT-20 (1974), pp. 146-181.

[15] ———, *Signal processing in the* VLSI *era*, in VLSI and Modern Signal Processing, S. Kung, H. Whitehouse and T. Kailath, eds., Prentice-Hall, Englewood Cliffs, NJ, 1985.

[16] T. KAILATH, A. M. BRUCKSTEIN AND D. MORGAN, *Fast matrix factorization via discrete transmission lines*, Linear Algebra Appl., 75 (1986), pp. 1-25.

[17] T. KAILATH, S. KUNG AND M. MORF, *Displacement ranks of matrices and linear equations*, J. Math. Anal. Appl., 68 (1979), pp. 395-407; Bull. Amer. Math. Soc., 1 (1979), pp. 769-773.

[18] T. KAILATH, B. LEVY, L. LJUNG AND M. MORF, *The factorization and representation of operators in the algebra generated by Toeplitz operators*, SIAM J. Appl. Math., 37 (1979), pp. 467-484.

[19] H. LEV-ARI AND T. KAILATH, *Triangular factorization of structured Hermitian matrices*, in Schur Methods in Operator Theory and Signal Processing, Operator Theory: Advances and Applications, I, Vol. 18, I. Gohberg, ed., Birkhäuser-Verlag, Basel, 1986, pp. 301-324.

[20] ———, *Lattice filter parameterizations and modeling of nonstationary process*, IEEE Trans. Inform. Theory, IT-30 (1984), pp. 2-16.

[21] N. LEVINSON, *The Wiener rms (root-mean-square) error criterion in filter design and prediction*, J. Math. Phys., 25 (1947), pp. 261–278.

[22] M. MORF, *Doubling algorithms for Toeplitz and related equations*, Proc. 1980 IEEE Internat. Conf. on Acoust., Speech, Signal Processing, 1980, pp. 954–959.

[23] B. PARLETT, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[24] I. SCHUR, *Über Potenzreihen, die im Innern des Einheitskreises beschrankt sind*, J. Reine Angew. Math., 147 (1917), pp. 205–232.

[25] G. STEWART, *The effects of rounding error on an algorithm for downdating a Cholesky factorization*, J. Inst. Math. Appl., 23 (1979), pp. 203–213.

[26] D. R. SWEET, *Fast Toeplitz orthogonalization*, Numer. Math., 43 (1984), pp. 1–21.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 6, November 1987
© 1987 Society for Industrial and Applied Mathematics

MULTIGRID ON COMPOSITE MESHES*

W. D. HENSHAW† AND G. CHESSHIRE‡

Abstract. The multigrid method is applied to the numerical solution of elliptic equations on general composite overlapping meshes. Computational results show that good convergence rates are obtained.

Key words. composite meshes, overlapping grids, multigrid

AMS(MOS) subject classification. 65N20

1. Introduction. We describe the application of the multigrid method to the solution of elliptic partial differential equations (PDEs) on two-dimensional regions which have been discretized using composite overlapping grids. A general purpose code, CGMG, has been developed (in FORTRAN) which can solve problems on composite meshes created by the grid construction program CMPGRD, Chesshire [5]. With CMPGRD the user may create a composite grid containing any number of component grids at any number of multigrid levels. CGMG can then be used to solve elliptic PDE boundary value problems.

A composite overlapping grid consists of a number of simpler *component* grids. These component grids cover a region and overlap where they meet. Functions defined on the composite mesh are matched by interpolation at the overlapping grid boundaries. The problem of generating grids for regions of complicated geometry can be difficult, especially for those grid generation algorithms which attempt to fit a single global grid. With a composite overlapping grid, however, the component grids can be generated almost independently of each other. Each component grid can be stretched and refined with little effect on the other component grids. The numerical solution of PDEs on such grids has been examined by, among others, Starius [10], [11], Reyna [9], Kreiss [8], Atta and Vadyak [1], Benek et al. [2], Henshaw [7] and Berger [3].

The multigrid method is a fast iterative method for the solution of elliptic problems. Multigrid utilizes a sequence of grids of varying degrees of coarseness to accelerate the convergence of the solution on the finest grid. The basic principle rests on the fact that it is possible to obtain iterative procedures (*smoothers*) for which the high frequency components of the solution converge rapidly. This means that after a few smoothing iterations the part of the solution yet to converge is smooth and hence can be accurately solved for on a coarser grid. Discussions of the multigrid method in general can be found, for example, in Brandt [4], or Stüben and Trottenberg [12]. Multigrid on a model composite mesh has been described in Stüben and Trottenberg [12] and for more general two component meshes in Henshaw [7].

In the next section we outline the implementation of the multigrid algorithm on fairly general composite meshes. This includes a description of the technique we use to discretize PDEs on overlapping meshes. In § 3 numerical results are presented for some test cases. These results show that the good convergence rates expected from multigrid can be obtained on composite meshes.

* Received by the editors March 3, 1986; accepted for publication (in revised form) December 8, 1986. This work was supported by National Science Foundation grant DMS-8312264 and Office of Naval Research grant N00014-83-K-0422.

† Department of Applied Mathematics, California Institute of Technology, Pasadena, California 91125. Present address, IBM, Thomas J. Watson Research Center, Yorktown Heights, New York 10598.

‡ Department of Applied Mathematics, California Institute of Technology, Pasadena, California 91125. Present address, Intel Scientific Computers, Beaverton, Oregon 97006.

2. Description. The multigrid code CGMG was written to solve linear, variable coefficient elliptic PDEs of the form

$$Lu := c_{xx}\frac{\partial^2 u}{\partial x^2} + c_{xy}\frac{\partial^2 u}{\partial x\,\partial y} + c_{yy}\frac{\partial^2 u}{\partial y^2} + c_x\frac{\partial u}{\partial x} + c_y\frac{\partial u}{\partial y} + c_c u = f \quad \text{in } \Omega, \tag{1a}$$

$$Bu := b_x\frac{\partial u}{\partial x} + b_y\frac{\partial u}{\partial y} + b_c u = g \quad \text{on } \partial\Omega. \tag{1b}$$

Periodic boundary conditions are also allowed. Currently a second-order difference approximation has been implemented.

Let us first describe how the above PDE boundary value problem is discretized on a composite mesh, without any references to multigrid. The grid construction program CMPGRD can be used to generate a composite overlapping mesh for the region Ω. This mesh is composed of one or more component meshes. See, for example, the composite meshes of Figs. 1 and 2. Each component mesh is logically rectangular,

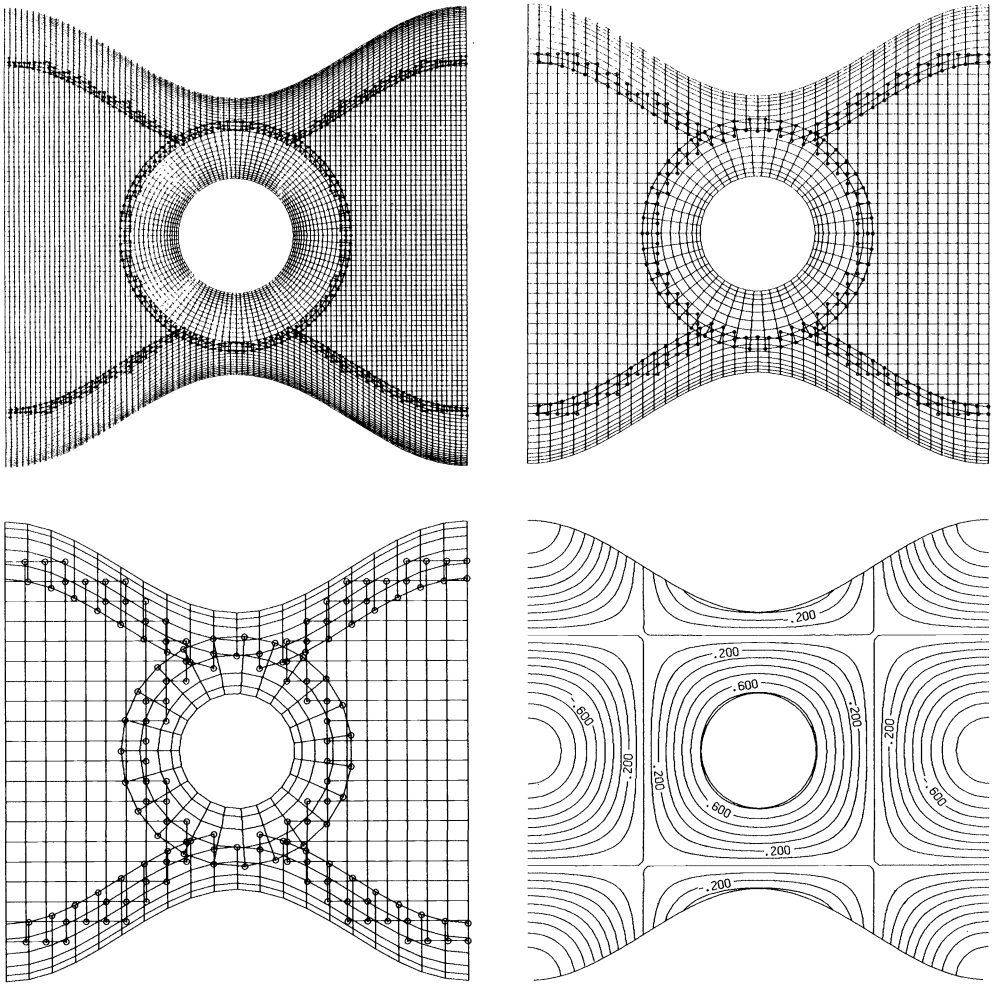


FIG. 1. *Grid* 1: *Composite meshes for multigrid levels* 1, 2 *and* 3 *and a contour plot of the calculated solution.*

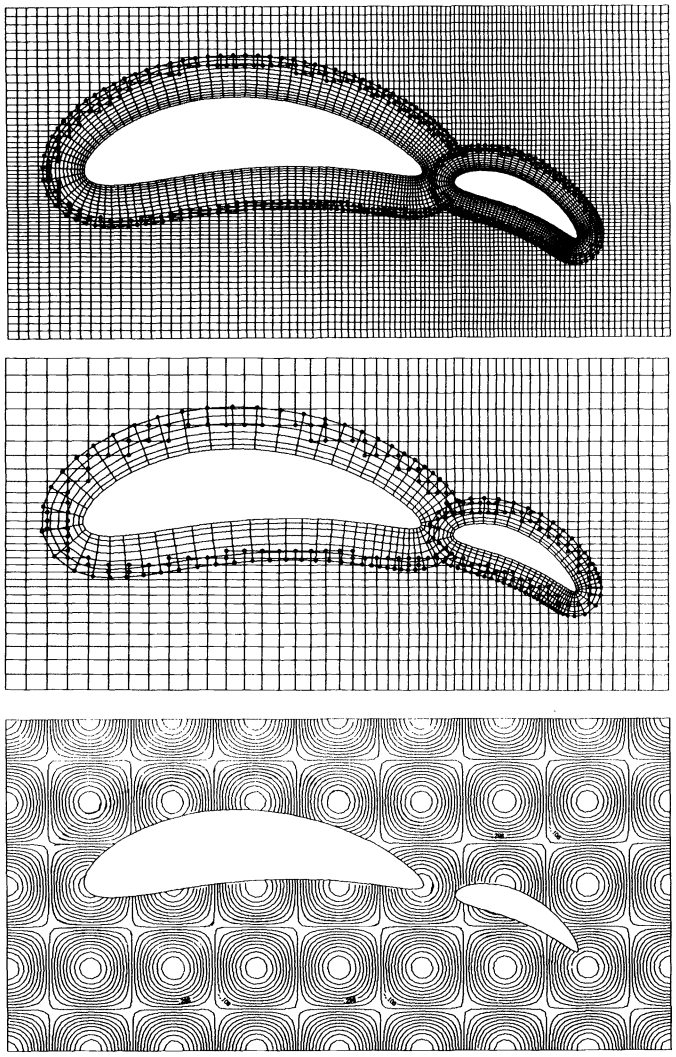


FIG. 2. *Grid* 2: *Composite meshes for multigrid levels* 1 *and* 2 *and a contour plot of the calculated solution.*

although some points in overlap regions are discarded. A point on a component grid will be one of the following four types:

(i) An *interior* point where the PDE (1a) should be applied.

(ii) A *boundary* point, which corresponds to $\partial\Omega$, where the boundary condition (1b) holds.

(iii) A point where the solution is matched, by interpolation to the solution on other component grids. We sometimes call the set of all such interpolation points the *interpolation boundary.*

(iv) A point in a region of overlap which is not used.

There are a number of possible ways to generate the discrete approximations to the PDE (1a), the boundary conditions (1b) and the interpolation equations. The approach we take is a *mapping* method and proceeds as follows. Each component grid is simple enough so that it can be mapped smoothly to a unit square (coordinates (r, s)). The

PDE can be written in these (r, s) coordinates.

$$Lu = c_{rr}\frac{\partial^2 u}{\partial r^2} + c_{rs}\frac{\partial^2 u}{\partial r\,\partial s} + c_{ss}\frac{\partial^2 u}{\partial s^2} + c_r\frac{\partial u}{\partial r} + c_s\frac{\partial u}{\partial s} + c_c u = f,$$

$$Bu = b_r\frac{\partial u}{\partial r} + b_s\frac{\partial u}{\partial s} + b_c u = g.$$

The coefficients in these equations depend on derivatives of the mapping. These derivatives are supplied as output from the composite grid construction program CMPGRD. The solution of a PDE on a composite mesh can then be considered as the solution of PDEs on a sequence of unit squares. The solutions on the squares are coupled to each other through the interpolation boundaries. Let the discrete solution at point (i, j) on component grid k be denoted by $v_k(i, j)$. Then for each k we obtain discrete approximations to the elliptic equation and the boundary conditions of the form

$$L_k v_k = f_k, \tag{2a}$$

$$B_k v_k = g_k. \tag{2b}$$

A point (i, j, k) on an interpolation boundary is interpolated from some other component grid k'. CMPGRD supplies the position (r', s') of the point (i, j, k) on grid k'. Hence standard interpolation formulae for rectangular grids can be applied:

$$v_k(i, j) = \sum_{i', j'} \alpha_{kk'}(i, j, i', j')\, v_{k'}(i', j'). \tag{2c}$$

These equations (2a), (2b) and (2c) will be written as a single linear system:

$$\mathbf{A}\mathbf{v} = \mathbf{f}. \tag{3}$$

The vector $\mathbf{v}$ of all unknowns will be called a composite mesh function. $\mathbf{A}$ is an example of a composite mesh operator, mapping one composite mesh function to another composite mesh function.

The mesh equations (3) can be solved in any number of ways; using multigrid is but one possibility. If there are not too many equations the system can be solved directly. Sparse matrix routines [6] have been used for this purpose. For large systems iterative methods become attractive. Many standard iterative methods are applicable. The matrix is not symmetric, however, so that some schemes do not apply. In general it is best to try and solve all the equations simultaneously, rather than iterating for too long on one component grid. The work of J. Linden, as reported in Stüben and Trottenberg [12], indicates that an iteration based on the Schwartz alternating procedure (where the equations on each component grid are solved exactly before updating interpolation boundaries) is slower and sensitive to the amount of overlap. In contrast it was found that by using the smoothers presented below, the multigrid convergence rate was fairly insensitive to the amount of overlap. The general principle to follow seems to be to iterate in such a way that, at any time, all equations have converged to about the same degree.

2.1. Multigrid. The multigrid algorithm can be applied to the solution of the mesh equations (3). Once a composite mesh has been constructed using CMPGRD it is a simple matter to have CMPGRD generate the sequence of coarser composite meshes which are used for the multigrid algorithm. Figures 1 and 2 show some composite meshes which have been generated for multigrid. Denote the finest composite mesh, level 1, by $\mathbf{M}^1$ and successively coarser meshes by $\mathbf{M}^l$, $l = 2, 3, \cdots$. Note that for simplicity the composite meshes at the different levels all have the same number of component meshes. The elliptic PDE boundary value problem can be discretized on each of the composite meshes $\mathbf{M}^l$. Let $\mathbf{v}^l$ denote the composite mesh function for level

l. Then at each level l there will be mesh equations of the form (3):

$$\mathbf{A}^l\mathbf{v}^l = \mathbf{f}^l. \tag{4}$$

Now let us outline the multigrid algorithm as it applies to composite meshes. Let $\mathbf{v}^1(p)$ be the pth iterate in the solution of the mesh equations on the finest mesh $\mathbf{M}^1$.

MULTIGRID ALGORITHM.
while not converged **do**
  smooth ν_1 times
    $\mathbf{v}^1(*) \leftarrow (\mathbf{S}^1)^{\nu_1}\mathbf{v}^1(p)$
  compute the defect and transfer to the coarser grid
    $\mathbf{f}^2 \leftarrow \mathbf{R}^{1\to 2}(\mathbf{f}^1 - \mathbf{A}^1\mathbf{v}^1(*))$
  "solve" the defect equation on the coarser grid
    $\mathbf{v}^2 \leftarrow (\mathbf{A}^2)^{-1}\mathbf{f}^2$
  correct the fine grid solution from coarse grid solution
    $\mathbf{v}^1(**) \leftarrow \mathbf{v}^1(*) + \mathbf{P}^{2\to 1}\mathbf{v}^2$
  smooth ν_2 times
    $\mathbf{v}^1(p+1) \leftarrow (\mathbf{S}^1)^{\nu_2}\mathbf{v}^1(**)$
end while

This is the basic defect correction scheme. The smoothing operator $\mathbf{S}^1$, the restriction operator $\mathbf{R}^{1\to 2}$ and the prolongation operator $\mathbf{P}^{2\to 1}$ will be described in the context of composite grids. The defect equation need only be "solved" approximately. This approximate solution can be obtained by multigrid, in which case the algorithm becomes recursive. At the coarsest level the equations are usually solved directly.

Smoothers. The smoothing operator $\mathbf{S}$ for composite grids consists of smoothing each component grid and updating the interpolation boundaries. A component grid may be smoothed with any of the standard smoothers that exist. The program CGMG allows the user to choose from a number of possibilities including Gauss-Seidel, Red-Black-Gauss-Seidel, Zebra line smoothers and alternating Zebra line smoothers. Each component grid can have a different smoother; the smoother can be tailored to the grid. For example, a particular component grid might be stretched to resolve a boundary layer, in which case one can use a line smoother in the appropriate direction.

There is some freedom as to the order of smoothing and interpolation. One possibility is to smooth all component grids before updating the interpolation boundaries:

$$\mathbf{S}^l = \begin{cases} \text{Smooth first component grid,} \\ \text{Smooth second component grid,} \\ \quad\vdots \\ \text{Smooth last component grid,} \\ \text{Interpolate.} \end{cases}$$

From experience, however, it seems that a good procedure involves interpolating after each component grid is smoothed:

$$\mathbf{S}^l = \begin{cases} \text{Smooth first component grid,} \\ \text{Interpolate.} \\ \text{Smooth second component grid,} \\ \text{Interpolate.} \\ \quad\vdots \end{cases}$$

The latter composite smoother requires more interpolations but this extra work is usually small compared to the smoothing operations.

Restriction operators (fine to coarse grid transfer). The defect computed on a given level is transferred to the next coarsest level by the restriction operator $\mathbf{R}^{1\to 2}$. (The superscript $1\to 2$ indicates that this operator maps mesh functions on $\mathbf{M}^1$ to mesh functions on $\mathbf{M}^2$.) A typical restriction operator determines the value at points on the coarse grid as some weighted average of the surrounding points on the fine grid. We use the so-called full weighting restriction. The defects in the boundary equations are averaged separately from the defects in the interior equations; boundary defects are averaged along the boundary line. Since the final stage of the smoothing operation involves an interpolation, the defects in the interpolation equations are all zero. Hence no defect need be transferred at these points.

Prolongation operators (coarse to fine grid transfer). The prolongation operator $\mathbf{P}^{2\to 1}$ maps the coarse grid solution to the fine grid. This mapping usually takes the form of an interpolation. We use second order interpolation. Interpolation boundary values could be corrected as well. However, once all other values have been corrected the interpolation equations can be solved to update the interpolation boundary. It turns out that less overlap is needed on finer meshes for the latter approach.

Choice of parameters and cycle. An important part of the multigrid algorithm is the choice of the parameters ν_1, ν_2, etc. and the choice of cycle. *Cycle* is the term used to denote the sequence in which the different levels of grids are traversed. The program dynamically determines the type of cycle and the values for ν_1 and ν_2 in a manner similar to that described by Brandt [4]. There are two basic principles:

(i) Perform smoothing iterations until the smoothing rate (the reduction in residual per iteration) becomes larger than some value η, where $\eta \approx .6$.

(ii) Return to a finer grid once the residual at this level has been reduced by a certain factor δ with $\delta \approx .1$.

We have found it very helpful to monitor the smoothing rates and component grid residuals as the iteration proceeds. This information can be used to determine what changes can be made to improve the convergence rate. If, for example, one component grid is converging slowly one can choose a better component smoother for that grid. Ideally the program should automatically choose the parameters η and δ and the types of component smoothers. A program with this level of sophistication has not yet been developed.

Remarks on grid construction. For practical reasons the amount of overlap between component grids is kept fairly small. This reduces the number of computational points. With sufficient overlap, points on an interpolation boundary can be explicitly interpolated from noninterpolation points on other component grids. However, as the amount of overlap decreases the interpolation equations may couple interpolation points from different component grids. In this case a system of equations must be solved to obtain values on the interpolation boundaries in terms of other values. Of course, as the overlap goes to zero these equations may become singular. The behaviour of the numerical solution to a model elliptic problem, as a function of the amount of overlap, was considered in Henshaw [7]. Suppose that the amount of overlap goes to zero as the grid is refined. To maintain accuracy, the order of accuracy of the interpolation formulae must be greater than the order of accuracy of the interior formulae. In the results presented here we use second order accurate approximations to the elliptic equations and the amount of overlap is required to be greater than one half a grid line. In this case the interpolation formulae should be third order accurate.

To make the restriction and prolongation operators simple the composite meshes at different multigrid levels are strongly related. For example, interior points on a

coarse grid coincide with interior points on finer grids. In Henshaw [7] the coarse grid was generated first and all finer grids were generated by simply doubling the number of grid lines. This gave reasonable results for composite grids with 2 component grids. However, this method does not immediately generalize to more component grids and it tends to lead to more overlap on the finer grids than is really necessary. Hence the grid construction program CMPGRD was designed explicitly to meet the requirements of the multigrid routine [5].

Remarks on programming. A computer code for solving problems on composite meshes is somewhat more complicated than a code written for a single grid. One of the major difficulties arises in the handling of all the data which describes the composite grid. We have worked out a scheme for storing the composite grid data in an efficient and flexible manner. Physically, all the data is stored on a single array. Logically, the data is stored in a "directory-tree" fashion. Variables can be stored or accessed by name using utility routines. Arrays can be stored with a minimum of wasted space. With this storage structure the composite grid data can be easily passed to subroutines.

3. Numerical results. In this section we present results from the multigrid solver CGMG. The composite grids used in the examples are shown in Figs. 1 and 2. Grid 1 (Fig. 1) has 3 levels and Grid 2 (Fig. 2) has 2 levels. The interpolation points are marked with small circles. Notice that the finer meshes are not simply the coarser meshes with double the number of lines. There is less overlap between component grids on the finer meshes. One restriction associated with composite overlapping grids is illustrated by these figures; for complicated regions the most coarse mesh that can be constructed will still have a substantial number of grid points.

In the two test examples which follow we solve

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=f, \qquad f=-2\pi^2 \cos \pi x \cos \pi y. \tag{5}$$

The boundary conditions are chosen so that the true solution is

$$u_{\text{true}}=\cos \pi x \cos \pi y.$$

One reason for solving this rather simple problem is to allow the reader to compare the results presented here to other well known results for Poisson's equation. In addition it is shown that similar convergence rates are obtained for solving the same problem on two widely different composite meshes.

Test 1. The Poisson equation (4) is solved on Grid 1. This grid might be used to study flow around an obstacle in a varying channel. Periodic boundary conditions are applied at the left and right ends of the channnel. Dirichlet boundary conditions, $u=u_{\text{true}}$, are given at all other boundaries. For smoothers we chose Red-Black on the rectangular grid and line-Zebra smoothers for the 3 curvilinear grids. The Zebra smoothers are on lines normal to the boundaries since this is the direction in which the grids are stretched. Smoothing was performed until the smoothing rate became greater than $\eta=.6$. The residual at level 2 had to be decreased by a factor of $\delta=.01$ before the program would return to level 1. These parameters resulted in a "W" cycle, that is the levels were traversed in the order (1, 2, 3, 2, 3, 2, 1). A value of $\delta=.1$ resulted in a "V" cycle, (1, 2, 3, 2, 1), which gave slightly inferior results.

Test 2. Equation (4) is solved on Grid 2 with Dirichlet boundary conditions at all boundaries. Grid 2 is meant to represent an airfoil with a flap. Alternating Zebra

smoothers were used on all component grids since they all are stretched in both the r and s directions. A value of $\eta = .36 = (.6)^2$ was chosen since the alternating smoothers comprise two sweeps.

Convergence results for the two tests are summarized in Table 1. $r(p)$ is the residual on the finest composite mesh after the pth multigrid iteration,

$$r(p) = \|\mathbf{f}^1 - \mathbf{A}^1\mathbf{v}^1(p)\|.$$

Define $WU(p)$ to be the number of work units used for the pth iteration. A work unit is the amount of work (number of multiplications) to perform one iteration of SOR on the composite mesh. The effective convergence rate, ECR, is defined as

$$\text{ECR}(p) = \left(\frac{r(p)}{r(p-1)}\right)^{\rho}, \qquad \rho = (WU(p))^{-1}.$$

Theoretically the ECR for multigrid should be independent of the grid spacing h as $h \to 0$. In contrast the (effective) convergence rate for many other standard iterative schemes deteriorates as $h \to 0$. For optimal SOR, $\text{ECR} \approx 1 - c_1 h$, and for Gauss-Seidel, $\text{ECR} \approx 1 - c_2 h^2$. In Table 1 the ECR increases on the fourth iteration for Grid 1 since the solution has almost converged to single precision accuracy.

In Tables 2 and 3 the errors in the solution to the elliptic problem are given. The problem was solved on the fine mesh (level 1) and also on each of the coarser meshes. The error on component grid k, multigrid level l, $e(k, l)$, is the maximum difference

TABLE 1
Convergence rates.

	Grid 1		Grid 2	
Iteration	$r(p)/r(p-1)$	ECR (p)	$r(p)/r(p-1)$	ECR (p)
$p=1$	.019	.71	.091	.73
$p=2$	.065	.72	.033	.63
$p=3$	.055	.71	.11	.75
$p=4$	.385	.89	.14	.77

TABLE 2
Errors in solution for Grid 1.

l	k	n_r	n_s	$e(k, l)$	$e(k, l)/e(k, 1)$
1	1	93	77	1.5×10^{-3}	1
	2	81	21	1.1×10^{-3}	1
	3	81	21	1.1×10^{-3}	1
	4	93	17	1.7×10^{-3}	1
2	1	47	39	7.4×10^{-3}	5
	2	41	11	5.4×10^{-3}	4.8
	3	41	11	5.4×10^{-3}	4.8
	4	47	9	7.4×10^{-3}	4.4
3	1	24	20	3.7×10^{-2}	25
	2	21	6	2.6×10^{-2}	23
	3	21	6	2.6×10^{-2}	23
	4	24	5	3.3×10^{-2}	20

TABLE 3
Errors in solution for Grid 2.

l	k	n_r	n_s	$e(k,l)$	$e(k,l)/e(k,1)$
1	1	101	61	1.8×10^{-2}	1
	2	121	13	2.0×10^{-2}	1
	3	111	13	5.7×10^{-2}	1
2	1	51	31	9.8×10^{-2}	5.4
	2	61	7	8.7×10^{-2}	4.4
	3	56	7	2.7×10^{-2}	4.7

between the calculated and true solution. The number of points on the component grids in the r and s directions is given by n_r and n_s, respectively. Not all the points are used since there is overlap between the grids. Level 1 of Grid 1 has 8431 computational points while there are 7431 computational points on the finest level of Grid 2. Contour plots of the calculated solutions are shown in Figs. 1 and 2.

4. Summary. We have given a brief description of a computer code, CGMG, which solves elliptic PDEs on general composite overlapping meshes using the multigrid algorithm. Multigrid meshes can be created using the composite grid construction program CMPGRD. Care has been taken in the grid construction to keep the amount of overlap to a minimum. The multigrid algorithm for composite meshes involves only minor modifications to its standard form. These changes are necessary in order to handle the interpolation points of the overlapping grids. We have shown the results of solving Poisson's equation with Dirichlet boundary conditions on two nontrivial composite meshes. The numerical solutions were shown to be accurate. Convergence rates were obtained which compare favourably with those rates which might be obtained on a more simple grid.

Acknowledgments. Computations were performed on the Fluid Dynamics VAX 11/750 at the California Institute of Technology. The authors would like to thank Professor Heinz-Otto Kreiss for his encouragement and support.

REFERENCES

[1] E. H. ATTA AND J. VADYAK, *A grid overlapping scheme for flowfield computations about multicomponent configurations*, AIAA J., 21 (1983), pp. 1271–1277.

[2] J. A. BENEK, J. L. STEGER AND F. C. DOUGHERTY, *A flexible grid embedding technique with application to the Euler equations*, AIAA Paper No. 83-1944, Danvers, MA, 1983.

[3] M. BERGER, *On conservation at grid interfaces*, ICASE report no. 84-43, NASA Langley Research Center, Hampton, VA, 1984.

[4] A. BRANDT, *Guide to multigrid development*, in Multigrid Methods, W. Hackbusch and U. Trottenberg, eds., Springer-Verlag, Berlin, 1982, pp. 220–312.

[5] G. S. CHESSHIRE, *Composite grid construction and applications*, Ph.D. thesis, Dept. Appl. Math., California Institute of Technology, Pasadena, CA, 1986.

[6] S. C. EISENSTAT, M. C. GURSKY, M. H. SCHULTZ AND A. H. SHERMAN, *Yale sparse matrix package*, Research Rpts. 112, 114, Yale University, Dept. Comput. Sci., New Haven, CT, May 1977.

[7] W. D. HENSHAW, *Part* I: *The numerical solution of hyperbolic systems of conservation laws*; *Part* II: *Composite overlapping grid techniques*, Ph.D. thesis, Dept. Appl. Math., California Institute of Technology, Pasadena, CA, 1985.

[8] B. KREISS, *Construction of a curvilinear grid*, this Journal, 4 (1983), pp. 270–279.

[9] L. G. M. REYNA, I. *Stability of Tchebyshev collocation*; II. *Interpolation for surfaces with* 1-D *discontinuities*; III. *On composite meshes*, Ph.D. thesis, Dept. Appl. Math., California Institute of Technology, Pasadena, CA, 1982.

[10] G. STARIUS, *Composite mesh difference methods for elliptic boundary value problems*, Numer. Math., 28 (1977), pp. 243-258.

[11] ———, *On composite mesh difference methods for hyperbolic difference equations*, Numer. Math., 35 (1980), pp. 241-255.

[12] K. STÜBEN AND U. TROTTENBERG, *Multigrid methods: fundamental algorithms, model problem analysis and applications*, in Multigrid Methods, W. Hackbusch and U. Trottenberg, eds., Springer-Verlag, Berlin, 1982, pp. 1-176.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 6, November 1987
© 1987 Society for Industrial and Applied Mathematics
004

COURANT-LIKE CONDITIONS LIMIT REASONABLE MESH REFINEMENT TO ORDER h^2 *

BLAIR SWARTZ†

Abstract. Under conditions (e.g., Courant-Friedrichs-Lewy conditions) on the size of all local time steps—namely, that they be proportional to the local spatial mesh size—a mesh refinement algorithm cannot be a "reasonable" algorithm if, near some space-time surface, the spatial mesh is refined to size $o(h^2)$, i.e., smaller than order h^2. Here h typifies the mesh size over most of the spatial region; and an algorithm is called "unreasonable" if, asymptotically, it spends all of its computational work exercising at the space-time surface. The limitation is even more stringent for problems with more than one spatial dimension unless the refinement occurs only normal to the surface. As the local time step constraints may be imposed for reasons of accuracy or efficiency as well as for stability, they may differ, on the coarse or fine meshes, from Courant-like conditions. In that case, the limitations on the order of reasonable refinement will generally change. But there is an absolute limitation which depends on the time step restriction away from the surface and on the number of spatial dimensions. One consequence of all this appears to be that $o(h^4)$ errors are unattainable in vast regions of space-time utilizing reasonable mesh refinement—at least when using more or less uniformly applied difference schemes, with Courant-like conditions, to approximate those solutions of hyperbolic systems whose first derivatives or values jump at some surface in space-time.

Key words. mesh refinement, static rezoning, evolutionary partial differential equations, discontinuities, adaptive grids

AMS(MOS) subject classification. 65M50

1. Introduction.

Mesh refinement algorithms are a currently popular technique for improving the efficiency of the approximate solution of evolutionary problems. This is so even for the problems which motivate this note—problems whose solutions are only piecewise smooth in space-time. For this paper's purposes, *mesh refinement* stands for an algorithm which advances the approximate solution on a given spatial mesh and then (at least every once in a while) locally refines or coarsens the old mesh depending upon the observed contemporary local roughness of the approximate solution on that old mesh. More specifically, we associate the term "mesh refinement" with adapting the spatial mesh to the current spatial character of the solution—what computational hydrodynamicists call "static rezoning." In this context, it is unfair to use information associated with the differential equation to anticipate the location of moving zones of roughness, or to relocate the entire mesh, for use at the end of the time-step; we associate the use of such techniques, respectively, with "shock fitting" or with "moving mesh methods" (neither of which we consider here).

Mesh refinement algorithms are sometimes complicated to analyze and to implement. It may be worthwhile, then, to attempt to contrast the computational efficiency of mesh refinement with the efficiencies associated with "shock fitting" or "moving mesh methods." People contemplating such contrasts may find this note useful. Those who consider using different difference methods in the refined and coarse mesh regions, who attempt to balance computational work with error size in those regions, and who contemplate both the interaction and the optimal balancing of these errors and this work may also find food for thought.

* Received by the editors August 12, 1981; accepted for publication (in revised form) November 13, 1986. This work was supported by the U.S. Department of Energy.

† T-7, Theoretical Division, University of California, Los Alamos National Laboratory, Los Alamos, New Mexico 87545.

2. The basic argument—one space dimension. We approximately solve an evolutionary system of partial differential equations with independent variables (x, t) in $[0, 1]\times[0, 1]$. It is presumed that the solution is only piecewise smooth. That is to say, the space-time curve in Fig. 1 separates $[0, 1]\times[0, 1]$ into two regions, and the solution can be extended to be smooth in the closure of each one.

A generic spatial increment h is associated with the difference scheme used where the solution is smooth. Furthermore, a condition analogous to a Courant condition for a hyperbolic equation—namely, that Δt_{local} is the order of Δx_{local} in size—is imposed on all time steps. Then the total number of space-time mesh squares associated with the smooth region, i.e., all those of order $h\times h$ in size, is of order $1/h^2$. Hence, for typical difference schemes, the computational work w_{smooth} associated with the smooth region is also of order $1/h^2$.

However, at the curve of discontinuity, we use a generic mesh size δ which is presumed $o(h)$ in size, i.e., δ goes to zero faster than h. The Courant-like condition then means that the curve is to be covered by space-time squares whose size is of order $\delta\times\delta$. It takes order $1/\delta$ of them to do so; thus, the work associated with the curve is of order $1/\delta$.

If we refine at each instant from order h to order δ in an exponentially fast way, then the work associated with this transition region is, in fact, bounded by some constant$/\delta$. So the work associated with the transition region is irrelevant to the result that, if $\delta = o(h^2)$, then we spend more work in connection with the curve of discontinuity than where the solution is smooth. Thus, under the above assumptions, the mesh refinement algorithm is *unreasonable* if $\delta = o(h^2)$.

3. General assumptions, analysis and results. Again, the context is the approximate solution of an evolutionary initial-boundary value problem—say, a hyperbolic system of partial differential equations—to be solved for $(x, t) := (x_1, x_2, \cdots, x_d, t)$ in the space-time domain $[0, 1]^{d+1}$. (However, the argument and conclusions to follow are basically geometrical, independent of the space-time type of the PDE.) One presumes there is some d-dimensional space-time surface

$$S = (s(t), t), \quad s(t) = (x_1(\xi_1, \cdots, \xi_{d-1}, t), \cdots, x_d(\xi_1, \cdots, \xi_{d-1}, t)),$$

$$DS \text{ piecewise-smooth},$$

at which the solution, or some of its derivatives, jump; but the solution can be extended to be smooth on the closure of each component of $[0, 1]^{d+1} - S$. Estimates will be made

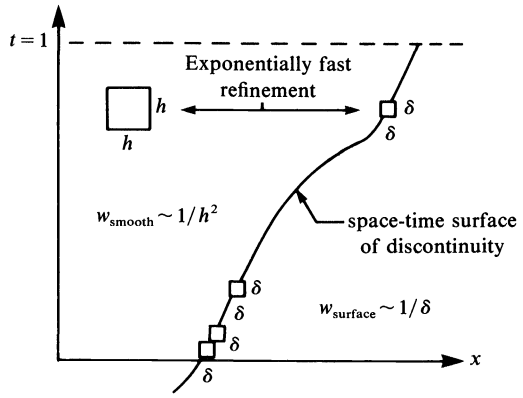


FIG. 1. *One space dimension.* (*Table* 1*'s first column.*)

separately for the computational work w_{smooth} associated with the smooth region and w_{rough} associated with the presence of the surface. If w_{rough} asymptotically exceeds w_{smooth}, i.e., if

$$w_{\text{smooth}} = o(w_{\text{rough}}), \tag{1}$$

then the mesh refinement algorithm is called *unreasonable.* This word is used because "shock fitting" appears offhand to lead to *reasonable* algorithms. "Moving mesh methods," on the other hand, may well allow smaller reasonable refinement than the mesh refinement algorithms defined in the Introduction.

Notation. Use is made of the usual order symbols, namely, $f = O(g)$ (f/g is bounded) and $f = o(g)$ (f/g goes to zero). Use is also made of a less standard notation—for the equivalence relation "f and g are of the same order" or "f is of order g," denoted by $f \asymp g$. Specifically,

$$f \asymp g \quad \text{means} \quad f = O(g) \quad \text{and} \quad g = O(f);$$

see, e.g., Hardy and Wright [1954, p. 7] or McLeod and Parter [1974, pp. 309–310]. It is a notion akin to equivalence of norms and is a more precise way of ignoring unknown constants than using the "big O."

At this point the reader could understand the general results via a slow study of Table 1 below in conjunction with Figs. 1–3, which go with the table's first three columns. But we proceed now with the general discussion.

The variable underlying all asymptotic estimates is the total computational work; it is assumed to increase without bound. With it all local spatial and time increments are assumed $o(1)$. Typical error estimates would be $o(1)$ as well, but some, like errors adjacent to discontinuities, might not be.

Work estimates are naturally based on dividing the total computational work into individual tasks accomplished during disjoint time intervals in the computer. The work, i.e., the computational time, associated with each task is presumed to be the same order as for any other task; and the total work is assumed to be of the order of the sum of the individual task times. In this note the task is the computation associated with a given space-time mesh cell. That the work for each such task is of the order of one arithmetic operation is a characteristic of explicit-in-time difference schemes involving a bounded number of neighboring cells. And, in a personal communication, A. Brandt has confided his confidence that this can also be true for suitably "multi-gridded" implementations of implicit schemes, even in many space dimensions.

We shall estimate the work associated with various space-time *mesh regions.* Such a region is actually a union of tasks rather than a geometric union of space-time mesh cells. Nevertheless, a careful count of all such tasks, whose corresponding cells often overlap geometrically in typical algorithms, would obscure the essential simplicity of the whole idea. So we count only the number of typical, disjoint cells necessary to fill a given region. These counts will be adequate if we assume that to each such cell corresponds a bounded number of tasks, that the volume associated with each such task is within two fixed percentages of that of the typical cell, and that we can so account for all tasks. This we assume.

The *smooth mesh region* M_{smooth} is the mesh region associated with a generic spatial mesh size h—a mesh size which one feels adequate to represent the solution where it is smooth. More precisely, it is the mesh region whose spatial mesh sizes are order h in all coordinates. If there are d space dimensions, then the number of cells in the smooth mesh region at any instant is order $1/h^d$. If one also assumes the

(hyperbolic) Courant-like condition

(2a) $$\Delta t_{\text{local}} \asymp \Delta x_{\text{local}}$$

holds in M_{smooth}, then the total work w_{smooth} associated with all of M_{smooth} (i.e. from $t=0$ to $t=1$) is

(2b) $$w_{\text{smooth}} \asymp 1/h^{d+1} \quad \text{(Courant-like condition).}$$

However, if the truncation errors in M_{smooth} are, in fact, all of the order $\Delta x_{\text{local}}^{p} + \Delta t_{\text{local}}^{q}$, then (for reasons of computational efficiency to compute to a given error level in M_{smooth}) one must balance the two terms using

(3a) $$\Delta t_{\text{smooth}} \asymp h^{\sigma} \quad (\sigma \text{ for smooth}, \sigma > 0),$$

with $\sigma := p/q$. This assumption (3a) implies instead of (2b) that

(3b) $$w_{\text{smooth}} \asymp 1/h^{\sigma+d} \quad \text{(general condition on time step).}$$

Now, the mesh is refined at the space-time discontinuity surface S to generic size $\delta = o(h)$. That is, at each instant $t=a$, the $t=a$ section of S, $s(a)$, is contained in the union of some space-time cells, each of whose width normal to $s(a)$ is order δ. The union of all cells having minimum size δ in *some* orthogonal spatial system is the *surface mesh region* M_{surface}. There are two cases concerning M_{surface} that we wish to consider.

The first case is that in which, at each instant of time, each and every one of the d spatial coordinates is refined to order δ at the surface. This case is illustrated in Fig. 2 for $d=2$ spatial dimensions under the Courant-like constraint $\Delta t_{\text{local}} \asymp \Delta x_{\text{local}}$ on all local time steps. In this case the number of spatial cells in M_{surface} at a given instant of time is *at least* order $1/\delta^{d-1}$. Let us also assume that

(4) at each instant t the number of cells of M_{surface} along any spatial normal to $s(t)$ is order one.

Then the number of spatial cells in M_{surface} at any instant is *exactly* of order $1/\delta^{d-1}$. Finally, let us assume that the local time steps in M_{surface} satisfy

(5a) $$\Delta t_{\text{surface}} \asymp \delta^{\rho} \quad (\rho \text{ for rough}).$$

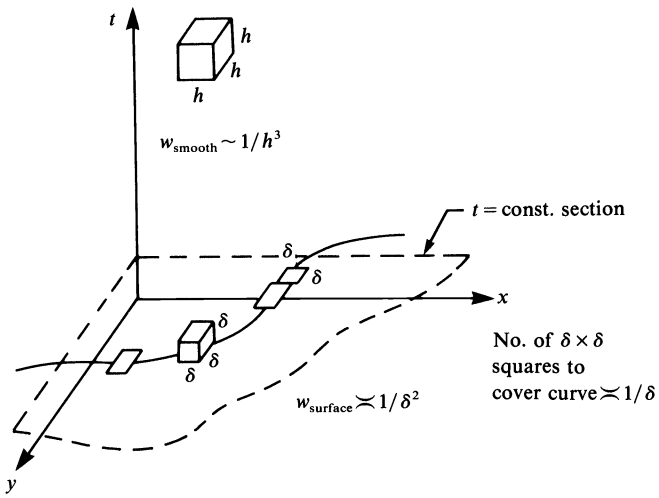


FIG. 2. *Two space dimensions: refinement in all spatial coordinates.*

There is a geometric restriction on ρ: for if $s(t)$ is almost always in motion, if the cells in M_{surface} touch in time but don't overlap, and if (4) holds; then one must have

$$\rho \geqq 1 \tag{5b}$$

since S is a subset of the geometric union of M_{surface}'s cells at each instant. Under these assumptions, then, the work associated with the surface itself, i.e., with M_{surface}, will be

$$w_{\text{surface}} \asymp 1/\delta^{\rho+d-1} \quad \text{(refinement in all spatial variables)}. \tag{6}$$

The second case concerning M_{surface} presumes refinement to order δ *only* normal (perpendicular) to the surface's intersection with each $t =$ constant plane, i.e., to $s(t)$. This case is illustrated in Fig. 3 for $d = 2$ spaced dimensions under a Courant-like condition on all local time steps. In this case, a cell can be much larger in the direction of the $d-1$ tangential variables (if any, i.e., if $d > 1$), but not too much larger if the surface's Gaussian curvature is nonzero. For then, locally, its distance away from its tangent plane is of order of the square of the distance from the point of tangency—at least it is in the $d-1$ coordinates associated with the $d-1$ curvature directions. Therefore, the normal height will become order δ exactly when that tangential distance is order $\delta^{1/2}$. At each instant, then, the number of cells of normal height δ required to cover a *curved* surface can be taken to be order $1/\delta^{(d-1)/2}$. This is also the order of the number of cells in M_{surface}, assuming (4) as before. With assumption (5) on the local time step at the surface, one concludes

$$w_{\text{surface}} \asymp 1/\delta^{\rho+(d-1)/2} \quad \text{(refinement only normal to the surface's time-sections)}. \tag{7}$$

It remains to account for the work associated with the space-time mesh cells in the *transition mesh*, $M_{\text{transition}}$, where the spatial mesh sizes are $o(h)$, yet not accounted for in M_{surface}. We assume the geometric region associated with $M_{\text{transition}}$ has spatial width $O(h)$. We shall also assume (and shall presently argue by example) that the associated work satisfies

$$w_{\text{transition}} = O(w_{\text{surface}}). \tag{8}$$

Then the work w_{rough} associated with the presence of the surface satisfies

$$w_{\text{rough}} (:= w_{\text{surface}} + w_{\text{transition}}) \asymp w_{\text{surface}}. \tag{9}$$

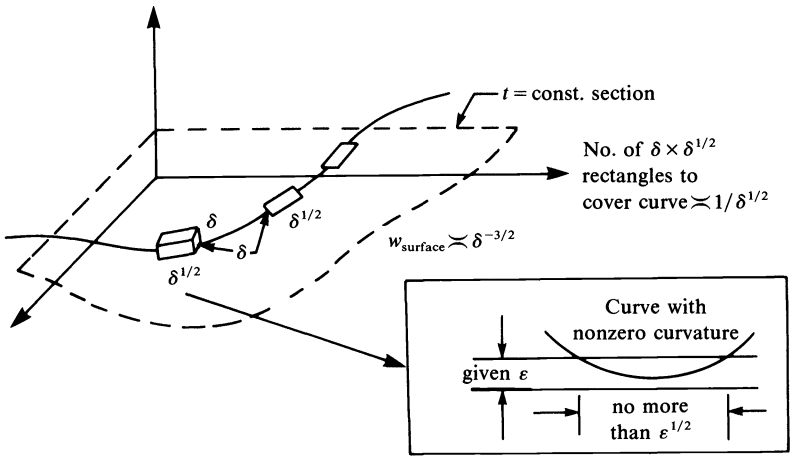


FIG. 3. *Two space dimensions: refinement only normal to surface's intersection with each* $t =$ const *plane.*

As a consequence we conclude a simplification of (1):

(10) the mesh refinement algorithm is unreasonable iff $w_{\text{smooth}} = o(w_{\text{surface}})$.

Table 1 displays results which now follow immediately from (3), (5)-(7) and (10).

In order that $\delta \asymp h^r$ (r for "reasonable") is $o(h)$, the parameter ρ must be bounded above in the two expressions limiting reasonable refinement in Table 1:

(11) $r = (d+\sigma)/(d+\rho-1), \quad 1 \leqq \rho < 1+\sigma$ (all coordinates refined),

(12a) $r = 2(d+\sigma)/(d+2\rho-1),$

$1 \leqq \rho < (d+2\sigma+1)/2$ (only normal refinement).

If our goal is reasonable refinement to as small a mesh as possible, i.e., if we seek large r, then, whether we refine all coordinates or only normally, we want to (a) take a large spatial increment relative to the time step in the smooth region (large σ in (3)), and/or (b) take a small spatial increment relative to the time step at the surface (small ρ in (5)). Consequently, (12a) implies the absolute limitation (taking $\rho = 1$)

(12b) $r_{\max} = 2(d+\sigma)/(d+1)$ (absolute restriction assuming (4))

on the exponent for reasonable refinement, given σ. Thus, for example, if the truncation errors in the smooth region were order $h^4 + \Delta t^2_{\text{smooth}}$, so that $\sigma = 2$ (for reasons of computational efficiency to compute to a given error level in the smooth region), then reasonable refinement for problems involving $d = 2$ space dimensions is limited to order $h^{8/3}$; and, to order $h^{5/2}$ for $d = 3$.

TABLE 1
Order of work and reasonable refinement restrictions.

	Courant-like constraints on all local time steps; i.e., $\Delta t_{\text{local}} \asymp \Delta x_{\text{local}}$, where $f \asymp g$ means $f = O(g)$ *and* $g = O(f)$					$\Delta t_{\text{smooth}} \asymp h^\sigma$, $\Delta t_{\text{surface}} \asymp \delta^\rho$, d spatial dimensions	
Spatial dimension	1	2_{all}	2_{normal}	d_{all}	d_{normal}	All coordinates refined	Refinement only along surface normal
$\text{work}_{\text{smooth}}$, spatial mesh h	$1/h^2$	$1/h^3$	$1/h^3$	$1/h^{d+1}$	$1/h^{d+1}$	$1/h^{\sigma+d}$	$1/h^{\sigma+d}$
$\text{work}_{\text{surface}}$, spatial mesh δ	$1/\delta$	$1/\delta^2$	$1/\delta^{3/2}$	$1/\delta^d$	$1/\delta^{(d+1)/2}$	$1/\delta^{\rho+d-1}$	$1/\delta^{\rho+(d-1)/2}$
Unreasonable if $\delta = o(h^r)$, $r =$	2	3/2	2	$1+1/d$	2	$(d+\sigma)/(d+\rho-1)$	$2(d+\sigma)/(d+2\rho-1)$

The absolute limitation (12b) can be relaxed, but not unboundedly relaxed, under conditions somewhat more general than (4). The conditions we have in mind are the following: the surface is still to be contained in the cells associated with normal grid size δ, but it is allowed to cross over a large number of them, both in a single time step and in a single tangential spatial increment. (It is far from clear whether stable and accurate schemes exist which can take advantage of such a space-time grid, but that is irrelevant to the present argument.) Thus, at any instant, the normal width of the refined region is presumed order δ^q, some positive $q \leqq 1$, so that the number of cells of M_{surface} along each spatial direction normal to $s(t)$ is order $1/\delta^{1-q}$. An additional constraint on the smallness of q is necessary so that the fine mesh not cover the entire

spatial region; this constraint is that

$$\delta^q = O(h). \tag{13}$$

If we now also assume only normal refinement, then the tangential refinement is to order only $\delta^{q/2}$; and the time step can also be larger, namely, $\Delta t_{\text{surface}} \asymp (\delta^q)^\rho$, $\rho \geqq 1$. Then $w_{\text{surface}} \asymp 1/\delta^{1-q+q\rho+q(d-1)/2}$. Hence the refinement will be reasonable, assuming (8) again along with (3b), and will also satisfy (13), if $\delta \asymp h^p$ with

(14)
$$1/q \leqq p \leqq r := (d+\sigma)/[1+q(\rho-1+(d-1)/2)], \quad \text{where } 1 \leqq \rho < (d+2\sigma+1)/2,$$
$$q_{\min} := 2/(d+2\sigma+3-2\rho) \leqq q \leqq 1.$$

It may be noted that, when $q=1$, (14) yields (12a) as it should. Furthermore, the right-hand side of (14) is a monotone, strictly decreasing function of q in (0, 1], unless both d and ρ are at their lower limits of one, when (14) and (12a) again coincide. Otherwise, one can attain finer reasonable refinement than allowed by (12a) by letting q decrease to $q_{\min}$. The resulting maximum corresponds to (12a) with this more general mesh:

$$r_M = (d+2\sigma+3-2\rho)/2, \qquad 1 \leqq \rho < (d+2\sigma+1)/2. \tag{15a}$$

It is given by $r_M = 1/q_{\min}$, so occurs with $\delta^{q_{\min}}$ of order h in size. If (15a) is maximized in turn over the allowed values of ρ, we obtain the analogue of (12b) at $\rho = 1$:

$$r_{\max} = (d+2\sigma+1)/2 \quad \text{(absolute restriction without (4))}. \tag{15b}$$

This changes the limitation in the example following (12b) only from order $h^{8/3}$ there to $h^{7/2}$ for $d=2$ space dimensions; and from order $h^{5/2}$ to order h^4 for $d=3$.

We are at the border between mesh refinement (as defined in the Introduction) and "shock fitting" or "moving mesh methods." Suppose, above (13), that we felt it possible to compute with only a *bounded* number of the $1/\delta^{1-q}$ mesh cells along each normal to $s(t)$. Then the denominator in the right-hand side of (14) would be $q[\rho + (d-1)/2]$, allowing unlimited reasonable refinement as q could now approach zero. To our mind, however, any implementation involving such a mesh will cross the border just mentioned; we consider it no further.

Finally, as promised near (8), we demonstrate how the work $w_{\text{transition}}$ in the transition region can be $O(w_{\text{surface}})$ if we refine exponentially fast. Suppose, in fact, we refine by halving, i.e., coarsen by doubling. For conceptual purposes, suppose the space-time surface S is a hyperplane whose normal is the x_1-axis. The fine mesh enclosing S is of thickness δ; and it has $1/\delta^\alpha$ nonoverlapping space-time cells in it where, in the two cases leading to (6) or to (7), $\alpha > 0$. In the next layer towards $x_1 > 0$, we double δ in the normal direction and adjust the tangential and time steps accordingly. The number of cells in this, the first layer in the transition region, is $1/(2\delta)^\alpha$ with associated work $c/(2\delta)^\alpha = w_{\text{surface}}/2^\alpha$. The second layer in the transition region has, similarly, a normal thickness of $2^2\delta$ and associated work $w_{\text{surface}}/(2^\alpha)^2$. One may stop this process at the kth layer, where both $h = O(2^k\delta)$ and $h^\sigma = O(2^k\delta)^\rho$. But, whatever value of k is sufficient, one has the bound

$$w_{\text{transition}} \leqq (\omega + \omega^2 + \omega^3 + \cdots) w_{\text{surface}}, \qquad \omega := 1/2^\alpha < 1.$$

Thus, $w_{\text{transition}} = O(w_{\text{surface}})$, as was to be shown.

It should be noted, however, that a modicum of care may be necessary in choosing the manner in which one switches from the surface's time step constraint, $\Delta t_{\text{local}} \asymp \Delta x_{\text{local}}^\rho$, to the smooth region's constraint, $\Delta t_{\text{local}} \asymp \Delta x_{\text{local}}^\sigma$. For example: it may well be that $\sigma > \rho$; and if one chooses to switch from ρ to σ within a *fixed* number of layers of the surface in this case, then $w_{\text{transition}}$ will *not* be $O(w_{\text{surface}})$.

In such a case as this last one, i.e., when it is false that

$$w_{\text{rough}}(:= w_{\text{surface}} + w_{\text{transition}}) = O(w_{\text{surface}});$$

it is still true that the numbers r in Table 1 *limit* reasonable mesh refinement. For their validity as restrictions is based only on the fact that the exhibited estimates for w_{surface} are *lower* bounds for w_{rough}. But, if $w_{\text{transition}}$ is *not* $O(w_{\text{surface}})$, then refinement of δ to order h^r may *also* be unreasonable, i.e., the numbers r in Table 1 may not be sharp.

4. Probable unattainability of $o(h^4)$ errors, using reasonable mesh refinement and more or less uniformly applied difference schemes under Courant-like conditions, for hyperbolic systems whose solutions are only piecewise smooth. The results of the previous sections are basically geometric facts concerning the approximation of surfaces. In this section, we apply them.

A certain body of knowledge exists concerning the approximation of hyperbolic problems using uniformly applied difference schemes on a uniform spatial grid. Some of this knowledge concerns problems whose solutions are only piecewise smooth. We review some of these results and presume they extrapolate to varying spatial meshes. Then we apply the previous restrictions associated with reasonable refinement to deduce limitations on the order of accuracy one can possibly attain using such refinement. These limitations will restrict accuracy in large subsets of space-time, subsets where the solution itself is, in fact, smooth.

The work of Hedstrom [1968; see, e.g., Thm. 5.3] (for $u_t = u_x$ with step function initial data; high-order, not necessarily centered, dissipative schemes on a spatial mesh of uniform size δ, fully discrete with Courant condition) tempts one to hope that bad errors near discontinuities need not necessarily affect high-order errors away from them. But Majda and Osher [1977] thoroughly analyzed a time-continuous example (otherwise the difference setup was similar), with either a step function or $|x|$-type initial data for a constant coefficient hyperbolic system much like the system below. And they proved that $o(\delta^2)$ errors were then unattainable in almost the entire "V" between the two characteristics emanating from the initial point of roughness ($o(\delta)$ errors, in fact, for the step function unless it was suitably centered on the uniform mesh) (for a simpler analysis, see Brown [1982, pp. 13–16]). This makes it easy to guess that $o(h^4)$ errors are unlikely to be attained when approximately solving general hyperbolic systems via reasonable mesh refinement—at least when using more or less uniformly applied difference schemes under a Courant condition—since, then, δ cannot be $o(h^2)$.

And, indeed, this would be unsurprising in view of the behavior of the solutions to the hyperbolic systems themselves. For consider how the right-traveling solution of the equation

$$u_t + u_x = 0, \qquad u(x, 0) = \begin{cases} 0, & x < 0, \\ 1, & 0 \leqq x \leqq \varepsilon, \\ 0, & x > \varepsilon \end{cases}$$

acts as a source in the coupled equation

$$v_t - v_x = u, \qquad v(x, 0) \equiv 0.$$

Integrating it up the characteristics for the latter, we see that $v \equiv \varepsilon/2$ between the two characteristics emanating from $(\varepsilon/2, \varepsilon/2)$. That is to say, a perturbation in the first equation's initial data is first propagated undamped up its own characteristics and

then spread by integration over all characteristics of the second equation. It is not damped in any other way.

The Majda-Osher δ^2 barrier (above) does *not* preclude $o(h^4)$ errors in the smooth region if the Courant-like condition of comparable spatial and temporal truncation errors there (namely $\sigma = 1$) is dropped. For example, an $O(h^6) + O(\Delta t_{\text{smooth}}^2)$ accurate scheme (with concomitant $\sigma = 3$) permits reasonable "normal" refinement to at least as small as $\delta \asymp h^3$ for problems with $d \leqq 3$ space dimensions, assuming at the surface the Courant-like condition $\Delta t_{\text{surface}} \asymp \delta$(i.e. $\rho = 1$). (To see this, use (12a).)

Finally, a reminder may be in order concerning the bad effect which *nondissipative* schemes can have where a piecewise smooth solution is, in fact, smooth. For, Hedstrom's bound (above) concerning $u_t = u_x$ (shown sharp by Serdyukova [1971] and extended to general piecewise smooth initial data by Brenner and Thomée [1971] using a technique of Apelkrans [1968]—who also considered a single equation with variable coefficients) concerned only *dissipative* schemes. And in that situation, the error in approximating a traveling discontinuity drops off exponentially fast (in the number of mesh points spatially separating one from the relevant characteristic), so that high-order truncation then has a chance to dominate in smooth regions. But Hedstrom [1968, Remark, p. 399] also considered other schemes for $u_t = u_x$, schemes whose amplification factor had magnitude *exactly* one. And in his bounds for that case, also made sharp by Serdyukova [1971], the error drops only with the reciprocal of the square root of that number of mesh points. It then swamps other truncation errors with its order $\delta^{1/2}$ effect on convergence. These results surely explain, too, the early analytic result of Trulio [1964, pp. 19-34] (the first-order system for the second-order wave equation, time continuous, simple centered spatial differences for stacked unknowns, a piecewise linear tent with fixed support as initial data, order $\delta^{3/2}$ convergence at advanced times at the initial position of the tent's peak) since his difference scheme diagonalizes simultaneously with the differential equation. And they almost surely explain (and the analysis there would explain) the order $\delta^{1/2}$ effect on remote convergence which is discernible in Chin's [1975] analysis (same equations and differencing, except for staggered unknowns and an initial-*boundary* value problem involving a traveling step function). Thus, the seeming incompatibility of these last two results with the slightly higher-order errors which Majda and Osher [1977] actually got—in spite of the additional terms in their differential equations—appears as a probable contrast, to their detriment, of nondissipative schemes with dissipative ones. Furthermore, we can easily guess, if we now use reasonable mesh refinement, that at best order h convergence to piecewise smooth solutions is possible when more or less uniformly applied *nondissipative* schemes are used under Courant-like conditions.

One cannot deny the need for mesh refinement when approximating solutions which are smooth but almost discontinuous. This is especially so when adequate resolution at a given time is clearly necessary for future accuracy of the solution; for example, in flame propagation problems with sources and coefficients strongly dependent on temperature. Indeed, the proposed definition of "reasonable refinement" is, in fact, unreasonable in such contexts. But we are forced to question the efficiency of mesh refinement for models whose merely *piecewise* smooth solutions adequately approximate the physical situation.

Acknowledgments. The author acknowledges fruitful conversations with M. Hyman, H.-O. Kreiss and B. Wendroff, and attributes any accuracies in the hyperbolic history above to G. Hedstrom's kind guidance. The concepts presented in this paper were first discussed publicly in Swartz [1981].

REFERENCES

M. Y. T. APELKRANS [1968], *On difference schemes for hyperbolic equations with discontinuous initial values*, Math. Comp., 22, pp. 525–539.

A. BRANDT [1981], personal communication.

P. BRENNER AND V. THOMÉE [1971], *Estimates near discontinuities for some difference schemes*, Math. Scand., 28, pp. 329–340.

D. BROWN [1982], *Solution adaptive mesh procedures for the numerical solution of singular perturbation problems*, Ph.D. thesis, California Inst. of Technology, Pasadena, CA.

R. C. Y. CHIN [1975], *Dispersion and Gibb's phenomenon associated with difference approximations to initial boundary-value problems for hyperbolic equations*, J. Comput. Phys., 18, pp. 233–247.

G. H. HARDY AND F. M. WRIGHT [1954], *An Introduction to the Theory of Numbers*, 3rd edition, Clarendon Press, Oxford, England.

G. W. HEDSTROM [1968], *The rate of convergence of some difference schemes*, SIAM J. Numer. Anal., 5, pp. 363–406.

A. MAJDA AND S. OSHER [1977], *Propagation of error into regions of smoothness for accurate difference approximations to hyperbolic equations*, Comm. Pure Appl. Math., 30, pp. 671–705.

J. B. MCLEOD AND S. V. PARTER [1974], *On the flow between two counter-rotating infinite plane discs*, Arch. Rational Mech. Anal., 54, pp. 301–327.

S. I. SERDYUKOVA [1971], *Oscillations which occur in the numerical computation of the discontinuous solutions of differential equations*, Zh. Vychisl. Mat. i Mat. Fiz., 11, pp. 411–424 (translated as U.S.S.R. Computational Math. and Math. Phys., 11, pp. 140–154).

B. SWARTZ [1981], *Courant-like conditions limit reasonable mesh refinement to order h^2*, U.S. Dept. of Energy Numerical Analysis Special Interest Group, Berkeley, CA, April 29–May 1 (abstract LA-UR-81-1285; report LA-UR-81-2037).

JOHN G. TRULIO [1964], *Studies of finite difference techniques for continuum mechanics*, Air Force Weapons Laboratory, report WL TDR-64-72 (report AD-613-315, National Technical Information Service, Springfield, VA), 390 pp.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 6, November 1987
© 1987 Society for Industrial and Applied Mathematics

OPTIMAL REGULARIZED RECONSTRUCTION IN COMPUTERIZED TOMOGRAPHY*

DIDIER A. GIRARD†

Abstract. We consider regularized least squares solutions for the picture reconstruction problem in computerized tomography. The choice of the regularization parameter ρ is crucial, but the known methods (like cross-validation) are expensive. This paper shows that the usual rotationally invariant geometry of tomographs greatly reduces their cost.

In the case of the $\mathbf{L}_2$-norm taken as the irregularity functional, this invariance was previously used to efficiently compute a solution f_ρ with an a priori fixed value of ρ. We present an algorithm which computes first an optimal value ρ^*, then f_{ρ^*} with only twice (four times in the Poisson noise situation) the number of operations and the same memory space as the computation of one solution with fixed ρ.

In the case of any rotationally invariant norm and with a suitable discretization of the picture, we describe a similar algorithm which, after a first step which is roughly the standard back-projection of the data, has the same complexity as the previous one.

Numerical applications show the performance of these adaptive reconstructions.

Key words. integral equations, optimal regularization, spline, cross-validation, image reconstruction, time-of-flight emission tomography

AMS(MOS) subject classifications. 65R10, 92A07

1. Introduction. The picture reconstruction problem in classical computed tomography consists in estimating a real valued function f^0 defined on a cross-section of some tissue, from m experimental data y_i, $i=1,\cdots,m$, which are approximate values of the integrals $y_i^0=\int_{Li} f^0$ on m given thin strips or lines L_i of the section.

For example, in X-ray tomography each measurement is the attenuation factor of the intensity of a thin X-ray beam which crosses the tissue along a given line, and the logarithm of this factor gives an estimate of the integral of the density along this line.

In positron emission tomography, a ring of detectors is placed around a positron emitting object in order to study the activity distribution f^0 of these emissions in the ring section (for a description of this technique, see [15]). Each positron emission produces, at the same point, two photons emitted in opposite directions. So if a pair, numbered i, of detectors receives two photons at the same time, this means that one positron emission has occurred on the strip L_i between these detectors. The measurement y_i is the number of such observed coincidences and is modelled as a Poisson random variable with mean, y_i^0, being:

$$y_i^0=\frac{1}{\pi}T(d_i/l_i)\iint_{Li} f^0(x,y)\,dx\,dy,$$

where d_i is the width, l_i the length of the strip L_i, and T the duration of the experiment. In addition, the time-of-flight information [1] provides a classification of these y_i coincidences by an approximate measurement of their origin along L_i. The data thus approximate more local integrals.

In this paper, we use the following general formulation for these imaging systems:

$$y_i^0=(\varphi_{t_i},f^0),\qquad i=1,\cdots,m \tag{1.1}$$

* Received by the editors November 14, 1985; accepted for publication (in revised form) September 29, 1986.

† Laboratoire d'Electronique et de Technologie de l 'Informatique, MCTE, C. E. N. Grenoble, France. Current address, Centre National de la Recherche Scientifique, Laboratoire TIM3, Institut de Mathématiques Appliquées de Grenoble, B.P.68, 38402 St. Martin d'Hères, France.

where:

—the support of f^0 is the unit disc $\mathbf{D} \subset \mathbb{R}^2$;

—(,) is the inner product in $\mathbf{L}_2$ (the space of the square integrable functions on $\mathbf{D}$): $(f, g) = \iint_{\mathbf{D}} f(x, y) g(x, y)\, dx\, dy$;

—for each parameter t, φ_t is a known function of $\mathbf{L}_2$ specific to the tomograph;

—$t_i \in [0, \pi] \times [-1, 1]$ in classical tomography ($[0, \pi] \times [-1, 1]^2$ in time-of-flight tomography) is the known coordinate of the measurement (φ_{t_i}, f).

The reconstruction algorithms presently in use in tomography are discretizations of explicit expressions giving f from the function $g : t \to (\varphi_t, f)$ (in the first two examples g is essentially the Radon transform of f) and are implemented with Fourier transformations. These analytical methods are really efficient as soon as one has enough good data (at least 10^4) in a uniform sampling, and when the discretization is done carefully (see e.g. [3], [11], [12]) with an attenuation of the high frequencies which must be even stronger when the data errors are bigger. The need of such smoothing is evident when considering the differential operator occurring in the inverse of the Radon transform. But the essential limitation of these algorithms is that the amount of smoothing is predetermined, whereas the optimal amount of smoothing really depends on the error level and the shape of f^0.

This paper describes new reconstruction algorithms which produce such an adaptation automatically derived from the data and which may be refined by using an estimate of their variance at nearly the same cost as the analytical methods.

Our approach proceeds by optimal regularization with discrete data (see [16]): denoting $s(f)$ the infidelity of $f \in \mathbf{L}_2$ from the data $y_1, y_2, \cdots, y_m$:

$$s(f) = \sum_{i=1,\cdots,m} [(\varphi_{t_i}, f) - y_i]^2$$

(this sum may be weighted when the errors are not of the same order); for any parameter δ, we can associate the set E_δ of functions which satisfy the data at the level δ:

$$E_\delta = \left\{ f \in \mathbf{L}_2 \Big/ \frac{1}{m} s(f) \leqq \delta^2 \right\}.$$

It is well known (see e.g. [13]) that E_δ, even when restricted to a suspace $\mathbf{F}$ of $\mathbf{L}_2$ which has to model well any available solution, still contains wildly oscillatory functions as long as it is nonempty (the data are very probably nonconsistent when the φ_{t_i} are nonindependent and so E_δ is empty when δ is small enough). Thus we need to use a measure of the irregularity of $f \in \mathbf{F}$, here defined by a norm $\| \cdot \|_{\mathbf{F}}$ giving a Hilbert space structure to $\mathbf{F}$, in order to define the least irregular function of $E_\delta \cap \mathbf{F}$ as the estimator with level δ.

In fact, this family of estimators with different levels is described by f_ρ, the function of $\mathbf{F}$ which minimizes $J_\rho(f) = s(f) + \rho \|f\|_{\mathbf{F}}^2$ in $\mathbf{F}$, when the real ρ varies in $]0, +\infty[$. It is this parameter which is explicitly used since the computation of f_ρ is provided by a linear system.

The rotationally invariant geometry of classical tomographs has already been exploited for an efficient implementation of this method when $\mathbf{F} = \mathbf{L}_2$. Natterer [9] has shown the capability of such a reconstruction compared with the analytical method when each one uses good parameters, but the proposed algorithm has a competitive cost only when the parameter ρ is a priori fixed.

Since this regularization parameter ρ controls the amount of smoothing, we need a good value of ρ which depends not only on the probabilistic properties of the data

error but also the data themselves. We study here how to compute such a good value. Three different methods of choosing ρ from the data (the discrepancy method, a statistical method and the cross-validation method) are recalled in § 2.

In § 3 we describe an algorithm for the case $\mathbf{F} = \mathbf{L}_2$, which computes the parameter ρ^* given by any one of these methods and f_{ρ^*} for a total cost (number of operations and memory space) equivalent to that with fixed ρ. The Poisson noise situation is also described.

Then if $\mathbf{F}$ is taken as a finite dimension subspace of $\mathbf{L}_2$, we show in § 4 how to compute such an optimal regularized solution equally efficiently, even when $\mathbf{F}$ has a large finite dimension, as soon as a spanning functions family of $\mathbf{F}$ and $\|\cdot\|_{\mathbf{F}}$ have a certain rotational invariance (we use the results of [6]). Finally, in § 5 we present numerical applications in classical and time-of-flight tomography (system of around 1,000,000 equations in 10,000 unknowns) which show the performances of these adaptive reconstructions.

2. Three methods of choosing the parameter of regularization. We assume that the space $\mathbf{F}$ and the norm $\|\cdot\|_{\mathbf{F}}$ defining the irregularity are chosen. It is a classical result of Hilbert space theory (see e.g. [8, Chap. 4]) that, since each $(\varphi_t, \cdot)$ is a bounded linear functional on the Hilbert space $\mathbf{F}$, for any $\rho > 0$ there exists only one minimizer f_ρ of $J_\rho(f) = s(f) + \rho\|f\|_{\mathbf{F}}^2$ in $\mathbf{F}$. And there is a $m \times m$ computable matrix $A(\rho)$ such that:

$$((\varphi_{t_1}, f_\rho), (\varphi_{t_2}, f_\rho), \cdots, (\varphi_{t_m}, f_\rho))^t = A(\rho)\mathbf{y},$$

where $y = (y_1, y_2, \cdots, y_m)^t$ is the data vector.

In the first two methods of choosing ρ, we have to assume some statistical knowledge of the data errors.

(1) A usual assumption is to model the errors as a white noise with known variance σ^2: the vector $\mathbf{y}$ is a realization of a random vector $Y = (Y_1, Y_2, \cdots, Y_m)^t$ with:

$$E(Y_i) = y_i^0, \quad i, j = 1, 2, \cdots, m,$$

$$E((Y_i - y_i^0)(Y_j - y_j^0)) = \begin{cases} \sigma^2, & i = j, \\ 0, & i \neq j, \end{cases}$$

where E denotes the expectation value.

A natural idea is then to say that a function $f \in \mathbf{F}$ is sufficiently adjusted to the data if:

$$\frac{1}{m} s(f) = \frac{1}{m} \sum_{i=1,\cdots,m} [(\varphi_{t_i}, f) - y_i]^2 \leqq \sigma^2.$$

The least irregular of these functions satisfying this discrepancy condition is generally obtained by taking ρ_s such that [10]:

$$\frac{1}{m} s(f_{\rho_s}) = \sigma^2,$$

i.e.,

$$\frac{1}{m} \|A(\rho_s)\mathbf{y} - \mathbf{y}\|^2 = \sigma^2 \tag{2.1}$$

where $\|\cdot\|$ is the natural Euclidean norm in $\mathbb{R}^m$.

If there is no such ρ_s, this means that either the space $\mathbf{F}$ is too poor or σ is too small.

(2) Let us now define the optimum value of ρ following Wahba [16]: ρ^0 which minimizes $r(\rho)$, the mean square error of prediction of the exact data:

$$r(\rho)=\frac{1}{m}\sum_{i=1,\cdots,m}[(\varphi_{t_i},f_\rho)-(\varphi_{t_i},f^0)]^2=\frac{1}{m}\|A(\rho)\mathbf{y}-\mathbf{y}^0\|^2 \tag{2.2}$$

where $\mathbf{y}^0=(y_1^0,y_2^0,\cdots,y_m^0)^t$ is the vector (unknown) of exact data. And let ρ_0 be the optimum value in average: ρ_0 minimizes the expected value $E(R(\rho))$ of the random variable:

$$R(\rho)=\frac{1}{m}\|A(\rho)Y-\mathbf{y}^0\|^2.$$

Although neither $r(\rho)$ nor its mean $E(R(\rho))$ can be known, we can show (e.g. [4]), that in the case of white noise with known variance σ^2, $\hat{r}(\rho)$ defined by:

$$\hat{r}(\rho)=\frac{1}{m}\|A(\rho)\mathbf{y}-\mathbf{y}\|^2+\frac{\sigma^2}{m}\operatorname{trace}(A(\rho)-I)^2+\frac{\sigma^2}{m}\operatorname{trace}A(\rho)^2 \tag{2.3}$$

where I is the $m\times m$ matrix identity, is an unbiased estimate of $E(R(\rho))$, theoretically computable. So the second method is to take the minimizer ρ_r of $\hat{r}(\rho)$ as an approximation of ρ^0.

In the case where y_i is modelled as a Poisson random variable of mean y_i^0, we still have another unbiased estimate $\hat{r}(\rho)$ of $E(R(\rho))$ (e.g. [5]):

$$\hat{r}(\rho)=\frac{1}{m}\|A(\rho)\mathbf{y}-\mathbf{y}\|^2+\frac{1}{m}\sum y_i-\frac{2}{m}\operatorname{trace}(\operatorname{Diag}(y_1,\cdots,y_m)[I-A(\rho)]). \tag{2.4}$$

(3) The third method needs no such knowledge on the data error variance. It consists of choosing ρ_v as the minimizer of $v(\rho)$:

$$v(\rho)=\frac{1/m\|(I-A(\rho))\mathbf{y}\|^2}{[1/m\operatorname{trace}(I-A(\rho))]^2}. \tag{2.5}$$

Wahba has shown that, with chiefly the assumption of a white noise, ρ_v is a good estimate of ρ_0, even better when m is large ([16] and [4]).

This method, called the cross-validation, originates from a simple idea [14]: we omit a datum for a moment and we study the validity of different ρ by measuring how the corresponding estimate can predict this datum. We choose the ρ which gives the best prediction in average for all the possible omissions.

3. Minimal energy reconstruction in classical tomography. We consider here the case $\mathbf{F}=\mathbf{L}_2$ and $\|f\|^2=\int_{\mathbf{D}}f^2$ (minimal energy reconstruction). The tomographs in which we are interested here are the first two described in the Introduction: frequently the size m of the data is greater than 10^4. Thus the straightforward search and processing of the matrix $A(\rho)$ is unrealistic. But we shall see that the usual rotational symmetries of these systems, already used [9] to compute f_ρ, minimizer of $\sum[\int_{Li}f-y_i]^2+\rho\int_{\mathbf{D}}f^2$, with a competitive cost when ρ is fixed, greatly simplifies also the computation of a good value of ρ.

To fix ideas, let us first describe the usual parallel geometry. Let $\mathcal{R}$ be the following operator:

$$\mathcal{R}:\mathbf{L}_2(\mathbf{D})\to\mathbf{L}_2([0,\pi]\times[-1,1]),$$

$$f\mapsto\mathcal{R}f:(\theta,r)\mapsto\int_{\mathbf{D}}\varphi_{\theta,r}(x,y)f(x,y)\,dx\,dy$$

where $\varphi_{\theta,r}$ is the characteristic function of the strip $L_{\theta,r}$ of width d, orthogonal to $(\cos\theta, \sin\theta)$ at the distance r from the origin, i.e.:

$$L_{\theta,r} = \{(x, y) \in \mathbf{D} : r - d/2 < x\cos\theta + y\sin\theta < r + d/2\},$$

$$\varphi_{\theta,r}(x, y) = \begin{cases} 1 & \text{if } (x, y) \in L_{\theta,r}, \\ 0 & \text{else.} \end{cases}$$

So $\mathcal{R}f$ is the locally averaged Radon transform of f. Now, in classical tomography with beams of constant width d, the ith measurement y_i is a noisy value of $\mathcal{R}f^0(t) = (\varphi_t, f^0)$ at a given point $t = t_i = (\theta_i, r_i)$.

If we set φ_i the function φ_{t_i} and R_m the operator:

$$R_m : f \in \mathbf{L}_2(\mathbf{D}) \mapsto R_m(f) = ((\varphi_1, f), (\varphi_2, f), \cdots, (\varphi_m, f))^t,$$

a classical result of the Hilbert space theory (e.g. [8, Chap. 4]) says that f_ρ is given by:

$$f_\rho = R_m^* \boldsymbol{\lambda}_\rho, \tag{3.1}$$

$$\boldsymbol{\lambda}_\rho = (R_m R_m^* + \rho I)^{-1} \mathbf{y} \tag{3.2}$$

where R_m^* is the adjoint of R_m:

$$R_m^* : \mathbb{R}^m \to \mathbf{L}_2(\mathbf{D}),$$

$$\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \cdots, \lambda_m)^t \mapsto R_m^* \boldsymbol{\lambda} = \sum_{i=1,\cdots,m} \lambda_i \varphi_i.$$

So the computation of f_ρ needs the solution of the linear system (3.2) where $B = R_m R_m^*$ is the $m \times m$ matrix with i, jth entry:

$$(B)_{i,j} = (\varphi_i, \varphi_j).$$

Note that this entry is simply the area of the intersection $L_{\theta_i,r_i} \cap L_{\theta_j,r_j}$ and that (3.1) is essentially a generalized form of the (well known in tomography) back projection occurring in the analytical method: the value of f_ρ at a point (x, y) of $\mathbf{D}$ is the sum of the components $\lambda_{\rho,i}$ of $\boldsymbol{\lambda}_\rho$ for i such that (x, y) belongs to the strip L_{θ_i,r_i}.

Now let us see how trace $(A(\rho) - I)$, $\|A(\rho)\mathbf{y} - \mathbf{y}\|$ and f_ρ can be easily computed in the frequent case of rotational invariance satisfying:

$$m = K \cdot P \quad \text{and} \quad \begin{cases} t_i = (\theta_k, r_p), \\ \theta_k = (k-1)2\pi/K, \end{cases}$$

$$k = 1, 2, \cdots, K, \quad p = 1, 2, \cdots, P, \quad i = (k-1)P + p.$$

Let $y_{k,p}$ denote the measurement at the point (θ_k, r_p) and $\varphi_{k,p}$ denote the function φ_{θ_k,r_p}. Let

$$\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_K)^t \quad \text{where } \mathbf{y}_k = (y_{k,1}, y_{k,2}, \cdots, y_{k,P})^t,$$

$$\boldsymbol{\lambda}_\rho = (\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \cdots, \boldsymbol{\lambda}_K)^t \quad \text{where } \boldsymbol{\lambda}_k = (\lambda_{k,1}, \lambda_{k,2}, \cdots, \lambda_{k,P})^t$$

(we omit ρ). Then the matrix B relating $\boldsymbol{\lambda}_\rho$ to $\mathbf{y}$ in (3.2), partitioned as follows:

$$B = R_m R_m^* = \begin{pmatrix} B_{1,1} & B_{1,2} & \cdots & B_{1,K} \\ B_{2,1} & \cdot & \cdots & B_{2,K} \\ \vdots & & & \vdots \\ B_{K,1} & \cdot & \cdots & B_{K,K} \end{pmatrix}$$

is block-circulant. Indeed

$$(B_{k,l})_{p,q} = (\varphi_{k,p}, \varphi_{l,q})$$

is the surface of the intersection $L_{\theta_k,r_p} \cap L_{\theta_l,r_q}$ and is invariant if one adds modulo K the same integer to k and l.

Thus as is well known, we can compute a block-diagonalization of B:

Denoting:

W the discrete Fourier transform matrix of order K: $W_{k,l} = \exp(-2\pi(k-1)(l-1)i/K)$,

Ψ the $K \cdot P \times K \cdot P$ permutation matrix: $\Psi \mathbf{y} = (y_{1,1}, y_{2,1}, \cdots, y_{K,1}, \cdots, y_{1,P}, \cdots, y_{K,P})^t$,

$$W_v = \Psi^t \begin{pmatrix} W & & & 0 \\ & W & & \\ & & \ddots & \\ & & & W \\ 0 & & & \end{pmatrix} \Psi \tag{3.3}$$

the matrix of the Fourier transforms of the P columns of the array $y_{k,p}$.

Then we have:

$$B = \frac{1}{K} \bar{W}_v \begin{pmatrix} M_1 & & & \\ & M_2 & & \\ & & \ddots & \\ & & & M_K \end{pmatrix} W_v \tag{3.4}$$

where the $P \times P$ matrices M_k are given by

$$\begin{pmatrix} M_1 \\ M_2 \\ \vdots \\ M_K \end{pmatrix} = W_v \begin{pmatrix} B_{1,1} \\ B_{1,2} \\ \vdots \\ B_{1,K} \end{pmatrix}. \tag{3.5}$$

When these matrices M_k are precomputed (by P^2 discrete Fourier transforms (D.F.T.) of order K) and stored for a fixed geometry, the computation of $\boldsymbol{\lambda}_\rho$ is reduced to $2P$ D.F.T.'s of order K and K linear systems of order P since (3.2) may now be written as:

$$\boldsymbol{\lambda}_\rho = \frac{1}{K} \bar{W}_v \begin{pmatrix} (M_1 + \rho I)^{-1} & & & \\ & (M_2 + \rho I)^{-1} & & \\ & & \ddots & \\ & & & (M_K + \rho I)^{-1} \end{pmatrix} W_v \mathbf{y}. \tag{3.6}$$

Furthermore we also have:

$$A(\rho)\mathbf{y} = B\boldsymbol{\lambda}_\rho = \frac{1}{K} \bar{W}_v \begin{pmatrix} M_1(M_1 + \rho I)^{-1} & & & \\ & M_2(M_2 + \rho I)^{-1} & & \\ & & \ddots & \\ & & & M_K(M_K + \rho I)^{-1} \end{pmatrix} W_v \mathbf{y}. \tag{3.7}$$

Now it is easy to verify that the $B_{1,k}$ are symmetric thus the M_k are symmetric; and

since B is also symmetric the matrices M_k are real. Note also that $M_{2+k} = M_{K-k}$ for $k = 0, \cdots, K-2$. Thus we can diagonalize each M_k by a unitary transform:

$$M_k = V_k D_k V_k^t \quad \text{where } D_k \text{ is the diagonal matrix Diag } (d_{k,1}, d_{k,2}, \cdots, d_{k,P}).$$

So we get with (3.7) the following diagonalization of $I - A(\rho)$ (and its trace):

$$I - A(\rho) = \frac{1}{K} \bar{W}_v \begin{pmatrix} V_1 & & & \\ & V_2 & & \\ & & \ddots & \\ & & & V_K \end{pmatrix} \begin{pmatrix} I - D_1(D_1+\rho I)^{-1} & & & \\ & I - D_2(D_2+\rho I)^{-1} & & \\ & & \ddots & \\ & & & I - D_K(D_K+\rho I)^{-1} \end{pmatrix} \times \begin{pmatrix} V_1^t & & & \\ & V_2^t & & \\ & & \ddots & \\ & & & V_K^t \end{pmatrix} W_v. \tag{3.8}$$

As $1/\sqrt{K}\, W_v$ does not change the Euclidean norm in $\mathbb{C}^n$, like

$$\begin{pmatrix} V_1 & & & \\ & V_2 & & \\ & & \ddots & \\ & & & V_K \end{pmatrix},$$

we get:

$$\|(I - A(\rho))\mathbf{y}\|^2 = \frac{1}{K} \sum_{\substack{k=1,\cdots,K \\ p=1,\cdots,p}} (1 - d_{k,p}/(d_{k,p}+\rho))^2 |z_{k,p}|^2 \tag{3.9}$$

where $\mathbf{z} = (\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_K)^t$ with $\mathbf{z}_k = (z_{k,1}, z_{k,2}, \cdots, z_{k,p})^t$ is given by:

$$\mathbf{z} = \begin{pmatrix} V_1^t & & & \\ & V_2^t & & \\ & & \ddots & \\ & & & V_K^t \end{pmatrix} W_v \mathbf{y}. \tag{3.10}$$

We see also that (3.6) becomes:

$$\boldsymbol{\lambda}_\rho = \frac{1}{K} \bar{W}_v \begin{pmatrix} V_1 & & & \\ & V_2 & & \\ & & \ddots & \\ & & & V_K \end{pmatrix} \begin{pmatrix} (D_1+\rho I)^{-1} & & & \\ & (D_2+\rho I)^{-1} & & \\ & & \ddots & \\ & & & (D_K+\rho I)^{-1} \end{pmatrix} \mathbf{z}. \tag{3.11}$$

Thus if, in place of storing the matrices $(M_k + \rho I)^{-1}$ as suggested in [9], we store the matrices V_k and the coefficients $d_{k,p}$, the computation of a good parameter ρ^* and the associated solution $f_{\rho^*} = R_m^* \boldsymbol{\lambda}_{\rho^*}$ is obtained by the three following steps:

(1) computation of $\mathbf{z}$ following (3.10) (this needs P D.F.T.'s of order $K + K \cdot P^2$ operations);

(2) iterative approximation of ρ^*, e.g. by the golden section search for the minimizing of $v(\rho)$ (computation of $v(\rho)$ needs $3K \cdot P$ operations);

(3) computation of $\boldsymbol{\lambda}_{\rho^*}$ following (3.11) (same cost as (1)).

Since in practice about twenty iterations are always enough to (2), the overall algorithm for this optimal regularization needs only about twice the number of operations of the regularization with a priori fixed ρ.

We turn now to the Poissonian case: the minimizing of $r\hat{}(\rho)$ given by (2.4) needs the evaluation of the trace of the product diag $(y_1, y_2, \cdots, y_m)(I - A(\rho))$, with different values of ρ. We show here how to greatly reduce the cost of this computation.

Denoting $Y = \text{Diag}\,(y_1, y_2, \cdots, y_m)$, and

$$U = \frac{1}{\sqrt{K}} \bar{W}_v \begin{pmatrix} V_1 & & & \\ & V_2 & & \\ & & \ddots & \\ & & & V_K \end{pmatrix} \quad \text{hence } U^{-1} = \frac{1}{\sqrt{K}} \begin{pmatrix} V_1^t & & & \\ & V_2^t & & \\ & & \ddots & \\ & & & V_K^t \end{pmatrix} W_v;$$

from (3.8) we have:

$$\begin{aligned} \text{trace}\,(Y(I - A(\rho))) &= \text{trace}\,(U^{-1}(Y(I - A(\rho))U)) \\ &= \text{trace}\,(L \text{ diag}\,(1 - (d_{k,p}/(d_{k,p} + \rho)), k = 1, \cdots, K, p = 1, \cdots, P)) \end{aligned}$$

where $L = U^{-1}YU$. Thus with $L_{k,p}$, $k = 1, \cdots, K$, $p = 1, \cdots, P$, the diagonal elements of L, we have:

$$\text{trace}\,(Y(I - A(\rho))) = \rho \sum_{\substack{k=1,\cdots,K \\ p=1,\cdots,p}} L_{k,p}[1 - d_{k,p}/(d_{k,p} + \rho)]. \tag{3.12}$$

Now since Y is diagonal, it is easy to verify that we can write:

$$\frac{1}{K} \bar{W}_v Y W_v = \begin{pmatrix} C_1 & C_2 & \cdots & C_K \\ C_K & C_1 & \cdots & C_{K-1} \\ \vdots & \vdots & & \vdots \\ C_2 & C_3 & \cdots & C_1 \end{pmatrix},$$

with C_k diagonal and $C_1 = \text{diag}\,(1/K \sum y_{k,p}, p = 1, \cdots, P)$, hence:

$$\bar{L} = \begin{pmatrix} V_1^t C_1 V_1 & V_1^t C_2 V_2 & \cdots & V_1^t C_K V_K \\ V_2^t C_K V_1 & V_2^t C_1 V_2 & \cdots & V_2^t C_{K-1} V_K \\ \vdots & \vdots & & \vdots \\ V_K^t C_2 V_1 & V_K^t C_3 V_2 & \cdots & V_K^t C_1 V_K \end{pmatrix}.$$

Thus the coefficients $L_{k,p}$ can be computed from y and the memorized matrices V_k with only $2K \cdot P^2$ operations.

We now see that an optimal ρ^* which minimizes $r\hat{}(\rho)$ (2.4) is obtained by the following two steps in place of the step (2) of the previous algorithm:

(2′) computation of the coefficients $L_{k,p}$ ($2K \cdot P^2$ operations);

(2″) minimization of $r\hat{}(\rho)$ (the computation of $r\hat{}(\rho)$ following (3.9) and (3.12) requires $4K \cdot P$ operations).

4. Optimal regularized reconstruction in a space of finite dimension. Here we suppose chosen a space F of finite dimension n which is able to model well any attainable reconstruction and is as simple as possible: by such a discretization we choose a degree of freedom for the reconstruction generally less than the number m of the data (m is about 10^6 in time-of-flight tomography).

Now the definition of a regularized reconstruction with a priori fixed ρ is like Bayesian reconstruction, well known in the tomography literature [2]: assuming known n basis functions $\psi_1, \psi_2, \cdots, \psi_n$ of F, the irregularity measure of any function of F

is defined by a quadratic functional with a nonnegative, symmetric $n \times n$ matrix Δ following:

$$f = \sum \lambda_j \psi_j \Rightarrow \|f\|_{\mathbf{F}}^2 = \boldsymbol{\lambda}^t \Delta \boldsymbol{\lambda} \quad \text{where } \boldsymbol{\lambda} = (\lambda_1, \lambda_2, \cdots, \lambda_n)^t.$$

Then, for any $\rho > 0$, the regularized reconstruction f_ρ is the minimizer on $\mathbf{F}$ of $J_\rho(f) = \sum [(\varphi_{t_i}, f) - y_i]^2 + \rho \|f\|_{\mathbf{F}}^2$.

It is easy to see that the expansion $f_\rho = \sum \lambda_{\rho,j} \psi_j$ is given by the normal equation:

$$\boldsymbol{\lambda}_\rho = (\lambda_{\rho,1}, \lambda_{\rho,2}, \cdots, \lambda_{\rho,n})^t \quad \text{solution of } (H^t H + \rho \Delta) \boldsymbol{\lambda}_\rho = H^t \mathbf{y} \tag{4.1}$$

where H is the $m \times n$ matrix with i,jth entry $H_{i,j} = (\varphi_{t_i}, \psi_j)$. That is still a huge linear system since we must choose n large enough (typically 10,000).

Such reconstructions using iterative methods for approximating f_ρ with a priori fixed ρ are not new in classical tomography [2], but here we give an algorithm of reconstruction using a direct inversion method which computes from the data, first the optimal value ρ^* of ρ, then f_{ρ^*}, for a total cost competing with analytical reconstructions, in the usual case of a rotationally invariant geometry.

It is still the use of some precomputations (following the general algorithm in [6]) which greatly reduces the cost of the direct inversion of this system provided that the basis functions of $\mathbf{F}$ and $\|\cdot\|_{\mathbf{F}}$ has a symmetry of this kind.

Let us make this symmetry precise. We assume that the data can be divided into K subsets, the kth subset containing the P data $y_{(k-1)p+1}, y_{(k-1)p+2}, \cdots, y_{(k-1)p+P}$, and, if $\varphi_{k,p}$ denotes the function $\varphi_{t_{(k+1)P+p}}$, then $\varphi_{k,p}$ is the "rotation" of $\varphi_{k-1,p}$ over angle $360°/K$. More precisely:

$$\varphi_{k,p}(x, y) = \varphi_{1,p}(x \cos \theta_k - y \sin \theta_k, x \sin \theta_k + y \cos \theta_k) \quad \text{where } \theta_k = (k-1)2\pi/K.$$

Now we choose $n = K \cdot Q$ and the ψ_j are generated by successive rotations over angle θ_k, $k = 1, \cdots, K$ of Q functions $\zeta_1, \zeta_2, \cdots, \zeta_Q$, i.e., for $k = 1, \cdots, K$, $q = 1, \cdots, Q$,

$$\psi_j(x, y) = \psi_{k,q}(x, y) = \zeta_q(x \cos \theta_k - y \sin \theta_k, x \sin \theta_k + y \cos \theta_k) \quad \text{with } j = (k-1)Q + q.$$

Thus we could easily verify that the value of:

$$\sum_{i=1}^{m} (\varphi_{t_i}, \psi_{k_0,q_0})(\varphi_{t_i}, \psi_{k_1,q_1}) = (H^t H)_{(k_0-1)K+q_0,(k_1-1)K+q_1}$$

stays invariant if one adds modulo K the same integer to k_0 and k_1. This means that $H^t H$ is a block-circulant matrix like $R_m R_m^*$ in § 3, but with submatrix blocks of order Q.

The simple example for such a base as we propose here is the discretization on a polar grid obtained by partitioning the disc $\mathbf{D}$ in K angular sectors with angle $2\pi/K$, each sector being next partitioned in Q small regions such that the relative pattern of this partition in the sector is not varied: $\psi_{k,q}$ is then the characteristic function of such a "polar pixel."

Since the support of $\psi_{k,q}$ is local, H is a sparse matrix. Note also that the ith component of the vector $H^t \mathbf{y}$:

$$(H^t \mathbf{y})_j = \sum_{i=1,\cdots,m} (\varphi_{t_i}, \psi_j) y_i = \left(\psi_j, \sum_{i=1,\cdots,m} y_i \varphi_{t_i} \right)$$

is the average on the pixel j of the function $\sum y_i \varphi_{t_i}$ which is obtained by the well-known operation of (confidence-weighted) back-projection of the data y_i. So the computation of $H^t \mathbf{y}$ has essentially the same complexity as this back-projection.

Let us turn now to the linear system (4.1). Due to the previous symmetry, only the first Q columns of H^tH have to be computed (that is equivalent to Q elementary projections and back-projections) for the precomputation and storage of the K matrices $M_1, M_2, \cdots, M_K$ of the block-diagonalization of H^tH (same as § 3 except that the order of each block is Q and multiplying by W_v involves now Q D.F.T.'s of order K):

$$H^tH = \frac{1}{K}\bar{W}_v \begin{pmatrix} M_1 & & & \\ & M_2 & & \\ & & \ddots & \\ & & & M_K \end{pmatrix} W_v. \tag{4.2}$$

Now in order to exploit this decomposition, we choose an irregularity measure defined by a matrix Δ which has the same block-circulant form as H^tH. Such a matrix is obtained by the discretization on the polar grid of any rotationally invariant norm, for example:

$$\|f\|_F^2 = \int_D [f(x,y)]^2\,dx\,dy + \int_D \left[\frac{\partial f}{\partial x}(x,y)\right]^2 dx\,dy + \int_D \left[\frac{\partial f}{\partial y}(x,y)\right]^2 dx\,dy. \tag{4.3}$$

The choice of Δ is also equivalent to the choice of K matrices of order Q, $\Delta_1, \Delta_2, \cdots, \Delta_K$, since we have:

$$\Delta = \frac{1}{K}\bar{W}_v \begin{pmatrix} \Delta_1 & & & \\ & \Delta_2 & & \\ & & \ddots & \\ & & & \Delta_K \end{pmatrix} W_v. \tag{4.4}$$

Note that in practice these matrices are explicitly computable and are easy to generate (they are tridiagonal in the upper example). Now from (4.1), (4.2) and (4.3), we get:

$$\boldsymbol{\lambda}_\rho = \frac{1}{K}\bar{W}_v \begin{pmatrix} (M_1+\rho\Delta_1)^{-1} & & & \\ & (M_2+\rho\Delta_2)^{-1} & & \\ & & \ddots & \\ & & & (M_K+\rho\Delta_K)^{-1} \end{pmatrix} W_v H^t\mathbf{y}. \tag{4.5}$$

This decomposition shows that when the $2K$ matrices M_k and Δ_k have been computed (and stored) for a fixed geometry, the computation of the values of f_ρ on the $K \times Q$ polar grid is reduced to the computation of $H^t\mathbf{y}$ followed by $2Q$ D.F.T.'s of order K and K linear systems of order Q.

Thus, if the computation of $H^t\mathbf{y}$ is not taken into account, the cost of the computation of $\boldsymbol{\lambda}_\rho$ is the same as in § 3 (3.6) where P is replaced by Q. Note that the computation of $H^t\mathbf{y}$ may be the most important work: in time-of-flight tomography, this vector is essentially the "confidence-weighted" preimage used in the standard analytical algorithms ([1], [7]) and is typically a 100×100 array derived from around 1,000,000 data.

Let us now turn to the implementation of the computation of the optimal ρ. We see in (4.5) that the computations of $\boldsymbol{\lambda}_\rho$ with any ρ are dependent on $\mathbf{y}$ only through the value of $H^t\mathbf{y}$. In fact, though $s(f_\rho) = \|H\boldsymbol{\lambda}_\rho - \mathbf{y}\|^2$ cannot be expressed as a function of $H^t\mathbf{y}$ independently of $\mathbf{y}$, we have shown in [6] that the optimal value of ρ is well estimated by the minimizer of the following function $\tilde{v}(\rho)$ where $\mathbf{y}$ is only required through $H^t\mathbf{y}$:

$$\tilde{v}(\rho) = \frac{l\sum_{j/d_j\neq 0}[\rho z_j/(d_j(d_j^2+\rho))]^2}{[\sum_{j/d_j\neq 0}\rho/(d_j^2+\rho)]^2} \tag{4.6}$$

where

$$\mathbf{z}=(z_1,\cdots,z_n)^t;\qquad \mathbf{z}=\bar{V}^tL^{-1}H^t\mathbf{y}; \tag{4.7}$$

(4.8a) the $n\times n$ matrix L is the symmetric square root of Δ;

(4.8b) V is a $n\times n$ unitary matrix/ V diag $(d_j^2, j=1,\cdots,n)\,V^t=L^{-1}H^tHL^{-1}$; l is the number of nonzero d_j.

This result can be obtained by transforming the initial problem of fitting m data in an equivalent problem with only l data. This latter has still the same optimal parameter. We can thus apply to it the cross-validation method which consists in minimizing (4.6) to estimate the searched parameter ρ^0 (see [6], also for a similar result when L is not invertible).

Let us now see how $L^{-1}V$ and the values $d_j, j=1,\cdots,n$ can be computed and stored for a reasonable cost, even for very large systems.

First, by computing only the submatrix blocks L_k^{-1}, where L_k denotes the symmetric square root of Δ_k for $k=1,\cdots,K$, we obtain the following form of L^{-1}:

$$L^{-1}=\frac{1}{K}\bar{W}_v\begin{pmatrix}L_1^{-1} & & & \\ & L_2^{-1} & & \\ & & \ddots & \\ & & & L_K^{-1}\end{pmatrix}W_v. \tag{4.9}$$

Second, a simple form for the matrix V is provided by the computation of the blocks $L_k^{-1}M_kL_k^{-1}$ followed by their diagonalization by unitary transform:

$$L_k^{-1}M_kL_k^{-1}=V_K\ \text{diag}\,(d_{k,1}^2, d_{k,2}^2,\cdots,d_{k,Q}^2)\,V_k^t,$$

since we easily verify with (4.2) and (4.9) that the matrix

$$V=\frac{1}{\sqrt{K}}\bar{W}_v\begin{pmatrix}V_1 & & & \\ & V_2 & & \\ & & \ddots & \\ & & & V_K\end{pmatrix}$$

is unitary and diagonalizes $L^{-1}H^tHL^{-1}$.

Finally, the computation of the K blocks $B_k=L_k^{-1}V_k$ gives $L^{-1}V$ required in (4.7), in the form:

$$L^{-1}V=\frac{1}{\sqrt{K}}\bar{W}_v\begin{pmatrix}B_1 & & & \\ & B_2 & & \\ & & \ddots & \\ & & & B_K\end{pmatrix}. \tag{4.10}$$

Thus $\mathbf{z}$ required in the fast computation of $v(\rho)$ can be computed by:

$$\mathbf{z}=\bar{V}^tL^{-1}H^t\mathbf{y}=\frac{1}{\sqrt{K}}\begin{pmatrix}B_1^t & & & \\ & B_2^t & & \\ & & \ddots & \\ & & & B_K^t\end{pmatrix}W_vH^t\mathbf{y}. \tag{4.11}$$

Furthermore, substituting $L^{-1}H^tHL^{-1}$ by its eigenvalues decomposition (4.8b), we see that:

$$\begin{aligned}\boldsymbol{\lambda}_\rho &= (H^tH+\rho L^2)^{-1}H^t\mathbf{y}\\ &= L^{-1}V\,\mathrm{Diag}\,(1/(d_j^2+\rho), j=1,\cdots,n)\,\bar{V}^tL^{-1}H^t\mathbf{y};\end{aligned}$$

hence with (4.7) and (4.10):

(4.12)

$$\boldsymbol{\lambda}_\rho = \frac{1}{\sqrt{K}}\bar{W}_v\begin{pmatrix} B_1 & & & 0\\ & B_2 & & \\ & & \ddots & \\ 0 & & & B_K\end{pmatrix}\mathrm{Diag}\,(1/(d_{k,q}^2+\rho), k=1,\cdots,q=1,\cdots,Q)\cdot\mathbf{z}.$$

Finally by storing the blocks B_k and the n eigenvalues $d_{k,q}^2$ in place of M_k and Δ_k, this reconstruction with optimal parameter is thus obtained by the following algorithm:

(0) computation of $H^t\mathbf{y}$

(1) computation of $\mathbf{z}$ by (4.11) (needs Q D.F.T.'s of order K and $K\cdot Q^2$ operations)

(2) iterative approximation of the minimizer ρ^* of $\tilde{v}(\rho)$ ($\tilde{v}(\rho)$ is computed with $3K\cdot Q$ operations)

(3) computation of $\boldsymbol{\lambda}_{\rho^*}$ by (4.12) (Q D.F.T.s of order K and $K\cdot Q^2$ operations).

Note that, if the step (0) is not taken into account, this algorithm has nearly the same cost and needs the same memory space as the algorithm of § 3 (3.6) where P is replaced by Q.

5. Numerical applications. These two reconstruction algorithms have been implemented (on a VAX 11/780) for the classical tomography as described in § 3 and the second one for the time-of-flight tomography. In classical tomography, we have taken $K=192$ directions, $P=64$ juxtaposed strips by view, uniformly covering the support disc D of diameter $2P$:

$$L_{k,p}=\{(x,y)\in\mathbf{D}\colon p-1<x\cos((k-1)2\pi/K)+y\sin((k-1)2\pi/K)<p\},$$
$$k=1,2,\cdots,K,\quad p=1,2,\cdots,P.$$

This is a typical parallel geometry of 96 projections, with 128 strip integral in each. Numerical tests presented here were made principally with the following test object:

$$f^0=\chi_{D(0,0,30)}+\chi_{D(0,15,15)}+\chi_{D(0,20,4)}+\chi_{D(0,10,2)},$$

where $\chi_{D(x,y,r)}$ is the characteristic function of the disc $D(x,y,r)$ centered in x,y of radius r. The exact data vector $\mathbf{y}^0$ was computed analytically and we simulated a data vector $\mathbf{y}$ by adding a white noise to $\mathbf{y}^0$, with a variance $\sigma^2=9$.

5.1. Regularization by energy (§ 3). In this algorithm we had to precompute and store $K/2+1=97$ matrices of order $P=64$ and 97 by 64 eigenvalues (S.V.D. made by procedures of IMSL). Note that the observation of these eigenvalues (in [7]) has shown the need of a regularization. To evaluate the efficiency of the methods of choosing ρ, we studied the variations with ρ of v, r ((2.5), (2.2)) and s ($s(\rho)$ denotes $s(f_\rho)$). Note that for any ρ, $r(\rho)$ is also computable with the same cost as the calculation of v or s in the step (2) as soon as the step (1) is also made with $\mathbf{y}^0$ in place of $\mathbf{y}$ in order to compute $\mathbf{z}^0$, since we easily see that

$$r(\rho)=\frac{1}{K\cdot P}\frac{1}{K}\sum\left|z_{k,p}^0-\left(\frac{d_{k,p}}{d_{k,p}+\rho}\right)z_{k,p}\right|^2.$$

Figure 1 shows that the cross-validation method (i.e., to search the minimizer ρ_v of the function $v(\rho)$) produces a very good estimate of the optimal parameter. The discrepancy method (i.e., to take ρ_s such that $s(\rho_s)=9$) gives an oversmoothing and a mean square error of prediction of the exact data $r(\rho_s)$ early twice as big as $r(\rho_v)$.

Then we created another data vector $\mathbf{y}$ by stimulating a Poissonian noise, but here with the geometry of 20 projections, each one containing 64 measurements (i.e., $K=40$, $P=32$) and

$$f^0 = 3(\chi_{D(0,0,28)} + \chi_{D(0,14,12)} + \chi_{D(0,15,2)}),$$

and we computed the estimate $r\hat{}$ (2.4) of r.

We observe in Fig. 2 that this estimate is very good. However the cross-validation method which is not so expensive, still works very well.

For reconstructions produced by this algorithm with good value of ρ, see [9], [7].

5.2. Regularization on a polar grid. Next we applied the optimal regularization algorithm on a polar grid of § 4 with the first geometry ($K=192$, $P=64$) and previous (white noise) data $\mathbf{y}$. The basis functions used here are simply the following polar "pixels" for $k=1,\cdots,K$, $p=1,\cdots,P$:

$$\psi_{k,p}(x,y)=\psi_{k,p}(r\cos\theta, r\sin\theta)=\begin{cases}1 & \text{if } |\theta-2\pi(k-1)/K|<\pi/_K \text{ and } p-1<r<p,\\ 0 & \text{otherwise,}\end{cases}$$

where $r=(x^2+y^2)^{1/2}$.

The irregularity measure taken in the following experiments is a classical discretization on this grid, of the sum (4.3) of the $\mathbf{L}_2$-norms of the function and his gradient. Note that the storage space needed is still $97\times64\times65$. Versus the previous algorithm, the mean square error of prediction of exact data $\tilde{r}(\rho)$ and the infidelity from the data $\tilde{s}(\rho)$ are now related to the equivalent problem with l data and are easily computed by ([6]):

$$\tilde{r}(\rho)=\frac{1}{l}\sum_{k,q/d_{k,q}\neq 0}|z^0_{k,q}-(d^2_{k,q}/(d^2_{k,q}+\rho))z_{k,q}|^2/d^2_{k,q},$$

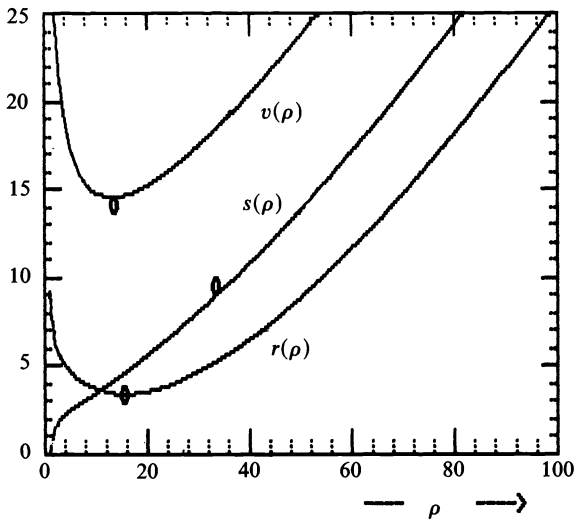


FIG. 1. *Exact error $r(\rho)$, infidelity $s(\rho)$ and cross-validation $v(\rho)$ functions of the regularization parameter ρ. Minimal energy reconstructions. 96 projections, 128 detectors; white noise.*

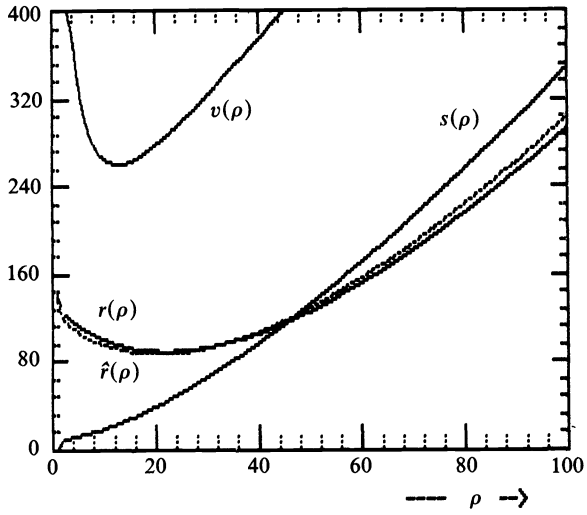


FIG 2. $r(\rho)$, $s(\rho)$, $v(\rho)$ *and estimated error* $\hat{r}(\rho)$. 20 *projections*, 64 *detectors*; *Poisson noise.*

and

$$\tilde{s}(\rho)=\frac{1}{l}\sum_{k,q/d_{k,q}\neq 0}(1-(d_{k,q}^2/(d_{k,q}^2+\rho)))|z_{k,q}|^2/d_{k,q}^2,$$

where $\mathbf{z}^0$ is computed from $\mathbf{y}^0$ in the same way that z is computed from $\mathbf{y}$ (i.e., steps (0) and (1)).

Note that the discrepancy method (on the equivalent problem) still consists of the equation: $\tilde{s}(\rho)=\sigma^2$, and recall that the optimal parameter ρ^0 is also the minimizer of $\tilde{r}(\rho)$.

We observe in Fig. 3 that the cross-validation method applied to the equivalent problem still produces a very good estimate of the optimal parameter, better than the estimate produced by the discrepancy method.

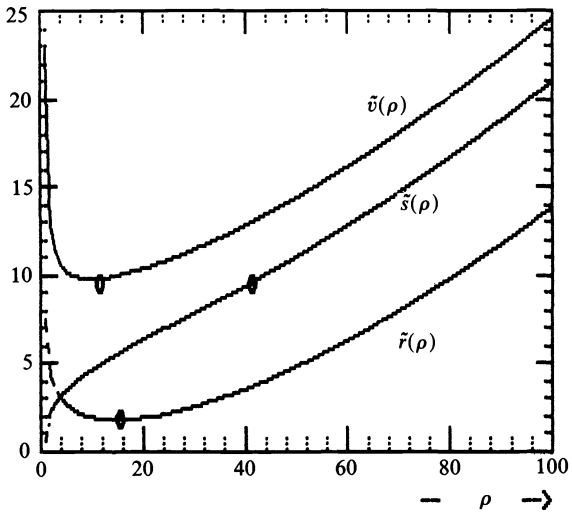
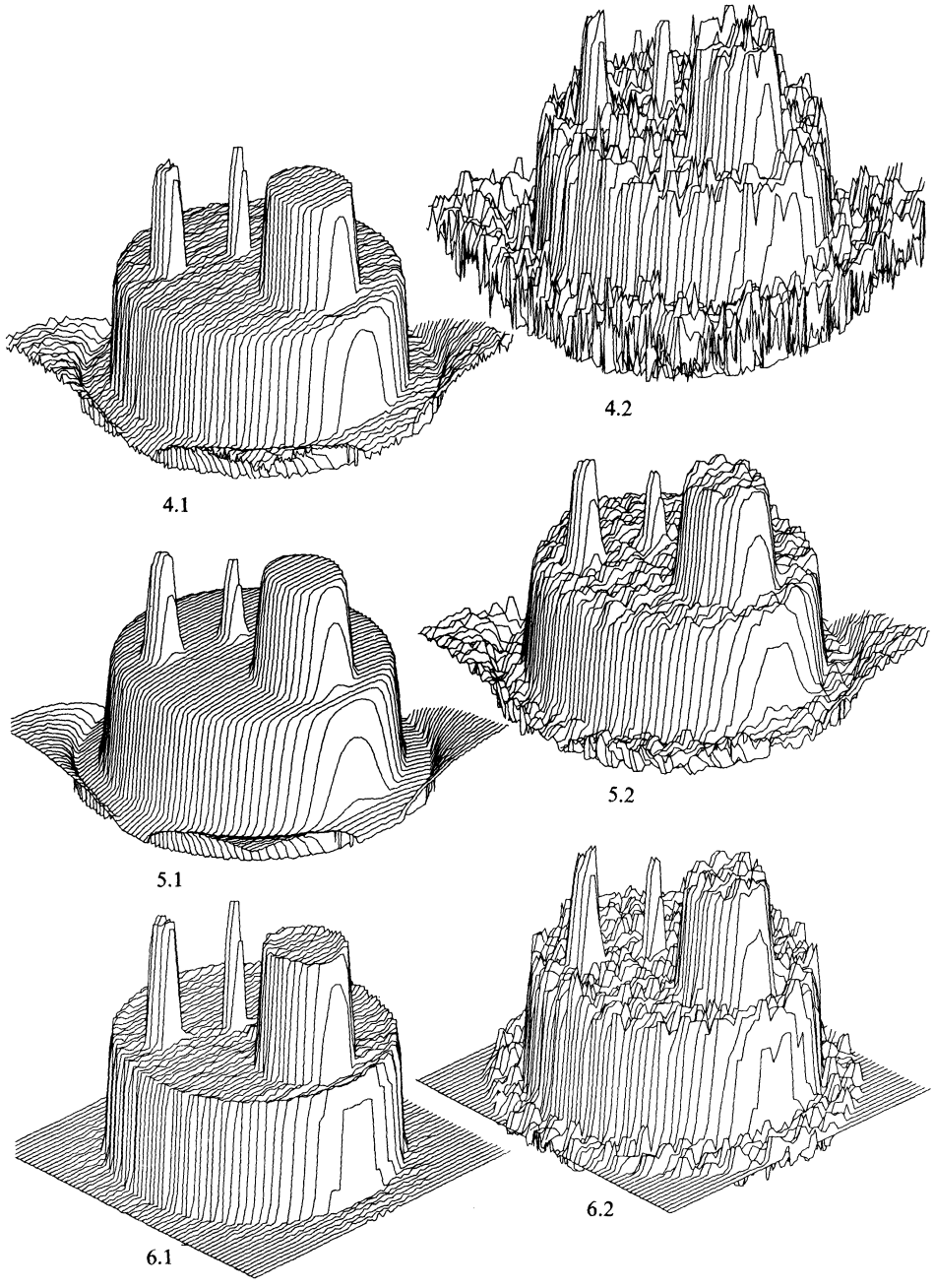


FIG 3. $\tilde{r}(\rho)$, $\tilde{s}(\rho)$ *and* $\tilde{v}(\rho)$. *Regularization on a* 96×128 *polar grid.* 96 *projections*, 128 *detectors*; *white noise.*



FIGS. 4, 5, 6. *Reconstructions from* 96 *projections in* 180°, 128 *detectors.*

	Exact data	*Noisy data*
1*st analytical algorithm*	Fig. 4.1	Fig. 4.2
2*nd analytical algorithm*	Fig. 5.1	Fig. 5.2
cross-validated regularization	Fig. 6.1	Fig. 6.2

Now let us see how this algorithm works when ρ^* is computed by the cross-validation method, compared to the standard analytical reconstruction algorithms ([11]). These analytical algorithms are filtered back-projection methods using linear interpolation in the back-projection step and are derived from various Hamming filters (see [12] for details).

For that we have computed exact and noisy data on a test object as simple as the previous one (given constant on 4 discs). Two analytical reconstruction algorithms have been chosen so that the first one produces a good reconstruction with the accurate data and the second one with the noisy data (i.e. the parameters of the Hamming filter in each one have been adjusted by trial and error). The reconstructions produced by these two methods and our algorithm are presented by 3D views in Figs. 4.1, 5.1 and 6.1 for the exact data and in the Figs. 4.2, 5.2 and 6.2 for the noisy data.

We see in Fig. 4.2 the numerical instability of the first analytical method with noisy data and in Fig. 5.1 the bad resolution in the second analytical reconstruction with accurate data. Comparing Fig. 4.1 with Fig. 6.1, and Fig. 5.2 with Fig. 6.2, we see that in both the two cases our algorithm produces a reconstruction as good as the best of the analytical methods.

This optimal regularization on this 96×128 polar grid has been applied to time-of-flight tomography with $96 \times 128 \times 64$ data. Numerical experiments have been presented in [6] and [7], with test objects of the previous type.

In all the experiments made, we have observed a very good efficiency of the cross-validation method, and that the statistical method often cannot produce a significant improvement though it uses the error variance.

We have experimented with other irregularity functionals. We have observed that the choice of this functional is not so crucial as the choice of the parameter, as soon as this functional includes a differential operator in order to control the local oscillation of any solution.

We can draw the conclusion that the images produced by these new reconstruction algorithms are often as good as the most suitable analytical reconstructions, even in the case of more than 10^4 data. As this paper shows that the cost of such an optimal regularization is competitive, it is hoped that these algorithms will find practical applications in tomography. Note also that these algorithms can take into account various geometries of tomographs (strip or cone in place of line, incomplete data like the so-called truncated projections) and have the same complexity for all the cases which preserve the rotational invariance.

REFERENCES

[1] R. ALLEMAND ET AL. (1980), *A new time-of-flight method for positron computed tomography*, in Biomedical Images and Computers, D. A. B. Linberg and P. L. Reichertz, eds., Springer, Berlin.

[2] A. ARTZY, T. ELFVING AND G. T. HERMAN (1979), *Quadratic optimisation for image reconstruction*, II, Computer Graphics and Image Processing, 11, pp. 242–261.

[3] T. CHANG AND G. T. HERMAN (1980), *A scientific study of filter selection for a fan-beam convolution reconstruction algorithm*, SIAM J. Appl. Math., 39, pp. 83–105.

[4] P. CRAVEN AND G. WAHBA (1979), *Smoothing noisy data with spline functions*, Numer. Math., 31, pp. 377–403.

[5] N. DYN AND G. WAHBA (1982), *On the estimation of functions of several variables from aggregated data*, SIAM J. Math. Anal., 13, pp. 134–152.

[6] D. GIRARD (1986), *Practical optimal regularization of large linear systems*, M^2AN, RAIRO Anal. Num., 20, pp. 75–87.

[7] ——— (1984), doctoral thesis, Institut National Polytechnique de Grenoble, France.

[8] P. J. LAURENT (1972), *Approximation et optimisation*, Hermann, Paris.

[9] F. NATTERER (1980), *Efficient implementation of "optimal" algorithms in computerized tomography*, Math. Methods Appl. Sci., 2, pp. 545-555.

[10] C. H. REINSCH (1971), *Smoothing by spline functions* II, Numer. Math., 16, pp. 451-454.

[11] L. A. SHEPP AND B. F. LOGAN (1974), *The Fourier reconstruction of a head section*, I.E.E.E. Trans. Nuclear Science, N.S. 21, pp. 21-43.

[12] S. W. ROWLAND (1979), *Computer implementation of image reconstruction formulas*, in Image Reconstruction from Projection: Implementation and Applications, G. T. Herman, ed., Springer-Verlag, Berlin and New York, pp. 9-80.

[13] K. T. SMITH, D. C. SOLMON AND S. L. WAGNER (1977), *Practical and mathematical aspects of the problem of reconstructing objects from radiographs*, Bull. Amer. Math. Soc., 83, pp. 1227-1270.

[14] M. STONE (1974), *Cross-validatory choice and assessment of statical prediction*, with discussion, J. Roy. Statist. Soc., Ser. B, 36, pp. 111-147.

[15] M. M. TER-POGOSSIAN ET AL. (1980), *Positron-emission tomography*, Scientific American, October, pp. 170-181.

[16] G. WAHBA, 1977, *Practical approximate solution to linear operator equations when the data are noisy*, SIAM J. Numer. Anal., 14, pp. 651-667.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 6, November 1987
© 1987 Society for Industrial and Applied Mathematics
006

A NEW ALGORITHM FOR THE NUMERICAL APPROXIMATION OF AN INVARIANT CURVE*

M. VAN VELDHUIZEN†

Abstract. This paper describes a new algorithm for the numerical approximation of an invariant curve for a system of ordinary differential equations. The algorithm is based on the stability properties of the invariant curve. It is possible to approximate stable invariant curves with vanishing rotation number by means of the new algorithm. Numerical examples, for the driven van der Pol equation and the delay logistic equation, illustrate the possibilities of the proposed algorithm.

Key words. invariant manifold, polygonal approximation

AMS(MOS) subject classifications. Primary 65L99; secondary 65H10, 34C40

1. Introduction.

In this paper we consider the ordinary differential equation

$$\frac{dx}{dt} = F(t, x) \tag{1.1}$$

where $x(t) \in \mathbb{R}^d$ and where $F: (t, x) \to F(t, x)$ is a smooth map from $\mathbb{R} \times \mathbb{R}^d$ into $\mathbb{R}^d$. In this paper F will be periodic with respect to the t-variable with period $\mathbf{p}$. Hence, for all times t we have $F(t+\mathbf{p}, x) = F(t, x)$. For such systems of ordinary differential equations one often investigates the existence of oscillatory solutions, both by analytic means and by numerical methods. Sometimes one is interested in periodic solutions with period $m\mathbf{p}$, m a natural number, but sometimes one looks for more complicated oscillatory motion. To that end we reformulate the problem by means of the Poincaré map $\mathbf{P}$.

Under reasonable conditions the differential equation (1.1) with initial vector x_0 at $t = 0$ has a unique solution $x(t; x_0)$ on $[0, \mathbf{p}]$. Hence, by assigning to x_0 at $t = 0$ the vector $x(\mathbf{p}; x_0)$ at $t = \mathbf{p}$ we define a map from (part of) $\mathbb{R}^d$ to $\mathbb{R}^d$, the Poincaré map $\mathbf{P}$ related to the differential equation (1.1) and the time-step $\mathbf{p}$. The Poincaré map describes how $\mathbb{R}^d$ is transformed by the differential equation (1.1) in one period $\mathbf{p}$. Clearly, the Poincaré map $\mathbf{P}$ is smooth if the right-hand side F of the differential equation (1.1) is smooth.

A fixed point of the Poincaré map $\mathbf{P}$ corresponds to a periodic solution of (1.1) with period $\mathbf{p}$. Indeed, $x_0 = \mathbf{P}x_0$ really means $x_0 = x(\mathbf{p}; x_0)$. Thus, periodic solutions of the differential equation (1.1) are determined by the solutions (if any) of the equation $x = \mathbf{P}^m x$, and a solution x_0 of this equation is the initial vector at $t = 0$ of a periodic solution with period $m\mathbf{p}$, $m \in \mathbb{N}$.

In what follows it is convenient to assume $F(t, x) = f(x) + \alpha g(t)$ and g $\mathbf{p}$-periodic, and α a small parameter. First, assume $\alpha = 0$, and let the autonomous equation $\dot{x} = f(x)$ possess a nontrivial periodic solution with period $\mathbf{p}$. This solution is not unique. Indeed, any shift in time of a periodic solution is again a periodic solution of $\dot{x} = f(x)$. Now draw the curve $\theta \to x(\theta)$ in $\mathbb{R}^d$, x a periodic solution. Any point on this curve is mapped by the Poincaré map on the curve. Thus, the graph of a periodic solution of $\dot{x} = f(x)$ is an invariant curve of the Poincaré map $\mathbf{P}$. For stable periodic solutions Levinson [L1] shows that the Poincaré map $\mathbf{P}$ of the differential equation $\dot{x} = f(x) + \alpha g(t)$

* Received by the editors November 11, 1985; accepted for publication (in revised form) November 23, 1986.

† Department of Mathematics and Computer Science, Vrije Universiteit, 1081 HV Amsterdam, the Netherlands.

possesses an invariant curve for $\alpha > 0$ small. That is, the invariant curve persists under perturbations, but this is not necessarily true for the periodic motion.

The existence of an invariant curve of **P** clearly implies the existence of oscillatory motion. The kind of oscillatory motion is determined by the *rotation number* ρ of the restriction of **P** to its invariant curve γ. For a formal definition of the rotation number, see Coddington and Levinson [CL1], Hale [H1] or Guckenheimer and Holmes [GH1]. Roughly speaking, the inverse of the rotation number ρ indicates how many time intervals of length **p** (the period) are necessary to map a point x_0 on the invariant curve on to itself by using it as an initial vector for the differential equation at time $t = 0$. Thus, a periodic motion corresponds with a rational rotation number and an irrational rotation number excludes periodic motion. Clearly, it is of some interest to be able to compute an invariant curve of the Poincaré map **P**. Once the invariant curve is known, we may approximate the rotation number. Of course, numerical approximations result in a rational rotation number. This means that we must distinguish between rational rotation numbers with large and small denominators, i.e., period motion on a relatively short time-scale and very long term periodic motion. The latter cannot be distinguished from nonperiodic oscillatory motion, at least not numerically.

Usually, bifurcations are studied by solving the bifurcation equation, i.e., by solving for those points in the parameter space where a Fréchet derivative is singular. In many cases this suffices, but sometimes a more subtle analysis is necessary. For instance, in the driven van der Pol equation a codimension two bifurcation takes place, and the analysis of the bifurcation equation alone does not suffice in obtaining a complete qualitative description of the oscillatory phenomena displayed by the differential equation. See Holmes and Rand [HR1], or Guckenheimer and Holmes [GH1]. In § 3 we shall give a numerical example related to the driven van der Pol equation.

Clearly, there is some interest in approximating invariant curves of the Poincaré map **P**. The first algorithms seem to be due to Jean [J1], and Thoulouze-Pratt and Jean [TJ1]. The idea behind these algorithms is best explained if the dimension $d = 2$. Suppose M is a point in the interior of an invariant Jordan curve γ of the differential equation (1.1), and let U be outside γ. Choose a point y on the half-line MU, and compute its iterates $\mathbf{P}^i y$, $i = 1, 2, \cdots$. If y is on the invariant curve γ, and if the rotation number is irrational (the most likely case), then there is a first index $i > 1$ such that the line segment $[\mathbf{P}^i y, \mathbf{P}^{i+1} y]$ cuts the half-line MU. The algorithm now determines the intersection of the half-line MU and the segment $[\mathbf{P}^i y, \mathbf{P}^{i+1} y]$. Let $\Gamma(y)$ be this intersection. Then one solves the equation $y = \Gamma(y)$ for y on MU. Once y is known, the invariant curve, or points on it, can be computed. In this sketch of the algorithm linear interpolation is used, but better interpolation improves the properties. For more details, see Jean [J1] or Thoulouze-Pratt [TP1].

As already observed by Jean [J1], this type of algorithm does not work if the rotation number is almost equal to 0 (mod 1). Indeed, in this case an almost infinite number of iterates $\mathbf{P}^i y$ is required before one passes the half-line MU for the first time. A result by Levinson [L1] shows that in a neighborhood of the bifurcation from a periodic orbit the rotation number is continuous in the parameter α. At the bifurcation the rotation number may be equal to 0. Thus, the above algorithms may fail in the neighborhood of a bifurcation. Also, the quality of the algorithm depends on the distribution of the Poincaré iterates along the invariant curve γ. In fact, a bad distribution may occur, resulting in a relatively large interpolation error and a relatively large approximation error for the invariant curve.

A different approach is used in the work of Doedel and Chan (cf. [Ch1]) and Kevrekidis et al. (cf. [K1]). In Chan's thesis the basis of one of the algorithms is the

conjugacy of the circle map (the restriction of the Poincaré map to the invariant curve γ) and a fixed rotation. See Guckenheimer and Holmes [GH1] for an introduction to these concepts. This conjugacy is used in defining a collocation method. A nonuniqueness problem, similar to the nonuniqueness of a periodic solution of an autonomous equation, is solved by introducing a suitable additional restriction, the so-called "anchor" equation. A second method described by Chan [Ch1] discretizes the relation

$$\mathbf{P}\gamma(t) = \gamma(\sigma(t))$$

with unknown invariant curve γ and unknown circle map σ. Both γ and σ are approximated by a truncated Fourier series in a collocation method. Additional equations are obtained by requiring a nice distribution of the values of σ in the collocation points. A final indeterminacy (similar to the indeterminacy in a period solution of an autonomous system) is again resolved by the use of an "anchor" equation.

The work of Kevrekidis et al. [K1] is based on a parametrization of the invariant curve by an angular coordinate θ and a radial coordinate (vector) r. The invariant curve is given by the points $(\theta, r(\theta))$. A discretization is given by the points (θ_i, r_i). A system of equations is obtained by requiring that the images of these points under the Poincaré map $\mathbf{P}$ belong to the curve obtained by interpolation between the points (θ_i, r_i). The nonlinear equations are then solved by Newton's method.

Both the Chan methods and the K-A-S-P [K1] methods are applicable to stable and repellent invariant curves. But there are a few drawbacks. The second Chan method can produce wrong answers (cf. [Ch1]). The K-A-S-P methods [K1] are promising, but these methods depend on the choice of the parametrization. In addition, no efficient step size control is described in [K1]. In spite of this, these methods are valuable tools and a basis for further research.

In the following section we shall present a new algorithm, based on the stability of the invariant curve. In some sense it may be viewed as a numerical implementation of Levinson's proof (cf. [L1]). In proving his results, Levinson [L1] exploits the attraction to the invariant curve. The algorithm described in the following section approximates the invariant curve γ by a polygon, and solves by iteration for this polygon. The iterative process relies on the attraction to the curve γ. In the iteration process step size adjustments are possible and often necessary. The method does not depend on a parametrization. For this we pay a price. Instead of interpolation we use a nonlinear projection. In principle, the behavior of the new algorithm does not depend on the rotation number. In fact, it is possible to approximate invariant curves on which the Poincaré map has a fixed point. An advantage of the algorithm seems to be its reliability. A disadvantage is its explicit dependence on the stability properties of the invariant curve.

2. The algorithm. The new algorithm for the numerical approximation of an invariant curve γ will now be described. The curve γ will be approximated by a polygon in $\mathbb{R}^d$. A polygon is completely determined by its vertices. The N vertices, vectors in $\mathbb{R}^d$, are denoted by $x_1, x_2, \cdots, x_N$, and the polygon $\mathbf{p}(\{x_i\}_{i=1}^N)$ is obtained as the set of segments $[x_1, x_2]$, $[x_2, x_3]$, $\cdots$, $[x_{N-1}, x_N]$ and $[x_N, x_1]$. A vertex x_i of an approximating polygon is mapped by the Poincaré map $\mathbf{P}$ (see § 1) on the image $\mathbf{P}x_i$, $i = 1, 2, \cdots, N$. In this way a polygon $\mathbf{p}(\{x_i\}_{i=1}^N)$ approximating the curve γ at $t = 0$ is mapped to another polygon $\mathbf{p}(\{\mathbf{P}x_i\}_{i=1}^N)$ approximating γ at $t = \mathbf{p}$. For a stable invariant torus, i.e., for an attracting curve γ, we expect that $\mathbf{p}(\{\mathbf{P}x_i\}_{i=1}^N)$ is a better approximation than $\mathbf{p}(\{x_i\}_{i=1}^N)$.

At first sight this suggests that the sequence of polygons $\mathbf{p}(\{\mathbf{P}^n x_i\}_{i=1}^N)$, $n = 1, 2, \cdots$, converges to γ, provided we start in the neighborhood of γ. This, however, may not

be the case. Of course, for an attracting curve γ the sequence of points $\mathbf{P}^n x_i$, $n = 1, 2, \cdots$, comes arbitrarily close to the curve γ, and it may even converge to a point on the curve γ. In fact, what happens on the curve γ is mainly determined by the rotation number of the restriction of the Poincaré map $\mathbf{P}$ to γ. Convergence takes place if this rotation number is rational. In this case it may even happen that the sequences $\mathbf{P}^n x_i$, $n = 1, 2, \cdots$, converge to the same point on γ for all indices i. Equivalently, the sequence of polygons $\mathbf{p}(\{\mathbf{P}^n x_i\}_{i=1}^N)$ may converge in the limit to a mere point on γ, not to γ itself. Thus, in an algorithm apart from applying the Poincaré map $\mathbf{P}$ over and over again, we must somehow distribute the vertices more or less homogeneously in an annular neighborhood of γ and keep them that way. To that end we project the vertices of the old polygon $\mathbf{p}(\{x_i\}_{i=1}^N)$ on the new polygon $\mathbf{p}(\{\mathbf{P}x_i\}_{i=1}^N)$ by means of a nonlinear projection H.

The nonlinear projection H is defined as follows. Let $x \in \mathbb{R}^d$ be a vector in the neighborhood of a polygon $\mathbf{p}(\{x_i\}_{i=1}^N)$. We assume that the distance of the vector $\mathbf{x}$ to the polygon is so small that the minimal distance of $\mathbf{x}$ to the vertices x_i is attained for at most two adjacent vertices. Of course, there is always a vertex, say x_l, such that for all $i = 1, 2, \cdots, N$ we have

$$\|x - x_l\| = \min_{i=1,\cdots,N} \|x - x_i\|.$$

Our assumption says that there are at most two indices with this property, say l and $m = l+1 \bmod N+1$. If there are two minimizing adjacent vertices for x then we define Hx with respect to the polygon by $Hx = \frac{1}{2}(x_l + x_m)$, i.e., the middle of the line segment $[x_l, x_m]$ of the polygon. In all other cases there is only one minimizing vertex x_l in the polygon. For the moment we assume that the three vectors x_k, x_l and x_m where $k = l-1 \bmod N+1$ determine a plane, a two-dimensional linear manifold, in $\mathbb{R}^d$. Later we shall indicate what to do if this is not so.

First, we determine the point $\hat{x}$ in the plane determined by the three vertices x_k, x_l, x_m as the euclidean projection of x on this plane. Second, in the plane determined by the three vertices x_k, x_l and x_m we determine the centre of the circumscribing circle of this triangle. Let S denote this centre (cf. Fig. 1). In order to define the projection H uniquely, we now make the additional assumption that the angle between the line segments of the polygon at x_l is obtuse. Since $\|x - x_l\|$ is smaller than both $\|x - x_m\|$ and $\|x - x_k\|$, it follows that the projection $\hat{x}$ must belong to the shaded quadrilateral

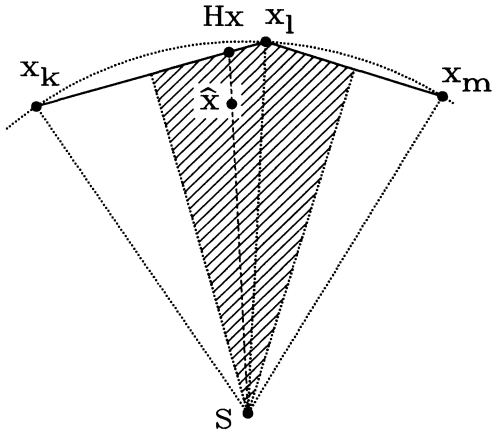


FIG. 1. *The definition of Hx in the plane determined by the successive vertices x_k, x_l and x_m. The projection of x on this plane is $\hat{x}$. The centre of the circle (dotted curve) through x_k, x_l, x_m is the point S.*

in Fig. 1. The projection Hx on the polygon is now defined as the intersection of the line connecting x and S with the polygonal line segment $[x_k, x_l]$ or $[x_l, x_m]$; see the dashed line in Fig. 1.

If the three vertices are all three of them on the line $[x_k, x_m]$, then we define Hx with respect to the polygon as the euclidean projection in $\mathbb{R}^d$ of x on the line segment $[x_k, x_m]$. Our assumptions guarantee that the projection belongs to this line segment.

It is easily seen that the nonlinear projection H is well defined. In fact, the following lemma holds true.

LEMMA 2.1. *Let $\mathbf{p}(\{x_i\}_{i=1}^N)$ be a given polygon and x a vector such that*

(i) *The euclidean distance of x to the polygon $\mathbf{p}(\{x_i\}_{i=1}^N)$ is small enough;*

(ii) *The angle between any two successive line segments in the polygon is obtuse. Then the nonlinear projection $x \to Hx$ described above is well-defined, and H is a continuous map.*

It is readily seen that both conditions in the above lemma are necessary for H to be continuous. First, let us show why the distance between x and the polygon must be sufficiently small. To that end, let $\mathbf{p}(\{x_i\}_{i=1}^N)$ be a regular polygon and let x be the centre of the circumscribing circle. Then H cannot be continuous in this centre. This argument shows that H as defined cannot be continuous everywhere. However, a much stronger result holds true. No continuous definition of H is possible as a consequence of the topological result that the sphere cannot be mapped continuously on its boundary.

As for the second condition, if sharp angles between successive line segments occur, then the line connecting the point S and the point $\hat{x}$ may fail to intersect any of the two line segments. This shows that the two conditions are necessary. Observe that the map $x \to Hx$ is continuous, but not Fréchet differentiable for all x. For instance, if Hx coincides with a vertex of the polygon, then H is not differentiable in this point x (unless the line segments joining in the vertex are linearly dependent).

ALGORITHM 2.2. Let a polygon $\mathbf{p}(\{x_i\}_{i=1}^N)$ be given. Let $N_{\max}$ be the maximum number of vertices in a polygon, and let $\varepsilon > 0$ denote a tolerance, and $\delta > 0$ a prescribed minimal distance between vertices.

(i) Stop the algorithm with an error message if $N > N_{\max}$. In all other instances, delete those vertices from the polygon which are within a distance δ from their predecessor. This results in a possibly smaller polygon, which we nevertheless denote again by $\mathbf{p}(\{x_i\}_{i=1}^N)$. Thus the value of N may change in this step.

(ii) Compute $\mathbf{p}(\{\mathbf{P}x_i\}_{i=1}^N)$ by integrating the differential equation (1.1) by means of an automatic integrator, e.g., the Shampine–Gordon code [SG1].

(iii) Compute the angles (or rather their cosines) between the successive line segments of the polygon $\mathbf{p}(\{\mathbf{P}x_i\}_{i=1}^N)$. If all angles are obtuse, accept the image $\mathbf{p}(\{\mathbf{P}x_i\}_{i=1}^N)$ and go to (v).

(iv) If the angle between the line segments at $\mathbf{P}x_k$ is sharp, then we add to $\mathbf{p}(\{x_i\}_{i=1}^N)$ the two vertices $\frac{1}{2}(x_k + x_{k-1})$ and $\frac{1}{2}(x_k + x_{k+1})$ in the right order for and after x_k (all indices should be counted mod $N+1$). Do this for all sharp angles. This results in a new polygon with more vertices than the previous one. Nevertheless, we denote it again by $\mathbf{p}(\{x_i\}_{i=1}^N)$, but the value of N has changed. If $N < N_{\max}$ then go to (ii); else stop with an error message.

(v) Compute the projections Hx_i for $i = 1, 2, \cdots, N$ with respect to the polygon $\mathbf{p}(\{\mathbf{P}x_i\}_{i=1}^N)$. That is, project the vertices on the polygon formed by the images. In this step we try to conserve a more or less homogeneous distribution of the vertices along the curve γ to be approximated. Define a new polygon consisting of the vertices Hx_1, $Hx_2, \cdots, Hx_N$. We have completed one iteration step. Repeat the process, i.e., go to

(i), unless the maximum number of iteration steps is reached, or unless the maximal distance $\max \|Hx_i - x_i\|$, the maximum taken over all indices $i = 1, \cdots, N$ is less than the prescribed small tolerance ε.

Clearly, this algorithm is finite. Either it converges, or it stops with an error message. For instance, if vertices of the polygon $\mathbf{p}(\{x_i\}_{i=1}^N)$ are deleted in step (i), and new vertices are added in step (iv), then it is theoretically just possible that the algorithm starts cycling, by deleting the vertices added in the previous step. Even if this happens the algorithm is finite, because between the addition and the deletion of vertices (steps (iv) and (i), respectively) the number of iteration steps is increased (in step (v)). In our implementation we have $\delta = 10^{-10}$ in a 48-bit mantissa arithmetic. From a computational point of view, step (ii) requires the largest amount of work and we shall discuss this step in some detail.

In step (ii) we integrate numerically a total of N ordinary differential equations (1.1) each of them with a different initial vector x_i. A typical value is $N = 150$. On a computer like a CY750 we integrate these equations by taking them together in one huge differential equation consisting of dN components, and we integrate this set of equations by means of the Shampine-Gordon code. An advantage of this approach is that the overhead of the algorithm is distributed over the integration of dN differential equations. A disadvantage is that a small step size used in the algorithm may be necessary for one equation (1.1), but not for the many other equations (1.1). By taking everything together we avoid as much overhead as possible, but the smoothness and stability properties of the solutions sought may be so different that most of the equations are integrated with too small a step size, i.e., inefficiently. For nonstiff problems the possible disadvantage seems acceptable. For stiff problems, with internal layers at positions determined by the initial vector x_i, the disadvantage certainly outweighs the reduction in the cost of the overhead. For such problems it might be better to integrate the N equations separately.

On a pipeline machine like a CY205 it is much more advantageous to take all equations together. This automatically results in large contiguous vectors, and a code like the Shampine-Gordon code is rather easily vectorized. A speed-up by a factor 6 is possible as compared with an optimized scalar code with do-loop unrolling. This has been measured on a 1-pipe CY205 with $N = 336$ and $d = 2$. It turns out that the integration of the differential equation makes up about 90 percent of the total computing time required by a typical iteration step of the algorithm. Of course, taking all equations together may be inefficient as a consequence of the step size selection (the smallest step wins), but this is not so great a disadvantage as in the case of a scalar processor. However, for stiff equations, e.g., a van der Pol equation with large nonlinear damping, the disadvantage of integrating all equations together is likely to outweigh the advantages of vector speed.

A network of many processors each of them acting more or less independently and each of them equipped with a reasonable amount of memory would suit the kind of integration of step (ii) best. Just assign to each of the processors the integration of an ordinary differential equation (1.1). During integration there is no memory conflict possible: all processes run independently. After the integration one collects all the results working with one processor, generating a new polygon. Then, using the vertices of this polygon as initial vectors, one starts the integration again on each of the many processors, etc. In this kind of computational environment it is also possible to treat stiff equations efficiently (thanks to the many processors available). It is hoped that such a process can be implemented in the near future on the (experimental) network

of processors being developed at the Department of Mathematics and Computer Science of the Vrije Universiteit.

The stopping criterion $\|Hx_i - x_i\| < \varepsilon$ for all indices i as described in step (v) is largely motivated by practical experience, less by theory. In fact, no convergence proof of the iterative process described by the algorithm (for an infinite number of iteration steps) is available. In actual computations convergence to a limiting polygon has been observed for a sufficiently accurate initial guess. Moreover, in all instances we observed a linear convergence rate. Unfortunately, no sound explanation of these phenomena can be given at present.

Nevertheless, some results can be proved. We shall prove a result which is of interest in the actual implementation of the algorithm. In order to formulate this result, we recall the existence of a so-called tubular (or moving) coordinate system in a tubular neighborhood of the curve γ. Let γ be parametrized as a 2π-periodic map $\theta \to u(\theta)$ with a parametrization of class C^∞. Also, let $du/d\theta$ be of constant length 1. Then any vector x in a tubular neighborhood of γ can be written as

$$x = u(\theta) + Z(\theta)\xi \tag{2.1}$$

with $Z(\theta)$ a d by $d-1$ matrix with kernel $v(\theta)$, $v(\theta)$ tangential to γ at $u(\theta)$. The columns of $Z(\theta)$ are normalized to length 1 and mutually orthogonal. Hale [H1] gives a detailed description of such a nonlinear coordinate system (θ, ξ), and he proves its existence. In these tubular coordinates $\|\xi\|$ measures the distance from a point $x = u(\theta) + Z(\theta)\xi$ to the curve γ. In what follows, we mean by the distance $d(x; \gamma)$ of a vector x to γ the euclidean norm of the tubular coordinate ξ of x given by (2.1). The stability of the invariant curve γ will be understood as contraction to γ with contraction factor κ. Now we are able to formulate and to prove the following proposition.

PROPOSITION 2.3. *Assume that for all x in a tubular neighborhood of γ we have $d(\mathbf{P}x; \gamma) \leqq \kappa d(x; \gamma)$, with $0 \leqq \kappa < 1$. Then, if $Hx_i \in [\mathbf{P}x_j, \mathbf{P}x_{j+1}]$ and if the x_i are close enough to γ*

$$d(Hx_i; \gamma) \leqq \kappa \max_{i=1,\cdots,N} d(x_i; \gamma) + O(\|\mathbf{P}x_j - \mathbf{P}x_{j+1}\|^2). \tag{2.2}$$

Proof. The distance between Hx_i and γ is less than or equal to the maximum distance between the line segment $[\mathbf{P}x_j, \mathbf{P}x_{j+1}]$ and γ. Let $\mathbf{P}x_j = u(\theta_j) + Z(\theta_j)\xi_j$ for all indices j. Consider the chord $[u(\theta_j), u(\theta_{j+1})]$. The distance between this chord and γ is at most of the order of $O(\|u(\theta_j) - u(\theta_{j+1})\|^2)$, but this is of the order of $O(\|\mathbf{P}x_j - \mathbf{P}x_{j+1}\|^2)$, at least close to γ. The distance between the chord $[u(\theta_j), u(\theta_{j+1})]$ and the line segment $[\mathbf{P}x_j, \mathbf{P}x_{j+1}]$ is at most $\kappa \max\{d(x_i; \gamma)\}$, the maximum taken over all indices i. The distance between Hx_i and γ is at most equal to the sum of both distances. This proves the result.

This result indicates that if the sequence x_i, Hx_i, $H^2x_i, \cdots$, converges for all indices i, then the limiting polygon differs from γ by at most $O(h^2)$, where

$$h = \max_{i=1,\cdots,N} \|\mathbf{P}x_i - \mathbf{P}x_{i+1}\| \tag{2.3}$$

with x_{N+1} understood as x_1. The actual implementation of the Algorithm 2.2 takes this into account; it computes h in every iteration step and it inserts additional points (by interpolation) if $h > 4|\mathbf{p}(\{x_i\}_{i=1}^N)|/N$, where $|\mathbf{p}(\{x_i\}_{i=1}^N)|$ denotes the sum of the lengths of all the line segments of the polygon. Observe $h > |\mathbf{p}(\{x_i\}_{i=1}^N)|/N$ unless all line segments have the same length. So we must allow $h > |\mathbf{p}(\{x_i\}_{i=1}^N)|/N$. In our implementation the factor 4 turned out to work well. A smaller factor increases the number of vertices rather rapidly.

In all instances we observed convergence of the algorithm under this additional test. After a few iterations the vertices settle down along γ, and a linear convergence process seems to take over the behavior in the iteration steps.

3. Examples. In this section we illustrate the algorithm of the previous section by applying it to the van der Pol equation (with periodic driving term)

$$\ddot{x}+\alpha(x^2-1)\dot{x}+x=\beta\cos(\omega t). \tag{3.1}$$

As in Holmes and Rand [HR1] we introduce the parameters

$$\kappa=\frac{\beta}{\alpha},\qquad \sigma=\frac{(1-\omega^2)}{\alpha}. \tag{3.2}$$

For $\alpha\to 0$ the asymptotic behavior of the solution is known; see Holmes and Rand [HR1] or Guckenheimer and Holmes [GH1]. The behavior is rather complicated, with up to three different periodic solutions for one parameter value pair (σ, κ). Here we are interested in almost periodic motion, in invariant tori. To that end we start with the bifurcation set. Let $x(t; x(0), \dot{x}(0))$ denote the solution at time t of (3.1) with initial values $x(0)$ and $\dot{x}(0)$. A periodic solution is determined by the equations

$$x\left(\frac{2\pi}{\omega}; x(0), \dot{x}(0)\right)=x(0), \tag{3.3a}$$

$$\dot{x}\left(\frac{2\pi}{\omega}; x(0), \dot{x}(0)\right)=\dot{x}(0). \tag{3.3b}$$

If the Jacobian matrix corresponding to the above equations is singular for a periodic solution and parameter values (σ, κ), then the point (σ, κ) belongs to the bifurcation set. Also, a point belongs to the bifurcation set if two (or more) eigenvalues of the Jacobian matrix corresponding to a solution cross the imaginary axis (Hopf bifurcation). For $\alpha\to 0$ this set can be computed by averaging. The result is given in Holmes and Rand [HR1], see also Guckenheimer and Holmes [GH1]. For $\alpha>0$ numerical techniques are necessary. The bifurcation set given by the properties of the Jacobian matrix is shown in Fig. 2 for $\alpha=0.4$. It is essentially the same set as for $\alpha\to 0$. So we may use the numbering and notation as in Guckenheimer and Holmes [GH1]. The meaning of regions I, II, III and IV is the same as in [GH1]. As for $\alpha\to 0$ region IV admits a natural splitting in two regions IVa and IVb. In region IVb there is a trivial invariant torus consisting of a saddle and a sink. There is a kind of "saddle connection" via the sink with a source in the middle. The invariant curve γ, plus the source, the sink and saddle are depicted in Fig. 3 for $\alpha=2/5$, $\sigma=2/5$ and $\kappa=2/5$. The approximation of this invariant curve required $N=336$. It follows that the point $(2/5, 2/5)$ belongs to region IVb. If κ increases, the distance between the source and the saddle diminishes rapidly. The Poincaré map pushes vertices out of the neighborhood of the saddle. In order to avoid too large a distance between successive vertices, the algorithm then inserts quite a few additional vertices in the neighborhood of the saddle. For $\kappa=0.4125$ already 608 vertices are required, most of them clustered in the neighborhood of the saddle.

In Fig. 4 we are in region IVa; invariant curves are drawn for $\alpha=2/5$, $\sigma=0.55$ and nine different values of κ. The value of κ increases from background to foreground. The x-axis is horizontal; the y-axis is positive upward. Part of the x, κ-plane is shown in perspective, namely the rectangle $|x|\leqq 2.5$, $0.4725\leqq\kappa\leqq 0.495$. This rectangle is shown twice. On the lower rectangle we show the projections on the x, κ-plane. At the edge of the plane (foreground) we have $\kappa=0.495$. For this value of κ there is one sink (on

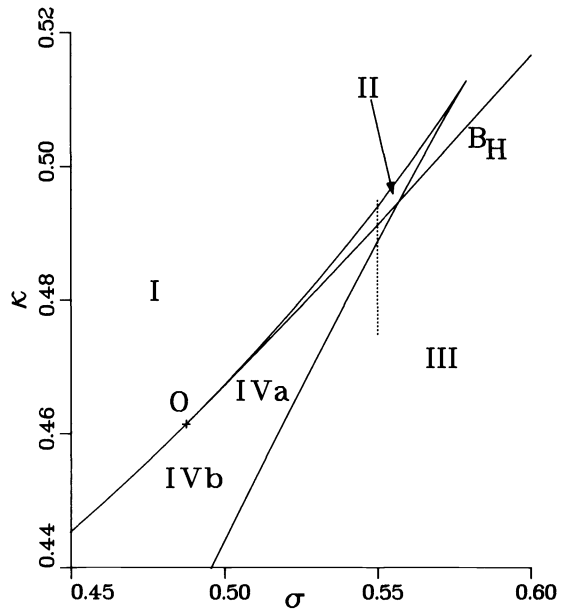


FIG. 2. *Part of the* σ, κ*-plane*; *cf.* (3.2) *for* $\alpha = 0.4$. *The meaning of regions* I–IV *is the same as in Guckenheimer and Holmes* [GH1]. *On the curve* B_H *starting in the point O a Hopf bifurcation takes place. In region* III *there is an invariant torus. The dotted line at* $\sigma = 0.55$ *indicates the region of interest in Figs.* 4 *and* 5.

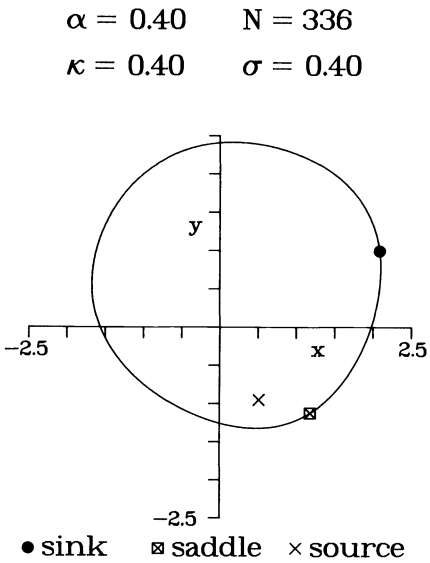


FIG. 3. *Intersection of the invariant torus with* $\mathbb{R}^2$ *for the differential equation* (3.1). *In fact, this invariant curve consists of two parts, both of them connecting the saddle with the sink. The computation required* $N = 336$ *vertices in the polygon.*

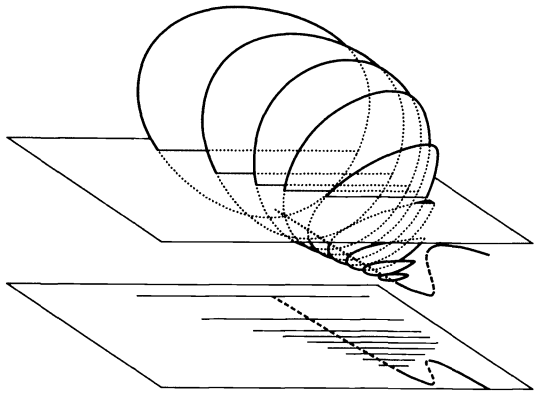


FIG. 4. *Invariant curves for the van der Pol equation* (3.1) *for* $\alpha = 2/5$, $\sigma = 0.55$. *The rectangle* $|x| < 2.5$, $0.4725 < \kappa < 0.495$ *is shown twice. The x-axis is horizontal, the y-axis is vertical, and the* κ*-axis points out of the plane. The lower rectangle shows the projections onto the x,* κ*-plane. The invariant curves are shown for* κ*-values given in* § 3. *Together with the invariant curve we have shown the fixed points of the Poincaré map. There are sinks (solid lines) connected by a saddle, and a source (within the invariant curves).*

the solid line). For $\kappa = 0.494031\cdots$ there is a first bifurcation; a saddle (dashed line) and a sink emerge. Then, for $\kappa = 0.491281\cdots$ one of the sinks becomes a source and a stable invariant curve appears. Between $\kappa = 0.491281\cdots$ and $\kappa = 0.488797\cdots$ we have a stable invariant curve, two sinks and a saddle. The first invariant curve shown corresponds with $\kappa = 0.49$. For $\kappa < 0.488797\cdots$ there is only a source (dotted line) and a stable invariant curve. In Fig. 4 we show the computed approximations to the invariant curve for $\kappa = 0.49$, 0.48875, 0.48750, 0.48625, 0.48500, 0.48375, 0.48250, 0.48000 and 0.47500. The rapid increase in the size of the curve is apparent. In order to better locate the invariant curves in relation to the x, y-coordinates, we have drawn them again in the x, y-plane in Fig. 5. In fact, each curve is a Jordan curve, and in this range the interior regions form a nicely ordered set (by set-inclusion).

The algorithm performed well for this problem. Although it is based on attraction of an invariant curve, it is able to approximate curves close to a bifurcation. However, in all instances the algorithm requires a good initial guess for the first iterate (polygon).

As a second example we consider the delayed logistic map

$$(x, y) \to \Phi(x, y) = (y, \lambda y(1-x)). \tag{3.4}$$

In this case we ask for an invariant curve of the map Φ. Aronson et al. [Ar1] consider this map in great detail, and Chan [Ch1] uses this map to illustrate his methods. For the parameter values $\lambda > 2.177$ the invariant curve is still topologically a circle, but no longer differentiable (cf. [Ar1]). The results in Chan "become twisted before $\lambda = 2.16$." Figure 6 shows the result obtained by the method of § 2, $\lambda = 2.18$. Also shown are the fixed points of the seventh iterate of the map Φ: a source within in the curve, seven sinks and saddles on the curve, and a saddle in the origin. The interior of the invariant curve is no longer convex. The fixed points of Φ^7 are computed independently by the Newton method.

4. Conclusions. In this paper we have presented a new algorithm for the numerical approximation of an invariant curve. The algorithm depends essentially on the stability

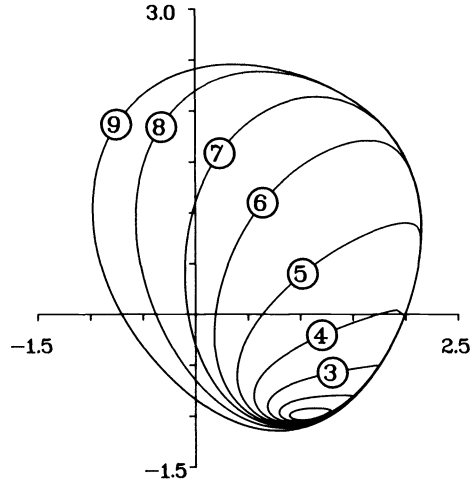


FIG. 5. *Projection on the x, y-plane (horizontal x-direction, vertical y-direction) of the invariant curves shown in Fig. 4. The number identifies the curve by its κ value. The successive κ-values are (from 1 to 9)* 0.49, 0.48875, 0.48750(→3), 0.48625(→4), 0.485(→5), 0.48375(→6), 0.4825(→7), 0.48(→8) *and* 0.475(→9). *The numbers 1 and 2 are not drawn because of lack of space.*

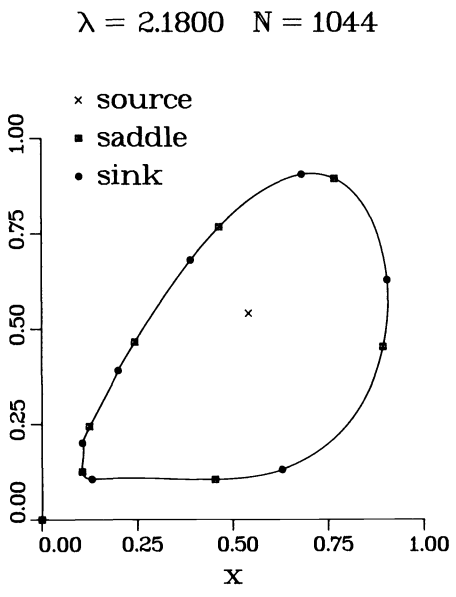


FIG. 6. *Invariant curve for the delayed logistic map* (3.4), $\lambda = 2.18$. *The sinks and saddles are fixed points of the seventh iterate of the map. The source in the middle and the saddle in the origin are fixed points of the map itself.*

properties of the curve. The algorithm compares favorably with the (second) Chan method [Ch1]. The K-A-S-P methods (cf. [K1]) are promising, but they still lack a mesh selection algorithm. The earlier algorithms by Jean [J1] and Thoulouze-Pratt and Jean [TJ1] are usually not accurate enough, but, being cheap, they may provide a good initial guess.

Finally, a remark about the use of a truncated Fourier series seems in order. Chan's second method [Ch1] uses a truncated Fourier series, and he reports unpleasant

behavior for more difficult cases. I have had a similar experience. In one of my first attempts the invariant curve γ is approximated by a trigonometric polynomial $\theta \to \gamma_N(\theta)$. We then minimize the sum of the distances

$$\sum_{i=1}^{N} d(\mathbf{P}\gamma_N(\theta_i); \gamma_N)^2 \tag{4.1}$$

with $\theta_i = 2\pi i/N$. Here we use the distance function $d(\cdot; \gamma_N)$ as in § 2. The minimization is done by a quasi-Newton method. Every now and then the points $\gamma(\theta_i)$ are examined and possibly redistributed in order to keep them nicely distributed along the curve. Unfortunately, this algorithm is rather expensive—in particular the computation of the distance function d is time-consuming—and not very reliable. For instance, this algorithm works fine for the van der Pol equation with $\alpha = 2/5$, $\sigma = 0.55$ and $\kappa \leqq 0.475$. It fails for the curves drawn in Fig. 4 by giving wrong answers. The global character of a truncated Fourier series approximation might be the cause of these failures.

Acknowledgments. I am grateful to Dr. S. A. van Gils for asking me "Can you approximate an invariant torus?", and to Dr. J. Sanders for many discussions on bifurcation theory. I am also grateful to a referee who made the work of Chan [Ch1] available to me, and who gave me the reference to the work of Kevrekidis et al. [K1].

REFERENCES

[Ar1] D. G. ARONSON, M. A. CHORY, G. R. HALL AND R. P. MCGEEHEE, *Bifurcations from an invariant circle for two-parameter families of maps of the plane*, Comm. Math. Phys., 83 (1982), pp. 303-354.

[Ch1] TZE NGON CHAN, *Numerical bifurcation analysis of simple dynamical systems*, thesis (unpublished), Dept. of Computer Science, Concordia Univ., Montreal, Canada.

[CL1] E. A. CODDINGTON AND N. LEVINSON, *Theory of Ordinary Differential Equations*, McGraw-Hill, New York, 1955.

[GH1] J. GUCKENHEIMER AND PH. HOLMES, *Nonlinear Oscillations, Dynamical Systems and Bifurcations of Vector Fields*, Springer-Verlag, New York, 1983.

[H1] J. K. HALE, *Ordinary Differential Equations*, Wiley-Interscience, New York, 1969.

[HR1] P. J. HOLMES AND D. A. RAND, *Bifurcations of the forced van der Pol oscillator*, Quart. Appl. Math., 35 (1978), pp. 495-509.

[J1] M. JEAN, *Sur la méthode des sections pour la recherche de certaines solutions presque périodiques de systèmes forcés périodiquement*, Internat. J. Non-Linear Mech., 15 (1980), pp. 367-376.

[K1] I. G. KEVREKIDIS, R. ARIS, L. D. SCHMIDT AND S. PELIKAN, *Numerical computations of invariant circles of maps*, Phys. D, 16 (1985), pp. 243-251.

[L1] N. LEVINSON, *Small periodic perturbations of an autonomous system with a stable orbit*, Ann. Math., 52 (1950), pp. 727-738.

[SG1] L. SHAMPINE AND M. K. GORDON, *Computer Solution of Ordinary Differential Equations*, Freeman, San Francisco, 1975.

[TP1] E. THOULOUZE-PRATT, *Existence theorem of an invariant torus of solutions to a periodic differential system*, Nonlinear Anal., 5 (1981), pp. 195-202.

[TJ1] E. THOULOUZE-PRATT AND M. JEAN, *Analyse numérique du comportement d'une solution presque périodique*, Internat. J. Non-Linear Mech., 17 (1982), pp. 319-326.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 6, November 1987

© 1987 Society for Industrial and Applied Mathematics
007

AN INVESTIGATION INTO THE STABILITY PROPERTIES OF SEMI-IMPLICIT BLENDED FORMULAE*

C. A. ADDISON† AND P. M. HANSON†

Abstract. Implementations of the blended Adams-Moulton and backward differentiation formulae for stiff oscillatory ordinary differential equations, based on perfect square approximation matrices, are considered. The stability of semi-implicit blended formulae is investigated together with the unreliability of certain perfect square implementations of the implicit formulae.

Key words. stiff oscillatory ordinary differential equations, blended formulae, absolute stability

AMS(MOS) subject classification. 65LO5

1. Introduction. Blended Adams-Moulton and backward differentiation formulae have been recommended by Skeel and Kong [11] for stiff differential equations, particularly stiff oscillatory ordinary differential equations whose Jacobian matrices have large eigenvalues near the imaginary axis. For stiff oscillatory problems the blended formulae have more attractive regions of absolute stability than the backward differentiation formulae, which are the more traditional formulae for solving stiff problems.

It has been observed by Gaffney [4], Berzins [2] and the authors that certain implementations of the blended formulae do not always perform as well as may be expected on some stiff oscillatory problems. At times, there appear to be stability difficulties. To solve the implicit blended formula, Skeel and Kong [11] use a perfect square approximation $(I - chJ)^2$ to the iteration matrix $(I - (\gamma\alpha + \beta)hJ + \gamma h^2 J^2)$, where J is an approximation to the Jacobian matrix, h is the step-size, and c, α, β and γ are formula-dependent constants. If the implicit formula is satisfied, the stability of the method is that of the implicit formula and it is not affected by the choice of c. If premature acceptance of a solution of the algebraic equations occurs on a number of successive steps due to unreliable convergence criteria, then the implicit formula is not satisfied and the stability characteristics are no longer governed by just the underlying implicit formula—they depend also on the details of the iteration.

To obtain some understanding of the effect on stability of failing to satisfy the implicit formula, this paper considers *semi-implicit blended formulae* that are produced by applying a *fixed* number m of iterations using the perfect square approximation $(I - chJ)^2$ to the iteration matrix. In this case, the stability is dependent upon the choice of c and the number of iterations m, in addition to the underlying implicit formula.

A stability analysis of the semi-implicit formulae is performed, and the absolute stability regions are determined and compared with those of the implicit formulae. With c chosen as in Skeel and Kong [11], it is shown, when only a few iterations are performed, that the stability regions are considerably less favorable than those of the implicit formulae. The stability regions can be bounded for orders greater than three, and in some cases consist of separated regions.

* Received by the editors September 3, 1985; accepted for publication (in revised form) September 26, 1986. This work was supported by Chr. Michelsen Institute and the Natural Sciences and Engineering Research Council of Canada.

† Centre for Computer Science, Chr. Michelsen Institute, Fantoftvegen 38, N-5036 Fantoft, Bergen, Norway.

Alternate semi-implicit formulae, obtained by choosing c to give the "best" possible regions of stability for $m = 1$, are considered. Some improvement on stability is achieved. Further improvements are obtained by considering semi-implicit formulae with perfect square matrices $(aI - bhJ)^2$, where a and b are chosen to give the "best" possible regions of stability.

The blended formulae and semi-implicit blended formulae are presented in § 2. The stability analysis of the semi-implicit formulae is performed in § 3. The stability regions and numerical results are presented in § 4, followed by conclusions in § 5.

2. Blended formulae and semi-implicit blended formulae. The $(k+1)$st order blended formula considered by Skeel and Kong [11] for systems of ordinary differential equations of the form

$$y' = f(t, y), \qquad y(t_0) = y_0 \tag{2.1}$$

is

$$y_i - y_{i-1} - h \sum_{j=0}^{k} \beta_j f_{i-j} - \gamma^{(k)} hJ \left\{ \sum_{j=0}^{k} \alpha_j y_{i-j} - h f_i \right\} = 0, \tag{2.2}$$

where the $\{B_j\}$ are the coefficients of the $(k+1)$st order Adams-Moulton formula, the $\{\alpha_j\}$ are the coefficients of the kth order backward differentiation formula, J is a Jacobian approximation, and $\gamma^{(k)}$ is chosen to maximise the angle α for which the formula is $A(\alpha)$-stable. (A formula is $A(\alpha)$-stable if every numerical solution of $y' = \lambda y$ tends to zero for $|\mathrm{Arg}\,(-\lambda)| < \alpha$. If $\alpha = \pi/2$, the formula is said to be A-stable.)

In the Nordsieck representation an approximation $\mathbf{y}_i = (y_i, hy_i', \cdots, (h^k/k!)y_i^{(k)})^T$, to the scaled derivatives $(y(t_i), hy'(t_i), \cdots, (h^k/k!)y^{(k)}(t_i))^T$, of the solution, is generated by a prediction

$$\mathbf{p}_i = P\mathbf{y}_{i-1}, \tag{2.3}$$

followed by a correction

$$\mathbf{y}_i = \mathbf{p}_i + \boldsymbol{l}\Delta_i, \tag{2.4}$$

where P is the Pascal triangle matrix given by

$$P_{ij} = \binom{j}{i}, \qquad 0 \leqq i, j \leqq k, \tag{2.5}$$

$\mathbf{p}_i = (p_i, hp_i', \cdots, (h^k/k!)p_i^{(k)})^T$, $\boldsymbol{l} = (l_0, l_1, \cdots, l_k)^T$, and Δ_i is defined so that $y_i' = f(t, y_i)$, that is, Δ_i satisfies the implicit equation

$$hp_i' + l_1\Delta_i - hf(t_i, p_i + l_0\Delta_i) = 0. \tag{2.6}$$

It is shown in Skeel [13] that the Nordsieck representation with

$$l_j = a_j^{(k)} I - \gamma^{(k)} b_j^{(k)} hJ, \qquad j = 0, 1, 2, \cdots, k \tag{2.7}$$

is equivalent to the $(k+1)$st order blended formula when

$$a_j^{(k)} = \frac{1}{j(k-1)!}\begin{bmatrix} k \\ j \end{bmatrix}, \qquad j = 1, 2, \cdots, k,$$

$$a_0^{(k)} = -\sum_{j=1}^{k+1} (-1)^j a_j^{(k+1)},$$

$$b_j^{(k)} = \frac{1}{k!}\begin{bmatrix} k+1 \\ j+1 \end{bmatrix}, \qquad j = 0, 1, \cdots, k,$$

with [ ] denoting Stirling numbers of the first kind (Knuth [10]). Notice that $a_0^{(k)}$ and $b_1^{(k)}$ are in fact equivalent to β_0 and α_0 of (2.2), and $a_1^{(k)}=1$, $b_0^{(k)}=1$.

The Newton iteration for the implicit equation (2.6) satisfied by Δ_i is

$$\Delta_i^{[0]}=0, \qquad \Delta_i^{[r]}=\Delta_i^{[r-1]}+\left(l_1-h\frac{\partial f_i^{[r-1]}}{\partial y}l_0\right)^{-1}(hf_i^{[r-1]}-(hp_i'+l_i\Delta_i^{[r-1]})), \tag{2.8}$$

$r=1,2,\cdots$, where for a function the superscript $[r-1]$ denotes evaluation at $(t_i, p_i+l_0\Delta_i^{[r-1]})$. In practice, a recent Jacobian approximation J is used in place of $\partial f_i^{[r-1]}/\partial y$ in (2.8),

$$\Delta_i^{[r]}=\Delta_i^{[r-1]}+(l_1-hJl_0)^{-1}(hf_i^{[r-1]}-(hp_i'+l_1\Delta_i^{[r-1]})). \tag{2.9}$$

From (2.7) and the remarks following (2.7),

$$\begin{aligned} l_1-hJl_0&=(a_1^{(k)}I-\gamma^{(k)}b_1^{(k)}hJ)-hJ(a_0^{(k)}I-\gamma^{(k)}b_0^{(k)}hJ)\\ &=(I-\gamma^{(k)}\alpha_0hJ)-hJ(\beta_0I-\gamma^{(k)}hJ)\\ &=I-(\gamma^{(k)}\alpha_0+\beta_0)hJ+\gamma^{(k)}h^2J^2, \end{aligned} \tag{2.10}$$

with α_0 and β_0 from the formula (2.2). That is, the iteration matrix is a quadratic polynomial in hJ.

Since forming J^2 is expensive and usually destroys sparseness, the quadratic iteration matrix (2.10) is approximated by a perfect square $(I-c^{(k)}hJ)^2$ and the iteration (2.9) becomes

$$\Delta_i^{[r]}=\Delta_i^{[r-1]}+(I-c^{(k)}hJ)^{-2}(hf_i^{[r-1]}-(hp_i'+l_1\Delta_i^{[r-1]})). \tag{2.11}$$

The corresponding iteration for the corrector equation is

$$\mathbf{y}_i^{[r]}=\mathbf{y}_i^{[r-1]}+\boldsymbol{l}(I-c^{(k)}hJ)^{-2}(hf_i^{[r-1]}-hy_i^{[r-1]\prime}), \tag{2.12}$$

with $y_i^{[0]}=p_i$. Iteration (2.11) converges linearly for the system $y'=Jy$, whereas (2.9) requires only one iteration. Therefore the value of $c^{(k)}$ is chosen so that convergence of (2.11) is rapid for this special case. In practice it is the eigenvalues with negative real parts that may cause convergence problems. When the iteration scheme (2.11) is applied to the scalar test equation

$$\begin{aligned} &y'(t)=\lambda y(t), \qquad y(t_0)=y_0,\\ &\operatorname{Re}(\lambda)\leqq 0 \end{aligned} \tag{2.13}$$

it becomes

$$\begin{aligned} \Delta_i^{[r]}=(1-c^{(k)}h\lambda)^{-2}&((\gamma^{(k)}\alpha_0+\beta_0-2c^{(k)})h\lambda+((c^{(k)})^2-\gamma^{(k)})h^2\lambda^2)\Delta_i^{(r-1)}\\ &+(1-c^{(k)}h\lambda)^{-2}h(\lambda p_i-p_i'). \end{aligned}$$

The rate of convergence depends on the coefficient of $\Delta_i^{(r-1)}$, and so $c^{(k)}$ is chosen to minimise

$$\phi(c^{(k)})=\max_{\operatorname{Re}(\lambda)\leqq 0}|(1-c^{(k)}h\lambda)^{-2}((\gamma^{(k)}\alpha_0+\beta_0-2c^{(k)})h\lambda+((c^{(k)})^2-\gamma^{(k)})h^2\lambda^2)|.$$

The values of $c^{(k)}$ together with the corresponding $\phi(c^{(k)})$ are given in Skeel and Kong [11], and for convenience are reproduced here for orders 2 to 7 in Table 1.

TABLE 1
Values of $c^{(k)}$ and $\phi(c^{(k)})$.

k	Order	$c^{(k)}$	$\phi(c^{(k)})$
1	2	$1-(0.5)^{1/2}$	0*
2	3	.33750	.11847
3	4	.33354	.11618
4	5	.34273	.11398
5	6	.31691	.09956
6	7	.29930	.08945

* For $k=1$ the iteration matrix is a perfect square.

A *semi-implicit blended formula* is obtained by applying a *fixed* number m, of iterations (2.11), rather than iterating to convergence. In this case the stability of the formula depends on the choice of $c^{(k)}$ and the number of iterations m, as well as the underlying implicit formula.

3. Stability analysis of the semi-implicit formulae. The absolute stability properties of the semi-implicit formulae are investigated by considering linear equations with constant coefficient matrices. It is assumed that the approximate Jacobian is in fact the true Jacobian. This is a more favorable situation than appears in practice; however, the stability analysis is still useful and interesting.

Consider the test equation (2.13). In this case the semi-implicit formula may be expressed as

$$(3.1)\quad (1-ch\lambda)^2\mathbf{y}_i^{[r]} = (1-ch\lambda)^2\mathbf{y}_i^{[r-1]} + l(h\lambda y_i^{[r-1]} - hy_i^{[r-1]\prime}), \qquad r=1,2,\cdots,m,$$

with $\mathbf{y}_i^{[0]} = \mathbf{p}_i$, and where $\mathbf{y}_i^{[m]}$ is the final approximation $\mathbf{y}_i$. The dependence on k has been suppressed to simplify the notation.

Let

$$\mathbf{x}_i = \left(y_i^{[m]}, hy_i^{[m]\prime}, \cdots, \frac{h^k}{k!}y_i^{[m](k)}, y_i^{[m-1]}, hy_i^{[m-1]\prime}, \cdots, \right.$$
$$\left. \frac{h^k}{k!}y_i^{[m-1](k)}, \cdots, y_i^{[0]}, hy_i^{[0]\prime}, \cdots, \frac{h^k}{k!}y_i^{[0](k)} \right)^T,$$

where $y_i^{[r](j)}$ is the jth derivative of $y_i^{[r]}$; then $\mathbf{x}_i$ contains the information that results from the ith step, together with all information from previous steps that is required to advance from t_i to t_{i+1}. Hence $\mathbf{x}_i$ satisfies

$$(3.2)\qquad T\mathbf{x}_i = S\mathbf{x}_{i-1},$$

with $\mathbf{x}_0$ given and

$$T = \begin{bmatrix} (1-ch\lambda)^2 I & M & & 0 \\ & (1-ch\lambda)^2 I & \ddots & \\ & & \ddots & M \\ 0 & & & I \end{bmatrix}, \qquad S = \begin{bmatrix} 0 & & & 0 \\ & \ddots & & \\ & & \ddots & \\ P & & & 0 \end{bmatrix},$$

$$M = \begin{bmatrix} -(1-ch\lambda)^2 - l_0 h\lambda & l_0 & 0 & \cdots & 0 \\ -l_1 h\lambda & -(1-ch\lambda)^2 + l_1 & 0 & \cdots & 0 \\ -l_2 h\lambda & l_2 & -(1-ch\lambda)^2 & & 0 \\ \vdots & \vdots & & \ddots & \\ -l_k h\lambda & l_k & 0 & & -(1-ch\lambda)^2 \end{bmatrix}$$

and

$$P = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ & 1 & 2 & 3 & \cdots & k \\ & & 1 & 3 & \cdots & \cdot \\ & & & 1 & \cdots & \cdot \\ & & & & \ddots & \vdots \\ 0 & & & & & 1 \end{bmatrix}$$

T and S are square matrices of dimension $(m+1)(k+1)$ with blocks I, M and P of dimension $(k+1)$. The first m blocks of (3.2) represent the m iterations of the corrector and the last block represents the predictor step.

The method is stable if and only if the eigenvalues of the generalised eigenvalue problem

$$(S-\omega T)\mathbf{z} = \mathbf{0}, \tag{3.3}$$

have modulus less than or equal to one with eigenvalues of modulus one possessing a complete set of eigenvectors (Gear [6]).

Now

$$S-\omega T = \begin{bmatrix} R & -\omega M & & 0 \\ & R & -\omega M & \\ & & \ddots & \ddots \\ & & & R \quad -\omega M \\ P & & & -\omega I \end{bmatrix},$$

with $R = -\omega(1-ch\lambda)^2 I$, can be factored as EF with

$$E = \begin{bmatrix} I & & & 0 & 0 \\ & \ddots & & & \vdots \\ 0 & & & I & 0 \\ PR^{-1} & \omega PMR^{-2} & \omega^2 PM^2 R^{-3} \cdots \omega^{m-1} PM^{m-1} R^{-m} & & (-1)^m R^{-m} \end{bmatrix}$$

and

$$F = \begin{bmatrix} R & -\omega M & & 0 \\ & R & -\omega M & \\ & & \ddots & \ddots \\ & & & R \quad -\omega M \\ 0 & & & C \end{bmatrix},$$

where

$$C = (-1)^m \omega^m PM^m + (-1)^{m-1} \omega R^m,$$

and therefore

$$\det(S-\omega T)=(-1)^{m(k+1)}\det(C).$$

To determine $\det(C)$, C may be written as

$$C=(-1)^{m-1}R^m(\omega I-PB),$$

where

$$B=\omega^m R^{-m}M^m$$

$$=\begin{bmatrix} 1-h\lambda b_0 & b_0 & 0 & \cdots & 0 \\ -h\lambda b_1 & 1+b_1 & 0 & \cdots & 0 \\ -h\lambda b_2 & b_2 & 1 & & 0 \\ \vdots & \vdots & & \ddots & \\ -h\lambda b_k & b_k & 0 & & 1 \end{bmatrix}$$

is a $(k+1)\times(k+1)$ matrix. The values b_i are given by

$$b_i=\frac{-l_i}{(1-ch\lambda)^{2m}}\left[\frac{\nu(h\lambda)^m-(1-ch\lambda)^{2m}}{\nu(h\lambda)-(1-ch\lambda)^2}\right],$$

where $\nu(h\lambda)=(1-ch\lambda)^2-(l_1-l_0h\lambda)$ is the difference between the perfect square matrix approximation and the true iteration matrix.

The matrix B can be expressed in the form

$$B=I+\mu\boldsymbol{l}(h\lambda\mathbf{e}_0^T-\mathbf{e}_1^T),$$

where

$$\mu=\frac{1}{(1-ch\lambda)^{2m}}\left[\frac{\nu(h\lambda)^m-(1-ch\lambda)^{2m}}{\nu(h\lambda)-(1-ch\lambda)^2}\right],\quad \mathbf{e}_0^T=[1,0,\cdots,0]^T,\quad \mathbf{e}_1^T=[0,1,0,\cdots,0]^T.$$

Then

$$\begin{aligned}\omega I-PB&=\omega I-P+\mu P\boldsymbol{l}(h\lambda\mathbf{e}_0^T-\mathbf{e}_1^T)\\&=(\omega I-P)(I+\mu(\omega I-P)^{-1}P\boldsymbol{l}(\mathbf{e}_1^T-h\lambda\mathbf{e}_0^T)),\end{aligned}$$

and

$$\begin{aligned}\det(\omega I-PB)&=(\omega-1)^{k+1}(1+\mu(\mathbf{e}_1^T-h\lambda\mathbf{e}_0^T)(\omega I-P)^{-1}P\boldsymbol{l})\\&=(\omega-1)^{k+1}(1+\mu(\mathbf{e}_1^T-h\lambda\mathbf{e}_0^T)(-I+\omega(\omega I-P)^{-1})\boldsymbol{l})\\&=(\omega-1)^{k+1}(1+\mu(h\lambda l_0-l_1))+\mu\omega(\rho(\omega)-h\lambda\sigma(\omega))\end{aligned}$$

using the fact (Skeel and Kong [11]) that

$$\begin{aligned}(\omega-1)^{k+1}\mathbf{e}_1^T(\omega I-P)^{-1}\boldsymbol{l}&=\rho(\omega)=\rho^{AM}(\omega)-\gamma h\lambda\rho^{BD}(\omega),\\(\omega-1)^{k+1}\mathbf{e}_0^T(\omega I-P)^{-1}\boldsymbol{l}&=\sigma(\omega)=\sigma^{AM}(\omega)-\gamma h\lambda\sigma^{BD}(\omega),\end{aligned}$$

with

$$\rho^{AB}(\omega)=\omega^k-\omega^{k-1},\quad \rho^{BD}(\omega)=\sum_{j=0}^{k}\alpha_j\omega^{k-j},\quad \sigma^{AB}(\omega)=\sum_{j=0}^{k}\beta_j\omega^{k-j},\quad \sigma^{BD}(\omega)=\omega^k.$$

From the above it follows that

$$\det(S-\omega T) = (-1)^{(m-1)(k+1)}\omega^{m(k+1)}(1-ch\lambda)^{2m}[(\omega-1)^{k+1}(1+\mu(h\lambda l_0 - l_1)) + \mu\omega(\rho(\omega)-h\lambda\sigma(\omega))]$$

$$= (-1)^{(m-1)(k+1)}\omega^{m(k+1)}(1-ch\lambda)^{k+1} \cdot \left[(\omega-1)^{k+1}\nu(h\lambda)^m + \left[\frac{\nu(h\lambda)^m-(1-ch\lambda)^{2m}}{\nu(h\lambda)-(1-ch\lambda)^2}\right]\omega(\rho(\omega)-h\lambda\sigma(\omega))\right],$$

and it is the polynomial

$$Q(\omega, h\lambda) = (\omega-1)^{k+1}\nu(h\lambda)^m + \left[\frac{\nu(h\lambda)^m-(1-ch\lambda)^{2m}}{\nu(h\lambda)-(1-ch\lambda)^2}\right]\omega(\rho(\omega)-h\lambda\sigma(\omega)) \tag{3.4}$$

that determines the stability properties of the semi-implicit method. That is, $Q(\omega, h\lambda)$ is the stability polynomial whose roots for a given $h\lambda$ value must be less than or equal to one in modulus, with roots of modulus one corresponding to eigenvalues possessing a complete set of eigenvectors.

The polynomial $Q(\omega, h\lambda)$ will be said to be *stable* if its roots satisfy the above stability requirements. The set of $h\lambda$ values for which $Q(\omega, h\lambda)$ is stable is called the *region of absolute stability.* This region will depend on the choice of perfect square approximation, that is the choice of c and the number of iterations m.

4. Stability regions and numerical results. Using the stability polynomial $Q(\omega, h\lambda)$, the regions of absolute stability have been determined for the semi-implicit blended formulae with $m=1$, 2, 3 and orders 3 to 12. Whereas all the underlying blended formulae have unbounded regions of stability, with the 2nd, 3rd and 4th order formulae being A-stable, the semi-implicit formulae result in regions which, for $m=1$, are bounded for all orders greater than 3, and separated for all orders greater than 6. As the order increases, still with $m=1$, the regions of stability become smaller and those for the higher orders consist of two extremely short and narrow regions around -8.0 and 0.0. As the number of iterations increases, the regions tend towards those of the underlying implicit blended formulae, but several iterations are required before the regions are similar, particularly for the higher order formulae.

The values of $\pi/2-\alpha$, where α is the largest angle for which the formulae are $A(\alpha)$-stable, and the maximum distance $d_{\max}$, that the stability boundaries extend into the left-half plane, are given for orders 3 to 7 in Table 2. The stability regions are shown in Figs. 1-4.

In an attempt to improve the stability regions of the semi-implicit blended formulae, alternative perfect square approximations were considered. First, the value of c which gave rise to the "best" region of stability for $m=1$ was determined using the code NLPQL (see Hock [9]) to maximise the angle α, and to minimise the distance $d_{\max}$. The results for orders 3 and 4 are given in Table 3.

For both orders the value of $\phi(c)$ has increased, but not drastically. For order 3 the stability region has improved slightly, A-stability being achieved after 2 iterations, while for order 4 an unbounded region of stability is obtained after only 1 iteration, and A-stability after 3 iterations.

To improve the stability regions further, perfect square approximations of the form $(aI-bhJ)^2$ were considered, as in Addison and Hanson [1]. The values of a and b were determined as for c above, except that zero stability had to be included as a constraint since $a \neq 1$. The results for orders 3 and 4 are given in Table 4.

For order 3, A-stability is achieved for $m=1$ with a smaller $\phi(a, b)$ value than the original $\phi(1, c)$. For order 4 the "best" a and b values give rise to a stability

TABLE 2
$\pi/2-\alpha$ and $d_{\max}$ for the blended formulae and perfect square iterations.

Order	Method	$\pi/2-\alpha$	$d_{\max}$	Figure No.
3	$m=1$	0.09356	0.22322	1
	$m=2$	0.00004	0.337×10^{-4}	
	$m=3$	A-stable		
	Underlying blended formula	A-stable		
4	$m=1$	Bounded	27.15101	2
	$m=2$	0.02164	0.04922	
	$m=3$	0.00001	0.201×10^{-6}	
	Underlying blended formula	A-stable		
5	$m=1$	Bounded	11.85251	Similar to
	$m=2$	0.07966	0.18785	Fig. 3
	$m=3$	0.01271	0.02151	
	Underlying blended formula	0.01019	0.01627	
6	$m=1$	Bounded	9.33994	3
	$m=2$	0.19208	0.69067	
	$m=3$	0.06209	0.15053	
	Underlying blended formula	0.05286	0.12197	
7	$m=1$	Separated	8.35840	4
	$m=2$	0.34863	1.63437	
	$m=3$	0.13979	0.40902	
	Underlying blended formula	0.12319	0.34456	

polynomial $Q(\omega, h\lambda)$ which is not quite A-stable after 1 iteration, but is A-stable after 2. The value of $\phi(a, b)$ is reasonable.

For order 5 the "best" values of a and b required at least 3 iterations to yield a $Q(\omega, h\lambda)$ with properties akin to those of the stability polynomial of the underlying blended formula, and for higher orders the regions all remained bounded for $m=1$ as a and b varied.

It should be noted that the predictor given by (2.3) is of order k, and when $a \neq 1$ the order of accuracy of the method may be reduced, depending on how many iterations are performed. The orders given in Table 4 correspond to those of the underlying blended formula, as in previous tables.

Most blended formulae codes are based on Gear's code DIFSUB [7], or descendants such as LSODE (Hindmarsh [8]). A maximum number of iterations is allowed per step, usually two or three, and an approximation may be accepted after only one iteration. If the convergence criteria are unreliable and premature acceptance of a solution of the implicit equation occurs on a number of successive steps, then stability difficulties, such as can occur with semi-implicit blended formulae, may arise on some stiff problems, particularly oscillatory problems.

To illustrate the difficulties that can arise, the following codes were considered.

(i) BLSODE: a variable-step, variable-order code based on LSODE modified to implement the blended formulae as described in § 2, with maximum order restricted to 4 so that the underlying blended formulae are A-stable.

(ii) BLSMOD: as in (i) but with the iteration matrix refactored whenever the step-size is changed. (When the step-size is changed from $\bar{h}$ to $h=\xi\bar{h}$, the use of the old matrix $(I-c\bar{h}J)^2$, rather than the new matrix $(I-chJ)^2$, is equivalent to using $(I-\bar{c}hJ)^2$ where $\bar{c}=c/\xi$, rather than $(I-chJ)^2$, at least when J is constant.)

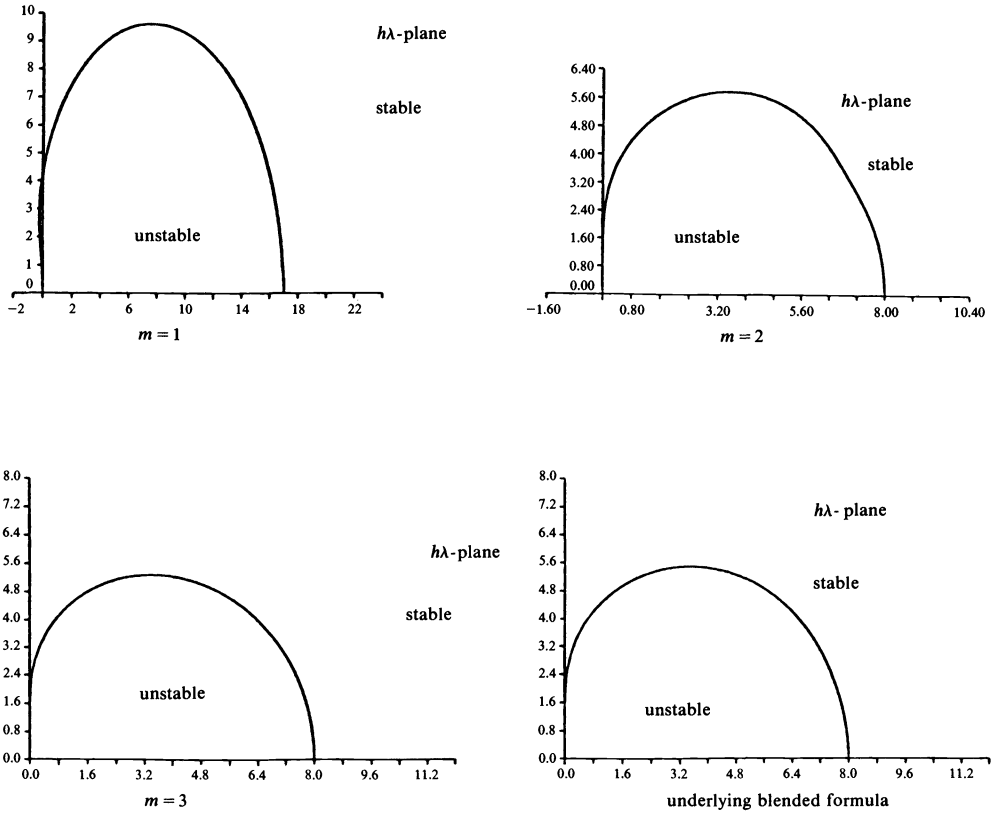


FIG. 1. *Stability regions for the 3rd order formulae.*

(iii) BLEND: a variable-step, variable-order code based on Gear's code DIFSUB modified to implement the blended formulae as in § 2.

(iv) BLEMOD: as in (iii) but with the iteration matrix refactored whenever the step-size is changed.

(v) BCOMP: as in (ii) but with the true iteration matrix (2.10) instead of a perfect square approximation. This quadratic matrix is factored into complex conjugate parts and the corrector equation solved without explicitly forming J^2 (see Enright [3]). The stability properties are those of the underlying blended formulae.

BLSODE and BLEND are codes in the NASTI library [5].

Each code was applied to the problem

$$\mathbf{y}'(t) = \begin{bmatrix} 0 & I \\ K & 0 \end{bmatrix} \mathbf{y}(t) + \mathbf{F}(t), \quad \mathbf{y}(0) = \mathbf{0}, \quad 0 \leqq t \leqq 40, \tag{4.1}$$

where

$$K = \begin{bmatrix} k_1 + k_2 + k_5 & -k_2 & -k_5 \\ -k_2 & k_2 + k_3 + k_4 & -k_3 \\ -k_5 & -k_3 & k_3 + k_5 \end{bmatrix},$$

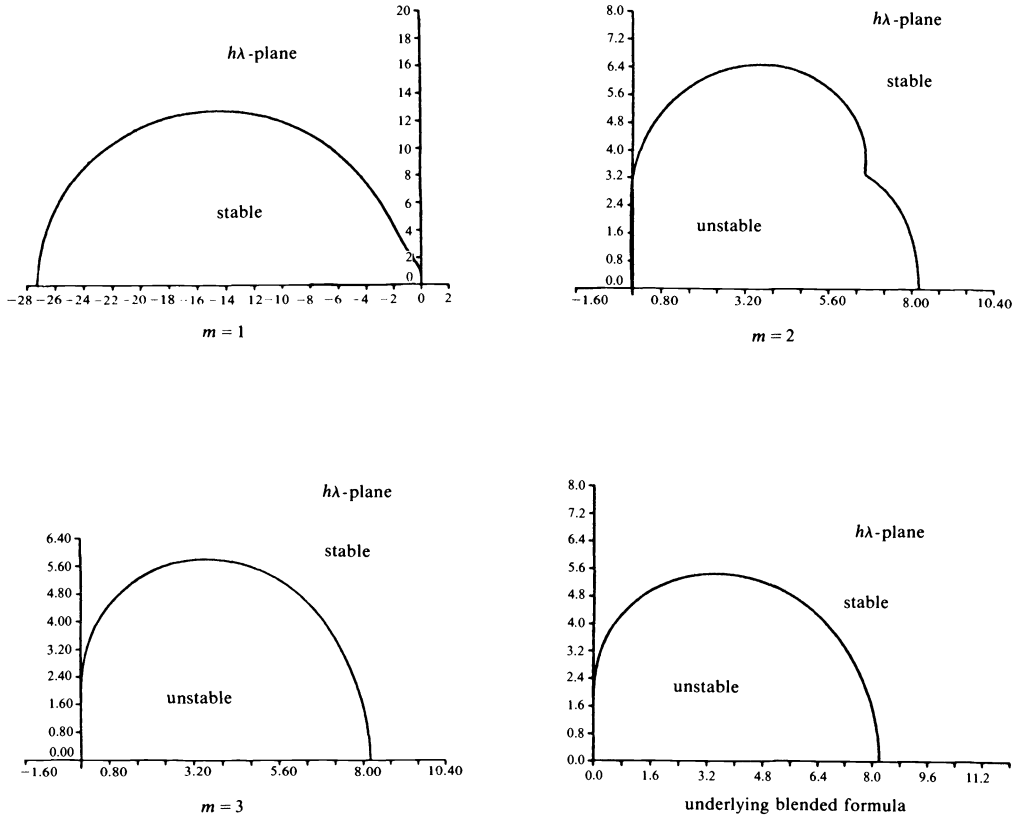


FIG. 2. *Stability regions for the 4th order formulae.*

with $k_1 = -1$, $k_2 = -100$, $k_3 = -10000$, $k_4 = -2$, $k_5 = -200$ and $\mathbf{F}(t) = (0, 0, 0, 0, 0, f(t))^T$ with

$$f(t) = \begin{cases} 2t & \text{if } 0 \leqq t < 5, \\ 20 - 2t & \text{if } 5 \leqq t < 10, \\ 0 & \text{if } 10 \leqq t \leqq 40. \end{cases}$$

For this problem the eigenvalues are approximately $\pm i, \pm 21i, \pm 140i$. Only the low frequency components are excited to any great extent by the forcing term. Statistics for each code with tolerances of 10^{-2}, 10^{-3}, and 10^{-4} (TOL) are presented in Table 5.

Whereas BCOMP has no difficulty solving the problem, both BLSODE and BLEND experience severe difficulties at the two highest tolerances, with BLSODE performing worse than BLEND. The modified codes in which the iteration matrix is refactored whenever the step-size changes also experienced difficulties, and BLEMOD was in fact worse than BLEND at the stringent tolerances. Only BCOMP achieved the order of accuracy expected.

It would appear that BLSODE and BLEND have unreliable convergence criteria, with BLSODE having the more unreliable of the two. The convergence criteria are sometimes satisfied and approximations accepted after only one or two iterations, but

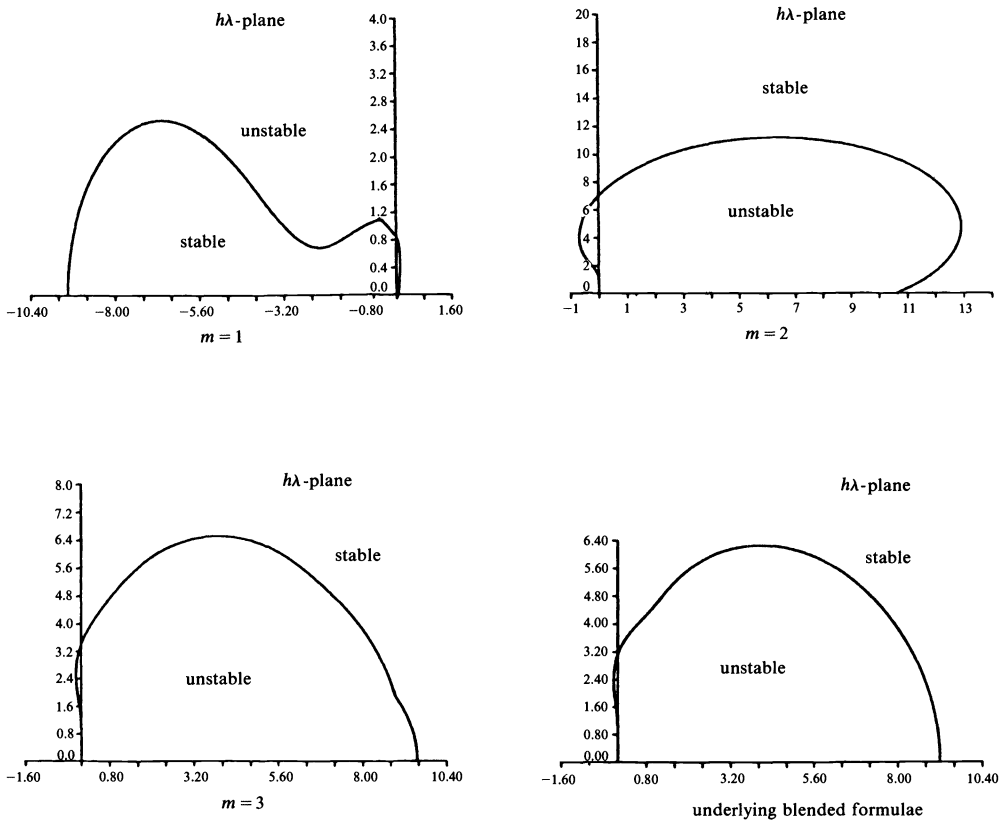


FIG. 3. *Stability regions for the 6th order formulae.*

these are not always sufficiently accurate approximations to the solution of the implicit blended formula.

To illustrate the improvement gained by using $(aI - bhJ)^2$ in place of $(I - chJ)^2$, problem (4.1) was solved again using the following codes:

(vi) BLSAB: as in (i) but with $(aI - bhJ)^2$ in place of $(I - chJ)^2$, where a and b are the optimum values shown in Table 4.

(vii) BLSMDAB: as in (vi) but with the iteration matrix refactored whenever the step-size is changed.

(viii) BLEAB: as in (iii) but with $(aI - bhJ)^2$ in place of $(I - chJ)^2$, where a and b are the optimum values shown in Table 4.

(ix) BLEMDAB: as in (viii) but with the iteration matrix refactored whenever the step-size is changed.

The results are shown in Table 6.

There is a substantial improvement in the behaviour of the codes, both in terms of work involved and accuracy achieved, particularly for the codes based on BLEND. However, the BLSODE based codes, although improved, still exhibit difficulties at the higher tolerance, and the difference between the results of these codes and those of BCOMP suggest that premature acceptance of an approximation to the solution of

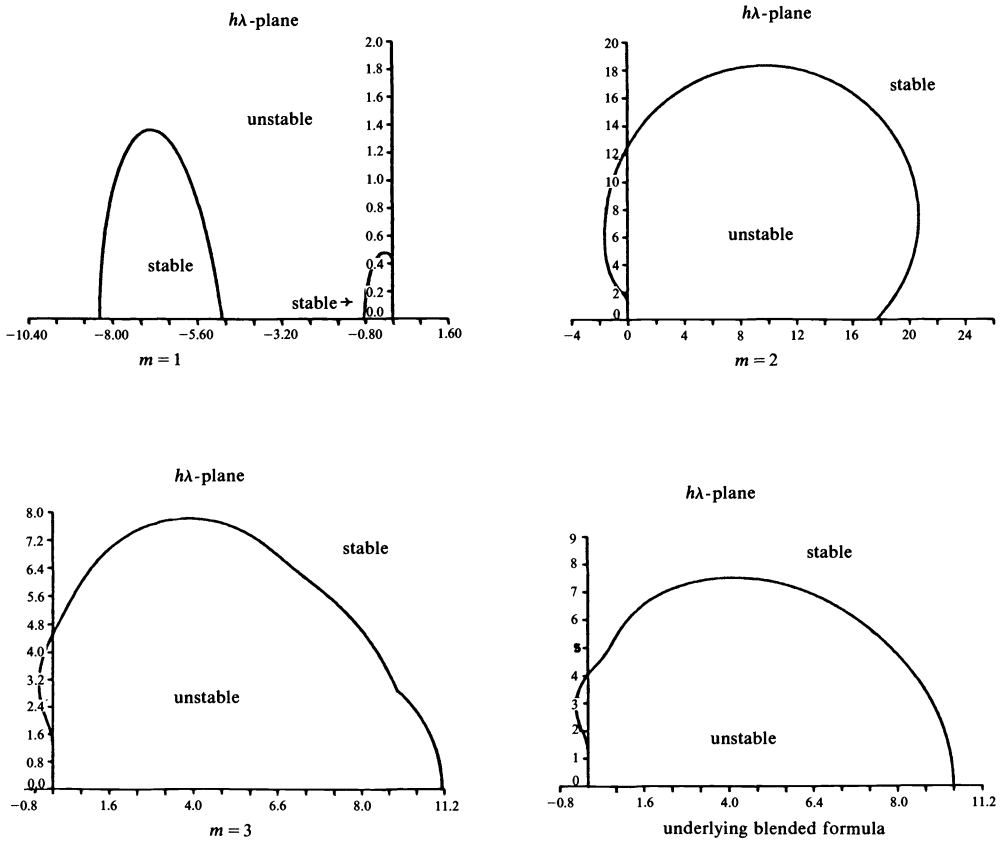


FIG. 4. *Stability regions for the 7th order formulae.*

the implicit blended formulae is still occurring, and that the attractive properties of the underlying blended formulae are still not fully exploited.

In essence, the convergence criterion used in BLSODE assumes convergence if an estimate of the next correction is sufficiently small. In BLEND convergence is assumed is *either* the current correction *or* an estimate of the next correction is sufficiently small. BLSODE allows a maximum of 3 iterations, BLEND a maximum of 2.

TABLE 3
Values of c that optimise α and d_{max} when m = 1.

Order	No. of iterations	c	$\phi(c)$	$\pi/2-\alpha$	d_{max}
3	1	0.36631	0.17878	0.08156	0.12860
	2			A-stable	
4	1	0.35983	0.17102	0.48161	1.73226
	2			0.01917	0.03161
	3			A-stable	

TABLE 4
Values of a and b that optimise α and $d_{\max}$ when $m = 1$.

Order	No. of iterations	a	b	$\phi(a, b)$	$\pi/2-\alpha$	$d_{\max}$
3	1	0.95825	0.35246	0.08904	A-stable	
4	1	0.93600	0.33810	0.14143	0.04308	0.05514
	2				A-stable	

TABLE 5
Results[1] illustrating difficulties with blended codes.

TOL	Code	No. of steps	No. of back solves	No. of matrix factorisations	End point error
10^{-2}	BLSODE	202	610	39	2.27×10^{-2}
	BLSMOD	158	462	44	2.34×10^{-2}
	BLEND	143	634	32	5.51×10^{-3}
	BLEMOD	131	434	23	3.14×10^{-2}
	BCOMP	114	136*	16*	4.10×10^{-2}
10^{-3}	BLSODE	13582	37530	1644	4.71×10^{-2}
	BLSMOD	10573	34380	2889	4.52×10^{-1}
	BLEND	1367	6500	435	1.12×10^{-2}
	BLEMOD	1952	7766	637	5.09×10^{-2}
	BCOMP	204	235*	21*	4.10×10^{-3}
10^{-4}	BLSODE	18493	52842	2512	7.71×10^{-2}
	BLSMOD	14290	45492	3590	5.34×10^{-2}
	BLEND	10337	47434	2533	1.13×10^{-2}
	BLEMOD	16150	42420	2373	5.71×10^{-2}
	BCOMP	794	1317*	275*	2.38×10^{-4}

[1] Computations performed on an IBM 4361 in double precision.
* Complex arithmetic.

TABLE 6
Results[1] obtained using $(aI - bhJ)^2$.

TOL	Code	No. of steps	No. of back solves	No. of matrix factorisations	End point error
10^{-3}	BLSAB	110	384	12	2.41×10^{-2}
	BLSMDAB	106	380	19	2.11×10^{-2}
	BLEAB	231	1010	99	4.18×10^{-3}
	BLEMDAB	133	534	21	2.85×10^{-2}
10^{-3}	BLSAB	1322	4496	255	2.19×10^{-1}
	BLSMDAB	226	772	41	5.17×10^{-3}
	BLEAB	1201	5606	731	1.29×10^{-3}
	BLEMDAB	439	1802	186	2.77×10^{-3}
10^{-4}	BLSAB	15383	68526	3008	3.28×10^{-2}
	BLSMDAB	1876	7502	563	3.40×10^{-2}
	BLEAB	1957	8926	1011	4.27×10^{-4}
	BLEMDAB	1270	5040	524	6.66×10^{-4}

[1] Computations performed on an IBM 4361 in double precision.

One possible way to improve the reliability of the convergence criteria is to augment the existing criteria with the requirement that the residual, arising from the use of the perfect square approximation instead of the quadratic iteration matrix, be sufficiently small.

From (2.11), $\Delta_i^{[r]} = \Delta_i^{[r-1]} + \delta_i^{[r-1]}$, where $\delta_i^{[r-1]}$ satisfies

$$(I - chJ)^2 \delta_i^{[r-1]} = g_i^{[r-1]}, \tag{4.2}$$

with $g_i^{[r-1]} = hf_i^{[r-1]} - (hp_i' + l_i \Delta_i^{[r-1]})$. The residual is

$$(I - (\gamma\alpha_0 + \beta_0)hJ + \gamma h^2 J^2)\delta_i^{[r-1]} - g_i^{[r-1]}. \tag{4.3}$$

This residual can be computed using only vector operations at the expense of storing one additional vector of information. Let z be the intermediate solution obtained when solving (4.2), that is, $(I - chJ)z = g_i^{[r-1]}$. Then, from (4.2),

$$J\delta_i^{[r-1]} = -\frac{1}{hc}(z - \delta_i^{[r-1]}). \tag{4.4}$$

Furthermore, using (4.4) with (4.2),

$$J^2 \delta_i^{[r-1]} = \frac{1}{h^2 c^2}(g_i^{[r-1]} + \delta_i^{[r-1]} - 2z), \tag{4.5}$$

and hence (4.3) can be computed using only vector operations, provided z is stored when solving (4.2).

Problem (4.1) was solved using the following codes with convergence assumed if the original convergence criteria are satisfied and the norm of the residual (4.3) is smaller than 10^{-2} times the local error tolerance.

(x) BLSIC: as in (i) but with improved convergence criteria.

(xi) BLSMDABIC: as in (vii) but with improved convergence criteria.

(xii) BLEIC: as in (iii) but with improved criteria and a maximum of 3 iterations instead of 2.

(xiii) BLEMDABIC: as in (ix) but with improved convergence criteria and a maximum of 3 iterations instead of 2.

The results are shown in Table 7.

TABLE 7

Results[1] *obtained using improved convergence criteria.*

TOL	Code	No. of steps	No. of back solves	No. of matrix factorisations	End point error
10^{-2}	BLSIC	112	494	14	1.36×10^{-3}
	BLSMDABIC	117	688	20	6.20×10^{-4}
	BLEIC	128	628	15	4.80×10^{-3}
	BLEMDABIC	122	706	18	4.64×10^{-3}
10^{-3}	BLSIC	739	4846	283	4.16×10^{-3}
	BLSMDABIC	365	2150	89	4.85×10^{-4}
	BLEIC	793	5502	123	2.15×10^{-3}
	BLEMDABIC	736	4802	207	1.76×10^{-4}
10^{-4}	BLSIC	1257	7688	201	1.14×10^{-3}
	BLSMDABIC	1180	7422	302	8.21×10^{-4}
	BLEIC	1210	7072	104	7.75×10^{-4}
	BLEMDABIC	1068	6704	241	1.13×10^{-4}

[1] Computations performed on an IBM 4361 in double precision.

The behaviour of the codes BLSODE and BLEND with the improved convergence criteria is considerably better than with the original convergence criteria. Furthermore, the effect of changing from $(I = chJ)^2$ to $(aI - bhJ)^2$ is less marked as the solution of the underlying implicit equation is more closely approximated.

It should be noted that the requirement that the residual be less than 10^{-2} times the local error tolerance may be unsatisfactory on some problems with high tolerances.

If, in BLEND, the original convergence criterion is only changed so as to allow a maximum of 3 iterations, instead of including the test on the residual, then the results are similar to, though not as good as, those in Table 7. Also, if the residual test is not included, but the original criterion is altered so that only a small *current* correction, and not a small estimate of the *next* correction, indicates convergence, then the results are improved, but again not as much as in Table 7. The achieved accuracy, in particular, is not as good, even when up to 3 iterations are allowed.

The blended DIFSUB code contained in a report [12] preceding Skeel and Kong [11] assumes convergence when the current correction is sufficiently small, allowing a maximum of 3 iterations. This convergence criterion seems much more reliable than those in BLSODE and BLEND. The code behaves well when applied to the same test problem (4.1). The results are shown in Table 8.

TABLE 8
Results[1] *obtained using blended* DIFSUB *of Skeel and Kong* [11].

TOL	Code	No. of steps	No. of back solves	No. of matrix factorisations	End point error
10^{-2}	blended DIFSUB	119	736	16	4.31×10^{-3}
10^{-3}	blended DIFSUB	932	6592	218	4.22×10^{-3}
10^{-4}	blended DIFSUB	1213	8600	257	7.91×10^{-4}

[1] Computations performed on an IBM 4361 in double precision.

5. Conclusions. The results of the stability analysis and the numerical tests show that, unless care is taken, perfect square implementations of the blended formulae can present severe difficulty in solving stiff oscillatory ordinary differential equations.

For semi-implicit blended formulae the choice of perfect square approximation and the number of iterations are critical in maintaining stability. When implicit blended formulae are implemented, it is essential that reliable convergence criteria be used.

The convergence criteria used in BLSODE and BLEND have been shown to be unreliable. Particular choices of perfect square approximations can help alleviate the difficulties caused by this, as would a suitable restriction on the minimum number of iterations allowed. Alternatively, more reliable convergence criteria could be sought. If it proves difficult or expensive to incorporate sufficiently reliable convergence criteria, alternative strategies for solving the implicit formulae, such as using complex arithmetic as in BCOMP, may be preferable to the perfect square approximation.

Acknowledgments. The authors would like to thank R. D. Skeel and the referees for very helpful comments and suggestions.

REFERENCES

[1] C. A. ADDISON AND P. M. HANSON, *An investigation into the stability properties of second derivative methods using perfect square iteration matrices*, this Journal, 7 (1986), pp. 1246–1264.

[2] M. BERZINS, private communication, 1986.
[3] W. H. ENRIGHT, *Optimal second derivative methods for stiff systems*, in Stiff Differential Systems, R. A. Willoughby, ed., Plenum Press, New York, 1974, pp. 95–111.
[4] P. W. GAFFNEY, *A performance evaluation of some* FORTRAN *subroutines for the solution of stiff oscillatory ordinary differential equations*, ACM Trans. Math. Software, 10 (1984), pp. 58–72.
[5] ———, NASTI *library*, private communication, 1985.
[6] C. W. GEAR, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1971.
[7] ———, *Algorithm* 407: DIFSUB *for solution of ordinary differential equations*, Comm. ACM, 14 (1971), pp. 176–179.
[8] A. C. HINDMARSH, LSODE *and* LSODI, *Two new initial value ordinary differential equation solvers*, ACM SIGNUM Newsletter, December, 1980.
[9] W. HOCK, *Test examples for non-linear programming codes*, Lecture Notes in Econom. and Math. Systems, 187, Springer-Verlag, Berlin-New York, 1981.
[10] D. E. KNUTH, *The Art of Computer Programming. Volume* 1: *Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1968.
[11] R. D. SKEEL AND A. K. KONG, *Blended linear multistep methods*, ACM Trans. Math. Software, 3 (1977), pp. 326–345.
[12] ———, *Blended linear multistep methods*, Technical Report UIUCDCS-R-76-800, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, Urbana, IL.
[13] R. D. SKEEL, *Equivalent forms of multistep formulas*, Math. Comp., 33 (1979), pp. 1229–1250.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 6, November 1987
© 1987 Society for Industrial and Applied Mathematics
008

NUMERICAL DIFFERENTIATION BY HIGH ORDER INTERPOLATION*

PETER HOFFMAN† AND K. C. REDDY‡

Abstract. High order interpolation is examined as a method for numerical approximation and differentiation. The approximation operators are proved to be bounded in the uniform norm. To reduce this bound for uniformly spaced points, a technique involving subsets of these points is introduced. The subsets are chosen to approximate the Chebyshev distribution. Numerical verification and an application to the solution of partial differential equations by pseudospectral methods are provided. Particular attention is given to the role of machine precision.

Key words. approximation, interpolation, numerical differentiation, finite difference, finite element, spectral

AMS(MOS) subject classifications. 39, 41, 65

1. Introduction. High order accurate spectral methods are becoming increasingly applicable [1]–[4] for the solution of multidimensional flow problems in complex geometries and for other physical phenomena. They are usually implemented over discrete grid points that cover the physical domain, which is transformed to a suitable computational space. Pseudospectral methods involve high order interpolations over a whole interval based on either sinusoidal functions for problems with periodic boundary conditions or Chebyshev or other polynomials for nonperiodic problems. While Fourier-spectral methods utilize equally spaced grid points, Chebyshev-spectral methods require an unequally spaced mesh such as the extrema of a high order Chebyshev polynomial. Interpolation by Chebyshev polynomials over equally spaced grid points is highly ill-conditioned and not always convergent. However, in practical multidimensional flow solver codes it is advantageous to have spectral differentiation approximations over equally spaced grid points in the computational domain for two reasons. One is that the implicit time integration or iteration methods can be carried out efficiently by approximating some of the spectral derivative operators by finite difference operators and by the use of approximate factorization methods. Also, a domain may be subdivided into smaller domains depending on the nature of the problem, and spectral approximations can be applied over overlapping grid points in each of the subdomains. These techniques can be incorporated into a larger code most conveniently if the grid in the computational domain is equally spaced in each direction.

This paper deals with high order accurate approximations for functions and their derivatives by means of Chebyshev polynomial interpolations. The approximations outlined have applications for solving partial differential equations by pseudo-spectral methods. First a theorem is proved on the bounds for numerical differentiation operators based on global interpolation polynomials. For uniformly spaced grids a subsetting procedure is outlined which can overcome the ill-conditioning of Chebyshev interpolation over such grids. Background information can be found in [2], [5] and [6].

* Received by the editors October 27, 1983; accepted for publication (in revised form) October 21, 1986.

† Calspan Field Services, Inc./AEDC Division, Arnold Engineering Development Center, Arnold Air Force Station, Tennessee 37389, and Department of Mathematics, University of Colorado, Denver, Colorado 80202. The work of this author was supported in part by Air Force Office of Scientific Research grant AFOSR-83-0101.

‡ Calspan Field Services, Inc./AEDC Division, Arnold Engineering Development Center, Arnold Air Force Station, Tennessee 37389, and Department of Mathematics, University of Tennessee Space Institute, Tullahoma, Tennessee 37388.

Let X_N denote any set of $(N+1)$ distinct points, $\{x_0, x_1 \cdots x_N\}$, in $[-1, 1]$. The interpolant of a function $f(x)$ at the points of X_N is an element of P_N, the space of polynomials of degree at most N, and is given by

$$Lf(x) = \sum_{i=0}^{N} f(x_i) l_i(x)$$

where

$$l_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{N} \frac{x - x_j}{x_i - x_j}.$$

The Lebesgue constant Λ for the mesh X_N is defined by

$$\Lambda = \operatorname*{Max}_{x \in [-1,1]} \sum_{i=0}^{N} |l_i(x)|,$$

which is equal to $\|L\|$ in the uniform norm. Furthermore, the error in the interpolant approximation satisfies the Lebesgue inequality [6],

$$\|f - Lf\| \leqq (1 + \Lambda) \|f - p^*\|$$

where p^* is the best approximation to f in P_N. Regardless of the spacing of the points, Λ always grows at least logarithmically with N. For a uniform mesh, denoted by E_N, Λ grows exponentially. There are, however, many meshes for which the growth in Λ is not more than logarithmic [7]. One of these is $U_N = \{u_0, u_1, \cdots, u_N\}$, where $u_j = \cos(j\pi/N)$. U_N is the set of extrema of the Nth degree Chebyshev polynomial, $T_N(x) = \cos N\theta$, where $x = \cos\theta$. When necessary for clarity, L_{U_N} will be used to designate the operator L based on U_N. The other quantity in the Lebesgue inequality is usually bounded by one of Jackson's theorems [6]. For example, if $f \in C[-1, 1]$ belongs to Lip α, $0 < \alpha \leqq 1$, then $\|f - p^*\| \leqq AN^{-\alpha}$. For such functions, the Lebesgue inequality implies that the interpolant converges to the underlying function when $X_N = U_N$, though not necessarily when $X_N = E_N$.

In this paper we consider numerical differentiation in which $D^k f$ is approximated by $D^k Lf$, where D denotes (d/dx). In § 2, a theorem is proved which asserts that $\|D^k L\| \leqq \Lambda T_N^{(k)}(1)$, $k > 0$, for any mesh X_N. In § 3, a technique is discussed for coping with the astronomical Lebesgue constant associated with a uniform mesh. The technique constructs subsets of $N_{\max}$ which resemble U_N for $N < N_{\max}$. For these subsets, the parameter $\eta = N^2/N_{\max}$ is shown to determine the Lebesgue constant. All of this is validated in § 4, where we account for finite precision arithmetic. Finally, § 5 demonstrates an application to the numerical solution of a partial differential equation by a pseudospectral method.

2. Derivatives of interpolated functions. There is a theorem due to Markov which states that for all $p \in P_N[-1, 1]$,

$$\|Dp\| \leqq N^2 \|p\|.$$

A more general version was proved by Duffin and Schaeffer [9]. Duffin and Karlovitz [10] recently gave an elegant new proof of these results.

MARKOV-DUFFIN-SCHAEFFER THEOREM. *Let* $p \in P_N[-1, 1]$. *If* $|p| \leqq 1$ *on* $U_N \subset [-1, 1]$; *then*

$$\|D^k p\| \leqq T_N^{(k)}(1) = \frac{N^2(N^2 - 1^2)(N^2 - 2^2) \cdots (N^2 - (k-1)^2)}{1 \cdot 3 \cdot 5 \cdots (2k-1)}$$

for $0 < k \leqq N$.

This theorem is essential to the proof of the next theorem, which is a reinterpretation of a result of Berman [8] to incorporate the Lebesgue constant.

THEOREM. *Let* $0 < k \leqq N$. *Then*

$$T_N^{(k)}(1) = \|D^k L_{U_N}\| \leqq \|D^k L_{X_N}\| \leqq \Lambda_{X_N} T_N^{(k)}(1).$$

Proof. (i) $\|D^k L_{U_N}\| \leqq T_N^{(k)}(1)$ because

$$\begin{aligned}\|D^k L_{U_N}\| &= \operatorname*{Sup}_{\|f\|=1} \|D^k L f\| \\ &= \operatorname*{Sup}_{\|f\|=1} \left\| \sum_{i=0}^{N} f(u_i) l_i^{(k)}(x) \right\| \\ &\leqq \operatorname*{Max}_{x \in [-1,1]} \sum_{i=0}^{N} |l_i^{(k)}(x)|\end{aligned}$$

where $l_i^{(k)}$ is the kth derivative of l_i based on U_N. This maximum is achieved at some point x^* in $[-1, 1]$. Define p in P_N by $p(u_i) = \operatorname{Sgn} l_i^{(k)}(x^*)$, $i = 0, 1, \cdots, N$. Then

$$\begin{aligned}\sum_{i=0}^{N} |l_i^{(k)}(x^*)| &= \sum_{i=0}^{N} \operatorname{Sgn} l_i^{(k)}(x^*) l_i^{(k)}(x^*) \\ &= p^{(k)}(x^*) \\ &\leqq T_N^{(k)}(1)\end{aligned}$$

by the preceding theorem.

(ii) $T_N^{(k)}(1) \leqq \|D^k L_{X_N}\|$, where X_N is arbitrary, because

$$\begin{aligned}T_N^{(k)}(1) &= \sum_{i=0}^{N} T_N(x_i) l_i^{(k)}(1) \\ &\leqq \left\| \sum_{i=0}^{N} T_N(x_i) l_i^{(k)}(x) \right\| \\ &\leqq \operatorname*{Sup}_{\|f\|=1} \left\| \sum_{i=0}^{N} f(x_i) l_i^{(k)}(x) \right\| \\ &= \|D^{(k)} L_{X_N}\|,\end{aligned}$$

where l_i is based on X_N. In particular, for $X_N = U_N$, $T_N^{(k)}(1) \leqq \|D^k L_{U_N}\|$. Hence, $T_N^{(k)}(1) = \|D^k L_{U_N}\|$.

(iii) Finally,

$$\begin{aligned}\|D^k L_{X_N}\| &= \|D^k L_{U_N} L_{X_N}\| \\ &\leqq \|D^k L_{U_N}\| \, \|L_{X_N}\| \\ &\leqq T_N^{(k)}(1) \Lambda_{X_N} \quad \text{by (i).}\end{aligned}$$

□

It follows from this theorem that U_N is optimal among all meshes X_N for numerical differentiation.

Although the differential operator is unbounded, the theorem provides a bound for the numerical derivative operator,

$$\|D^k L\| \leqq \Lambda T_N^{(k)}(1) \quad \text{for } k \geqq 0.$$

Although each differentiation increases this bound by roughly N^2, the Lebesgue constant is the dominant factor when $X_N = E_N$.

3. Subsetting technique. This is a method to reduce the Lebesgue constant for uniform mesh problems. Let $N_{\max}$ denote the number of nodes of the uniform mesh, $E_{N_{\max}}$. Let $N < N_{\max}$. Because the Lebesgue constant of $E_{N_{\max}}$ increases exponentially with $N_{\max}$ while that of U_N increases only logarithmically with N, this method is based on selecting subsets of $E_{N_{\max}}$ which resemble U_N.

If $e_k = 1 - (2k/N_{\max})$ and $u_j = \cos(j\pi/N)$, then e_k approximates u_j, written $e_k \doteq u_j$, if

$$k \doteq \frac{N_{\max}}{2}\left(1 - \cos\frac{j\pi}{N}\right), \qquad j = 0, 1, 2, \cdots, N.$$

The numerical results in this paper are obtained by taking k to be the integer part of this expression as j varies. If this value of k has been previously selected, choose the next available value, working always from the ends of the mesh toward the center.

In the numerical results of the next section, the Lebesgue constant of the submesh is seen to behave exponentially, not with N or $N_{\max}$, but with the smaller parameter,

$$\eta = \frac{N^2}{N_{\max}}$$

This has a heuristic explanation. Consider the distance between adjacent points,

$$\Delta e_k = e_k - e_{k+1} = \frac{2}{N_{\max}} = \Delta e$$

and

$$\begin{aligned}\Delta u_j &= u_j - u_{j+1} \\ &= \cos\frac{j\pi}{N} - \cos\frac{(j+1)\pi}{N} \\ &\doteq \frac{\pi^2}{N^2}\left(j + \frac{1}{2}\right).\end{aligned}$$

If $\Delta u_j < \Delta e$, then the approximation $e_k \doteq u_j$ becomes impossible to satisfy. Setting $\Delta u_j = \Delta e$ and solving for j gives $j = (2/\pi^2)\eta - \frac{1}{2}$, where $\eta = (N^2/N_{\max})$.

The exponential growth of the Lebesgue constant is characterized in some sense by the number of uniformly spaced points. For the submesh, the number of points which fail to approximate the points of U_N and which, therefore, retain uniform spacing is

$$j = \frac{2}{\pi^2}\eta - \frac{1}{2}.$$

Furthermore, this reasoning predicts that exponential growth vanishes when j vanishes for $\eta = (\pi^2/4)$.

4. Numerical verification. For the meshes under consideration, the Lebesgue constant, Λ, can be easily estimated in the following manner. Define the matrix M whose entries are

$$M_{ij} = l_j(u_i).$$

Then

$$\Lambda \doteq \|M\|$$

where $\|\cdot\|$ denotes the matrix norm analogous to the uniform norm, namely the row norm [11],

$$\|M\| = \operatorname{Max}_i \sum_j |m_{ij}|.$$

Briefly, this approximation is valid because, once M is used to provide data at the points of U_N, further interpolation introduces only a negligible logarithmic error.

With similar reasoning, the Lebesgue constant can be related to the ill-conditioning of the interpolation by

$$\Lambda \doteq \|M\| \doteq \kappa(M),$$

where κ is the condition number of M based on the row norm.

Numerical results are presented with a logarithmic scale so that exponential relationships yield linear plots. Moreover, the base ten logarithm is used so that the data can be directly interpreted in decimal places of accuracy.

In Fig. 1, a solid line connects those circles which represent numerical data for uniform meshes. Numerically, we observe a linear relationship, for the most part, between $\log \Lambda$ and $N_{\max}$. Analytically, the exact slope of this line is unknown. However, these numerical results do fall within known bounds [6]. Also in Fig. 1, dotted lines connect those circles which represent data for meshes created by the subsetting technique. Numerically, we observe that Λ is nearly constant for meshes sharing the same value of $\eta = (N^2/N_{\max})$.

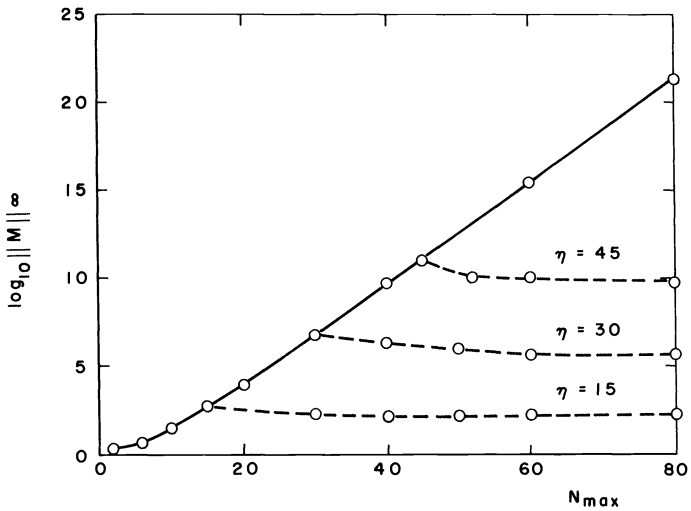


FIG. 1. *Verification that* $\|M\|_\infty$ *is determined by* η.

In Fig. 2, the solid line of Fig. 1 is again shown for reference. Figure 2 shows the implication of the Lebesgue inequality

$$\|f - Lf\| \leqq (1+\Lambda)\|f - p^*\|$$

for floating point arithmetic. The error in the best approximation, $\|f - p^*\|$, can be no smaller than the machine precision, θ. It follows that the calculations will have an apparent precision of Λ times θ. The norms of the errors found in interpolating a constant function are plotted as functions of η in Fig. 2 for both single precision

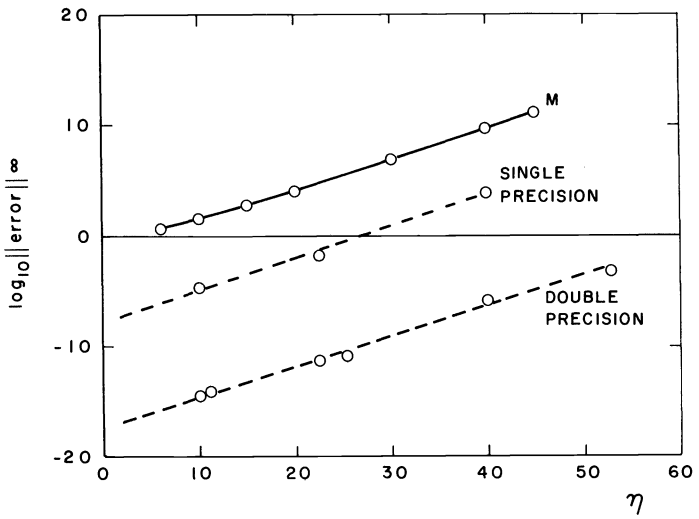


FIG. 2. *Perturbation caused by floating point arithmetic is linear in* η.

($\theta = 10^{-7.2}$) and double precision ($\theta = 10^{-16.8}$) calculations. The dotted lines, obtained by least squares, show that the increase in error parallels the increase in the Lebesgue constant. They also indicate that the perturbations caused by floating point arithmetic are nearly linear in η. As we mentioned in the previous section, this effect is predicted to vanish when $\eta = (\pi^2/4) \doteq 2.5$; the plot does indeed show that machine precision is attained at this value.

Figure 3 illustrates the theorem of § 2, with some test calculations that the error in numerical differentiation is amplified by N^2 for each differentiation. With $N = 24$, 35 and 52 and the corresponding values of $\eta = 15$, 30 and 45, Chebyshev polynomials of degrees both above and below N are chosen as test functions. Interpolation approximations and numerical differentiations of first and second order are carried out using double precision arithmetic. For polynomials of degree $\leqq N$, the error in best

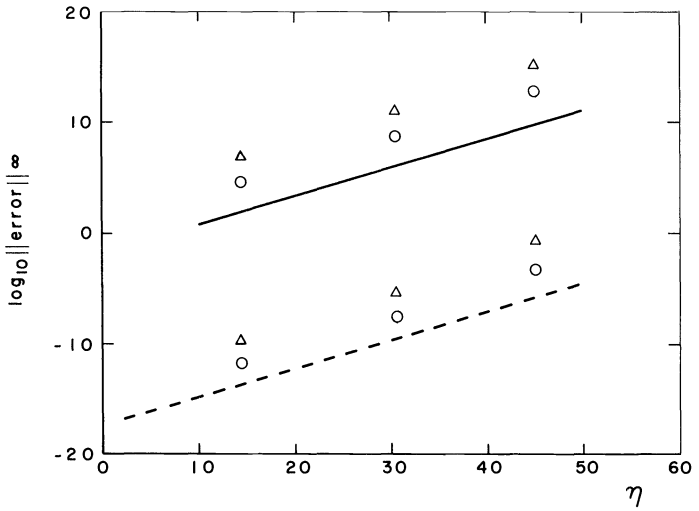


FIG. 3. *Largest errors observed in first derivative (circle) and second derivative (triangle), for varying* η.

approximation satisfies $\|f-p^*\| = \theta = 10^{-16.8}$, whereas for polynomials of degree $> N$, $\|f-p^*\| \simeq 1$. The dotted line in the figure represents the worst errors in approximating the Chebyshev polynomials of degree $\leqq N$ and the circles above the dotted line show the largest errors in approximating the corresponding first derivatives and the triangles above them show similar errors for the second derivatives. The solid line and the circles and triangles in the upper half of the figure show the largest errors in approximating the Chebyshev polynomials of degree $> N$ and their first two derivatives. In these experiments, $\log_{10} N^2$ lies between 2.7 and 3.4, which is seen to bound the distance of the circles above the lines and the distance of the triangles above the circles.

TABLE 1

Pointwise errors in the solution of Burgers' equation across the interval $[-1, 1]$.

Second order finite difference	Pseudospectral $\eta = 30$	Pseudospectral $\eta = 15$
0.00000000000000000	0.00000000000000000	0.00000000000000000
0.00000135057895684	0.00000094621951063	0.00000000000004129
0.00000272523219137	0.00000090694500574	0.00000000000006382
0.00000411825489322	0.00000087670840566	0.00000000000009065
0.00000552334496008	0.00000084427558521	0.00000000000011635
0.00000693360770194	0.00000081019764776	0.00000000000014244
0.00000834156870344	0.00000077436228052	0.00000000000016963
0.00000973919570321	0.00000073675962554	0.00000000000019615
0.00001111793026767	0.00000069735365135	0.00000000000022153
0.00001246872997063	0.00000065612470432	0.00000000000024707
0.00001378212164146	0.00000061305087458	0.00000000000027257
0.00001504826614059	0.00000056811897392	0.00000000000029696
0.00001625703487530	0.00000052132903509	0.00000000000031936
0.00001739809809344	0.00000047268620941	0.00000000000034017
0.00001846102473287	0.00000042220053285	0.00000000000035973
0.00001943539332416	0.00000036989081102	0.00000000000037806
0.00002031091313676	0.00000031578195865	0.00000000000039430
0.00002107755441612	0.00000025992290624	0.00000000000040762
0.00002172568626382	0.00000020235373688	0.00000000000041830
0.00002224622031814	0.00000014313183479	0.00000000000042670
0.00002263075811198	0.00000008232912148	0.00000000000043287
0.00002287173963966	0.00000002002481540	0.00000000000043624
0.00002296259039407	0.00000004369380968	0.00000000000043635
0.00002289786390282	0.00000010872348891	0.00000000000043294
0.00002267337661725	0.00000017495659255	0.00000000000042646
0.00002228633189778	0.00000024226926827	0.00000000000041745
0.00002173542985056	0.00000031053470543	0.00000000000040547
0.00002102095980108	0.00000037961056375	0.00000000000039029
0.00002014487239805	0.00000044934681809	0.00000000000037151
0.00001911082860850	0.00000051958836066	0.00000000000034967
0.00001792422325085	0.00000059017292176	0.00000000000032578
0.00001659218118471	0.00000066092989762	0.00000000000029929
0.00001512352483902	0.00000073168473498	0.00000000000026994
0.00001352871241565	0.00000080225654475	0.00000000000023786
0.00001181974679061	0.00000087246461522	0.00000000000020434
0.00001001005587202	0.00000094212781319	0.00000000000016896
0.00000811434593984	0.00000101106459370	0.00000000000013158
0.00000614843019747	0.00000107912234809	0.00000000000009332
0.00000412903543596	0.00000114552003409	0.00000000000005463
0.00000207359041524	0.00000122422936164	0.00000000000001419
0.00000000000000000	0.00000000000000000	0.00000000000000000

Although these examples involve only Chebyshev polynomials, excellent results have been obtained for many other functions encountered in our computations. Other examples are not particularly instructive, however, as they introduce the details of estimating the rate of exponential decay of Chebyschev coefficients.

5. Application to numerical partial differential equations. Consider the Burgers' equation

$$u_t = \nu u_{xx} - u u_x,$$

with boundary conditions such that the steady state solution is $u(x) = -\tanh((x-1)/2\nu)$. When $\nu = 1$, this solution will be sufficiently smooth to ensure that $\|u - p^*\| < \theta$ in the examples which follow.

Numerical calculations were performed on $[-1, 1]$ using a 40-point uniform mesh for discretization of the spatial derivatives. The time derivative was discretized with the implicit backward Euler method. In each time step, the resulting nonlinear system was solved by Newton's method; the Newton iteration operator uses efficient, tridiagonal finite difference approximations to the spatial derivatives. The calculation was continued until no further time dependence could be observed ($t = 30$).

In Table 1, errors are tabulated for points uniformly spaced from left to right. For comparison, the first column contains results for the usual three-point finite difference solution on this mesh. Four-place accuracy is observed.

Numerical differentiation based on a single interpolant through all the points is usually called a pseudospectral method. Table 1 contains no results for this case because the computation fails. Indeed, Fig. 3 reveals that little accuracy can be expected for second derivative terms when $\eta = (N^2/N_{\max}) = ((40)^2/40) = 40$.

Discarding five points with the subsetting technique gives $\eta = ((35)^2/40) \doteq 30$. Results for this case are tabulated in the second column. Already, six-place accuracy is observed; this is predicted by Fig. 3.

Discarding fifteen points with subsetting gives $\eta = ((25)^2/40) \doteq 15$. The third column of Table 1 shows a remarkable 12-place accuracy; this is again consistent with Fig. 3.

If explicit, rather than implicit, time integration is used, one expects a stability restriction on the time step, $\Delta t \simeq O(1/N_{\max}^2)$. Since $N_{\max} = N^2/\eta$, subsetting may not relax the asymptotic time step restriction but it is still practical because relatively large values of η can give good results as demonstrated here.

Although this analysis is performed for $[-1, 1]$, it can easily be done for $[-h, h]$ where h becomes small. However, the Lebesgue constant remains unchanged as h decreases; only the relative spacing of points is important. Hence, results similar to those of this paper hold for high order finite differences and for the p version of the finite element method.

Acknowledgments. This paper has benefited from helpful suggestions from Dick Duffin, Max Gunzburger, Jim Jacocks, Julius Smith and one of the reviewers.

REFERENCES

[1] D. GOTTLIEB AND S. A. ORSZAG, *Numerical Analysis of Spectral Methods*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1977.

[2] R. PEYRET AND T. D. TAYLOR, *Computational Methods for Fluid Flow*, Springer-Verlag, New York, 1983.

[3] K. C. REDDY, *Pseudospectral approximation in a 3-D Navier-Stokes code*, AIAA J., 21 (1983), pp. 1208-1210.

[4] R. G. VOIGT, D. GOTTLIEB AND M. Y. HUSSAINI, EDS., *Proceedings of the Symposium on Spectral Methods for Partial Differential Equations*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1984.
[5] C. LANCZOS, *Applied Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1956.
[6] T. J. RIVLIN, *An Introduction to the Approximation of Functions*, Dover, New York, 1969.
[7] L. BRUTMAN, *On the Lebesgue function for polynomial interpolation*, SIAM J. Numer. Anal., 15 (1978), pp. 694–704.
[8] T. J. RIVLIN, *The Chebyshev Polynomials*, John Wiley, New York, 1974.
[9] R. J. DUFFIN AND A. C. SCHAEFFER, *A refinement of an inequality of the Brothers Markoff*, Trans. Amer. Math. Soc., 50 (1941), pp. 517–528.
[10] R. J. DUFFIN AND L. A. KARLOVITZ, *The Markoff-Duffin-Schaeffer inequalities abstracted*, Proc. Nat. Acad. Sci. USA (1982), pp. 955–957.
[11] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins Univ. Press, Baltimore, MD, 1983.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 6, November 1987
© 1987 Society for Industrial and Applied Mathematics
009

RECURRENCE IN SEMIDISCRETE APPROXIMATIONS OF THE NONLINEAR SCHRÖDINGER EQUATION*

J. A. C. WEIDEMAN† AND B. M. HERBST†

Abstract. Recurrence in the nonlinear Schrödinger equation is studied by means of three semidiscrete approximations, namely a spectral, a pseudospectral and a finite difference method. These methods are analyzed for conservation and stability properties. A linear stability analysis is presented, as well as a nonlinear two-mode analysis. A comparison of the results reveals some disadvantages of the pseudospectral and finite difference methods. The role of aliasing is examined and some numerical results are given.

Key words. recurrence, nonlinear Schrödinger equation

AMS(MOS) subject classification. 65P05

1. Introduction. The interest in nonlinear wave phenomena has escalated in recent years. To a large extent this was initiated by the famous numerical experiments of Fermi, Pasta and Ulam [1]. They considered a dynamical system consisting of 64 particles vibrating transversely on an elastic string. Their main concern was the effect of nonlinear interactions among the particles. In particular, they argued that if the initial energy distribution consisted of energy in a few low-frequency Fourier modes, the nonlinear interaction would spread energy to the high frequencies. It was suspected that ultimately the energy would be equipartitioned between all frequencies, a process called thermalization. In an attempt to investigate the rate at which thermalization is approached, Fermi, Pasta and Ulam integrated their system numerically. Surprisingly, thermalization did not occur. Instead, after an initial spread of energy, the process was reversed and the initial energy configuration was recovered. This phenomenon is known as recurrence.

Subsequently, recurrence has been encountered in other nonlinear wave problems, for example in waves described by the Korteweg–de Vries equation [2] (which approximates the Fermi–Pasta–Ulam model) and the modified Korteweg–de Vries equation [3]. In this paper we shall focus attention on yet another nonlinear wave equation which is known to exhibit recurrence, namely the nonlinear Schrödinger equation (NLS), given by

$$u_t = iu_{xx} + iq|u|^2 u. \tag{1}$$

Here u is a complex valued function, q a positive real parameter and $i^2 = -1$. Recurrence of unsteady wave train solutions of (1) was first demonstrated numerically by Yuen and Ferguson [4]. They showed that recurrence is the long-time evolution of an initial side-band growth, the onset of which is predicted by a linear instability analysis (Benjamin and Feir [5]). A *qualitative* description of recurrence in the NLS has been given by Thyagaraja [6]–[8]. He showed that two of the conservation laws satisfied by the NLS are sufficient to restrict the Fourier modes actively taking part to effectively a finite number. Hence the nonthermalization was demonstrated. Furthermore, by use of some well-known results from dynamical systems theory, recurrence could be proved. Analyses which *quantitatively* describe recurrence have been given by Infeld [9], [10] and Stiassni and Kroszynski [11]. All these authors were able to

* Received by the editors November 18, 1985; accepted for publication (in revised form) October 17, 1986. This work was supported in part by the Council for Scientific and Industrial Research.

† Department of Applied Mathematics, University of the Orange Free State, Bloemfontein 9300, South Africa.

establish periodicity in time, by assuming that Thyagaraja's finite number of effective modes was in fact precisely two, namely the two modes of lowest frequency. It should be noted here that the term "recurrence" as used by Thyagaraja is a weaker concept than periodicity, in the sense that recurrent behaviour need not be periodic. In the present paper recurrent behaviour turns out to be periodic.

From a numerical point of view, the methods employed by Infeld and by Stiassni and Kroszynski amount to using a spectral method to solve the NLS. In practice, however, a full spectral method is rarely used, due to the expensive computation it requires. Instead, pseudospectral, finite difference or finite element methods are cheaper and thus more commonly used.

The main thrust of this paper, therefore, is to undertake a systematic investigation of the use of numerical methods for a study of recurrence in the NLS. We selected three semidiscrete schemes, namely a spectral, a pseudospectral and a finite difference method. The term pseudospectral is used in the sense intended by Orszag [12], who advocated the use of Fast Fourier Transform (FFT) methods to approximate the second derivative in (1). As will be pointed out later, the theoretical difference between a full spectral method and the pseudospectral method is the presence of aliasing errors in the latter method. Particular attention will therefore be given to the effect of aliasing in the present situation.

We shall analyse our semidiscrete schemes for conservation properties (§ 4) and for linear Benjamin–Feir side-band instability (§ 5). It will be shown that all our methods satisfy discrete analogues of the two conservation laws upon which Thyagaraja's results are based. For this reason the finite element method is omitted from our discussion, since it does not conserve these quantities if the device called product-approximation is used for the nonlinear term. (For details see [16].) It will also be shown that the spectral-type methods possess the exact Benjamin–Feir instability limit, whereas only an approximation of this limit is yielded by the finite difference method. Similar analyses have been undertaken in [14] and [15].

In the main part of the paper, § 6, the two-mode approach of Infeld, Stiassni and Kroszynski is extended to our semidiscrete schemes. Approximations to the true recurrence time and the maximal side-band growth will be derived, and the results for the different methods will be compared. In § 7 some numerical results will be presented. Although a detailed comparison between the different methods falls outside the scope of the present study, the finite difference method will be shown to possess certain basic shortcomings.

In the present paper the emphasis is placed on the numerical simulation of recurrence in the NLS. For a full account of related issues involved in solving the NLS numerically, such as convergence, nonlinear blowup, time integration schemes, etc., the reader is referred to [13]–[23].

2. Overview of conservation, instability and recurrence. Before we briefly outline the analytical properties of the NLS which are relevant to our study we require the following definitions. Consider a function $u(x, t)$ which is periodic on $-\frac{1}{2}L \leqq x \leqq \frac{1}{2}L$. The Fourier transform of u will be denoted by

$$\hat{u}_n(t) = \frac{1}{L}\int_{-L/2}^{L/2} u(x, t) \exp(-i\mu_n x)\, dx, \qquad n = -\infty, \cdots, \infty \tag{2}$$

with frequencies

$$\mu_n = \frac{2\pi n}{L}. \tag{3}$$

The inverse formula, or Fourier-expansion of u, is given by

$$u(x, t) = \sum_{n=-\infty}^{\infty} \hat{u}_n(t) \exp(i\mu_n x). \tag{4}$$

In deriving (2) from (4), or vice versa, the orthogonality condition

$$\int_{-L/2}^{L/2} \exp(i(\mu_n - \mu_m))\, dx = L\delta_{n,m} \tag{5}$$

may be employed in the usual manner. Here δ denotes the Kronecker symbol.

We shall also need two of the infinite number of conservation laws satisfied by the NLS. They are the conservation of energy

$$\int_{-L/2}^{L/2} |u|^2\, dx = c_1 \tag{6}$$

as well as

$$\int_{-L/2}^{L/2} \left(|u_x|^2 - \frac{1}{2} q|u|^4\right) dx = c_2 \tag{7}$$

where c_1 and c_2 remain invariant with time.

We review the notion of Benjamin–Feir side-band instability by considering the x-independent solution

$$u = a \exp(iq|a|^2 t) \tag{8}$$

where a is a constant. When (8) is perturbed to

$$u = a \exp(iq|a|^2 t)(1 + \varepsilon_+(t) \exp(i\mu x) + \varepsilon_-(t) \exp(-i\mu x)) \tag{9}$$

a linear analysis establishes that the side-band amplitudes $\varepsilon_+(t)$ and $\varepsilon_-(t)$ will exhibit exponential growth whenever

$$0 < \mu^2 < 2q|a|^2. \tag{10}$$

The maximal growth rate in the side-bands occurs for

$$\mu^2 = q|a|^2. \tag{11}$$

For details of these derivations the reader is referred to [5] or [15].

It can be shown (see e.g. [6] or [24]) that the conservation laws (6) and (7) prevent the instabilities from becoming unbounded in time. The numerical investigations by Yuen and Ferguson concerning the long time evolution of these instabilities subsequently lead to their discovery of recurrence in the NLS. For reference purposes we will describe a numerical experiment similar to those conducted by them.

Consider

$$\varepsilon_+(0) = \varepsilon_-(0) := \tfrac{1}{2}\varepsilon, \tag{12}$$

say, where

$$|\varepsilon|^2 \ll 1. \tag{13}$$

Then the initial condition corresponding to the perturbed solution (9) becomes

$$u(x, 0) = a(1 + \varepsilon \cos \mu x), \qquad -\frac{\pi}{\mu} \leqq x \leqq \frac{\pi}{\mu}. \tag{14}$$

In particular we choose

$$q=2, \quad a=\frac{1}{2}, \quad \varepsilon=\frac{1}{10}, \quad \mu=\frac{1}{\sqrt{2}}. \tag{15}$$

The choice $\mu=1/\sqrt{2}$ falls inside the instability limit (10). In fact, for illustrative purposes we have chosen μ to yield the maximal growth rate given by (11).

The analytical solution of the NLS corresponding to the initial condition (14) is not known. We take as the correct solution a numerical solution obtained from a pseudospectral method which has converged. This solution is depicted in Fig. 1(a),

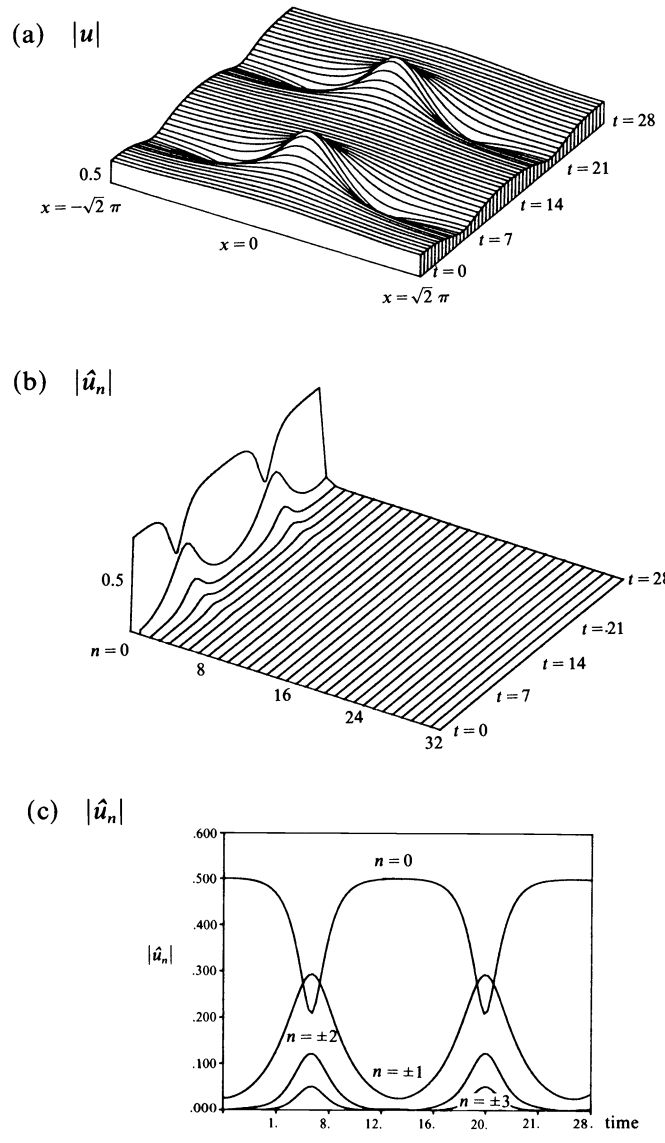


FIG. 1. *Recurrence in the* NLS *for the parameters* (15). (a) *Physical space solution* $|\hat{u}(x, t)|$; (b) *Fourier space solution* $|\hat{u}_n(t)|$, $n=0, \cdots, 32$; (c) *Fourier space solution* $|\hat{u}_n(t)|$, $n=0, \pm 1, \pm 2, \pm 3$.

where $|u|$ is plotted for one space-period $-\sqrt{2}\pi \leqq x \leqq \sqrt{2}\pi$ and for $0 \leqq t \leqq 28$. Figure 1(b) represents the corresponding Fourier space solution, that is, we have plotted $|\hat{u}_n|$ for $n = 1, \cdots, 32$. Note that we have $|\hat{u}_n| = |\hat{u}_{-n}|$ for all n due to the fact that the initial condition is an even function. In Fig. 1(c) the evolution of the four low-frequency components are represented.

In Figs. 1(b) and 1(c) the initial exponential side-band growth in $|\hat{u}_1| = |\hat{u}_{-1}|$ can be seen. This growth results in the single humped instability on the uniform solution in Fig. 1(a). At $t \approx 4$ the growth in the side-bands appears to be inhibited by conservation and gradually it starts to decline. Eventually, at $t \approx 13.3$, the initial condition is approximately reconstructed and the process repeats itself. This periodic return to the initial state will be referred to as recurrence. It should be noted that energy is transferred to the higher harmonics through the nonlinear interaction. However, the amplitudes of the higher harmonics $|\hat{u}_{\pm 2}|$, $|\hat{u}_{\pm 3}|, \cdots$, do not become dominant at any stage. Most of the energy remains confined to the fundamental, $n = 0$, and the unstable, $n = \pm 1$, modes.

We point out that Fig. 1 is only representative of the solution when

$$\tfrac{1}{2}q|a|^2 < \mu^2 < 2q|a|^2. \tag{16}$$

If $\mu^2 > 2q|a|^2$ no instability is observed as is predicted by the linear theory. The perturbation remains neutrally stable. On the other hand, if

$$\mu^2 < \tfrac{1}{2}q|a|^2 \tag{17}$$

some of the generated higher harmonics also fall inside the unstable range (10). In this case the unstable modes take turns in dominating the solution. A more complex pattern than the simple periodicity of Fig. 1 is observed. For pictures of these solutions either [4] or [15] may be consulted.

In the following section we introduce the numerical methods. In subsequent sections we analyse these methods for their conservation and stability properties.

3. Semidiscrete approximations of the NLS. We define an N-grid on $[-\frac{1}{2}L, \frac{1}{2}L]$ by

$$x_j = jh, \qquad j = -\tfrac{1}{2}N, \cdots, \tfrac{1}{2}N \tag{18}$$

with uniform grid size

$$h = \frac{L}{N}. \tag{19}$$

We shall take N to be an even number of intervals. On this grid an $O(h^2)$ finite difference replacement of the NLS yields the semidiscrete system

$$\dot{U}_j = i(U_{j+1} - 2U_j + U_{j-1})/h^2 + iq|U_j|^2 U_j, \qquad j = -\tfrac{1}{2}N, \cdots, \tfrac{1}{2}N - 1 \tag{20}$$

with initial condition

$$U_j(0) = u(x_j, 0).$$

Here $U_j(t)$ denotes the approximation to $u(x_j, t)$.

Also

$$U_{-(N/2)-1}(t) := U_{(N/2)-1}(t), \qquad U_{N/2}(t) := U_{-N/2}(t), \qquad t \geqq 0 \tag{21}$$

to ensure periodicity on $[-\frac{1}{2}L, \frac{1}{2}L]$.

Since periodic boundary conditions are under consideration the Fourier spectral method provides a viable alternative to the finite difference discretization. The basis

of the method is the Fourier expansion (4). It is assumed that the solution possesses sufficient smoothness so that the contribution of the high frequencies in (4) may be neglected. Thus, if $N+1$ terms in (4) are retained, the approximate solution to $u(x, t)$ may be expanded as

$$U(x, t) = \sum_{n=-N/2}^{N/2} A_n(t) \exp(i\mu_n x) \tag{22}$$

where $A_n(t)$ denotes the approximation to $\hat{u}_n(t)$.

Substitution of (22) into the NLS yields the semidiscrete system

$$\begin{aligned} &\dot{A}_n = -i\mu_n^2 A_n + iq \sum_{l_1+l_2-l_3=n} A_{l_1} A_{l_2} A_{l_3}^*, \\ &n = -\frac{1}{2}N, \cdots, \frac{1}{2}N, \quad -\frac{1}{2}N \leqq l_1, l_2, l_3 \leqq \frac{1}{2}N \end{aligned} \tag{23}$$

with initial conditions

$$A_n(0) = \frac{1}{L} \int_{-L/2}^{L/2} u(x, 0) \exp(-i\mu_n x)\, dx. \tag{24}$$

The systems (20) or (23) may be solved by any standard package for ODEs. In the computations which will be mentioned later, a Runge-Kutta-Merson time integration scheme was employed. Note, however, the complication in (23). The cubic nonlinearity, which is a local product in (20), gives rise to the convolution sum in (23). Thus the number of operations required to compute the right-hand side (RHS) of (23) increases as $O(N^3)$. This expensive procedure may be avoided by the pseudospectral approach [12], [25]. The idea is to use the very efficient FFT to transform the Fourier space solution to the physical space. The nonlinearity is then inexpensively computed in the physical space followed by another FFT to return to the Fourier space. This method reduces the number of operations to $O(N \log N)$. The penalty we pay, however, is that aliasing errors are introduced into the approximation. We now present a description of the pseudospectral method and aliasing.

On the grid (18) the discrete Fourier transform pair corresponding to (2) and (4) is given by

$$F_n U_j := A_n := \frac{1}{N} \sum_{j=-N/2}^{(N/2)-1} U_j \exp(-i\mu_n x_j), \qquad n = -\frac{1}{2}N, \cdots, \frac{1}{2}N - 1, \tag{25}$$

$$F_j^{-1} A_n := U_j := \sum_{n=-N/2}^{(N/2)-1} A_n \exp(i\mu_n x_j), \qquad j = -\frac{1}{2}N, \cdots, \frac{1}{2}N - 1. \tag{26}$$

The orthogonality condition is

$$\sum_{j=-N/2}^{(N/2)-1} \exp(i(\mu_n - \mu_m)x_j) = N\delta_{n,m(\mathrm{mod}\ N)}. \tag{27}$$

The origin of aliasing lies in the RHS of (27) which should be compared with (5). The discrete N-grid cannot distinguish between frequencies μ_n and $\mu_{n(\mathrm{mod}\ N)}$, and the high frequencies are "folded" onto the low ones. Thus, whenever an approximation to (2) is computed by means of the formula (25), the exact Fourier amplitude $\hat{u}_n$ and the aliased discrete amplitude A_n are related through

$$A_n = \hat{u}_n + \sum_{j=1}^{\infty} \hat{u}_{n \pm jN}. \tag{28}$$

Note that for a sufficiently smooth function the aliasing contribution in (28) becomes negligible as $N \to \infty$. The relationship (28) also explains the absence of the amplitude $A_{N/2}$ from (25) and (26). In this formulation $A_{-N/2}$ represents the approximation to both $2\hat{u}_{-N/2}$ and $2\hat{u}_{N/2}$.

When we use (25) and (26) the pseudospectral method may now be defined by

$$\dot{A}_n = -i\mu_n^2 A_n + iqF_n(U_j|U_j|^2) \tag{29}$$

or, equivalently, by

$$\dot{U}_j = -iF_j^{-1}(\mu_n^2 F_n U_j) + iq|U_j|^2 U_j, \qquad j = -\tfrac{1}{2}N, \cdots, \tfrac{1}{2}N - 1, \tag{30}$$

$$U_{-N/2}(t) = U_{N/2}(t), \qquad t \geqq 0. \tag{31}$$

We compare (29) and (23). Utilizing (27) we can easily verify that

$$F_n(U_j|U_j|^2) = \sum_{l_1+l_2-l_3=n} A_{l_1}A_{l_2}A_{l_3}^* + \sum_{l_1+l_2-l_3=n\pm N} A_{l_1}A_{l_2}A_{l_3}^*, \qquad n = -\frac{1}{2}N, \cdots, \frac{1}{2}N - 1. \tag{32}$$

Thus the pseudospectral scheme (29) differs from the spectral scheme in the second term on the RHS of (32). This is recognized to be the aliasing contribution.

Several procedures exist by which means the aliasing error can be removed. One method is simply to compute the transforms of the RHS of (29) on a $3N$-grid instead of an N-grid, and then to put $A_n = 0$ for $|n| > \frac{1}{2}N$. An alternative method would be the idea of a shifted grid as explained in [26]. Both these methods require more computational effort than (29) but are nevertheless cheaper than (23).

In the next sections we investigate some of the effects of aliasing. We start by indicating that aliasing does not destroy the conservation properties of the problem.

4. Conservation of semidiscrete approximations. It is easy to check that the finite difference method, like both the pseudospectral and spectral methods, satisfies discrete analogues of conservation laws (6) and (7).

The finite difference method (20) satisfies

$$h \sum_{j=-N/2}^{(N/2)-1} |U_j|^2 = C_1 \tag{33}$$

as well as

$$h \sum_{j=-N/2}^{(N/2)-1} \left\{ \frac{|U_{j+1} - U_j|^2}{h^2} - \frac{1}{2}q|U_j|^4 \right\} = C_2. \tag{34}$$

The pseudospectral scheme (30) satisfies (33) as well as

$$L \sum_{n=-N/2}^{(N/2)-1} \mu_n^2 |A_n|^2 - \frac{1}{2}qh \sum_{j=-N/2}^{(N/2)-1} |U_j|^4 = C_2. \tag{35}$$

Finally, the spectral method (23) satisfies

$$L \sum_{n=-N/2}^{N/2} |A_n|^2 = C_1 \tag{36}$$

and

$$L\left[\sum_{n=-N/2}^{N/2} \mu_n^2 |A_n|^2 - \frac{1}{2}q \sum_{-N/2 \leqq l_1, l_2, l_3 \leqq N/2} A_{l_1}A_{l_2}A_{l_3}^* A_{l_1+l_2-l_3}^* \right] = C_2. \tag{37}$$

It should be noted that the values of C_1 and C_2 need not be the same on each occurrence in (33)–(37). Each method conserves approximations to c_1 and c_2 in (6) and (7).

5. A linear stability analysis. We first consider the spectral scheme (23). For a Benjamin–Feir side-band analysis we assume

$$|A_n| \ll |A_0|, \qquad |n| = 1, \cdots, \tfrac{1}{2}N \tag{38}$$

and retain only first-order terms in A_n, $n \neq 0$. Hence the evolution of the fundamental mode is described by

$$\dot{A}_0 = iqA_0|A_0|^2. \tag{39}$$

This yields the x-independent solution

$$A_0(t) = a \exp(iq|a|^2 t) \tag{40}$$

where a is a constant.

The evolution of the side-band amplitudes of frequency $\mu_{\pm n}$ is governed by

$$\begin{aligned} \dot{A}_n &= -i\mu_n^2 A_n + iq[2|A_0|^2 A_n + A_0^2 A_{-n}^*], \\ \dot{A}_{-n} &= -i\mu_n^2 A_{-n} + iq[2|A_0|^2 A_{-n} + A_0^2 A_n^*], \end{aligned} \qquad |n| = 1, \cdots, \tfrac{1}{2}N. \tag{41}$$

We define

$$\varepsilon_{\pm n}(t) := \frac{A_{\pm n}(t)}{A_0(t)}. \tag{42}$$

Substitution of (40) and (42) into (41) yields the growth equation

$$\frac{d}{dt}\begin{pmatrix} \varepsilon_n \\ \varepsilon_{-n}^* \end{pmatrix} = i\begin{pmatrix} q|a|^2 - \mu_n^2 & q|a|^2 \\ -q|a|^2 & -q|a|^2 + \mu_n^2 \end{pmatrix}\begin{pmatrix} \varepsilon_n \\ \varepsilon_{-n}^* \end{pmatrix}. \tag{43}$$

The eigenvalues of the growth matrix in (43) are given by

$$\lambda_n = \pm\mu_n\sqrt{2q|a|^2 - \mu_n^2}. \tag{44}$$

The growth in the side-bands $|A_{\pm n}|$ will be exponential if one of the eigenvalues has a positive real part, that is if

$$0 < \mu_n^2 < 2q|a|^2. \tag{45}$$

This is identical to result (10), where the μ in (10) may be any one of the frequencies μ_n present in the computation. Note that it is the low frequencies which are unstable and that the number of unstable frequencies increases with the strength of the nonlinearity $q|a|^2$. In a practical computation the number of frequencies included should allow for at least the unstable ones. However, as we shall demonstrate in a later section, the stable high frequencies do have a marked effect on the quality of the numerical solution. Therefore a number of stable frequencies should also be included, which amounts to selecting

$$N \gg \frac{L}{\pi}(2q|a|)^{1/2}. \tag{46}$$

According to (44) the maximal growth rate occurs for

$$\mu_n^2 = q|a|^2, \tag{47}$$

which agrees with (11). If we choose $\mu = \mu_1$ to satisfy (47) and $A_1(0) = A_{-1}(0) = \frac{1}{2}a\varepsilon$ then we have the situation of (12)-(14). In this case the linear analysis predicts that the evolution of the side-bands will be

$$|A_{\pm 1}| = \tfrac{1}{2}|a||\varepsilon| \cosh^{1/2} 2q|a|^2 t. \tag{48}$$

For the choice of parameters (15) solution (48) is shown in Fig. 2(a). In comparison to the accurate solution (broken line) it is clear that the linear analysis describes the true behaviour accurately only in the very early stages, that is, while the perturbation is small. It gives no account of the periodicity in the solution. In the next section a nonlinear analysis will be given which yields a periodic description of the side-band evolution.

We turn to the pseudospectral method (29). An examination of (32) reveals that aliasing in the pseudospectral method does not change any of the results (39)-(48). It is therefore concluded that both the spectral and pseudospectral methods are capable of mimicking the linear instability properties of the true solution exactly. On the other hand, the finite difference method yields an $O(h^2)$ approximation of the analytical result. Proceeding along the lines of [14], we can show that all the modes satisfying

$$0 < \frac{2}{h^2} \sin^2 \frac{1}{2} \mu_n^2 h < q|a|^2 \tag{49}$$

will be unstable. This places a restriction on the grid size h. For instance, if

$$h^2 > \frac{2}{q|a|^2}$$

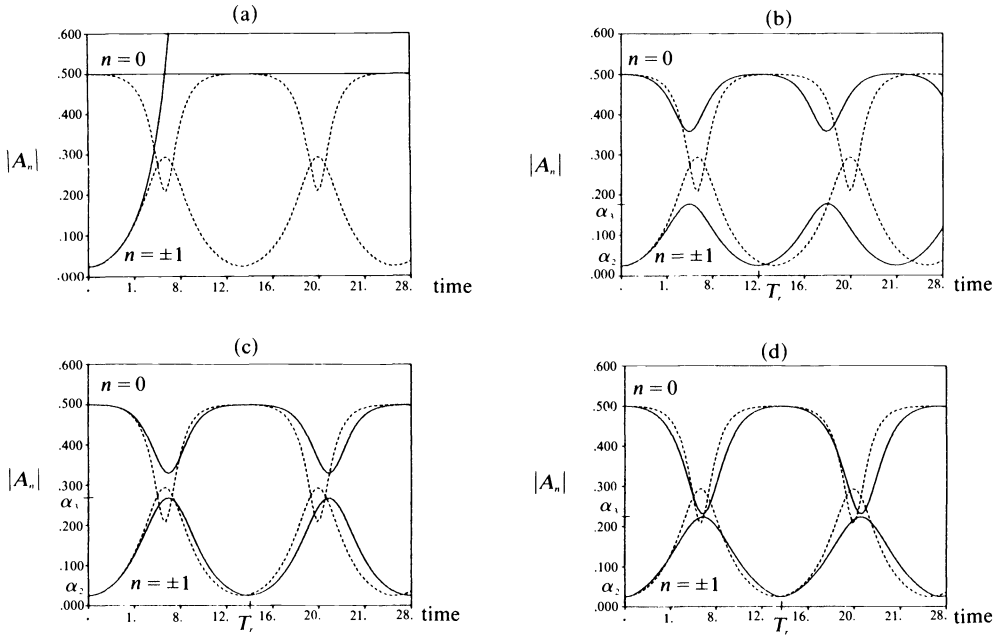


FIG. 2. *Approximate side-band evolution in the* NLS *as described by*: (a) *A linear analysis of* (41); (b) *A nonlinear analysis of the pseudospectral method* (54); (c) *A nonlinear analysis of the spectral method* (65); (d) *A nonlinear analysis of the finite difference method* (71). - - - -: *accurate solution*; ——: $N = 2$ *approximation.*

then all frequencies appearing in the solution will be unstable. A necessary but by no means sufficient restriction on h is therefore

$$h^2 < \frac{2}{q|a|^2}. \tag{50}$$

Furthermore, note that

$$\sin^2(\tfrac{1}{2}\mu_n h) < (\tfrac{1}{2}\mu_n h)^2 \tag{51}$$

for all nonzero values of $\frac{1}{2}\mu_n h$. Hence a comparison of (49) and (10) shows that the range of unstable wave numbers μ_n in the finite difference method is larger than in the analytical case. In particular, a mode lying on the edge of analytical stability, i.e., $\mu_n^2 = 2q|a|^2$, will always be treated as unstable by the finite difference method, however small h may be.

6. A nonlinear stability analysis. Inspired by the results of Fig. 1, we attempt the following nonlinear side-band analysis of the semidiscrete schemes (20), (23) and (29). We assume initial conditions of form (14) and furthermore restrict ourselves to case (16). That is, we assume that the energy will be transferred mainly between the fundamental mode and the single unstable mode of lowest frequency. We shall neglect the transfer of energy to the higher harmonics, which amounts to taking $N = 2$ in the numerical schemes. This approach should be compared to the analysis of the previous section. In the linear instability analysis all frequencies were considered, but quadratic and cubic products of the side-band amplitudes were ignored. In the present analysis we retain all nonlinearities but consider the generated higher harmonics to be negligible. The pseudospectral scheme (30) is considered first. On an $N = 2$ grid of $[-\pi/\mu, \pi/\mu]$ the discrete transforms (25) and (26) become simply

$$A_0 = \tfrac{1}{2}(U_0 + U_{-1}), \qquad A_{-1} = \tfrac{1}{2}(U_0 - U_{-1}), \tag{52a}$$

$$U_0 = A_0 + A_{-1}, \qquad U_{-1} = A_0 - A_{-1} \tag{52b}$$

where $\mu = \mu_1$. Hence (30) yields

$$\dot{U}_{-1} = \tfrac{1}{2} i\mu^2 (U_0 - U_{-1}) + iq|U_{-1}|^2 U_{-1}, \tag{53a}$$

$$\dot{U}_0 = \tfrac{1}{2} i\mu^2 (U_{-1} - U_0) + iq|U_0|^2 U_0. \tag{53b}$$

The functions $U_{-1}(t)$ and $U_0(t)$ are approximations to $u(-\pi/\mu, t)$ and $u(0, t)$, respectively. The aim is to solve (53) theoretically.

Although (53) may be solved in its present form, we transform it to Fourier space by means of (52) in order to facilitate the comparison with the spectral method. This yields

$$\dot{A}_0 = iq[A_0|A_0|^2 + 8A_0|A_1|^2 + 4A_1^2 A_0^*], \tag{54a}$$

$$\dot{A}_1 = -i\mu^2 A_1 + iq[4A_1|A_1|^2 + 2A_1|A_0|^2 + A_0^2 A_1^*]. \tag{54b}$$

In deriving (54) we have substituted A_1 for $\frac{1}{2}A_{-1}$, and so $A_1(t)$ now represents the approximation to $\hat{u}_1(t) = \hat{u}_{-1}(t)$. According to (14) the initial conditions are

$$A_0(0) = a, \tag{55a}$$

$$A_1(0) = \tfrac{1}{2} a\varepsilon. \tag{55b}$$

For simplicity ε will be considered real and $\varepsilon^2 \ll 1$. Invoking the conservation laws (33) and (35), we obtain two integrals of system (54), namely

$$|A_0|^2 + 4|A_1|^2 = D_1 = |a|^2(1 + \varepsilon^2) \tag{56a}$$

and

$$\mu^2|A_1|^2-\tfrac{1}{2}q(A_0A_1^*+A_1A_0^*)^2=D_2=\tfrac{1}{4}\varepsilon^2|a|^2(\mu^2-2q|a|^2). \tag{56b}$$

We now apply these integrals to (54) to obtain an equation which governs the side-band evolution $|A_1(t)|$. Starting from (54b), we can easily deduce that

$$\frac{d}{dt}|A_1|^2=iq[A_1^{*2}A_0^2-A_1^2A_0^{*2}]. \tag{57}$$

Squaring yields

$$\left(\frac{d}{dt}|A_1|^2\right)^2=-q^2(A_1^{*4}A_0^4+A_1^4A_0^{*4})+2q^2|A_0|^4|A_1|^4. \tag{58}$$

We now turn to (56b) and rewrite it as

$$q(A_0^2A_1^{*2}+A_1^2A_0^{*2})=2(\mu^2|A_1|^2-D_2)-2q|A_0|^2|A_1|^2. \tag{59}$$

When this equation is squared, it is obvious that the first term on the RHS of (58) may be expressed in terms of $|A_0|^2$ and $|A_1|^2$. Finally (56a) is used to eliminate all dependence on $|A_0|^2$ and (58) becomes

$$\dot{\alpha}^2=4(D_2-\mu^2\alpha)(8q\alpha^2+(\mu^2-2qD_1)\alpha-D_2) \tag{60}$$

where $\alpha:=|A_1|^2$.

From (60) it is clear that the side-band evolution is described by an elliptic function which accounts for the periodicity and recurrence. This is in contrast to the linear analysis which predicted exponentially growing side-bands.

To examine the qualitative and quantitative behaviour of the solution of (60) we shall find it convenient to introduce the parameter ϕ defined by

$$\phi:=\frac{\mu^2}{q|a|^2}. \tag{61}$$

We now examine the roots of the RHS of (60). They are

$$\alpha_1=\tfrac{1}{4}\varepsilon^2|a|^2(1-2/\phi), \tag{62a}$$

$$\alpha_2=\tfrac{1}{4}\varepsilon^2|a|^2, \tag{62b}$$

$$\alpha_3=\tfrac{1}{4}|a|^2(1-\tfrac{1}{2}\phi). \tag{62c}$$

The root α_2 is recognized to be the initial value $|A_1(0)|^2$.

First consider $\phi<2$. This corresponds to the linear result (10). However, in this case it does not ensure growth of α. In addition $\alpha_3>\alpha_2$ is required. If we combine this with (16), ϕ is restricted to

$$\tfrac{1}{2}<\phi<2(1-\varepsilon^2), \tag{63}$$

which is an $O(\varepsilon^2)$ modification of the linear result (16).

In this case $|A_1(t)|^2$ grows from its initial value α_2 to its maximum value α_3 before the initial condition recurs. The situation is depicted in Fig. 2(b) for the values (15) that corresponds to Fig. 1.

Next suppose $\phi\geqq 2(1-\varepsilon^2)$. Then either $\alpha_1<0<\alpha_3<\alpha_2$ if $\phi<2$ or $\alpha_3<0<\alpha_1<\alpha_2$ if $\phi<2$. In this case α oscillates between α_2 and whichever value α_3 or α_1 is positive. Since we have assumed $\varepsilon^2\ll 1$ we have $|A_1(t)|\approx|A_1(0)|$ for all time, that is, the solution remains approximately uniform.

Equation (60) also provides an explicit approximation to the recurrence time. This is the elliptic integral

$$T_r(\phi;\varepsilon;q|a|^2)=\frac{1}{\phi^{1/2}q|a|^2}\int_{\varepsilon^2}^{1-\phi/2}\frac{d\beta}{\{[\varepsilon^2(1-2/\phi)-\beta][\beta-\varepsilon^2][2\beta-(2-\phi)]\}^{1/2}} \tag{64}$$

where we have substituted $\beta=4\alpha/|a|^2$.

It is worth noting that for fixed ϕ and ε the integral (64) predicts a recurrence time which is inversely proportional to the strength of the nonlinearity $q|a|^2$. Thus, steeper temporal gradients are predicted for increasing nonlinearities.

The same analysis may be applied to the spectral method (23). The system that governs the evolution of the low frequencies is

$$\dot{A}_0=iq[A_0|A_0|^2+4A_0|A_1|^2+2A_1^2A_0^*], \tag{65a}$$

$$\dot{A}_1=-i\mu^2A_1+iq[2A_1|A_0|^2+3A_1|A_1|^2+A_0^2A_1^*] \tag{65b}$$

where we have used the fact that $A_1(t)=A_{-1}(t)$ if initial conditions of the form (14) are considered. Again we have written μ for μ_1. It is instructive to note that the difference between the systems (65) and (54) is simply aliasing error on the $N=2$ grid.

System (65) has integrals

$$|A_0|^2+2|A_1|^2=D_1=|a|^2(1+\tfrac{1}{2}\varepsilon^2), \tag{66a}$$

$$\mu^2|A_1|^2-\tfrac{1}{2}q[|A_1|^4+(A_0A_1^*+A_0^*A_1)^2]=D_2=\tfrac{1}{4}\varepsilon^2|a|^2[\mu^2-2q|a|^2(1+\tfrac{1}{16}\varepsilon^2)]. \tag{66b}$$

When (66) is substituted into (65) in the same fashion as before, the evolution of the side-bands is found to be described by

$$\dot{\alpha}^2=7q^2(\alpha-\alpha_1)(\alpha-\alpha_2)(\alpha-\alpha_3)(\alpha-\alpha_4) \tag{67}$$

where

$$\alpha=|A_1|^2, \tag{68a}$$

$$\alpha_1=|a|^2(\phi-\Delta^{1/2}), \tag{68b}$$

$$\alpha_2=\tfrac{1}{4}\varepsilon^2|a|^2, \tag{68c}$$

$$\alpha_3=\tfrac{1}{7}|a|^2(4-2\phi+\tfrac{1}{4}\varepsilon^2), \tag{68d}$$

$$\alpha_4=|a|^2(\phi+\Delta^{1/2}), \tag{68e}$$

$$\phi=\frac{\mu^2}{q|a|^2}, \tag{68f}$$

$$\Delta=\phi^2+(1-\tfrac{1}{2}\phi)\varepsilon^2+\tfrac{1}{16}\varepsilon^4. \tag{68g}$$

As before, the solution of (67) is an elliptic function and hence periodic in time. An inspection of the roots of the RHS of (67) reveals that α_2 is the initial value $|A_1(0)|^2$. Also, since we have assumed $\varepsilon^2\ll 1$ and $\frac{1}{2}<\phi<2$ the largest of the four roots (68b)–(68e) is α_4. Hence, whenever

$$\tfrac{1}{2}<\phi<2(1-\tfrac{3}{8}\varepsilon^2) \tag{69}$$

we have the ordering $\alpha_1<0<\alpha_2<\alpha_3<\alpha_4$, which implies that the solution of (67) oscillates between α_2 and α_3. The maximum value which the side-band amplitude $|A_1(t)|$ assumes is thus given by (68d). Again the nonlinear result (69) is an $O(\varepsilon^2)$ modification of the linear result (16).

As with formula (64), an approximation to the recurrence time may be derived from (67) as the elliptic integral

$$T_r(\phi;\ \varepsilon;\ q|a|^2)=\frac{8}{\sqrt{7}\,q|a|^2}\int_{\varepsilon^2}^{\beta_3}\frac{d\beta}{\{(\beta-\beta_1)(\beta-\beta_2)(\beta-\beta_3)(\beta-\beta_4)\}^{1/2}} \tag{70}$$

where

$$\beta_j=\frac{4\alpha_j}{|a|^2},\qquad j=1,\cdots,4.$$

Finally we turn to the nonlinear analysis of the finite difference scheme. On an $N=2$ grid the scheme (20) becomes

$$\dot{U}_{-1}=\frac{2i}{h^2}(U_0-U_{-1})+iq|U_{-1}|^2U_{-1}, \tag{71a}$$

$$\dot{U}_0=\frac{2i}{h^2}(U_{-1}-U_0)+iq|U_0|^2U_0 \tag{71b}$$

where $U_{-1}(t)$ and $U_0(t)$ are approximations to $u(-\pi/\mu,\ t)$ and $u(0,\ t)$ respectively. Since $h=\pi/\mu$, a comparison of (53) and (71) reveals that the analysis given for the pseudospectral method holds identically for the finite difference method, provided μ^2 is replaced by $4\mu^2/\pi^2$. Hence the finite difference method predicts a recurrence time of

$$T_r\left(\frac{4}{\pi^2}\phi;\ \varepsilon;\ q|a|^2\right) \tag{72}$$

where T_r is defined by (64). Also the maximum value of $|A_1(t)|^2$ is approximated by

$$\alpha_3=\tfrac{1}{4}|a|^2(1-2\phi/\pi^2). \tag{73}$$

An important point to be noticed is that the analysis of the finite difference method predicts growth in the side-bands when

$$\mu^2<\tfrac{1}{2}\pi^2q|a|^2(1-\varepsilon^2). \tag{74}$$

Thus the linear result (10) is not recovered when ε^2 is ignored. This is in agreement with the remarks below (51), namely that the finite difference method allows a larger range of unstable frequencies than the actual problem.

In this section we have presented an analysis which gives a qualitative as well as quantitative description of recurrence in semidiscrete approximations of the NLS. It improves on the linear analysis in the sense that the periodicity of the solution is demonstrated and an explicit approximation to the recurrence time is obtained. Bearing in mind that the quantitative results were obtained on an $N=2$ grid, it is to be expected that the numbers may be inaccurate. In the next section the validity of these results will be investigated numerically.

7. Numerical experiments. We commence this section by an investigation into the accuracy of the approximations derived in the previous section. In Table 1(a) the actual recurrence time is compared with the recurrence times (64), (70) and (72) as predicted by the two-mode analysis. In Table 1(b) the maximum value of $|A_1(t)|$ as computed by (62c), (68d) and (73) is compared to the actual value. All results are for the choice $\varepsilon=1/10$, $q|a|^2=\frac{1}{2}$ with values of ϕ ranging from $\frac{1}{2}$ to 2. For the computation of the actual solution a pseudospectral method with $N=32$ was found to yield sufficient

TABLE 1

A comparison of: (a) *The actual recurrence time and the predicted recurrence time as computed by* (64), (70) *and* (72). (b) *The actual maximum value of* $|A_1(t)|$ *and the predicted value as computed by* (62c), (68d) *and* (73).

(a) Recurrence times

ϕ	0.5	0.75	1.00	1.25	1.50	1.75	2.00
Actual	13.1	13.0	13.3	14.1	15.6	19.3	∞
Spectral Method	16.3	14.5	13.9	14.1	15.2	18.2	∞
Pseudospectral Method	13.1	12.2	12.0	12.3	13.2	15.8	∞
Finite Difference Method	16.3	14.7	13.7	13.1	12.6	12.3	12.1

(b) Maximum side-band growth $|A_1(t)|$

ϕ	0.5	0.75	1.00	1.25	1.50	1.75	2.00
Actual	0.287	0.301	0.294	0.271	0.233	0.173	0.025
Spectral Method	0.327	0.299	0.267	0.232	0.189	0.134	0.025
Pseudospectral Method	0.217	0.198	0.177	0.153	0.125	0.088	0.025
Finite Difference Method	0.237	0.230	0.223	0.216	0.209	0.201	0.193

accuracy. For the time integration, the tolerance in the Runge-Kutta-Merson procedure [27] was set to conserve the quantities (33) and (35) to a relative accuracy of 10^{-6}.

The solution corresponding to the choice $\phi = 1$ in Table 1 is also presented graphically in Figs. 2(b)-(d). These figures show an extra drain on the energy of the fundamental mode which is not accounted for by our approximations. This is of course due to the effects of the higher harmonics $|\hat{u}_{\pm 2}|$, $|\hat{u}_{\pm 3}| \cdots$ which were neglected in the two-mode approach.

We briefly comment on the results shown in Table 1. The agreement between the actual recurrence times and maximum value of $|A_1(t)|$ as compared to our approximations is good. The results, however, do not provide a clear-cut criterion to which method is best. Nevertheless it indicates that the analysis of the finite difference method does not produce satisfactory results when $\phi \approx 2$. The reason for this is outlined in the paragraph that follows (74). Despite this fact, it is remarkable how well the true behaviour is simulated by the finite difference method on an $N = 2$ grid when $\phi = 1$ (Fig. 2(d)).

Some effects of aliasing can also be deduced by comparing the results of the spectral and pseudospectral methods. It appears that aliasing error tends to decrease the recurrence time and also to decrease the maximum growth in the side-bands. In general it seems that the spectral method provides the most accurate description of the true behaviour. This is, of course, not unexpected, since it is the only method free of aliasing error.

A drawback of the analysis of § 6 is the restriction to a single unstable mode. To extend the analysis to include the interplay between different unstable modes, the case $N > 2$ will have to be considered. The analysis of this more general situation is at present not within our grasp.

We can, however, investigate numerically the convergence of the methods as the number of degrees of freedom, N, in the space discretization is increased. For this purpose the time integration was performed very accurately, and so the difference between the computed and the accurate solution may be taken to depend entirely on

N. We take as model problem the solution depicted in Fig. 1. The error E is computed as the Fourier space equivalent of the L_2 error, defined by

$$E^2 = L\left(\sum_{n=-N/2}^{N/2} |A_n - \hat{u}_n|^2 + \sum_{|n|>N/2} |\hat{u}_n|^2 \right). \tag{75}$$

The accurate solution $\hat{u}_n$ was computed by a spectral method with $N = 32$. In Fig. 3, the error E is depicted as a function of N for each of the three methods. The error is computed at time $t \approx 20$, i.e., when the unstable mode reaches its maximum for the second time.

We comment briefly on the results of Fig. 3. Of the three methods the spectral method converges most rapidly to the accurate solution, which is indicative of exponential convergence. For $N = 12$ the spectral method is practically exact, which could have been predicted from the results of Fig. 1(b). For values $N \leqq 10$ the pseudospectral method does not exhibit a tendency to converge. For these values of N aliasing is not negligible, and to complicate matters the generated high-frequency contributions are folded onto the unstable low frequency, $|n| = 1$. However, once a sufficient number of frequencies is included to cover all effective modes, aliasing becomes negligible and the method converges rapidly. The finite difference method exhibits a similar pattern, namely that of aliasing preventing convergence in the region $N \leqq 16$, before it starts to converge as $O(1/N^2)$.

The results of Fig. 3 can be taken as fairly typical of the situation when μ^2 is restricted to values in the range (16). For values of μ^2 in the range (17) the tendency for the energy to spread is greater, and we will have to include a larger number of frequencies in the computation to achieve convergence. The general effects of aliasing, however, remain the same.

8. Conclusions. The objective of this paper was to investigate the use of numerical methods for a study of recurrence in the NLS. Three semidiscrete methods were considered, namely a spectral, a pseudospectral and a finite difference method. All three methods were shown to conserve discrete analogues of two of the conservation

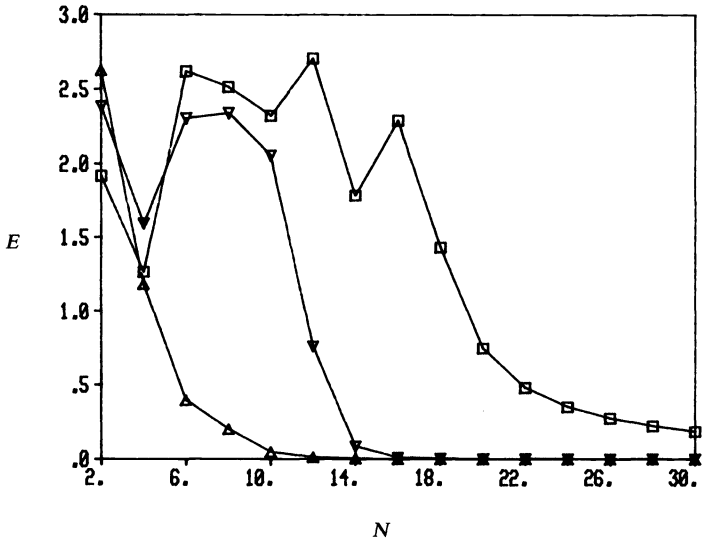


FIG. 3. *The L_2-error* (74) *versus the number of degrees of freedom in the space discretization for* Δ: *The spectral method* (23); ▽: *The pseudospectral method* (29); □: *The finite difference method* (20).

laws satisfied by the NLS. This property is very desirable for the numerical simulation of recurrence in the NLS, since the conservation laws under consideration are known to be sufficient to prove nonthermalization and necessary for recurrence in the analytical case (Thyagaraja [6]–[8]).

The spectral and pseudospectral methods were shown to possess the exact Benjamin–Feir instability limit. In contrast, the finite difference method was shown to have a larger region of instability. In particular, it was found that a frequency which lies on the theoretical stability limit becomes numerically unstable if the finite difference method is used. A nonlinear two-mode analysis confirmed these observations. The finite difference method should therefore be used with care, and a small spatial discretization is required in order to restrict the number of undesirable unstable modes. The results of this paper thus support the conclusions drawn by Taha and Ablowitz [23], namely that there is much to say in favour of the spectral methods for the solution of nonlinear dispersive wave problems.

Apart from computational considerations, the main difference between the spectral and pseudospectral methods is the aliasing error in the latter. Since the higher frequencies created by the nonlinear interactions are reflected onto unstable lower frequencies, aliasing error in this application has a marked effect on the quality of the solution (see Fig. 3). In particular, the nonlinear two-mode analysis showed that aliasing tend to decrease the recurrence time as well as the maximum value of the side-band amplitudes. The use of de-aliasing procedures is therefore advised.

Finally, we briefly comment on time integration procedures for the semidiscrete schemes. The Runge–Kutta–Merson scheme which was used in this paper is generally considered to be expensive and one-step methods are usually used. One of the main guidelines to the selection of a particular method would be its conservation properties, since the conservation of the quantities c_1 and c_2 is important in the simulation of recurrence. This points to the modified Crank–Nicolson scheme of Delfour et al. [22] as the judicious choice, since it conserves both quantities. However, it is a fully implicit method and thus also computationally expensive. No explicit method is capable of conserving any of the quantities. This is undesirable since the nonconservation of c_1 is known to lead to nonlinear blowup [14], [21]. The only implicit-explicit method known to satisfy c_1 is the split-step method of Tappert (see [28], [15]). However, it does not conserve c_2. It is possible that schemes which conserve c_1 but not c_2 may leak energy from the low to the high frequencies, i.e., numerical thermalization. This is partly substantiated by the results of [15]. Investigations in this direction are currently being undertaken.

Although the stability analyses in this paper are restricted, either to a linearized situation or to a limited number of degrees of freedom, it does provide valuable clues to the behaviour of these numerical methods in more general circumstances.

Acknowledgments. We have benefited from discussions with Professor J. Ll. Morris and are grateful to Mr. M. F. Maritz for assistance with the computer graphics.

REFERENCES

[1] E. FERMI, J. PASTA AND S. ULAM, *Studies of nonlinear problems* I. Reprinted in Nonlinear wave motion, A. C. Newell, ed., American Mathematical Society, Providence, RI, 1974.

[2] N. J. ZABUSKY AND M. D. KRUSKAL, *Interaction of "solitons" in a collisionless plasma and the recurrence of initial states*, Phys. Rev. Lett., 15 (1965), pp. 240–243.

[3] B. FORNBERG AND G. B. WHITHAM, *A numerical and theoretical study of certain nonlinear wave phenomena*, Philos. Trans. Roy. Soc. London Ser. A, 289 (1978), pp. 373–404.

[4] H. C. YUEN AND W. E. FERGUSON, *Relationship between Benjamin-Feir instability and recurrence in the nonlinear Schrödinger equation*, Phys. Fluids, 21 (1978), pp. 1275-1278.

[5] T. B. BENJAMIN AND J. E. FEIR, *The disintegration of wave trains on deep water. Part* 1. *Theory*, J. Fluid Mech., 27 (1967), pp. 417-430.

[6] A. THYAGARAJA, *Recurrent motions in certain continuum dynamical systems*, Phys. Fluids, 22 (1979), pp. 2093-2096.

[7] ———, *Recurrence, dimensionality and Lagrange stability of solutions of the nonlinear Schrödinger equation*, Phys. Fluids, 24 (1981), pp. 1973-1975.

[8] ———, *Recurrence phenomena and the number of effective degrees of freedom in nonlinear wave motions*, in Nonlinear Waves, L. Debnath, ed., Cambridge, Univ. Press, Cambridge, 1983.

[9] E. INFELD, *Quantitative theory of the Fermi-Pasta-Ulam recurrence in the nonlinear Schrödinger equation*, Phys. Rev. Lett., 47 (1981), pp. 717-718.

[10] ———, *Nonlinear waves: from hydrodynamics to plasma theory*, in Advances in Nonlinear Waves, L. Debnath, ed., Pitman, Boston, 1984.

[11] M. STIASSNI AND U. I. KROSZYNSKI, *Long-time evolution of an unstable water-wave train*, J. Fluid Mech., 116 (1982), pp. 207-225.

[12] S. A. ORSZAG, *Comparison of pseudospectral and spectral approximations*, Stud. Appl. Math., 51 (1972), pp. 253-259.

[13] B. M. HERBST, A. R. MITCHELL AND J. LL. MORRIS, *Numerical experience with the nonlinear Schrödinger equation*, J. Comp. Phys., 15 (1985), pp. 263-281.

[14] B. M. HERBST, A. R. MITCHELL AND J. A. C. WEIDEMAN, *On the stability of the nonlinear Schrödinger equation*, J. Comput. Phys., 15 (1985), pp. 282-304.

[15] J. A. C. WEIDEMAN AND B. M. HERBST, *Split-step methods for the solution of the nonlinear Schrödinger equation*, SIAM J. Numer. Anal., 23 (1986), pp. 485-507.

[16] D. F. GRIFFITHS, A. R. MITCHELL AND J. LL. MORRIS, *A numerical study of the nonlinear Schrödinger equation*, Comp. Meth. Appl. Mech., 45 (1984), pp. 177-215.

[17] A. R. MITCHELL AND J. LL. MORRIS, *A self adaptive difference scheme for the nonlinear Schrödinger equation*, Arab Gulf J. Sci. Res., 1 (1983), pp. 461-472.

[18] J. M. SANZ-SERNA, *An explicit finite difference scheme with exact conservation properties*, J. Comp. Phys., 52 (1982), pp. 273-289.

[19] J. M. SANZ-SERNA AND V. S. MANORANJAN, *A method for the integration in time of certain partial differential equations*, J. Comput. Phys., 52 (1983), pp. 273-289.

[20] J. M. SANZ-SERNA, *Methods for the numerical solution of the nonlinear Schrödinger equation*, Math. Comp., 43 (1984), pp. 21-27.

[21] J. M. SANZ-SERNA AND J. G. VERWER, *Conservative and nonconservative schemes for the solution of the nonlinear Schrödinger equation*, IMA J. Numer. Anal., 6 (1986), pp. 25-42.

[22] M. DELFOUR, M. FORTIN AND G. PAYRE, *Finite difference solutions of a nonlinear Schrödinger equation*, J. Comput. Phys., 44 (1981), pp. 277-288.

[23] T. R. TAHA AND M. J. ABLOWITZ, *Analytical and numerical aspects of certain nonlinear evolution equations.* II. *Numerical, Nonlinear Schrödinger equation*, J. Comput. Phys., 55 (1984), pp. 203-230.

[24] R. T. GLASSEY, *On the blowing up of solutions to the Cauchy problem for nonlinear Schrödinger equations*, J. Math. Phys., 18 (1977), pp. 1794-1797.

[25] H. SCHAMEL AND K. ELSÄSSER, *The application of the spectral method to nonlinear wave propagation*, J. Comput. Phys., 22 (1976), pp. 501-516.

[26] G. S. PATTERSON JR. AND S. A. ORSZAG, *Spectral calculations of isotropic turbulence: Efficient removal of aliasing interactions*, Phys. Fluids, 14 (1971), pp. 2538-2541.

[27] D02BBF, NAG Software Library, Mk 7: Dec 78.

[28] F. D. TAPPERT, *Numerical solutions of the Korteweg-de Vries equation and its generalizations by the split-step Fourier method*, AMS Lecture Notes in Appl. Math., 13 (1974), pp. 215-216.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 6, November 1987
© 1987 Society for Industrial and Applied Mathematics
010

CALCULATING FOURIER TRANSFORMS OF LONG-TAILED FUNCTIONS*

J. N. LYNESS† AND TASSO J. KAPER†

Abstract. In this paper, we describe a method for evaluating Fourier Transform functions numerically when function values $f(x)$ are available for any value of x. Our method is based on the Möbius inversion of the Poisson summation formula, and employs a nonlinear series acceleration technique. Numerical evidence suggests that, for relatively smooth functions having long tails, this approach may be very useful.

Key words. Fourier transform, long-tailed functions, Fast Fourier Transform, Poisson summation formula, trapezoidal quadrature rule, nonlinear series acceleration

AMS(MOS) subject classifications. 65B05, 65R10, 65B20, 42A38

1. Introduction. In this paper we describe a method for calculating Fourier transform functions

$$g_c(y) = \int_{-\infty}^{\infty} f(x) \cos xy \, dx, \qquad g_s(y) = \int_{-\infty}^{\infty} f(x) \sin xy \, dx \tag{1.1}$$

in terms of function values of $f(x)$. We shall assume for convenience that $f(x)$ is a real bounded function of a real variable x and that the Fourier transforms

$$g_e(y) = g_c(y) - ig_s(y) \tag{1.2}$$

exist for all y. While many of the formulas on which this and other methods are based do not require some of these conditions, in practice the successful use of such methods seems to require even more stringent conditions.

The method described here is based on the Möbius inversion of the Poisson summation formula. This technique seems to have been first mentioned by Duffin (1941). A fundamental paper on this theory is by Goldberg and Varga (1956), and its first computational use is described in Stetter (1966). In Lyness (1970) and subsequent papers by the same author, an extension of this method for systematic evaluation of Fourier coefficients is described. A brief, self-contained description of the part of this theory we require is given in § 3.

The method requires the evaluation of alternating sums $W(h)$ of $f(x)$ (see (3.3) below) for several values of h. This would be unduly expensive unless series acceleration methods (such as the epsilon algorithm) were to be used. Information about the reliability and performance of such methods is largely based on numerical experiments. The work of Smith and Ford (1979), (1982) and Fessler, Ford and Smith (1983) has been extremely useful to the present authors, not so much from the point of view of understanding the behavior of such processes as in being able to predict it on the basis of past performance.

Our results, presented in § 5, show clearly that for smooth functions $f(x)$ with long tails, the method described here may be more cost-effective than the Fast Fourier Transform (FFT) by several orders of magnitude. For functions without long tails, the FFT is more efficient than this method. However, for functions with significant oscillating components, our method should only be used with human supervision. While it

* Received by the editors June 23, 1986; accepted for publication (in revised form) February 4, 1987. This work was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under contract W-31-109-Eng-38.

† Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois 60439-4844.

works well most of the time for these functions, occasional flamboyant failures may occur. It seems to the present authors that significantly more work on the behavior of series acceleration processes will be necessary before we can understand fully how these particular cases should be handled.

In order to have some standard against which to measure the cost-effectiveness of this method, we describe briefly in § 2 the sort of results one can expect from the Fast Fourier Transform.

2. The Fast Fourier Transform and long tailed functions. For a wide class of functions $f(x)$, an efficient approach to the computation of Fourier transforms is simply to apply the trapezoidal rule to approximate the integral (1.1). A general trapezoidal rule applied to the cosine transform has the form

$$\hat{g}_c(y) = h \sum_{j=-\infty}^{\infty} f(jh + x_0) \cos((jh + x_0)y) \tag{2.1}$$

where h is the step length and x_0 a parameter which allows translation. One of the reasons that this approximation is popular is that, by proper choice of parameters mentioned below, one can use the Fast Fourier Transform algorithm (FFT). This algorithm is possibly one of the most cost-effective algorithms ever constructed.

We now outline some of the steps which have to be taken in order to be able to apply the FFT. We choose limits A and B at which to truncate the infinite sum. We choose an integer m in terms of which the step length is

$$h = (B - A)/m \tag{2.2}$$

and (for convenience) we set $x_0 = A$. When these choices (A, B and m) are made, we may apply the Fast Fourier Transform algorithm, which will yield approximations to precisely m different Fourier transforms; when m is even these are

$$\begin{aligned} &\hat{g}_c(y_s), \qquad s = 0, 1, \cdots, m/2, \\ &\hat{g}_s(y_s), \qquad s = 1, 2, \cdots, (m/2) - 1 \end{aligned} \tag{2.3}$$

where

$$y_s = \frac{2\pi s}{B - A}.$$

Here

$$\hat{g}_c(y_s) = h\left(\frac{(f(A) + f(B))}{2} \cos Ay_s + \sum_{j=1}^{m-1} f(jh + A) \cos((jh + A)y_s)\right)$$

and $\hat{g}_s(y_s)$ is defined in a similar way.

Different FFT algorithms may differ in detail from the description given above. However, they all have in common the following feature. To use any FFT algorithms in this context, we must assign A, B and m. Then we obtain m results at $y = y_s$ $s = 0, 1, \cdots, m/2$. While for many functions this is quite a reasonable procedure, for some, namely the long-tailed ones, it turns out to be cumbersome and expensive. We find this as soon as we assign numerical values to A, B and h.

Let us consider the long-tailed function, and its transform

$$f(x) = \frac{1}{1 + x^2}, \qquad g_c(y) = \pi e^{-|y|}. \tag{2.4}$$

If we wish to calculate $g_c(y)$ from function values of $f(x)$, the long tail imposes a large value of B or $-A$ and this results in a large value of m. A "back of an envelope" calculation shows that to obtain low accuracy results, say $\varepsilon = 10^{-2}$, we need $B = -A \simeq 200$ and $m \simeq 720$. However, these numbers increase rapidly as the required accuracy decreases. For $\varepsilon = 10^{-3}$ we need $B = 2000$ and $m \simeq 10{,}000$ and for $\varepsilon = 10^{-6}$ we find the ridiculous estimates $B = 2 \times 10^6$ and $m \simeq 1.9 \times 10^7$. Even when the data manipulation is fast, we do not want to handle a problem of this size if we can avoid it. The method described later, based on the Poisson summation formula, handles this problem efficiently, obtaining ten-figure accuracy using 169 function values.

We must hasten to add that the trapezoidal rule approach requires this large number of function values only when $f(x)$ has a long tail. The similar problems

$$f(x) = \operatorname{sech} x, \qquad g(y) = \pi \operatorname{sech} y/2, \tag{2.5}$$

$$f(x) = e^{-x^2}, \qquad g(y) = \sqrt{\pi}\, e^{-y^2/4} \tag{2.6}$$

are handled efficiently by the FFT. Ten-figure accuracy may be obtained using $m = 200$ and $m = 32$, respectively.

In § 5 we discuss in more detail the functions for which the new method is suitable. It is suitable for (2.5) and (2.6) above, and the cost is of the same order as in (2.4). However, the FFT gives more complete results, is easier to implement, and the cost in terms of function values is either comparable (2.5) or much less (2.6). It is when $f(x)$ has a long tail and $g(y)$ does not that the method described here is preferable.

3. The Poisson Summation Formula, variants and inversion. The Poisson Summation Formula is a classical formula. A relatively straightforward proof is given in Titchmarsh (1948). It relates an infinite sum of function values with an infinite sum of transform values as follows.

$$h \sum_{j=-\infty}^{\infty} f(jh) = \sum_{s=-\infty}^{\infty} g_e(2\pi s/h). \tag{3.1}$$

Since the transforms $g_e(y)$ decrease with increasing y, the general approach will be to calculate numerically approximations to the sums on the left for several values of h, and use these to infer the values of leading elements on the right. We shall deal with real $f(x)$, and introduce some notation for offset alternating sums W. Specifically, we define the trapezoidal rule operator

$$T(h; x_0; f) = h \sum_{j=-\infty}^{\infty} f(jh + x_0) \tag{3.2}$$

and an alternating sum operator

$$\begin{aligned} W(h; x_0; f) &= h \sum_{j=-\infty}^{\infty} (-1)^j f(jh + x_0) \\ &\equiv T(2h; x_0; f) - T(h; x_0; f). \end{aligned} \tag{3.3}$$

If we use standard translation techniques, it follows from (3.1) that

$$T(h; x_0; f) = \sum_{s=-\infty}^{\infty} \exp\left(\frac{2\pi i s}{h} x_0\right) g_e\left(\frac{2\pi s}{h}\right), \tag{3.4}$$

and

$$W(h; x_0; f) = \sum_{\substack{s=-\infty \\ s \text{ odd}}}^{\infty} \exp\left(\frac{\pi i s}{h} x_0\right)\left(g_e\left(\frac{\pi s}{h}\right)\right). \tag{3.5}$$

We recall that $f(x)$ is real and so $g_c(y)=g_c(-y)$ and $g_s(y)=-g_s(-y)$. Using this we may write some special cases of (3.4) and (3.5) that particularly interest us in the form

$$T(h;0;f)=g_c(0)+2\sum_{s=1}^{\infty} g_c\left(\frac{2\pi s}{h}\right), \tag{3.6}$$

$$W(h;0;f)=\sum_{\substack{s=1\\ s\text{ odd}}}^{\infty} 2g_c\left(\frac{\pi s}{h}\right), \tag{3.7}$$

$$W\left(h;\frac{h}{2};f\right)=\sum_{\substack{s=1\\ s\text{ odd}}}^{\infty} 2g_s\left(\frac{\pi s}{h}\right)(-1)^{(s-1)/2}. \tag{3.8}$$

The Möbius inversion of these formulas is described in some detail in Stetter (1966) and in Lyness (1970). In a simpler finite case in which $g_c(\pi s/h)=0$ for all s greater than some cutoff value, the inversion of (3.7) reduces to the inversion of a finite upper triangular matrix and (3.11) and (3.12) below are readily verified. In the general case, i.e., producing the general set of formulas (3.11) from the set (3.7) above, we need justification along the lines given in the references (see, for example, Hardy and Wright (1954)). The coefficients in (3.11)-(3.13) turn out to be Möbius numbers μ_j defined by

$$\begin{aligned} &\mu_j=0 && \text{when } j \text{ has a square factor other than 1.}\\ &\mu_j=(-1)^k && \text{where } k \text{ is the number of distinct prime factors of } j. \end{aligned} \tag{3.9}$$

(The first twelve Möbius numbers are 1, −1, −1, 0, −1, 1, −1, 0, 0, 1, −1, 0.) The inverted forms of (3.6), (3.7) and (3.8) are respectively,

$$2g_c(y)=\sum_{k=1}^{\infty}\mu_k\left(T\left(\frac{2\pi}{yk};0;f\right)-g_c(0)\right), \tag{3.10}$$

$$2g_c(y)=\sum_{\substack{k=1\\ k\text{ odd}}}^{\infty}\mu_k W\left(\frac{\pi}{yk};0;f\right), \tag{3.11}$$

$$2g_s(y)=\sum_{\substack{k=1\\ k\text{ odd}}}^{\infty}\mu_k(-1)^{(k-1)/2}W\left(\frac{\pi}{yk};\frac{\pi}{2yk};f\right). \tag{3.12}$$

The latter two form the basis of our method for calculating these transforms.

4. Outline of the calculation. Let us consider formula (3.11) with a view to its use in calculating a particular Fourier cosine transform $g_c(y_0)$ with $y_0>0$. We set

$$h_0=\pi/y_0 \tag{4.1}$$

and write this formula in the form

$$2g_c(y_0)=\sum_{\substack{k=1\\ k\text{ odd}}}^{\infty}\mu_k W(h_0/k) \tag{4.2}$$

where we have abbreviated $W(h_0/k;0;f)$ by $W(h_0/k)$. In a calculation, it will be necessary to use some sort of approximation $\tilde{W}(h_0/k)$ for each infinite sum $W(h_0/k)$. It will also be necessary to truncate the infinite sum in (4.2). Thus we shall in fact calculate

$$2\tilde{g}_c(y_0)=\sum_{\substack{k=1\\ k\text{ odd}}}^{\bar{k}}\mu_k\tilde{W}(h_0/k). \tag{4.3}$$

Let us suppose that each approximation satisfies

$$|\tilde{W}(h) - W(h)| \leqq \varepsilon_W \tag{4.4}$$

and that $\bar{k}$ is chosen so that

$$\sum_{\substack{k=\bar{k}+2 \\ k \text{ odd}}}^{\infty} |W(h_0/k)| \leqq \varepsilon_T. \tag{4.5}$$

Then it follows immediately that

$$2|\tilde{g}_c(y_0) - g_c(y_0)| \leqq K_1 \varepsilon_W + \varepsilon_T \tag{4.6}$$

where

$$K_1 = \sum_{\substack{k=1 \\ k \text{ odd}}}^{\bar{k}} |\mu_k| \leqq (\bar{k}+1)/2. \tag{4.7}$$

This approximation $\tilde{g}_c(y_0)$ to $g_c(y_0)$ is based on most of the quantities $\tilde{W}(h_0)$, $W(h_0/k), \cdots, \tilde{W}(h_0/\bar{k})$. Clearly, approximations to $g_c(jy_0)$ j odd can be based on a subset of these quantities. That is, we may define

$$2\tilde{g}_c(jy_0) = \sum_{\substack{k \text{ odd} \\ 1 \leqq k \leqq \bar{k}/j}} \mu_k \tilde{W}(h_0/jk). \tag{4.8}$$

It follows that each of these approximations satisfy

$$2|\tilde{g}_c(jy_0) - g_c(jy_0)| \leqq K_j \varepsilon_W + \varepsilon_T \tag{4.9}$$

where

$$K_j = \sum_{\substack{k \text{ odd} \\ 1 \leqq k \leqq \bar{k}/j}} |\mu_k| \leqq [(\bar{k}+j)/2j] \leqq \frac{\bar{k}}{2j} + \frac{1}{2} \tag{4.10}$$

and $K_j \leqq K_{j-1} \leqq \cdots \leqq K_1$ and ε_W and ε_T are the same quantities which occur in (4.6). Consequently, an implementation of this method, based on calculating $\tilde{W}(h_0/k)$ $k = 1, 3, \cdots, \bar{k}$, should produce approximations $\tilde{g}_c(jy_0)$ $j = 1, 3, \cdots, \bar{k}$ of roughly the same quality.

We are interested in whether this approach is cost-effective. To this end, it is convenient to think in terms of a range $|y| \leqq Y$ of interest of $g_c(y)$. Since $g_c(y)$ is integrable and bounded, such a range could be defined in terms of the required accuracy ε, as the smallest values of Y for which

$$|g_c(y)| < \varepsilon \quad \forall |y| > Y.$$

Now, in view of (3.7), it appears that for large y the function $W(2\pi/y)$ has the same behavior as $2g_c(y)$. So, if we take

$$y_0 = Y/\bar{k}, \tag{4.11}$$

we should find that the sequence $W(h_0/k)$ $k = 1, 3, \cdots$, falls below 2ε for $k > \bar{k}$. That is because $h_0/\bar{k} = 2\pi/y_0\bar{k} = 2\pi/Y$. To sum up, we obtain a roughly uniform coverage of $g_c(y)$ involving $(\bar{k}+1)/2$ distinct values of y for the cost of $(\bar{k}+1)/2$ evaluations of $W(h)$. And the accuracy of each approximation is determined by (4.9), where the value of ε_T has to be estimated from the ultimate values of $W(h)$.

The critical cost, then, is that required to evaluate approximations to $W(h)$. This is treated in the next section.

5. The epsilon algorithm and results. The cost-effective use of this simple technique depends critically on our ability to calculate

$$W(h) = h \sum_{j=-\infty}^{\infty} (-1)^j f(jh) \tag{5.1}$$

economically. As mentioned in the section on the FFT, these sums of function values can be very expensive when $f(x)$ has a long tail. For example, with $h = 1$ and

$$f(x) = 1/(1+x^2), \tag{5.2}$$

one thousand function values are required before $f(x)$ falls below 10^{-6} and truncating at this stage may leave an error of about 10^{-4}. When h is smaller than 1, even more function values are required to sum this series directly to this sort of accuracy. However, if we use the epsilon algorithm to accelerate the convergence of this series with $h = 1$, we can attain eight-figure accuracy using 13 terms, and twelve-figure accuracy using 16 terms. Moreover, with other values of h, the same sort of results appear. That is to say, there is a saving in cost of many orders of magnitude over what the unversed numerical analyst might expect.

In example (5.2), using $y_0 = \pi/2$, we evaluated $g_c(jy_0)$ $j = 1, 3, 5, \cdots, 19$, these being the only ones greater than 10^{-13} in magnitude using 330 function values. (No advantage of symmetry was taken.)

Incidentally, there is only marginal scope for the re-use of function values required by two or more different sums $W(h)$. A rough estimate indicates that about a 10% saving in function evaluation would result from retaining function values in case they were needed again. We found similar results for the sine Fourier transform $g_s(y)$ using (3.12).

We are able to program short subroutines for this calculation in which the routine returns its own estimate of the error. We used the QUADPACK epsilon algorithm QELG (or QEXT) (see Piessens (1983)) which is ultracautious and bases its stopping criterion on whether three previous reasonable approximations are available. This usually results in two unnecessary terms being calculated and an accuracy margin too stringent by factors of between 10^2 and 10^3. We also used a somewhat cautious stopping criterion for the sequence of $W(h/j)$, demanding three "reductions" in $|W(h/j)|$ after $|W(h/j)|$ had become less than ε. ($W(h/j)$, like $g_c(y)$, is occasionally sinusoidal and this guards to some extent against precipitate termination near a local zero.)

We also used our routines for more sophisticated examples. At no time did these routines fail in the sense of producing a result less accurate than that claimed. This is probably because the formula on which it is based is so simple. However, for some oscillatory functions, for certain values of h the series for $W(h)$ consists of long sequences of positive terms alternating with equally long sequences of negative terms. Such series are notoriously difficult to sum numerically, and the epsilon algorithm did not handle these series efficiently. With no cutoff, this could have involved thousands of function values instead of about thirty. With a cutoff of fifty function values in place, this had the effect that those values $g_c(jy)$ which required this particular $W(h)$ were inaccurate—and reported to be inaccurate. The values $g_c(jy)$ for other values of j were, of course, unaffected.

Our view is that further investigation of these features is required. At the moment we can assert only that this method works well automatically for smooth nonoscillatory

functions $f(x)$, such as linear combinations of scaled and shifted versions of functions (5.1), (2.5) and (2.6); it can be made to work under human supervision for a very much wider class of functions.

Acknowledgment. We are both grateful to Dr. Hans Stetter for his helpful advice about the presentation of this material.

REFERENCES

R. J. DUFFIN (1941), *Möbius transforms and Fourier transforms*, Bull. Amer. Math. Soc., 47, p. 383.

T. FESSLER, W. F. FORD AND D. A. SMITH (1983), HURRY: *An acceleration algorithm for scalar sequences and series*, ACM Trans. Math. Software, 9, pp. 346-354.

R. R. GOLDBERG AND R. S. VARGA (1956), *Moebius inversion of Fourier transforms*, Duke Math. J., 23, pp. 553-559.

E. C. HARDY AND E. M. WRIGHT (1954), *An Introduction to the Theory of Numbers*, Clarendon Press, Oxford, p. 237.

J. N. LYNESS (1970), *The calculation of Fourier coefficients by the Möbius inversion of the Poisson summation formula, Part* I., Math. Comp., 24, pp. 101-135.

R. PIESSENS, E. DE DONCKER-KAPENGA, C. W. ÜBERHUBER AND D. K. KAHANER (1983), *Quadpack*, Springer-Verlag, Berlin-New York, p. 258.

D. A. SMITH AND W. F. FORD (1979), *Acceleration of linear and logarithmic convergence*, SIAM J. Numer. Anal., 16, pp. 223-240.

———, (1982), *Numerical comparisons of nonlinear convergence accelerators*, Math. Comput., 38, pp. 481-499.

H. STETTER (1966), *Numerical approximation of Fourier transforms*, Numer. Math., 8, pp. 235-249.

E. C. TITCHMARSH (1948), *Introduction to the Theory of Fourier Integrals*, Clarendon Press, Oxford, p. 60.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 6, November 1987
© 1987 Society for Industrial and Applied Mathematics
011

AN ALGORITHM FOR CONSTRAINED INTERPOLATION*

LARS-ERIK ANDERSSON† AND TOMMY ELFVING†‡

Abstract. A Newton type algorithm is proposed for the solution of the following problem: Find a function of one variable which interpolates given points and whose kth derivative is greater than or equal to a given number δ and has minimal L^p-norm.

Key words. convex spline interpolation, nonlinear system of equations, iterative methods

AMS(MOS) subject classifications. 65D05, 65D07, 41A29

1. Introduction. When dealing with the common problem of interpolating given datapoints, we often use spline functions. However one difficulty which sometimes occurs is that the resulting spline interpolant has oscillations not supported by the data. This could, e.g., be the case when interpolating a function known to be convex and when the observed datavalues are not evenly distributed over the interval of interest.

In the present work we consider the efficient solution of the following interpolation problem. Find a function of one variable which interpolates given points and whose kth derivative is greater than or equal to some given number δ and has minimal L^p-norm. Here $k \geqq 2$ and $1 < p < \infty$ is assumed throughout the report. For practical applications the most interesting case is perhaps $k = 2$, $\delta = 0$ giving convex interpolation. Note, for this case, that the interpolant is a straight line on intervals where the constraint is active. This might be an unwanted effect, which is the reason for considering $\delta \neq 0$. An even more general case is when $\delta = \delta(t)$, a piecewise linear function of t. This increased flexibility could, e.g., be used to improve the approximation of functions having strongly increasing (or even singular) second (for the case $k = 2$) derivatives; cf. the numerical example in § 4. We also mention the possibility of using both upper and lower bounds on the derivative. This can be useful when approximating functions which are convex in part of the interval and concave in some other parts (see [6]).

In a previous paper [1], the problem of constrained interpolation was treated from a theoretical point of view and those results are used here to construct a numerical algorithm for the problem. The theoretical problem has been treated earlier by several authors. For instance the case $k = p = 2$ was considered by Hornung [7] and the case $k = 2$, $1 < p \leqq \infty$ by Iliev and Pollul [8], [9]. More recently the problem was treated in [12].

We now briefly mention some related approaches to convex and/or monotone interpolation. McAllister and Roulier [11] and more recently Shumaker [16] present algorithms for interpolation by shape preserving quadratic splines with knots at the given interpolation points and at most one additional knot between each interpolation point. Other methods for introducing additional knots are described in de Boor [3, Chap. 16]. In [13] and [2], the problem of computing a convex cubic C^1-spline, using fixed knots, is considered. This problem may not always have a solution (cf. also [15]). A recent paper on convex rational interpolation is [5]. The papers [10] and [12] have recently come to our attention. The paper by Irvine, Marin and Smith [10] contains, for the case $k = p = 2$, an approach related to ours. We also mention a generalization to least squares data approximation [6], [10].

* Received by the editors December 30, 1985; accepted for publication (in revised form) October 14, 1986.

† Department of Mathematics, Linköping University, Linköping, Sweden.

‡ The work of this author was supported by the Swedish Natural Science Research Council under contract F8519-100.

Before proceeding we introduce some notation and definitions. Let $[a, b]$ be an interval of $\mathbf{R}$ and let $t_1, t_2, \cdots, t_{N+k}$ be points of $[a, b]$ with $a \leqq t_1 < t_2 < \cdots < t_{N+k} \leqq b$. Here N and k are integers with $N \geqq 1$, $k \geqq 2$. Let $y_1, y_2, \cdots, y_{N+k}$ be real numbers. For $l = 0, 1, 2, \cdots, k$, the lth divided differences of the set of data $\{(t_i, y_i);\ i = 1, 2, \cdots, N+k\}$ are denoted by Δ_i^l, $i = 1, 2, \cdots, N+k-l$, and defined recursively by

$$\Delta_i^0 = y_i, \qquad \Delta_i^l = \frac{l}{t_{i+l} - t_i}(\Delta_{i+1}^{l-1} - \Delta_i^{l-1}) \quad \text{for } l \geqq 1.$$

For a continuous function f on $[a, b]$, the kth difference quotients $f[t_i, t_{i+1}, \cdots, t_{i+k}]$ are defined as the kth divided differences of the data $\{(t_1, f(t_1)), (t_2, f(t_2)), \cdots, (t_{N+k}, f(t_{N+k}))\}$. It is easily seen that $f[t_i, t_{i+1}, \cdots, t_{i+k}] = 0$ if f is a polynomial of degree less than k.

Let p be a real number with $1 < p < \infty$ and let q be its dual exponent: $1/p + 1/q = 1$. $W^{k,p}(a, b)$ is the Sobolev space of measurable functions on $[a, b]$, having kth order derivatives belonging to $L^p(a, b)$, i.e.,

$$W^{k,p}(a, b) = \left\{ f \in C^{k-1}[a, b];\ f^{k-1} \text{ is absolutely continuous, } \int_a^b |f^{(k)}(t)|^p\, dt < \infty \right\}.$$

The normalized B-splines $M_{i,k}$, $i = 1, 2, \cdots, N$ are defined by

$$M_{i,k}(t) = g_{k,t}[t_i, t_{i+1}, \cdots, t_{i+k}] \quad \text{where } g_{k,t}(s) = k(s-t)_+^{k-1}$$

and a_+ is the positive part of the number a: $a_+ = \max(a, 0)$. Also $a_- = \max(0, -a)$ will denote the negative part of a. For each $i \in \{1, 2, \cdots, N\}$, $M_{i,k}$ is then a nonnegative function with support $[t_i, t_{i+k}]$ and $\int_a^b M_{i,k}(t)\, dt = k!$.

For $f \in W^{k,p}(a, b)$, the relation

$$f[x_i, x_{i+1}, \cdots, x_{i+k}] = \frac{1}{k!}\int_a^b f^{(k)}(t) M_{i,k}(t)\, dt, \tag{1.1}$$

known as Peano's theorem, now follows immediately from Taylor's theorem

$$f(s) = \sum_{\nu=0}^{k-1} \frac{f^{(\nu)}(a)}{\nu!}(s-a)^\nu + \frac{1}{k!}\int_a^b f^{(k)}(t) g_{k,t}(s)\, dt.$$

We introduce the classes

$$F(\delta) = \{f \in W^{k,p}(a, b):\ f^{(k)} \geqq \delta \text{ and } f(t_i) = y_i,\ i = 1, 2, \cdots, N+k\} \tag{1.2}$$

and

$$G(\Delta^k, \delta) = \left\{ g \in L^p(a, b):\ g \geqq \delta \text{ and } \int_a^b g(t) M_{i,k}(t)\, dt = k! \cdot \Delta_i^k,\ i = 1, 2, \cdots, N \right\}. \tag{1.3}$$

Using (1.1) it is easy to prove that the operator d^k/dt^k is a bijection of F onto $G(\Delta^k, \delta)$. Thus the following problems are equivalent:

$P_1(\delta)$. Find $f \in F$ minimizing $\int_a^b |f^{(k)}(t)|^p\, dt$.

$P_2(\delta)$. Find $g \in G(\Delta^k, \delta)$ minimizing $\int_a^b |g(t)|^p\, dt$.

In § 2 we shall propose a Newton algorithm to solve problem $P_2(\delta)$, i.e., to determine $g(t) = f^{(k)}(t)$. The method will produce a solution for arbitrary δ (assuming that the data allow a solution), p with $1 < p < \infty$ and $k \geqq 2$. The asymptotic rate of convergence is always a superlinear and for some cases, e.g., when $k = p = 2$ also quadratic.

In § 3, where for simplicity only the case $k=p=2$ is considered, an algorithm for calculating the function f of problem $P_1(\delta)$, given $g(t)=f^{(2)}(t)$, is described. Finally in § 4 we discuss implementation of our algorithms and present some numerical results.

2. An algorithm for computing the optimal $g=f^{(k)}$.

2.1. The case $\delta=0$. Suppose that the data $\{(t_i, y_i)\}_{i=1}^{N+k}$ are such that $\Delta_i^k>0$ for $i=1,2,\cdots,N$ and so that the class $G(\Delta^k, 0)$ is nonempty. We remark that for $k=2$ the condition $\Delta_i^k>0$, $i=1,2,\cdots,N$, is both necessary and sufficient for the class $G(\Delta^k, 0)$ to be nonempty. This can be seen as follows. It is always possible to fit a $C^\infty[a, b]$ convex function to the data (we leave it to the interested reader to verify this). However such a function belongs to $W^{2,p}[a, b]$ and hence is a candidate for the solution of $P_1(0)$. For $k>2$ the condition $\Delta_i^k>0$ is no longer sufficient for the class $G(\Delta^k, 0)$ to be nonempty and it is fairly easy to produce counterexamples.

In [1] the following result is given.

THEOREM 1. *The problem $P_2(0)$ has a unique solution g and there exists a vector $\alpha\in\mathbf{R}^N$ such that*

$$g(t)=\left(\sum_{i=1}^{N}\alpha_i M_{i,k}(t)\right)_+^{q-1} \quad a.e.$$

Moreover, there is only one function in $G(\Delta^k, 0)$ of the form $(\sum\beta_i M_{i,k}(t))_+^{q-1}$, β_i real, namely the minimizing function for problem $P_2(0)$. □

It follows from Theorem 1 that the optimal g is continuous. Hence we may conclude that the optimal $f(x)\in C^k[a, b]$. For the case $k=2$ the resulting interpolant is hence a convex cubic spline function.

Next consider the following problems:

P_3. Find $\alpha\in\mathbf{R}^N$ which minimizes the convex and differentiable (note that $q>1$ since $p<\infty$) functional Φ given by

$$\alpha\mapsto\Phi(\alpha)=\frac{1}{q}\int_a^b\left(\sum_{j=1}^{N}\alpha_j M_{j,k}(t)\right)_+^q dt-k!\cdot\sum_{j=1}^{N}\alpha_j\Delta_j.$$

P_4. Find $\alpha\in\mathbf{R}^N$ such that $(\Phi'(\alpha))_i=\int_a^b(\sum\alpha_j M_{j,k}(t))_+^{q-1}M_{i,k}(t)\,dt-k!\Delta_i=0$ for $i=1,2,\cdots,N$.

Here Φ' denotes the gradient of Φ.

THEOREM 2. *Problems $P_1(0)$, $P_2(0)$, P_3 and P_4 are equivalent and all of them have a unique solution.*

Proof. From Theorem 1 it follows that $P_1(0)$, $P_2(0)$ and P_4 are equivalent. Since P_4 has a solution α with $\Phi'(\alpha)=0$ and since Φ is convex it follows that α minimizes Φ, i.e., solves P_3. Conversely if α solves P_3 then $\Phi'(\alpha)=0$, i.e., α solves P_4. This completes the proof. □

In order to solve problem $P_1(0)$ we propose the use of Newton's method for problem P_4. With a given initial vector α^0 the iterates are then defined by

$$\Phi'(\alpha^n)+\Phi''(\alpha^n)(\alpha^{n+1}-\alpha^n)=0. \tag{2.1}$$

Now by taking derivates of $\Phi'(\alpha)$, as given in P_4, one gets

$$(\Phi''(\alpha))_{ij}=(q-1)\int_a^b\left(\sum_{r=1}^{N}\alpha_r M_{r,k}\right)^{q-2}\chi_\alpha M_{i,k}M_{j,k}\,dt, \tag{2.2}$$

where

$$\chi_\alpha(t)=\begin{cases}1 & \text{if } \sum_{j=1}^{N}\alpha_j M_{j,k}(t)>0,\\ 0 & \text{else}\end{cases} \tag{2.3}$$

and provided that the zeros of $\sum_{j=1}^{N}\alpha_j M_{j,k}$ are simple. Therefore with

$$\Phi''(\alpha)=(q-1)T(\alpha) \tag{2.4}$$

the (i,j)th element of the symmetric matrix $T(\alpha)$ is given by

$$t_{i,j}=\int_a^b (\sum \alpha_r M_{r,k}(t))^{q-2}\chi_\alpha M_{i,k}(t)M_{j,k}(t)\,dt. \tag{2.5}$$

We immediately obtain that $t_{i,j}\geqq 0$ for all i,j and that $t_{ij}=0$ if $|i-j|\geqq k$. The restriction that the roots of $\sum \alpha_r M_{r,k}(t)=0$ are simple has two purposes. First, for the case $q<2$ it guarantees that Φ'' exists. Second, since $\Sigma\,\alpha_r M_{r,k}(t)$ cannot be zero on a whole interval, $\Phi''(\alpha)$ will be continuous everywhere.

Concerning the convergence of the proposed Newton-method it is well known (see [14]) that the method converges superlinearly if

(i) $\Phi''(\alpha)$ is continuous in a neighborhood of the solution α^*;

(ii) $\Phi''(\alpha^*)$ is invertible;

(iii) The initial value α^0 is chosen close enough to α^*.

Moreover, the method converges at least quadratically if in addition to (i)-(iii) the following is true:

(iv) The Gateaux derivative $\Phi'''(\alpha)$ is bounded in a neighborhood of α^*.

Inspecting the expression (2.2) we see that $\Phi''(\alpha)$ is a continuous function of α for all α and $1<p<\infty$. To prove that $\Phi''(x^*)=(q-1)T(\alpha^*)$ is invertible, in fact positive definite, we shall need a few lemmata.

LEMMA 3. *If $\alpha\in\mathbf{R}^N$ is such that*

$$\max_{t\in[t_i,t_{i+k}]}\left(\sum_{j=1}^{N}\alpha_j M_{j,k}(t)\right)>0 \quad \textit{for } i=1,2,\cdots,N$$

and the zeros of $\sum_{r=1}^{N}\alpha_j M_{j,k}$ are simple then $T(\alpha)$ is positive definite.

Proof. Let $\beta\in\mathbf{R}^N$ be arbitrary.

$$\beta^t T(\alpha)\beta=\sum_{i,j}t_{ij}\beta_i\beta_j=\int_a^b\left(\sum_{r=1}^{N}\alpha_r M_{r,k}\right)^{q-2}\chi_\alpha\left(\sum_{j=1}^{N}\beta_j M_{j,k}\right)^2 dt\geqq 0.$$

If $\beta^t T(\alpha)\beta=0$ then $\chi_\alpha\cdot\sum\beta_j M_{j,k}=0$ for all t. Now because of the condition on α, for every i the interval $[t_i,t_{i+k}]$ has some subinterval (s_i,s_{i+1}) on which $\chi_\alpha>0$. Consequently $\sum\beta_j M_{j,k}(t)=0$ on (s_i,s_{i+1}). Further (s_i,s_{i+1}) may be taken as a subinterval of one of the intervals $[t_{i+r},t_{i+r+1}]$, $r=0,1,\cdots,k-1$. Since the functions $M_{j,k}$ are linearly independent on (s_i,s_{i+1}) for $i+r-k+1\leqq j\leqq i+r$ we conclude that $\beta_j=0$ for those j-values. In particular $\beta_i=0$, for $i=1,2,\cdots,N$. This concludes the proof. □

LEMMA 4. *Suppose that $\alpha\in\mathbf{R}^N$ is such that $\max_{t\in[t_i,t_{i+k}]}\sum_{j=1}^{N}\alpha_j M_{j,k}(t)>0$ for $i=1,2,\cdots,N$ and so that the zeros of $\sum_{j=1}^{N}\alpha_j M_{j,k}$ are simple. Let $\gamma\in\mathbf{R}^N$ with $\gamma_i>0$, $i=1,2,\cdots,N$ be given. Then the solution β of the equation $T(\alpha)\beta=\gamma$ also satisfies the condition $\max_{t\in[t_i,t_{i+k}]}\sum_{j=1}^{N}\beta_j M_{j,k}(t)>0$ for $i=1,2,\cdots,N$.*

Proof. By the previous lemma $T(\alpha)^{-1}$ exists so the solution β certainly exists. Now $T(\alpha)\beta=\gamma$ means that, for $i=1,2,\cdots,N$,

$$\sum_{j=1}^{N}\beta_j\int_a^b(\sum\alpha_r M_{r,k})^{q-2}\chi_\alpha M_{i,k}M_{j,k}\,dt=\gamma_i,$$

i.e., that

$$\int_a^b \left(\sum_{r=1}^N \alpha_r M_{r,k}\right)^{q-2} \chi_\alpha \left(\sum_{j=1}^N \beta_j M_{j,k}\right) M_{i,k}\, dt = \gamma_i.$$

Since $\gamma_i > 0$, $M_{i,k} \geqq 0$, and supp $M_{i,k} = [t_i, t_{i+k}]$ we see that we cannot have $\sum_{j=1}^N \beta_j M_{j,k} \leqq 0$ on any interval $[t_i, t_{i+k}]$ and the proof is complete. □

LEMMA 5. *Suppose that $\Phi'(\alpha^*) = 0$, i.e., that α^* is the solution of problem P_4, and that the zeros of $\sum_{j=1}^N \alpha^* M_{j,k}$ are simple. Then $\Phi''(\alpha^*)$ is positive definite.*

Proof. $\Phi'(\alpha^*) = 0$ can be written (see P_4) as

$$\int_a^b \left(\sum \alpha_j^* M_{j,k}(t)\right)_+^{q-1} M_{i,k}(t)\, dt = \Delta_i^k \cdot k!$$

where by assumption the right-hand side is positive. This implies that

$$\max_{t \in [t_i, t_{i+k}]} \left(\sum \alpha_j^* M_{j,k}(t)\right) > 0$$

and so by Lemma 3, we conclude that $T(\alpha^*)$, and hence by (2.4) $\Phi''(\alpha^*)$, is positive definite. □

Next we observe that, using the definition of $\Phi'(\alpha)$ as given in P_4 and formula (2.5),

$$\Phi'(\alpha) = T(\alpha)\alpha - k!\Delta^k. \tag{2.6}$$

Hence the proposed method (2.1) takes the form

$$T(\alpha^n)\alpha^{n+1} = \frac{k!}{q-1}\Delta^k + \frac{q-2}{q-1} T(\alpha^n)\alpha^n. \tag{2.7}$$

In particular for $p = 2$ we have

$$T(\alpha^n)\alpha^{n+1} = k!\Delta^k, \tag{2.8}$$

where by (2.5)

$$t_{ij} = \int_a^b M_{i,k} M_{j,k} \chi_\alpha\, dt. \tag{2.9}$$

We summarize our results in the following theorem.

THEOREM 6. *Suppose that the solution α^* of problem P_4 has the property that the zeros of $\sum_{j=1}^N \alpha_j^* M_{j,k}$ are simple. Consider the following iteration method:*

(1) *Select $\alpha^0 \in \mathbf{R}^n$ such that $\max_{t \in [t_i, t_{i+k}]} \sum_{j=1}^N \alpha_j^0 M_{j,k}(t) > 0$ for $i = 1, 2, \cdots, N$.*

(2) *Let for $n \geqq 0$ α^{n+1} be defined by*

$$\alpha^{n+1} = \frac{k!}{q-1} \cdot T^{-1}(\alpha^n)\Delta^k + \frac{q-2}{q-1} \cdot \alpha^n.$$

Then if α^0 is close enough to α^*, all the matrices $T(\alpha^n)$ are positive definite so that the sequence $\{\alpha^n\}$ is well defined and $\alpha^n \to \alpha^*$ as $n \to \infty$. The convergence is at least quadratic if $1 < p \leqq 2$ and superlinear otherwise.

Proof. We have to prove that the convergence is quadratic for $1 < p \leqq 2$, i.e., that $\Phi'''(\alpha)$ is bounded in a neighborhood of α^*. However

$$(\Phi'''(\alpha))_{ijl} = (q-1)(q-2) \int_a^b \left(\sum \alpha_r M_{r,k}\right)_+^{q-3} M_{i,k} M_{j,k} M_{r,k}\, dt$$

for $1 < p < 2$, i.e., $2 < q < \infty$, which implies boundedness if the roots of $\sum \alpha_r M_{r,k} = 0$ are simple or if $k = 2$. We treat the case $p = q = 2$ separately.

A formal calculation, using (2.2), gives

$$(\Phi'''(\alpha))_{ijl} = \frac{\partial}{\partial \alpha_l}(\Phi''(\alpha))_{ij} = \frac{\partial}{\partial \alpha_l}\int_a^b \chi_\alpha(t) M_{i,k}(t) M_{j,k}(t)\, dt$$

$$= \int_a^b \frac{\partial \chi_\alpha(t)}{\partial \alpha_l} M_{i,k}(t) M_{j,k}(t)\, dt.$$

Formula (2.3) implies that we can write $(\partial/\partial\alpha_l)\chi_\alpha(t) = \sum_m c_m(\alpha)\delta(t - t_m(\alpha))$, where $t_m(\alpha)$ are the zeros of $\sum_{r=1}^N \alpha_r M_{r,k}(t)$ which depend continuously on α, $c_m(\alpha)$ are real valued bounded functions and $\delta(t)$ denotes Dirac's delta function. Inserting this in the previous equation we obtain $(\Phi'''(\alpha))_{ijl} = \sum c_m(\alpha) M_{i,k}(t_m(\alpha)) M_{j,k}(t_m(\alpha))$, which is a bounded function of α in a neighborhood of α^*. The validity of this formal calculation can certainly be verified. Note that the derivative $(\partial/\partial\alpha_l)(\Phi''(\alpha))_{ij}$ can have different values if we differentiate in the positive and negative α_l-direction, respectively. Therefore $\Phi'''(\alpha)$ is not continuous everywhere but can have jump discontinuities.

Consequently we have proved that the convergence is quadratic if $1 < p \leqq 2$ and the roots of $\sum \alpha_j^* M_{j,k}(t) = 0$ are simple. □

2.2. The case of an arbitrary δ. Suppose that the data $\{(t_i, y_i)\}^{N+k}$ are such that $\Delta_i^k \cdot k! > \delta \int_a^b M_{i,k}\, dt = \delta \cdot k!$, i.e., $\Delta_i^k > \delta$ for $i = 1, 2, \cdots, N$ and that the set $G(\Delta^k, \delta)$ is nonempty.

Following [1] we have the following theorem.

THEOREM 1′. *The problem $P_2(\delta)$ has a unique solution g and there exists a vector $\alpha \in \mathbf{R}^N$ such that*

$$g(t) = \begin{cases} \left(\sum_{j=1}^N \alpha_j M_{j,k}(t)\right)^{q-1} & \text{if } \sum \alpha_j M_{j,k}(t) > \delta, \\ \delta & \text{else.} \end{cases}$$

Moreover there is only one function in $G(\Delta^k, \delta)$ of this form. □

Next let

$$\varphi(t) = \begin{cases} |t|^q/q & \text{if } t > |\delta|^{1/(q-1)} \cdot \operatorname{sgn}(\delta), \\ \delta \cdot t - |\delta|^p/p & \text{else.} \end{cases} \tag{2.10}$$

Then

$$\varphi'(t) = \begin{cases} |t|^{q-1} \operatorname{sgn}(t) & \text{if } t > |\delta|^{1/(q-1)} \cdot \operatorname{sgn}(\delta), \\ \delta & \text{else.} \end{cases} \tag{2.11}$$

Define the convex functional $\alpha \mapsto \Phi(\alpha)$ by

$$\Phi(\alpha) = \int_a^b \varphi\left(\sum_{j=1}^N \alpha_j M_{j,k}(t)\right) dt - k! \cdot \sum_{j=1}^N \alpha_j \Delta_j^k,$$

and consider, besides $P_1(\delta)$ and $P_2(\delta)$, the following problems.

$P_3(\delta)$. Find $\alpha \in \mathbf{R}^N$ minimizing Φ.

$P_4(\delta)$. Find $\alpha \in \mathbf{R}^N$ so that $\Phi'(\alpha) = 0$.

From Theorem 1′ it follows that $P_1(\delta)$-$P_4(\delta)$ are all equivalent and have a unique solution. We will now show how our previously defined iteration method (2.7) shall be modified so as to solve $P_4(\delta)$ for arbitrary δ. In order to simplify notation we shall specialize to the case $k = p = 2$.

Now

$$(\Phi'(\alpha))_i = \int_a^b \varphi'(\textstyle\sum \alpha_j M_{j,2}) M_{i,2}\, dt - 2\Delta_i^2, \tag{2.12}$$

and

$$(\Phi''(\alpha))_{i,j} = \int_a^b \varphi''(\textstyle\sum \alpha_r M_{r,2}) M_{i,2} M_{j,2}\, dt. \tag{2.13}$$

Since $\varphi''(t) = 1$ for $t > \delta$ and $\varphi''(t) = 0$ otherwise, we obtain

$$\Phi''(\alpha) = T_\delta(\alpha) \tag{2.14}$$

where the (i, j)th element of the symmetric matrix $T_\delta(\alpha)$ is defined by

$$t_{ij} = \int_a^b \chi_{\alpha,\delta} M_{i,2} M_{j,2}\, dt. \tag{2.15}$$

Here

$$\chi_{\alpha,\delta} = \begin{cases} 1 & \text{if } \sum \alpha_j M_{j,2}(t) > \delta, \\ 0 & \text{else.} \end{cases} \tag{2.16}$$

Further from (2.11) and (2.12),

$$(\Phi'(\alpha))_i = \sum_{j=1}^N \alpha_j \int_a^b M_{j,2} M_{i,2} \chi_{\alpha,\delta}\, dt + \delta \int_a^b (1 - \chi_{\alpha,\delta}) M_{i,2}\, dt - 2\Delta_i^2.$$

Hence by (2.15) we can write

$$\Phi'(\alpha) = T_\delta(\alpha)\alpha + \delta \int_a^b (1 - \chi_{\alpha,\delta}) M\, dt - 2\Delta^2, \tag{2.17}$$

where

$$M = (M_{1,2}(t), M_{2,2}(t), \cdots, M_{N,2}(t))^T. \tag{2.18}$$

Therefore iteration (2.1), using (2.14) and (2.17), takes the form

$$T_\delta(\alpha^n)\alpha^{n+1} = 2\Delta^2 - \delta \int_a^b (1 - \chi_{\alpha^n,\delta}) M\, dt. \tag{2.19}$$

THEOREM 6′. *Consider the following iteration method*:

(1) *Select* $\alpha^0 \in \mathbf{R}^N$ *such that*

$$\max_{t \in [t_i, t_{i+2}]} \left(\sum_{j=1}^N \alpha_j^0 M_{j,2}(t) \right) > \delta \quad \text{for } i = 1, 2, \cdots, N.$$

(2) *Let for* $n \geqq 0$ α^{n+1} *be defined by*

$$T_\delta(\alpha^n)\alpha^{n+1} = 2\Delta^2 - \delta \cdot \int_a^b (1 - \chi_{\alpha^n,\delta}) M\, dt.$$

Then if α^0 *is close enough to the solution* α^* *to problem* $P_4(\delta)$ *and if the zeros of* $\sum a_j^* M_{j,2} - \delta$ *are simple, all the matrices* $T_\delta(\alpha^n)$ *are positive definite, so that the sequence is well defined, and* $\alpha^n \to \alpha^*$ *as* $n \to \infty$. *The convergence is at least quadratic* $(k = p = 2)$.

Proof. It suffices to prove that $\Phi''(\alpha^*) = T_\delta(\alpha^*)$ is positive definite and that $\Phi'''(\alpha)$ is bounded in a neighborhood of α^*.

Now from the fact that $g(t) = \max\{\sum_{j=1}^N \alpha_j^* M_{j,2}, \delta\}$ solves problem $P_2(\delta)$ and that $\int_a^b g(t)M_{i,2}\,dt = 2\Delta_i^2 > 2\delta$ we conclude that $\max_{t\in[t_i,t_{i+2}]} \sum_{j=1}^N \alpha_j^* M_{j,2} > \delta$ for $i = 1, 2, \cdots, N$. In fact, the reverse inequality on some interval $[t_i, t_{i+2}]$ would imply that $\int gM_{i,2}\,dt \leqq 2\delta$. Arguing as in the proof of Lemma 3 we obtain that $T_\delta(\alpha^*)$ is positive definite. That $\Phi'''(\alpha)$ is bounded in a neighborhood of α^* is proved in the same way as in Theorem 6. This finishes the proof. □

Now let us denote the algorithm described in Theorem 6′ by $\mathcal{A}(\delta, \alpha^0)$, where α^0 is an arbitrary initial vector.

Suppose that δ is fixed and that the problem $P_2(\delta)$ is given with $\Delta_i^k > \delta$, $i = 1, 2, \cdots, N$. To solve this problem we propose the following **main algorithm** (to simplify notation we again assume that $k = p = 2$).

(1) Choose $\alpha_i^0 = 1$ for $i = 1, 2, \cdots, N$.
(2) Compute α^1 from the equation $T(\alpha^0)\alpha^1 = 2\Delta^2$. This is equivalent to solving the interpolation problem without any restriction. Take $d = \min_{t\in[a,b]} \sum_{j=1}^N \alpha_j^1 M_{j,2}(t)$. If $d \geqq \delta$ then we are finished. Else put $\alpha^0 = \alpha^1$.
(3) $\delta' = d$.
(4) $\delta'' = \delta$.
(5) Run the algorithm $\mathcal{A}(\delta'', \alpha^0)$.
(6) If it diverges then $\delta'' := (\delta'' + \delta')/2$, goto 5. Else goto 7.
(7) $\mathcal{A}(\delta'', \alpha^0)$ converges producing a solution α^*. If $\delta'' < \delta$ then $\delta' := \delta''$ and $\delta'' := \delta$. $\alpha^0 := \alpha^*$. Goto 5.
(8) $\delta'' = \delta$ and we are finished.

THEOREM 7. *For some prescribed accuracy the algorithm above converges after a finite number of steps.*

Proof. The solution $\alpha^*(s)$ of the problem $P_2(s)$ must be a continuous function of s. Therefore the function $r(s)$ defined by $r(s) = \inf_{\alpha^0}\{|\alpha_0 - \alpha^*(s)|: \mathcal{A}(s, \alpha^0)$ diverges$\}$ is continuous on the compact interval $[d, \delta]$ and we conclude that $\min_{s\in[d,\delta]} r(s) = r_{\min} > 0$. Furthermore $\alpha^*(s)$ is uniformly continuous on $[d, \delta]$. Therefore there exists an $\varepsilon(r_{\min}) > 0$ so that $|\Delta s| < \varepsilon(r_{\min}) \Rightarrow |\alpha^*(s + \Delta s) - \alpha^*(s)| < r_{\min}$. The number of necessary outer iteration cycles in the given procedure (i.e., the number of times a new α^0 is selected in step (7)) is then less than $2d/\varepsilon(r_{\min})$. For each of these cycles we perform at most so many inner loops k (steps 5-6-5) so that $\varepsilon(r_{\min})/2 \leqq 2^{-k}d$. This means that $k \leqq {}^2\log(2d/\varepsilon(r_{\min}))$. Finally, by the superlinear convergence, for each of these loops we need only a fixed number, say L, iterations to establish either divergence or convergence. Consequently the number of necessary iterations is less than $2d/\varepsilon(r_{\min}) \cdot {}^2\log(2d/\varepsilon(r_{\min})) \cdot L < \infty$. □

3. Computation of the interpolant. In this section we will discuss how to compute the unique constrained interpolant $f(x)$ from its computed derivative $f^{(k)}(t)$. For simplicity we will only consider the case $k = p = q = 2$. Then $f(x)$ is twice continuously differentiable. We further have, by Theorem 1′,

$$f''(t) = \max\left(\sum_{j=1}^N \alpha_j M_{j,2}(t), \delta\right), \tag{3.1}$$

which is a piecewise linear function. Let $\{k_i\}$ denote the knotsequence for $f''(t)$ (a knot k_i is a point where a new segment of $f''(t)$ starts) and $\{z_i\}$ the corresponding values of f'' (i.e. $z_i = f''(k_i)$). The knotsequence consists of a subset of the initially

given interpolation abscissas $\{t_i\}^{N+2}$, and of new knots introduced when performing the max operation in (3.1). Observe that with

$$\bar{\alpha}_i = 4\alpha_i/(t_{i+2}-t_i), \tag{3.2}$$

$$\sum_j \alpha_j M_{j,2}(t_i) = \alpha_{i-1} M_{i-1,2}(t_i) = \bar{\alpha}_{i-1}. \tag{3.3}$$

Each double interval $[t_i, t_{i+2}]$ may have two additional knots. This case occurs when $\bar{\alpha}_{i-1}, \bar{\alpha}_{i+1} > \delta$ and $\bar{\alpha}_i < \delta$ or $\bar{\alpha}_{i-1}, \bar{\alpha}_{i+1} < \delta$ and $\bar{\alpha}_i > \delta$. It is also easy to verify that, with $N = 2r+1$, there could at most be $3r$ knots out of which $2r$ are new ones and r knots are a subset of the given interpolation abscissas (cf. [6]). The following formulas hold for computing the knotsequence.

(3.4)

Put $k_1 = t_1$, $z_1 = \max(0, \delta)$, and $s = 1$.
for $i = 2, N+1$
  if $\bar{\alpha}_{i-1} > \delta$ **then** $k_s = t_i$, $z_s = \bar{\alpha}_{i-1}$, $s = s+1$.
  if $\min(\bar{\alpha}_{i-1}, \bar{\alpha}_i) < \delta$ and $\max(\bar{\alpha}_{i-1}, \bar{\alpha}_i) > \delta$
    then $k_s = d_i$, $z_s = \delta$, $s = s+1$.
  $k_s = t_{N+2}$, $z_s = \max(0, \delta)$.

Here

$$d_i = \frac{\delta(t_{i+1}-t_i) + \bar{\alpha}_i t_i - \bar{\alpha}_{i-1} t_{i+1}}{\bar{\alpha}_i - \bar{\alpha}_{i-1}}. \tag{3.5}$$

We now give an explicit representation of $f(t)$ given the knotsequence $\{k_i\}$ and corresponding values $\{z_i\}$. Put

$$f''(t) = s_i''(t), \qquad t \in [k_i, k_{i+1}] \tag{3.6}$$

and let

$$h_i = k_{i+1} - k_i. \tag{3.7}$$

It follows that

$$s_i''(t) = \frac{z_{i+1}}{h_i}(t-k_i) + \frac{z_i}{h_i}(k_{i+1}-t) \tag{3.8}$$

and by integration

$$s_i'(t) = \frac{z_{i+1}}{2h_i}(t-k_i)^2 - \frac{z_i}{2h_i}(k_{i+1}-t)^2 + C_i - D_i, \tag{3.9}$$

$$s_i(t) = \frac{z_{i+1}}{6h_i}(t-k_i)^3 + \frac{z_i}{6h_i}(k_{i+1}-t)^3 + C_i(t-k_i) + D_i(k_{i+1}-t). \tag{3.10}$$

If we assume that there are m knots altogether, there are $2(m-1)$ numbers $\{C_i, D_i\}$ to determine in (3.10). We now discuss one way of specifying $2(m-1)$ linearly independent equations for computing $\{C_i, D_i\}$ such that the resulting function is indeed the unique interpolant, i.e.,

$$f(t) = s_i(t), \qquad t \in [k_i, k_{i+1}]. \tag{3.11}$$

For all knots that coincide with an interpolation abscissa we use the interpolation constraint

$$\begin{aligned} &s_{i-1}(k_i)=f(k_i)\\ &\qquad\qquad \text{if } k_i=t_j \quad \text{for some } j.\\ &s_i(k_i)=f(k_i) \end{aligned} \tag{3.12}$$

For the remaining knots continuity of s and s' yields:

$$\begin{aligned} &s_{i-1}(k_i)=s_i(k_i)\\ &\qquad\qquad \text{if } k_i\neq t_j \quad \forall j.\\ &s'_{i-1}(k_i)=s'_i(k_i) \end{aligned} \tag{3.13}$$

From (3.10) formula (3.12) becomes

$$h_{i-1}C_{i-1}=f(k_i)-\frac{z_i}{6}h_{i-1}^2, \qquad h_iD_i=f(k_i)-\frac{z_i}{6}h_i^2. \tag{3.14}$$

Similarly (3.13) becomes

$$\begin{aligned} &h_{i-1}C_{i-1}-h_iD_i=\frac{\delta}{6}\cdot(h_i^2-h_{i-1}^2),\\ &C_{i-1}-D_{i-1}-C_i+D_i=-\frac{\delta}{2}(h_i+h_{i-1}). \end{aligned} \tag{3.15}$$

We will distinguish between knots that coincide with interpolation abscissas and new knots, denoting them by z^ia and nk, respectively. First observe that at most two consecutive new knots can occur (the reason being that the second derivative is constant $(=\delta)$ between new knots). It follows that the only possible combinations of new knots and interpolation abscissas are the two following ones:

$$(ia,\ nk,\ ia), \tag{3.16}$$

which can only occur at the two endintervals and

$$(ia,\ nk,\ nk,\ ia). \tag{3.17}$$

Let the knotsequence corresponding to (3.17) be $(k_{i-1}, k_i, k_{i+1}, k_{i+2})$. Then this case results in a linear system

$$A_ix_i=b_i, \tag{3.18}$$

where

$$x_i=(D_{i-1}, C_{i-1}, D_i, C_i, D_{i+1}, C_{i+1})^T.$$

From (3.14) and (3.15) it is easy to work out the following formulas:

$$A_i=\begin{pmatrix} h_{i-1} & & & & & \\ & h_{i-1} & -h_i & & & \\ -1 & 1 & 1 & -1 & & \\ & & & h_i & -h_{i+1} & \\ & & -1 & 1 & 1 & -1 \\ & & & & & h_{i+1} \end{pmatrix}, \qquad b_i=\begin{pmatrix} f(k_{i-1})-\frac{1}{6}z_{i-1}h_{i-1}^2 \\ \frac{1}{6}\delta(h_i^2-h_{i-1}^2) \\ -\frac{1}{2}\delta(h_i+h_{i-1}) \\ \frac{1}{6}\delta(h_{i+1}^2-h_i^2) \\ -\frac{1}{2}\delta(h_{i+1}+h_i) \\ f(k_{i+2})-\frac{1}{6}z_{i+2}\cdot h_{i+1}^2 \end{pmatrix}.$$

It can readily be verified that $\det(A_i)=h_{i-1}\cdot h_i\cdot h_{i+1}(h_{i-1}+h_i+h_{i+1})$ and hence, if we assume distinct knots, the matrix A_i is nonsingular. However it also follows quite easily due to the assumption of distinct interpolation abscissas that the knots are distinct.

We leave it to the reader to work out the structure of the 4×4 nonsingular matrix corresponding to case (3.16).

In summary we suggest the following method for computing $\{C_i, D_i\}$ from constraints (3.12), (3.13). For knots of type *ia*, we use (3.14). For new knots, unless they occur at the endintervals, (3.18) is used. Finally, for a new knot occurring at one of the endintervals, a 4×4 systems must be solved. Since the constraints introduced by (3.12), (3.13) lead to linearly independent equations, it follows that the computed function equals the unique constrained interpolant.

4. Numerical results. Throughout this section only the case $p = k = 2$ is considered. We will discuss experiences using method (2.19). Since it turned out, in all of our experiments, that this method converged globally, there was no need for using the continuation algorithm discussed at the end of § 2.

We first briefly outline the method used for computing the elements of the symmetric, tridiagonal matrix $T_\delta(\alpha^n)$ needed in the iteration (2.19). By (2.15) and the fact that $M_{i,2}$ has support on $[t_i, t_{i+2}]$,

$$t_{i-1,i} = \int_{t_i}^{t_{i+1}} M_{i-1,2} M_{i,2} \chi_{\alpha,\delta}\, dt = \int_{\text{low}}^{\text{lup}} M_{i-1,2} M_{i,2}\, dt, \tag{4.1}$$

$$t_{i,i} = \int_{t_i}^{t_{i+2}} M_{i,2}^2 \chi_{\alpha,\delta}\, dt = \int_{\text{low}}^{\text{lup}} M_{i,2}^2\, dt + \int_{t_{i+1}}^{t_{i+2}} M_{i,2}^2 \chi_{\alpha,\delta}\, dt. \tag{4.2}$$

For each inner interval $[t_i, t_{i+1}]$, $i = 2, 3, \cdots, N$, condition (3.4) is checked:

If (3.4) **then** if $\bar{\alpha}_{i-1} > \delta$ **then** low $= t_i$, lup $= d_i$; else low $= d_i$, lup $= t_{i+1}$.
If not (3.4) and $\bar{\alpha}_{i-1}, \bar{\alpha}_i > \delta$ **then** low $= t_i$, lup $= t_{i+1}$.

Note that there will also be a contribution to t_{ii} from the next interval $[t_{i+1}, t_{i+2}]$. Also t_{11} will get a contribution from $[t_1, t_2]$ and t_{NN} from the interval $[t_{N+1}, t_{N+2}]$. It is also convenient to compute the function $\int_a^b (1-\chi_{\alpha^n,\delta}) M\, dt$ needed in (2.19) during the same scan through the interpolation abscissas. The integrals above were evaluated exactly using Simpson's rule. In order to solve the linear 3-diagonal system in (2.19) we used Gaussian elimination as, e.g., described in [4]. The number of operations are only $3N$, since no pivoting is needed.

In the tests we used

$$\alpha_j^0 = (1+\delta)\times(t_{j+2} - t_j)/4$$

which (cf. (3.3)) gives a permissible (according to condition 1 in Theorem 6′) starting vector. The program was written in Fortran and run, using single precision arithmetic, on a DEC-20 computer.

Table 1 shows the number of iterations needed for satisfying the criterion, $\|\alpha^{n+1} - \alpha^n\| / \|\alpha^n\| \leqq 10^{-5}$, when interpolating the function $f_s(t) = t^{-s}$, $s = 2$. The interpo-

TABLE 1
Number of necessary iterations when interpolating $1/x^2$, $\delta_1 = 0.5 * \delta_{\max}$, $\delta_2 = 0.9 * \delta_{\max}$.

$N1$	$\delta = 0$	$\delta = \delta_1$	$\delta = \delta_2$	$\delta_{\max}$
5	6	6	7	28
10	6	6	6	0.73
15	6	6	6	0.11
20	4	4	4	0.03

lation abscissas were taken to be, $t_i = a + 0.2i$, $i = 1, 2, \cdots, N+2 = N1$ and $b = -0.1$. Further $\delta_{\max}$ denotes the largest value of δ such that the data allow a solution (i.e. $\delta_{\max} = \min \Delta_i^2$). The results in Table 1 indicate that the number of iterations is insensitive to $N1$, the dimension of the problem. The same tests as in Table 1 were also performed on $f_4 = t^{-4}$. Here we obtained 9 necessary iterations for all 15 cases.

We remark that we do not advocate the use of polynomial spline interpolation for problems with very strong singularities. For $s = 4$ the largest α-coefficient become of the order 10^5. Here probably a rational spline interpolation is to be preferred (see [5]).

Tables 2 and 3 correspond to the case $N1 = 5$ in Table 1, i.e., interpolation is made at the points $-0.9(0.2)-0.1$. In Table 2 we list the error $\|\alpha^{n+1} - \alpha^n\|/\|\alpha^n\|$ as a function of n. The asymptotic quadratic convergence shows quite clearly.

We give in Table 3 the computed knotsequence for two different values of δ and, for comparison, for the cubic spline (which is obtained after the first step in (2.19)). The examples of Table 3 are quite ill conditioned: a new knot is introduced lying very close to the interpolation abscissa -0.7. In spite of this the algorithm converged relatively fast. In some other tests (not reported here) convergence was obtained after 2 or 3 iterations. However in these cases the new knots and the interpolation abscissas were better separated than in the above example.

Figure 1 shows the behavior of the convex interpolant ($\delta = 0$), the cubic spline and the "true" solution $1/x^2$. We remark that it is quite easy to dampen the oscillations shown by the cubic spline by adding extra points, e.g., by inserting -0.35, -0.15 as new points. The vector α^1 in (2.19) becomes positive and hence the cubic spline then coincides with the convex interpolant.

When the condition $f''(x) \geqq \delta$ is active the interpolant becomes a straight line when $\delta = 0$ and a quadratic otherwise. Figure 1 also illustrates this effect, which however is quite small due to the restrictive condition on $\delta (\delta \leqq \min \Delta_i^2)$. Also a larger δ-value

TABLE 2
Behavior of error when interpolating $1/x^2$ *on* $-0.9(0.2)-0.1$.

Iteration	Error $\delta = 0$	Error $\delta = 25$
0	190	6.9
1	0.47	0.52
2	0.38	0.46
3	0.17	0.24
4	0.03	0.06
5	10^{-3}	$4 \cdot 10^{-3}$
6	10^{-6}	$2 \cdot 10^{-5}$
7	$3 \cdot 10^{-8}$	$2 \cdot 10^{-8}$

TABLE 3
Knotsequence, $\{k_i\}$, *and second derivative values,* $\{z_i\}$, *when interpolating* $1/x^2$ *on* $-0.9(0.2)-0.1$.

Cubic spline	k_i	-0.9	-0.7	-0.5	-0.3	-0.1
	z_i	0	210	-668	3233	0
$\delta = 0$	k_i	-0.9	-0.7 -0.697	-0.39	-0.3	-0.1
	z_i	0	84 0	0	3912	0
$\delta = 25$	k_i	-0.9 -0.79	-0.7 -0.699	-0.38	-0.3 -0.101	-0.1
	z_i	25 25	45 25	25	4036 25	25

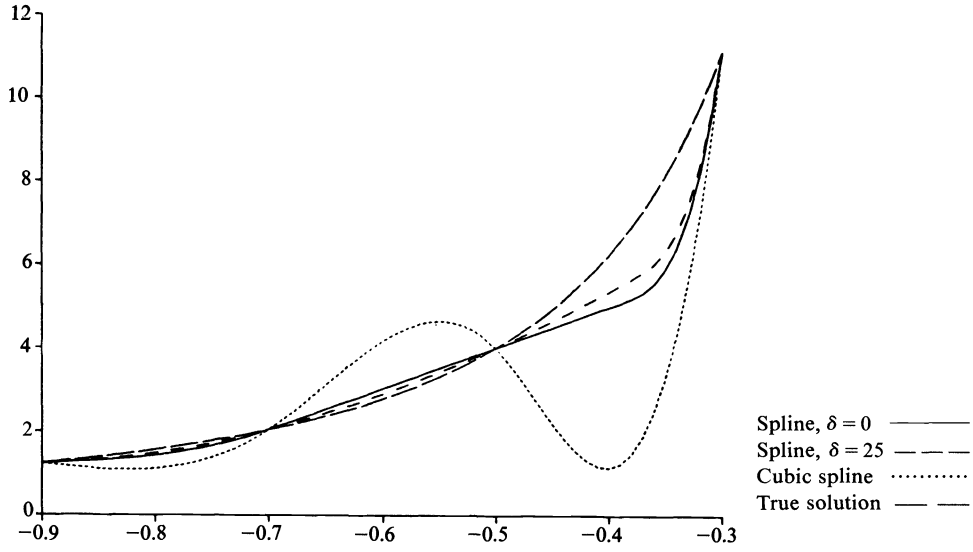


FIG. 1. *Interpolation in* $t = -0.9(0.2) - 0.1$.

would be needed to handle the singularity better at the origin. This is due to the fact that by Theorem 1′ (assuming $\delta \geqq 0$) the constrained interpolant must fulfill the conditions $f(a) = f(b) = \delta$. A possible remedy here is to generalize the algorithm to handle the case when $\delta = \delta(t)$, a function of t, e.g., $\delta(t)$ piecewise linear (see [6]).

Acknowledgment. A first version of algorithm (2.8) was skillfully implemented by Mr. Henrik Åkesäter and Mr. Claes Utterström. We also thank Mrs. Lena Östling for her efficient way of mastering the TEX editing program, a necessary tool for producing this manuscript.

REFERENCES

[1] L.-E. ANDERSSON AND P.-A. IVERT, *Constrained interpolants with minimal* $W^{k,p}$*-norm*, J. Approx. Theory, to appear.

[2] B. W. BURMEISTER, W. HESS AND J. W. SCHMIDT, *Convex spline interpolants with minimal curvature*, Computing, 35 (1985), pp. 219-229.

[3] C. DE BOOR, *A Practical Guide to Splines*, Springer, Berlin-New York, 1978.

[4] G. DAHLQUIST AND Å. BJÖRCK, *Numerical Methods*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[5] R. DELBOURGO AND J. A. GREGORY, *Shape preserving piecewise rational interpolation*, this Journal, 6 (1985), pp. 967-976.

[6] T. ELFVING AND L.-E. ANDERSSON, *An algorithm for computing constrained smoothing spline functions*, LiTH-MAT-R-86-07, Technical Report, Dept. of Mathematics, Linköping Univérsity, Sweden, 1986.

[7] U. HORNUNG, *Interpolation by smooth functions under restrictions on the derivatives*, J. Approx. Theory, 28 (1980), pp. 227-237.

[8] G. L. ILIEV AND W. POLLUL, *Convex interpolation with minimal* L_∞*-norm of the second derivative*, Math. Z., to appear.

[9] ———, *Convex interpolation by functions with minimal* L_p*-norm* $(1 < p < \infty)$ *of the kth derivative*, Sonderforschungsbereich 72, Preprint 665, Univ. Bonn, West Germany, 1984.

[10] L. D. IRVINE, S. P. MARIN AND P. W. SMITH, *Constrained interpolation and smoothing*, Constr. Approx., 2 (1986), pp. 129-151.

[11] D. F. MCALLISTER AND J. A. ROULIER, *Interpolation by convex quadratic splines*, Math. Comp., 32 (1978), pp. 1154-1162.

[12] C. A. MICCELI, P. W. SMITH, I. SWETITS AND I. D. WARD, *Constrained* L_p *approximation*, Constr. Approx., 1 (1985), pp. 93-102.

[13] E. NEUMAN, *Uniform approximation by some Hermite interpolating splines*, J. Comput. Appl. Math., 4 (1978), pp. 7–9.

[14] J. M. ORTEGA AND W. C. RHEINBOLDT, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.

[15] E. PASSOW AND J. A. ROULIER, *Monotone and convex spline interpolation*, SIAM J. Numer. Anal., 14 (1977), pp. 904–909.

[16] L. S. SHUMAKER, *On shape preserving quadratic spline interpolation*, SIAM J. Numer. Anal., 20 (1983), pp. 854–864.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 6, November 1987
© 1987 Society for Industrial and Applied Mathematics
012

CO-MONOTONE INTERPOLATING SPLINES OF ARBITRARY DEGREE—A LOCAL APPROACH*

PAOLO COSTANTINI†

Abstract. The purpose of this note is to describe a method for constructing C^k splines of degree n $(0<k<n-k)$ which interpolate an arbitrary set of data. These splines are obtained using Bernstein polynomials of suitable continuous piecewise linear functions and preserve the monotonicity of the data, i.e., are locally monotone in each subinterval defined by the knots.

Key words. splines, co-monotone interpolation, Bernstein polynomials

AMS(MOS) subject classifications. Primary 65D05, 65D07; secondary 41A05

1. Introduction. Let (x_i, y_i); $i=0, 1, \cdots, N$; $N \geqq 2$ be a set of data points with $x_0<x_1<\cdots<x_N$ and, setting $h_i=x_{i+1}-x_i$; $i=0, 1, \cdots, N-1$, let $\Delta_i=(y_{i+1}-y_i)/h_i$, $i=0, 1, \cdots, N-1$. Given the integers n, k, where $0<k<n-k$, let us consider the following set of splines:

$$BS_n^k=\{s\in C^k[x_0, x_N]: s(x)\in BP_{n,k,i} \text{ for } x\in[x_i, x_{i+1}],\ i=0, 1, \cdots, N-1\} \tag{1.1}$$

where $BP_{n,k,i}$ is defined by

$$\begin{aligned} BP_{n,k,i}=\{p\in P_n:\ & p=\phi+\psi;\ \phi\in P_n,\ \psi\in P_n;\ \phi^{(j)}(x_i)=\psi^{(j)}(x_{i+1})=0, \\ & j=2, \cdots, k;\ \phi^{(j)}(x_{i+1})=\psi^{(j)}(x_i)=0, j=2, \cdots, n-k\} \end{aligned} \tag{1.2}$$

(here P_n stands, as customary, for the space of n-degree polynomials and it is assumed that for $k=1$ the corresponding conditions are vacuous). It is shown in [2] that BS_n^k is a linear space of dimension $2N+2$, and that any $s\in BS_n^k$ can be defined giving its values and slopes at the knots. As a consequence, any problem concerning splines of BS_n^k interpolating a set of data at the knots can be reformulated in terms of the sequence $d_0, d_1, \cdots, d_N$, where $d_i=s^{(1)}(x_i)$, $i=0, 1, \cdots, N$. Further, we see immediately that $BP_{3,1,i}=P_3$ for every i, and so the space BS_3^1 coincides with the space of cubic, C^1 splines. The main results of [2] are concerned with the existence of monotone and/or convex interpolation and furnish the basis for the algorithms proposed in [3]. More specifically, in the latter paper methods are described for computing co-monotone and/or co-convex interpolating splines, that is, $s\in BS_n^k$ such that the polynomials' pieces are (nonstrictly) monotone and/or convex in $[x_i, x_{i+1}]$, $i=0, 1, \cdots, N-1$, according to the shape of data. The algorithms are very similar, and basically consist of a method for selecting the values $d_0, d_1, \cdots, d_N$ in such a way that (d_i, d_{i+1}), $i=0, 1, \cdots, N-1$, belong to those subsets of R^2 which guarantee the monotonicity and/or the convexity for the corresponding element of $BP_{n,k,i}$. For example, the monotonicity regions used in [3] are given by the triangles of vertices $(0, 0)$, $(n\Delta_i/k, 0)$, $(0, n\Delta_i/k)$.

However, we note that the algorithms suffer from the following defects: (1) they require two passes over the data; (2) the results are dependent upon the order of data processing; (3) they are not local. These drawbacks look even more relevant in the case $n=3$, $k=1$ for co-monotone interpolation, if compared with the simple and efficient techniques proposed in [4]. It is worthwhile to recall that in [5] it is shown

* Received by the editors June 10, 1985; accepted for publication (in revised form) January 13, 1987. This work was supported by Ministero della Pubblica Istruzione.

† Dipartimento di Energetica, Università di Firenze, Via di Santa Marta, 3, 50139 Firenze, Italy.

that if (d_i, d_{i+1}) lies in the square of vertices $(0,0)$, $(3\Delta_i, 0)$, $(3\Delta_i, 3\Delta_i)$, $(0, 3\Delta_i)$ the corresponding cubic polynomial is monotone, and that the techniques of [4] consist of local formulas which force every couple (d_i, d_{i+1}), $i=0, 1, \cdots, N-1$, to lie in the square. We note parenthetically that the triangle used in [3] for polynomials of $BP_{3,1,i}$ (which coincides with P_3) is half of the above region.

The purpose of this note is to extend the results of [5] for the monotonicity of polynomials of $BP_{n,k,i}$ and the formulas of [4] to the space BS_n^k, giving an improvement of the method proposed in [3] for co-monotone interpolation.

The resulting algorithm is, to quote Fritsch and Butland [4, p. 303], "simple, symmetrical, completely local and produces reasonable results when applied to piecewise monotone data." From a practical point of view its advantage relies essentially on the possibility of obtaining splines of arbitrary continuity class.

2. The method. Let the data set (x_i, y_i), $i=0, 1, \cdots. N$, the values h_i, $i=0, 1, \cdots, N-1$, the slopes Δ_i, $i=0, 1, \cdots, N-1$ and the integers n, k be given as in § 1. Setting $r_i = h_i/n$, $i=0, 1, \cdots, N-1$, we introduce the sequence t_i, $i=0, 1, \cdots, 2N+1$, where

$$t_0 = x_0; \quad t_{2i+1} = x_i + kr_i, \quad t_{2i+2} = x_{i+1} - kr_i, \quad i=0, 1, \cdots, N-1, \quad t_{2N+1} = x_N,$$

which we use to define

$$L = \{l \in C[t_0, t_{2N+1}]: l(x) \in P_1 \text{ for } x \in [t_i, t_{i+1}], i=0, 1, \cdots, 2N\}. \tag{2.1}$$

We note that L is the space of linear splines with two knots in each subinterval (x_i, x_{i+1}), and that it is possible to select any $l \in L$ by its values and slopes at x_i, $i=0, 1, \cdots, N$. We recall that the n-degree Bernstein polynomial of $f(x)$, $x \in [x_i, x_{i+1}]$ has the properties [6]

$$B_i^{(j)}(x; f, n) = \frac{1}{h_i^n} \frac{n!}{(n-j)!} \sum_{q=0}^{n-j} \binom{n-j}{q} \Delta^j f(x_i + qr_i)(x - x_i)^q (x_{i+1} - x_i)^{n-j-q}, \tag{2.2}$$

$$B_i^{(j)}(x_i; f, n) = \frac{1}{h_i^j} \frac{n!}{(n-j)!} \Delta^j f(x_i), \tag{2.3}$$

$$B_i^{(j)}(x_{i+1}; f, n) = \frac{1}{h_i^j} \frac{n!}{(n-j)!} \nabla^j f(x_{i+1}) \tag{2.4}$$

for $j=0, 1, \cdots, n$, where $\Delta^j(\nabla^j)$ is the jth forward (backward) difference with step r_i. It is easy to obtain from (2.1)-(2.4) [2] that, for any $l \in L$, $B_i(x; l, n) \in BP_{n,k,i}$ (see (1.2)), and that if we introduce the linear operator

$$B: C[x_0, x_N] \to C[x_0, x_N]$$

defined by

$$(Bf)(x) = B_i(x; f, n) \quad \text{for } x \in [x_i, x_{i+1}], \quad i=0, 1, \cdots, N-1,$$

Bl is a polynomial spline of degree n and deficiency $n-k$ at the knots x_i, $i=1, \cdots, N-1$. More specifically, recalling definitions (1.1), (1.2) it is possible to show [2, Lemma 1], that the restriction of B to L is an invertible map from L to BS_n^k. Since BS_n^k is a linear space of dimension $2N+2$ it is equivalent to the space of C^1 cubic splines for $n=3$, $k=1$.

Now, we want to obtain $s \in BS_n^k$ which are interpolant, that is,

$$s(x_i) = y_i \qquad i=0, 1, \cdots, N,$$

and co-monotone, that is,

$$\begin{aligned} &s^{(1)}(x)\Delta_i > 0 \quad \text{for all } x \in (x_i, x_{i+1}) \text{ when } \Delta_i \neq 0, \\ &s^{(1)}(x) = 0 \quad \text{for all } x \in [x_i, x_{i+1}] \text{ when } \Delta_i = 0. \end{aligned} \tag{2.5}$$

We have from (2.3), (2.4) that

$$(Bl)(x_i) = l(x_i), \quad (Bl)^{(1)}(x_i) = l^{(1)}(x_i), \quad i = 0, 1, \cdots, N$$

and, since L and BS_n^k are isomorphic, $s = Bl \in BS_n^k$ is interpolant if and only if, $l(x_i) = y_i$, $i = 0, 1, \cdots, N$. Recalling the fact that any $l \in L$ is univocally determined by its values and slopes at x_i, $i = 0, 1, \cdots, N$, we can reformulate the problem into the following one: to find a sequence $d_0, d_1, \cdots, d_N$ such that for the linear spline $l \in L$ satisfying

$$l(x_i) = y_i;\ l^{(1)}(x_i) = d_i, \qquad i = 0, 1, \cdots, N \tag{2.6}$$

the corresponding $s = Bl$ satisfies (2.5). Further, for any $x \in [x_i, x_{i+1}]$, $i = 0, 1, \cdots, N-1$, $s(x)$ coincides with the polynomial $B_i(x; l, n)$ and $B_i(\cdot; l, n)$ is univocally determined by the (local) values y_i, y_{i+1} and d_i, d_{i+1}. Let M_i denote the "monotonicity region" for $B_i(\cdot; l, n)$, that is the subset of R^2 such that $B_i(\cdot; l, n)$ is monotone if and only if $(d_i, d_{i+1}) \in M_i$. The final formulation of the problem is now the following: to find a sequence of slopes $d_0, d_1, \cdots, d_N$ such that $(d_i, d_{i+1}) \in M_i$, $i = 0, 1, \cdots, N-1$. We note parenthetically that a solution for an abstract formulation of this problem is given in [2, Lemma 4].

It is practically impossible to compute the sets M_i for arbitrary values of n and k, and consequently the next steps are: (1) to determine a set Q_i, $Q_i \subset M_i$ useful for applications, and (2) to give a method for calculating the above sequences of slopes. (Obviously, these steps can be viewed in reverse order, in the sense that, having in mind local formulas for computing the slopes d_i, d_{i+1}, we would prove that the couple (d_i, d_{i+1}) lies in a subset Q_i of the monotonicity region M_i.)

A solution for (1) is provided in the theorem below, which is a consequence of the following Lemmas 1 and 2. The simple proofs of the two lemmas are reported in § 3 for easier reading. Let

$$\rho(n, k) = \frac{\dfrac{n}{n-2k} \displaystyle\sum_{q=k}^{n-k-1} \binom{n-1}{q}}{\dfrac{2k}{n-2k} \displaystyle\sum_{q=k}^{n-k-1} \binom{n-1}{q} - 2 \sum_{q=0}^{k-1} \binom{n-1}{q}} \tag{2.7}$$

(since the denominator of the right side is positive (see the proof of Lemma 2 in § 3) we immediately have $\rho(n, k) > 1$). Then we have the first lemma.

LEMMA 1. *The points* $(0, 0)$, $(n\Delta_i/k, 0)$, $(\rho(n, k)\Delta_i, \rho(n, k)\Delta_i)$, $(0, n\Delta_i/k)$ *lie on the boundary of the monotonicity region* M_i *and are the vertices of a polygon contained in* M_i.

Moreover, the second lemma follows.

LEMMA 2. *Let* n *and* k *be such that* $0 < k < n - k$. *Then* $\rho(n, k) \leqq n/k$.

As a consequence we can state the following theorem.

THEOREM. *The square* Q_i *of vertices* $(0, 0)$, $(\rho(n, k)\Delta_i, 0)$, $(\rho(n, k)\Delta_i, \rho(n, k)\Delta_i)$, $(0, \rho(n, k)\Delta_i)$ *is the largest square contained in the monotonicity region* M_i, $i = 0, 1, \cdots, N-1$.

The set Q_i and some values of $\rho(n, k)$ are shown in Fig. 1 and Table 1, respectively.

We note that, in the particular case $n = 3$, $k = 1$ we obtain the well-known results presented in [5]. Now, following [4], we choose the slopes using the formula

$$d_i = G(\Delta_{i-1}, \Delta_i), \qquad i = 1, 2, \cdots, N-1 \tag{2.8}$$

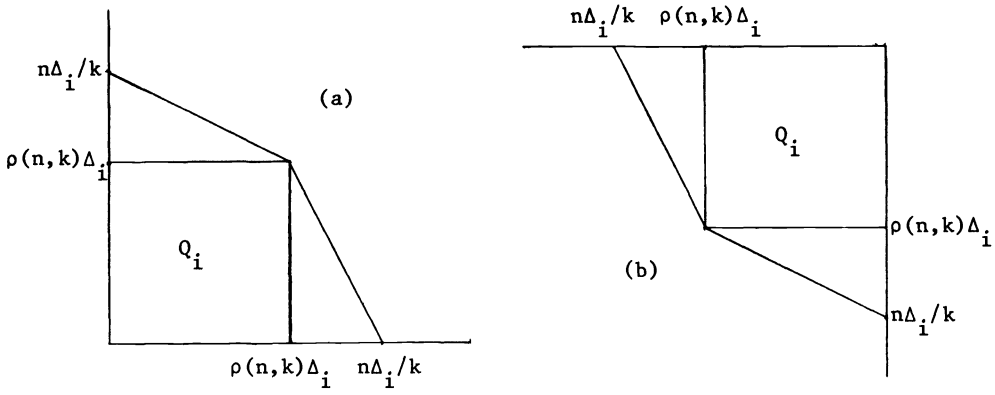


FIG. 1. *The polygon and the square Q_i for*: (a) $\Delta_i > 0$; (b) $\Delta_i < 0$.

TABLE 1
Some values of $\rho(n, k)$.

k \ n	3	4	5	6	7	8	9	10	11
1	3	3	$\frac{35}{11}$	$\frac{45}{13}$	$\frac{217}{57}$	$\frac{21}{5}$	$\frac{1143}{247}$	$\frac{1275}{251}$	$\frac{5621}{1013}$
2			$\frac{15}{7}$	$\frac{15}{7}$	$\frac{175}{79}$	$\frac{7}{3}$	$\frac{1071}{431}$	$\frac{205}{77}$	$\frac{5511}{1927}$
3					$\frac{35}{19}$	$\frac{35}{19}$	$\frac{273}{145}$	$\frac{525}{269}$	$\frac{5016}{2456}$
4							$\frac{315}{187}$	$\frac{315}{187}$	$\frac{77}{45}$
5									$\frac{693}{437}$

and the noncentered three-points difference formula as suggested in [5, p. 242] for d_0 and d_N. The function G is given by

$$G(\Delta_{i-1}, \Delta_i) = \begin{cases} 0 & \text{if } \Delta_{i-1}\Delta_i \leqq 0, \\ \dfrac{\rho(n,k)\Delta_i\Delta_{i-1}}{\Delta_i + (\rho(n,k)-1)\Delta_{i-1}} & \text{if } |\Delta_{i-1}| \leqq |\Delta_i|, \\ \dfrac{\rho(n,k)\Delta_i\Delta_{i-1}}{\Delta_{i-1} + (\rho(n,k)-1)\Delta_i} & \text{if } |\Delta_i| \leqq |\Delta_{i-1}| \end{cases} \tag{2.9}$$

and corresponds to formula (3) of [4] for $\rho(n, k) = \rho(3, 1) = 3$.

It is easy to verify that the function G satisfies the following properties (see [4, p. 301])

$$G(\Delta_{i-1}, \Delta_i) = G(\Delta_i, \Delta_{i-1})$$

and so it is independent of the order of data processing;

$$\min(\Delta_{i-1}, \Delta_i) \leqq G(\Delta_{i-1}, \Delta_i) \leqq \max(\Delta_{i-1}, \Delta_i),$$

that is, the spline has a slope intermediate between the adjacent slopes of data segments,

$$G(\Delta_{i-1}, \Delta_i) = 0 \quad \text{if } \Delta_i\Delta_{i-1} \leqq 0,$$
$$|G(\Delta_{i-1}, \Delta_i)| \leqq \rho(n, k) \min(|\Delta_{i-1}|, |\Delta_i|)$$

and this implies that $(d_i, d_{i+1}) \in Q_i$, $i = 0, 1, \cdots, N-1$ or, in other words, that the corresponding polynomials $B_i(\cdot, l, n)$ are monotone in $[x_i, x_{i+1}]$, $i = 0, 1, \cdots, N-1$.

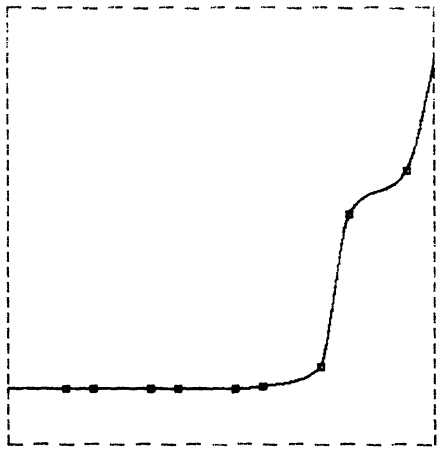


FIG. 2

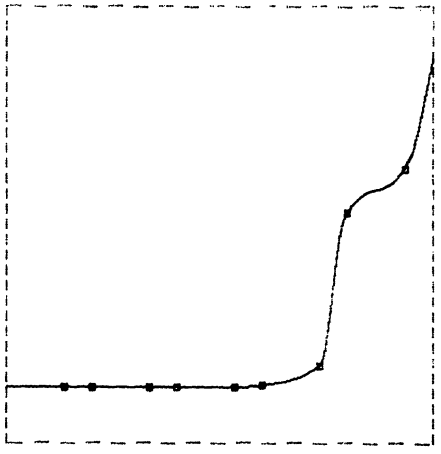


FIG. 3

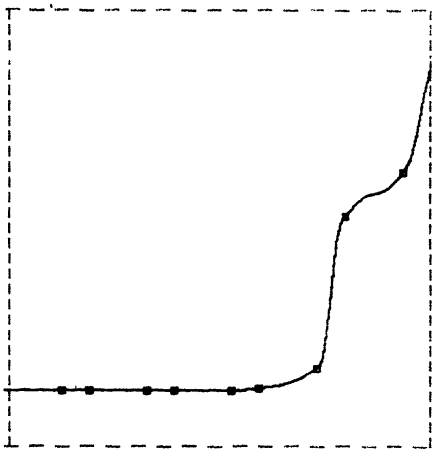


FIG. 4

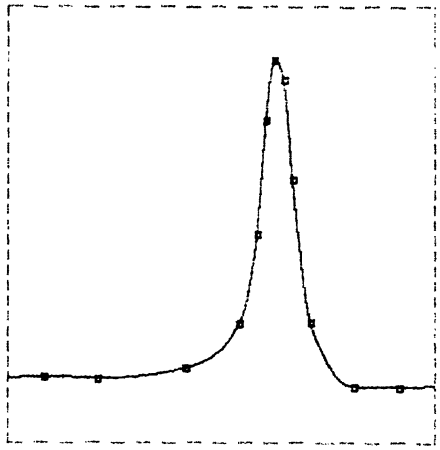


FIG. 5

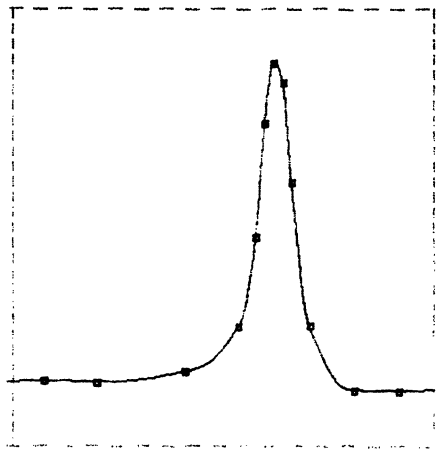


FIG. 6

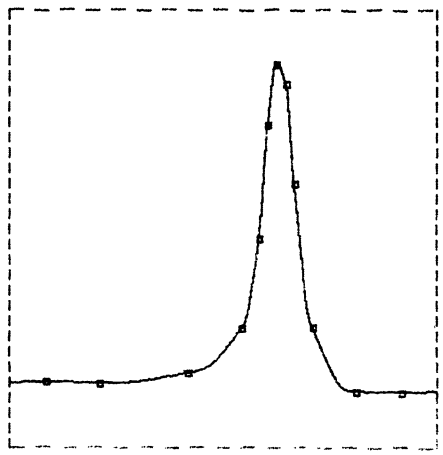


FIG. 7

TABLE 2

i	x_i	y_i
0	0	10
1	2	10
2	3	10
3	5	10
4	6	10
5	8	10
6	9	10.5
7	11	15
8	12	50
9	14	60
10	15	85

TABLE 3

i	x_i	y_i
0	595	.644
1	635	.652
2	695	.644
3	795	.694
4	855	.907
5	875	1.336
6	885	1.881
7	895	2.169
8	905	2.075
9	915	1.598
10	935	.916
11	985	.607
12	1035	.603
13	1075	.608

In Figs. 2-4 we exhibit the curves produced by the method described above for $n=3$, $k=1$; $n=5$, $k=2$; $n=7$, $k=3$ for the set of Table 2, corresponding to the test Akima3 of [5] and [4]. Figures 5-7 show the curves of identical tests applied to some titanium heat data [1] given in Table 3.

3. Proofs of Lemmas 1 and 2.

Proof of Lemma 1. From the shape-preserving properties of Bernstein polynomials [6, p. 23] we have that the monotonicity of $l \in L$ in $[x_i, x_{i+1}]$ implies the monotonicity of the corresponding $Bl \in BS_n^k$ in the same interval. If we impose the increase (decrease) of l in $[x_i, x_{i+1}]$, that is,

$$l(x_i) \leqq l(t_{2i+1}) \leqq l(t_{2i+2}) \leqq l(x_{i+1}), \qquad (l(x_i) \geqq l(t_{2i+1}) \geqq l_{(t2i+2)} \geqq l(x_{i+1})),$$

we obtain immediately (see [2, Lemma 3] for related results) that the set

$$\begin{aligned} &\{(d_i, d_{i+1}) \in R^2 \colon d_i \geqq 0,\ d_{i+1} \geqq 0,\ d_{i+1} \leqq -d_i + n\Delta_i/k\}, \\ &(\{(d_i, d_{i+1}) \in R^2 \colon d_i \leqq 0,\ d_{i+1} \leqq 0,\ d_{i+1} \geqq -d_i + n\Delta_i/k\}) \end{aligned} \tag{3.1}$$

is included in M_i. We note that the set (3.1) is the triangle of vertices $(0,0)$, $(n\Delta_i/k, 0)$, $(0, n\Delta_i/k)$. Now we observe that, for any $l \in L$ and satisfying (2.6), one has

$$\Delta l(x_i + qr_i) = \begin{cases} r_i d_i & \text{for } q = 0, \cdots, k-1, \\ r_i(n\Delta_i - kd_{i+1} - kd_i)/(n-2k) & \text{for } q = k, \cdots, n-k-1, \\ r_i d_{i+1} & \text{for } q = n-k, \cdots, n-1, \end{cases} \tag{3.2}$$

$$\Delta^2 l(x_i + qr_i) = \begin{cases} r_i[(n\Delta_i - kd_{i+1} - kd_i)/(n-2k) - d_i] & \text{for } q = k-1, \\ r_i[d_{i+1} - (n\Delta_i - kd_{i+1} - kd_i)/(n-k)] & \text{for } q = n-k-1, \\ 0 & \text{elsewhere.} \end{cases} \tag{3.3}$$

Hence, from (2.2) we obtain

$$\begin{aligned} B_i^{(2)}(x; l, n) &= [(x - x_i)(x_{i+1} - x)]^{k-1}[\lambda(x - x_i)^{n-2k} + \mu(x_{i+1} - x)^{n-2k}], \\ \lambda &= \frac{n(n-1)}{h_i^2}\binom{n-2}{k-1}\Delta^2 l(x_i + (n-k-1)r_i), \\ \mu &= \frac{n(n-1)}{h_i^2}\binom{n-2}{k-1}\Delta^2 l(x_i + (k-1)r_i). \end{aligned} \tag{3.4}$$

From (3.4) we see immediately that, except for the trivial case $\lambda = \mu = 0$, the polynomial $B_i^{(2)}(x; l, n)$ has, at most, one zero in (x_i, x_{i+1}) when $\lambda\mu < 0$. For the particular choice $d_i = d_{i+1}$ (which implies, via (3.3), $\lambda = -\mu$) this is located at $x_i + h_i/2$. In this point the polynomial $B_i^{(1)}(x; l, n)$ achieves its unique extreme of (x_i, x_{i+1}). From (2.2) and (3.2) we have, for $d_i = d_{i+1}$,

$$B_i^{(1)}(x_i + h_i/2; l, n) = \frac{1}{2^{n-1}}\left[d_i\left(2\sum_{q=0}^{k-1}\binom{n-1}{q} - \frac{2k}{n-2k}\sum_{q=k}^{n-k-1}\binom{n-1}{q}\right) + \frac{n\Delta_i}{n-2k}\sum_{q=k}^{n-k-1}\binom{n-1}{q}\right]$$

and consequently,

$$B_i^{(1)}(x_i + h_i; l, n) \geqq 0 \quad (\leqq 0) \quad \text{iff } d_i \leqq \rho(n,k)\Delta_i \quad (d_i \geqq \rho(n,k)\Delta_i),$$

where $\rho(n, k)$ is given by formula (2.7). Thus $(\rho(n,k)\Delta_i, \rho(n,k)\Delta_i) \in M_i$ and, since the linearity of the operator B and definition (2.1) imply that M_i is convex, the proof is then complete.

Proof of Lemma 2. Using (2.7) the inequality $\rho(n, k) \leq n/k$ is equivalent to

$$\rho(n,k) = \frac{\frac{n}{n-2k}\sum_{q=k}^{n-k-1}\binom{n-1}{q}}{\frac{2k}{n-2k}\sum_{q=k}^{n-k-1}\binom{n-1}{q} - 2\sum_{q=0}^{k-1}\binom{n-1}{q}} \leqq \frac{n}{k}. \tag{3.5}$$

We note that

$$\frac{2k}{n-2k}\sum_{q=k}^{n-k-1}\binom{n-1}{q} - 2\sum_{q=0}^{k-1}\binom{n-1}{q} \geqq 0$$

since the two terms

$$\frac{1}{n-2k}\sum_{q=k}^{n-k-1}\binom{n-1}{q} \quad \text{and} \quad \frac{1}{k}\sum_{q=0}^{k-1}\binom{n-1}{q}$$

are, respectively, the average of the first k elements and the central $n-2k$ elements of the binomial terms $\binom{n-1}{q}$, $q=0,1,\cdots,n-1$. Hence (3.5) reads

$$\frac{2}{k}\sum_{q=0}^{k-1}\binom{n-1}{q}\leqq\frac{1}{n-2k}\sum_{q=k}^{n-k-1}\binom{n-1}{q}.$$

Now we want to show that

$$\frac{2}{k}\sum_{q=0}^{k-1}\binom{n-1}{q}\leqq\binom{n-1}{k}.$$

For $k=1$ this is obviously true since $n\geqq 3$. Now suppose that

$$2\sum_{q=0}^{k-2}\binom{n-1}{q}\leqq(k-1)\binom{n-1}{k-1}.$$

Then, since

$$2\sum_{q=0}^{k-1}\binom{n-1}{q}=2\sum_{q=0}^{k-2}\binom{n-1}{q}+2\binom{n-1}{k-1}\leqq(k+1)\binom{n-1}{k-1}\leqq(k+1)\binom{n-1}{k-1}\frac{n-k}{k-1}$$
$$=k\binom{n-1}{k},$$

we can apply the induction principle. The proof is completed by observing that

$$\binom{n-1}{k}\leqq\frac{1}{n-2k}\sum_{q=k}^{n-k-1}\binom{n-1}{q},$$

since the second term is the average of elements which are greater than or equal to the first term.

REFERENCES

[1] C. DE BOOR, *A Practical Guide to Spline*, Springer-Verlag, New York, 1978.

[2] P. COSTANTINI, *On monotone and convex spline interpolation*, Math. Comp., 46 (1986), pp. 203-214.

[3] ———, *An algorithm for computing shape-preserving interpolating splines of arbitrary degree*, J. Comput. Appl. Math., submitted.

[4] F. N. FRITSCH AND J. BUTLAND, *A method for constructing local monotone piecewise cubic interpolants*, this Journal, 5 (1984), pp. 300-304.

[5] F. N. FRITSCH AND R. E. CARLSON, *Monotone piecewise cubic interpolation*, SIAM J. Numer. Anal., 17 (1980), pp. 238-246.

[6] G. G. LORENTZ, *Bernstein Polynomials*, University of Toronto Press, Toronto, Canada, 1953.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 6, November 1987
© 1987 Society for Industrial and Applied Mathematics
013

A STATISTICAL ANALYSIS OF GENERALIZED FEEDBACK SHIFT REGISTER PSEUDORANDOM NUMBER GENERATORS*

HARALD NIEDERREITER†

Abstract. We analyze the statistical performance of uniform pseudorandom numbers generated by the generalized feedback shift register (GFSR) algorithm of Lewis and Payne. Tests for equidistribution and for statistical independence of successive pseudorandom numbers are considered. We establish effective criteria for the choice of parameters in the GFSR algorithm which guarantee a good performance under these tests.

Key words. uniform pseudorandom numbers, GFSR algorithm, uniformity test, serial test, discrepancy

AMS(MOS) subject classification. 65C10

1. Introduction. Feedback shift registers are standard sources of pseudorandom digits, but they can also be used to generate uniform pseudorandom numbers in the interval [0, 1] if suitable transformations from strings of digits to numbers in [0, 1] are employed. Consider a feedback shift register with modular arithmetic relative to the prime p (in the most common binary case we have $p = 2$). The output of such a feedback shift register can be described as a sequence $y_0, y_1, \cdots$ of integers with $0 \leqq y_n < p$ satisfying the k-step recursion

$$y_{n+k} \equiv a_{k-1}y_{n+k-1} + \cdots + a_0 y_n \bmod p \quad \text{for } n = 0, 1, \cdots, \tag{1}$$

where the a_i are constant integral coefficients. In the *digital method* due to Tausworthe [17], the sequence of y_n is transformed into a sequence of uniform pseudorandom numbers in the following way: choose a block length m and set

$$x_n = \sum_{j=1}^{m} y_{mn+j-1} p^{-j} \quad \text{for } n = 0, 1, \cdots,$$

i.e., interpret consecutive blocks of y_n of length m as the digit expansions in the base p of numbers in [0, 1). To generate one new uniform pseudorandom number, we need to calculate m new terms y_n.

A method that is in a sense more efficient was proposed by Lewis and Payne [4] and called the *generalized feedback shift register* (GFSR) *algorithm.* Here we choose m distinct nonnegative integers $k_1, \cdots, k_m$ and set

$$x_n = \sum_{j=1}^{m} y_{n+k_j} p^{-j} \quad \text{for } n = 0, 1, \cdots. \tag{2}$$

We note that the blocks $X_n = y_{n+k_1} y_{n+k_2} \cdots y_{n+k_m}$ of length m satisfy the k-step recursion

$$X_{n+k} = a_{k-1}X_{n+k-1} + \cdots + a_0 X_n \quad \text{for } n = 0, 1, \cdots,$$

if arithmetic operations with blocks are defined termwise and modulo p. Therefore, to generate one new uniform pseudorandom number, we only have to form a linear combination of blocks that are known already. Various properties of GFSR pseudorandom number generators are discussed in Lewis and Payne [4] as well as in the later papers of Arvillias and Maritsas [1], Fushimi and Tezuka [2], Kirkpatrick and Stoll [3], Marsaglia and Tsay [7], and Pavlov and Pohodzeĭ [16].

* Received by the editors August 19, 1985; accepted for publication (in revised form) August 4, 1986. This work was supported by the Austrian Ministry for Science and Research.

† Mathematical Institute, Austrian Academy of Sciences, A-1010 Vienna, Austria.

For many numerical applications of uniform pseudorandom numbers, the most important statistical properties are *equidistribution* in [0, 1] and *statistical independence* of successive pseudorandom numbers. Statistical tests for both of these properties can be set up in a unified manner. Let $x_0, x_1, \cdots$ be a sequence of uniform pseudorandom numbers in [0, 1] and $s \geqq 1$ a given dimension. For the s-tuples

$$\mathbf{x}_n = (x_n, x_{n+1}, \cdots, x_{n+s-1}) \in [0, 1]^s \tag{3}$$

consider the maximum deviation between the empirical distribution of $\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_{N-1}$ and the uniform distribution on $[0, 1]^s$. The latter distribution corresponds to the ideal case, in the sense that it is the distribution of the random vector $\mathbf{x}_n$ when $x_0, x_1, \cdots$ are uniformly distributed independent random variables. In detail, the maximum deviation is given by the *discrepancy*

$$D_N^{(s)} = \sup_J |E_N(J) - V(J)| \quad \text{for } N \geqq 1, \tag{4}$$

where $E_N(J)$ is N^{-1} times the number of terms among $\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_{N-1}$ falling into the interval J, $V(J)$ denotes the volume of J, and the supremum is extended over all subintervals J of $[0, 1]^s$ of the form

$$J = [0, t_1] \times \cdots \times [0, t_s].$$

In this way we test for $s = 1$ the equidistribution of the sequence $x_0, x_1, \cdots$ (*uniformity test*) and for $s \geqq 2$ the statistical independence of s successive terms in this sequence (*s-dimensional serial test*). The sequence of pseudorandom numbers passes the uniformity test if the discrepancy $D_N^{(1)}$ is small for large N and it passes the s-dimensional serial test if the discrepancy $D_N^{(s)}$ is small for large N.

In this paper we establish upper and lower bounds for the discrepancy $D_N^{(s)}$ in the case where the sequence $x_0, x_1, \cdots$ is generated by the GFSR algorithm. We note that the discrepancy occurs in error bounds for quasi-Monte Carlo integration over the s-dimensional unit cube $[0, 1]^s$ (see [9], [13]), so that the results of this paper have immediate applications to numerical integration. Furthermore, other statistical quantities can be bounded in terms of an appropriate $D_N^{(s)}$. For instance, for the serial correlation coefficient

$$\sigma_N = \frac{M[(x_n - M(x_n))(x_{n+1} - M(x_{n+1}))]}{M[(x_n - M(x_n))^2]^{1/2} M[(x_{n+1} - M(x_{n+1}))^2]^{1/2}},$$

with $M(u_n)$ denoting the mean value of the numbers $u_0, u_1, \cdots, u_{N-1}$, we have

$$|\sigma_N| < 73 D_N^{(2)}$$

by a result in [12]. We remark that in the case where the sequence $x_0, x_1, \cdots$ is generated by the digital method, bounds for $D_N^{(s)}$ were established in [10], [11], [15].

With regard to the feedback shift register sequence $y_0, y_1, \cdots$ defined by (1) *we assume that $a_0 \not\equiv 0 \bmod p$ and that the initial values $y_0, y_1, \cdots, y_{k-1}$ are not all* 0. We associate with this sequence the *characteristic polynomial*

$$f(x) = x^k - a_{k-1}x^{k-1} - \cdots - a_0, \tag{5}$$

considered as a polynomial over the finite field $F_p = \mathbb{Z}/p\mathbb{Z}$, and *we assume that $f(x)$ is irreducible over F_p*. Then the sequence $y_0, y_1, \cdots$ is periodic and its least period τ can be described as follows. Take a fixed root α of $f(x)$ in the finite field F_q with $q = p^k$ elements; then τ is equal to the order of α in the multiplicative group F_q^* of nonzero elements of F_q. Consequently, the maximal value of τ is $\tau = p^k - 1$, and this

value can be achieved if we choose for $f(x)$ a *primitive polynomial* over F_p, i.e., a monic polynomial over F_p of degree k which has a generator of the cyclic group F_q^* as a root. Let Tr denote the trace function from F_q onto F_p, which is an F_p-linear mapping. Then we have the following explicit formula for the y_n viewed as elements of F_p: there exists $\beta \in F_q^*$ such that

$$y_n = \mathrm{Tr}(\beta \alpha^n) \quad \text{for } n = 0, 1, \cdots. \tag{6}$$

These facts can be found in Lidl and Niederreiter [5], [6]. If the sequence $y_0, y_1, \cdots$ has least period τ, then it is seen immediately that the sequence $x_0, x_1, \cdots$ defined by (2) has least period τ. Because of the periodic nature of the x_n, it is only of interest to study $D_N^{(s)}$ for $1 \leqq N \leqq \tau$. The italicized assumptions in this paragraph will be standing hypotheses for the rest of the paper.

A brief survey of the paper follows. In § 2 we present some auxiliary results. In § 3 we establish upper bounds for the discrepancy $D_N^{(s)}$ for the full period, i.e., for $N = \tau$, as well as for parts of the period, i.e., for $N < \tau$. Although most of the research so far has been devoted to the behavior over the full period, the performance for parts of the period is of great practical significance since in a typical Monte Carlo application one will only work with a segment of the period. The results attain a somewhat simpler form in the most common special case, namely when $p = 2$ and the characteristic polynomial $f(x)$ is a primitive polynomial over F_2. The bounds in § 3 depend in a rather complicated way on the essential parameters (the characteristic polynomial $f(x)$ and the integers $k_1, \cdots, k_m$), and so we introduce in § 4 a positive integer (called the "figure of merit") that measures the suitability of the parameters in a more convenient way. Section 5 contains lower bounds for the discrepancy $D_N^{(s)}$, in particular a lower bound in terms of the figure of merit. This shows that the upper bounds in § 3 are in a sense best possible. A discussion of the results is presented in § 6. The author is grateful to two anonymous referees for their detailed comments on the paper.

2. Preparatory results. The following elementary inequality is needed in the proof of the main result of this section, an upper bound for the discrepancy in terms of exponential sums given in Lemma 2.

LEMMA 1. *Let* $u_1, \cdots, u_s, t_1, \cdots, t_s \in [0, 1]$ *be such that for some* $v \in [0, 1]$ *we have*

$$0 \leqq u_i - t_i \leqq v \quad \textit{for } 1 \leqq i \leqq s.$$

Then

$$u_1 \cdots u_s - t_1 \cdots t_s \leqq 1 - (1 - v)^s.$$

Proof. We proceed by induction on s and note that the inequality is trivial for $s = 1$. Now suppose the inequality holds for some $s \geqq 1$; then

$$\begin{aligned} u_1 \cdots u_s u_{s+1} - t_1 \cdots t_s t_{s+1} &= u_1 \cdots u_s (u_{s+1} - t_{s+1}) + (u_1 \cdots u_s - t_1 \cdots t_s) t_{s+1} \\ &\leqq u_{s+1} - t_{s+1} + (1 - (1 - v)^s) t_{s+1} \\ &= (1 - (1 - v)^s) u_{s+1} + (1 - v)^s (u_{s+1} - t_{s+1}) \\ &\leqq 1 - (1 - v)^s + (1 - v)^s v = 1 - (1 - v)^{s+1}, \end{aligned}$$

and the induction is complete. □

Let $\mathbf{w}_0, \mathbf{w}_1, \cdots, \mathbf{w}_{N-1}$ be arbitrary points in $[0, 1]^s$ with the property that all their coordinates have finite digit expansions (relative to an integer base $b \geqq 2$) of the form

$\sum_{j=1}^{m} v_j b^{-j}$, where $v_j \in \{0, 1, \cdots, b-1\}$ and m is fixed. In detail, let $\mathbf{w}_n = (w_{n1}, \cdots, w_{ns})$ with

$$w_{ni} = \sum_{j=1}^{m} w_{ni}^{(j)} b^{-j} \quad \text{for } 0 \leqq n < N, \quad 1 \leqq i \leqq s,$$

where the digits $w_{ni}^{(j)}$ are integers in the interval $[0, b-1]$.

Let $C(b)$ be the set of all integers h with $-b/2 < h \leqq b/2$. For $H = (h_1, \cdots, h_m)$ with $h_j \in C(b)$ for $1 \leqq j \leqq m$ we define

$$d(H) = d(h_1, \cdots, h_m) = \begin{cases} 0 & \text{if } H = (0, \cdots, 0), \\ \text{largest index } d \text{ with } h_d \neq 0 & \text{if } H \neq (0, \cdots, 0). \end{cases}$$

For $b = 2$ we set

$$Q(H) = Q(h_1, \cdots, h_m) = 2^{-d(H)} \tag{7}$$

and for $b > 2$ we set

$$Q(H) = Q(h_1, \cdots, h_m) = \begin{cases} 1 & \text{if } H = (0, \cdots, 0), \\ b^{-d}((1/\sin \pi |h_d/b|) + \delta_d) & \text{if } H \neq (0, \cdots, 0), \end{cases} \tag{8}$$

where $d = d(H)$ and where $\delta_d = 1$ if $d < m$, but $\delta_m = 0$. Now let $C_{ms}(b)$ be the set of ms-dimensional lattice points of the form

$$\mathbf{h} = (h_{ij}) = (h_{11}, \cdots, h_{1m}, \cdots, h_{s1}, \cdots, h_{sm})$$

with $h_{ij} \in C(b)$ for $1 \leqq i \leqq s$, $1 \leqq j \leqq m$. For each such $\mathbf{h}$ we define the weight

$$P(\mathbf{h}) = \prod_{i=1}^{s} Q(h_{i1}, \cdots, h_{im}). \tag{9}$$

In the case $b = 2$ this reduces to the expression

$$P(\mathbf{h}) = 2^{-D(\mathbf{h})} \quad \text{with } D(\mathbf{h}) = \sum_{i=1}^{s} d(h_{i1}, \cdots, h_{im}). \tag{10}$$

For integers h we use the complex exponential function $e_b(h) = e^{2\pi\sqrt{-1}h/b}$. The following result is a multidimensional version of Lemma 1 in [11] and a refinement of Lemma 1 in [10].

LEMMA 2. *For the discrepancy $D_N^{(s)}$ of the points $\mathbf{w}_n$, $0 \leqq n < N$, we have*

$$D_N^{(s)} \leqq 1 - (1 - b^{-m})^s + \sum_{\mathbf{h} \neq \mathbf{0}} P(\mathbf{h}) \left| \frac{1}{N} \sum_{n=0}^{N-1} e_b\left(\sum_{i=1}^{s} \sum_{j=1}^{m} h_{ij} w_{ni}^{(j)} \right) \right|,$$

where the outer sum is extended over all nonzero lattice points $\mathbf{h} = (h_{ij}) \in C_{ms}(b)$.

Proof. Let $J = [0, t_1] \times \cdots \times [0, t_s]$ be a subinterval of $[0, 1]^s$ and let $A(J; N)$ denote the number of $n < N$ with $\mathbf{w}_n \in J$. Since $\mathbf{w}_n \in J$ if and only if $w_{ni} \leqq t_i$ for $1 \leqq i \leqq s$, we have

$$A(J; N) = \sum_{n=0}^{N-1} c_{t_1}(w_{n1}) \cdots c_{t_s}(w_{ns}), \tag{11}$$

where c_t denotes the characteristic function of the interval $[0, t]$. For $1 \leqq i \leqq s$ let

$$t_i = \sum_{r=1}^{\infty} t_i^{(r)} b^{-r}$$

be the digit expansion of t_i to the base b, with the usual condition that $t_i^{(r)} < b-1$ for infinitely many r, except in the case $t_i = 1$ where we take $t_i^{(r)} = b-1$ for all r. Put

$$u_i^{(r)} = \begin{cases} t_i^{(r)} & \text{for } 1 \leqq r < m, \\ t_i^{(r)} + 1 & \text{for } r = m. \end{cases} \tag{12}$$

For integers $v, w \in [0, b-1]$ let $c(v, w) = 1$ if $v = w$ and $c(v, w) = 0$ if $v \neq w$. Then we have

$$c_{t_i}(w_{ni}) = \sum_{r=1}^{m} c(u_i^{(1)}, w_{ni}^{(1)}) \cdots c(u_i^{(r-1)}, w_{ni}^{(r-1)}) \sum_{v=0}^{u_i^{(r)}-1} c(v, w_{ni}^{(r)})$$

by an argument in the proof of Lemma 1 in [11], where we use the standard conventions that empty sums are interpreted as 0 and empty products are interpreted as 1. Now

$$c(v, w) = \frac{1}{b} \sum_{h \in C(b)} e_b(h(w-v)),$$

and so

$$\begin{aligned} c_{t_i}(w_{ni}) &= \sum_{r=1}^{m} b^{-r} \sum_{h_1, \cdots, h_r \in C(b)} e_b\left(\sum_{j=1}^{r} h_j w_{ni}^{(j)} - \sum_{j=1}^{r-1} h_j u_i^{(j)} \right) \sum_{v=0}^{u_i^{(r)}-1} e_b(-h_r v) \\ &= \sum_{r=1}^{m} b^{-r} \sum_{h_1, \cdots, h_r \in C(b)} e_b\left(\sum_{j=1}^{r} h_j w_{ni}^{(j)} - \sum_{j=1}^{r} h_j u_i^{(j)} \right) \sum_{v=0}^{u_i^{(r)}-1} e_b(h_r(u_i^{(r)} - v)) \\ &= \sum_{r=1}^{m} b^{-r} \sum_{h_1, \cdots, h_r \in C(b)} e_b\left(\sum_{j=1}^{r} h_j (w_{ni}^{(j)} - u_i^{(j)}) \right) T(h_r, u_i^{(r)}) \end{aligned}$$

with

$$T(h, L) = \sum_{v=1}^{L} e_b(hv).$$

Inserting this in (11), we get

$$\begin{aligned} A(J; N) = \sum_{n=0}^{N-1} \sum_{r_1, \cdots, r_s = 1}^{m} b^{-r_1 - \cdots - r_s} \sum_{\substack{h_{ij} \in C(b) \\ 1 \leqq j \leqq r_i, 1 \leqq i \leqq s}} e_b\left(\sum_{i=1}^{s} \sum_{j=1}^{r_i} h_{ij}(w_{ni}^{(j)} - u_i^{(j)}) \right) \\ \cdot T(h_{1,r_1}, u_1^{(r_1)}) \cdots T(h_{s,r_s}, u_s^{(r_s)}). \end{aligned}$$

Separating the contribution from the term in the inner sum in which all $h_{ij} = 0$ and denoting by an asterisk the deletion of this term, we obtain using $T(0, L) = L$,

$$\begin{aligned} A(J; N) = N \sum_{r_1, \cdots, r_s = 1}^{m} u_1^{(r_1)} \cdots u_s^{(r_s)} b^{-r_1 - \cdots - r_s} \\ + \sum_{r_1, \cdots, r_s = 1}^{m} b^{-r_1 - \cdots - r_s} \sum_{\substack{h_{ij} \in C(b) \\ 1 \leqq j \leqq r_i, 1 \leqq i \leqq s}}^{*} e_b\left(-\sum_{i=1}^{s} \sum_{j=1}^{r_i} h_{ij} u_i^{(j)} \right) \\ \cdot T(h_{1,r_1}, u_1^{(r_1)}) \cdots T(h_{s,r_s}, u_s^{(r_s)}) \sum_{n=0}^{N-1} e_b\left(\sum_{i=1}^{s} \sum_{j=1}^{r_i} h_{ij} w_{ni}^{(j)} \right). \end{aligned}$$

Hence

$$A(J;N)-N\prod_{i=1}^{s}\left(\sum_{r=1}^{m}u_i^{(r)}b^{-r}\right)$$

$$=\sum_{r_1,\cdots,r_s=1}^{m}b^{-r_1-\cdots-r_s}\sum_{\substack{h_{ij}\in C(b)\\ 1\leqq j\leqq m,1\leqq i\leqq s\\ h_{ij}=0\text{ for }j>r_i}}^{*}e_b\left(-\sum_{i=1}^{s}\sum_{j=1}^{m}h_{ij}u_i^{(j)}\right)$$

$$\cdot\,T(h_{1,r_1},u_1^{(r_1)})\cdots T(h_{s,r_s},u_s^{(r_s)})\sum_{n=0}^{N-1}e_b\left(\sum_{i=1}^{s}\sum_{j=1}^{m}h_{ij}w_{ni}^{(j)}\right).$$

Now we interchange the order of summation and sum first over the h_{ij}, thus obtaining an outer sum over all nonzero $\mathbf{h}=(h_{ij})\in C_{ms}(b)$. We note that the condition $h_{ij}=0$ for $j>r_i$, $1\leqq i\leqq s$, is equivalent to $r_i\geqq a(\mathbf{h}_i)$ for $1\leqq i\leqq s$, where $\mathbf{h}_i=(h_{i1},\cdots,h_{im})$, $a(\mathbf{h}_i)=d(h_{i1},\cdots,h_{im})$ for $\mathbf{h}_i\neq\mathbf{0}$ and $a(\mathbf{0})=1$. Therefore we get

$$A(J;N)-N\prod_{i=1}^{s}\left(\sum_{r=1}^{m}u_i^{(r)}b^{-r}\right)$$

$$=\sum_{\mathbf{h}\neq\mathbf{0}}e_b\left(-\sum_{i=1}^{s}\sum_{j=1}^{m}h_{ij}u_i^{(j)}\right)\left(\sum_{n=0}^{N-1}e_b\left(\sum_{i=1}^{s}\sum_{j=1}^{m}h_{ij}w_{ni}^{(j)}\right)\right)$$

$$\cdot\sum_{r_1=a(\mathbf{h}_1)}^{m}\cdots\sum_{r_s=a(\mathbf{h}_s)}^{m}b^{-r_1-\cdots-r_s}T(h_{1,r_1},u_1^{(r_1)})\cdots T(h_{s,r_s},u_s^{(r_s)}).$$

It follows that

$$\left|\frac{A(J;N)}{N}-V(J)\right|\leqq\left|\prod_{i=1}^{s}\left(\sum_{r=1}^{m}u_i^{(r)}b^{-r}\right)-\prod_{i=1}^{s}t_i\right|$$

$$+\sum_{\mathbf{h}\neq\mathbf{0}}\left|\frac{1}{N}\sum_{n=0}^{N-1}e_b\left(\sum_{i=1}^{s}\sum_{j=1}^{m}h_{ij}w_{ni}^{(j)}\right)\right|\prod_{i=1}^{s}\left|\sum_{r=a(\mathbf{h}_i)}^{m}b^{-r}T(h_{ir},u_i^{(r)})\right|.$$

For $1\leqq i\leqq s$ we set

$$u_i=\sum_{r=1}^{m}u_i^{(r)}b^{-r}.$$

Then it follows from (12) that

$$0\leqq u_i-t_i\leqq b^{-m}\quad\text{for }1\leqq i\leqq s.$$

Also $u_i,\,t_i\in[0,1]$, and so by Lemma 1

$$\left|\frac{A(J;N)}{N}-V(J)\right|\leqq 1-(1-b^{-m})^s \tag{13}$$

$$+\sum_{\mathbf{h}\neq\mathbf{0}}\left|\frac{1}{N}\sum_{n=0}^{N-1}e_b\left(\sum_{i=1}^{s}\sum_{j=1}^{m}h_{ij}w_{ni}^{(j)}\right)\right|\prod_{i=1}^{s}|R(\mathbf{h}_i,u_i)|$$

with

$$R(\mathbf{h}_i,u_i)=\sum_{r=a(\mathbf{h}_i)}^{m}b^{-r}T(h_{ir},u_i^{(r)}).$$

If $\mathbf{h}_i = \mathbf{0}$, then

$$|R(\mathbf{h}_i, u_i)| = \left| \sum_{r=1}^{m} b^{-r} T(0, u_i^{(r)}) \right| = \left| \sum_{r=1}^{m} b^{-r} u_i^{(r)} \right|$$

$$= u_i \leqq 1 = Q(h_{i1}, \cdots, h_{im})$$

by (7) and (8). If $b > 2$, $\mathbf{h}_i \neq \mathbf{0}$, we put $d = a(\mathbf{h}_i) = d(h_{i1}, \cdots, h_{im})$ and get

$$|R(\mathbf{h}_i, u_i)| = \left| \sum_{r=d}^{m} b^{-r} T(h_{ir}, u_i^{(r)}) \right| = \left| b^{-d} T(h_{id}, u_i^{(d)}) + \sum_{r=d+1}^{m} b^{-r} T(0, u_i^{(r)}) \right|$$

$$\leqq b^{-d} |T(h_{id}, u_i^{(d)})| + \sum_{r=d+1}^{m} b^{-r} u_i^{(r)} \leqq b^{-d} |T(h_{id}, u_i^{(d)})| + b^{-d} \delta_d.$$

For $h \in C(b)$, $h \neq 0$, we have

$$|T(h, L)| = \frac{|e_b(hL) - 1|}{|e_b(h) - 1|} \leqq \frac{2}{|e_b(h) - 1|} = \frac{1}{\sin \pi |h/b|},$$

and so

$$|R(\mathbf{h}_i, u_i)| \leqq Q(h_{i1}, \cdots, h_{im})$$

by (8). In the case $b = 2$, $\mathbf{h}_i \neq \mathbf{0}$, a bound for $|R(\mathbf{h}_i, u_i)|$ is obtained as follows. If $h \in C(2)$, $h \neq 0$, then necessarily $h = 1$, and

$$T(1, L) = \sum_{v=1}^{L} e_2(v) = \sum_{v=1}^{L} (-1)^v = \begin{cases} 0 & \text{for even } L \geqq 0, \\ -1 & \text{for odd } L \geqq 1. \end{cases}$$

Therefore with $d = d(h_{i1}, \cdots, h_{im})$,

$$R(\mathbf{h}_i, u_i) = 2^{-d} T(1, u_i^{(d)}) + \sum_{r=d+1}^{m} 2^{-r} u_i^{(r)} \leqq \sum_{r=d+1}^{m} 2^{-r} u_i^{(r)} \leqq 2^{-d}$$

and

$$R(\mathbf{h}_i, u_i) = 2^{-d} T(1, u_i^{(d)}) + \sum_{r=d+1}^{m} 2^{-r} u_i^{(r)} \geqq -2^{-d},$$

so that

$$|R(\mathbf{h}_i, u_i)| \leqq 2^{-d} = Q(h_{i1}, \cdots, h_{im})$$

by (7). Thus in all cases we have

$$|R(\mathbf{h}_i, u_i)| \leqq Q(h_{i1}, \cdots, h_{im}),$$

and so by (9)

$$\prod_{i=1}^{s} |R(\mathbf{h}_i, u_i)| \leqq P(\mathbf{h}).$$

The desired result follows now from (13) if we recall the definition (4) of $D_N^{(s)}$ and note that $E_N(J) = A(J; N)/N$. □

LEMMA 3. *For the weights $P(\mathbf{h})$ defined by* (9) *we have*

$$\sum_{\mathbf{h} \neq \mathbf{0}} P(\mathbf{h}) < \left(\frac{2}{\pi} m \log b + \frac{7}{5} m - \frac{m-1}{b} \right)^s - 1, \tag{14}$$

where the sum is extended over all nonzero lattice points $\mathbf{h} \in C_{ms}(b)$. *In the case* $b = 2$ *we have*

$$\sum_{\mathbf{h} \neq \mathbf{0}} P(\mathbf{h}) = \left(\frac{m}{2} + 1\right)^s - 1. \tag{15}$$

Proof. We have

$$\sum_{\mathbf{h} \neq \mathbf{0}} P(\mathbf{h}) = \sum_{\substack{h_{ij} \in C(b) \\ 1 \leqq i \leqq s, 1 \leqq j \leqq m}} Q(h_{11}, \cdots, h_{1m}) \cdots Q(h_{s1}, \cdots, h_{sm}) - 1$$

$$= \left(\sum_{h_1, \cdots, h_m \in C(b)} Q(h_1, \cdots, h_m) \right)^s - 1.$$

To evaluate the last sum, we split it up according to the value of $d = d(h_1, \cdots, h_m)$ for nonzero lattice points $(h_1, \cdots, h_m)$. There are b^{d-1} such lattice points with a fixed nonzero value of h_d. Writing $C^*(b)$ for the set of nonzero elements of $C(b)$, we get for $b > 2$,

$$\begin{aligned} \sum_{h_1, \cdots, h_m \in C(b)} Q(h_1, \cdots, h_m) &= 1 + \sum_{d=1}^{m} b^{d-1} b^{-d} \sum_{h \in C^*(b)} \left(\frac{1}{\sin \pi |h/b|} + \delta_d \right) \\ &= 1 + \frac{m}{b} \sum_{h \in C^*(b)} \frac{1}{\sin \pi |h/b|} + \frac{b-1}{b} \sum_{d=1}^{m} \delta_d \\ &= \frac{m}{b} \sum_{h \in C^*(b)} \frac{1}{\sin \pi |h/b|} + \frac{(m-1)(b-1)+b}{b} \\ &= \frac{m}{b} \sum_{h \in C^*(b)} \frac{1}{\sin \pi |h/b|} + m - \frac{m-1}{b}. \end{aligned}$$

By [8, p. 574] we have

$$\sum_{h \in C^*(b)} \frac{1}{\sin \pi |h/b|} < \frac{2}{\pi} b \log b + \frac{2}{5} b,$$

and combining these results we obtain (14). In the case $b = 2$ we use (7) to get

$$\sum_{h_1, \cdots, h_m \in C(2)} Q(h_1, \cdots, h_m) = 1 + \sum_{d=1}^{m} 2^{d-1} 2^{-d} = \frac{m}{2} + 1,$$

and (15) follows. □

We state some bounds for character sums over the finite field F_q with $q = p^k$ elements. We note that

$$\chi(\beta) = e_p(\mathrm{Tr}(\beta)) \quad \text{for all } \beta \in F_q \tag{16}$$

defines a character of the additive group of F_q (compare with [5, Chap. 5]). The following results were shown in [11, Lemma 4] and [11, Lemma 5], respectively.

LEMMA 4. *If* α *is an element of* F_q^* *of order* τ, *then for any* $\gamma \in F_q^*$ *we have*

$$\left| \sum_{n=0}^{\tau-1} \chi(\gamma \alpha^n) \right| \leqq B(\tau) := q^{1/2} - \frac{\tau}{1 + q^{1/2}}.$$

LEMMA 5. *If* α *is an element of* F_q^* *of order* τ, *then for any* $\gamma \in F_q^*$ *we have*

$$\left| \sum_{n=0}^{N-1} \chi(\gamma \alpha^n) \right| < B(N, \tau) := q^{1/2} \left(\frac{2}{\pi} \log \tau + \frac{2}{5} \right) + \frac{N q^{1/2}}{\tau} - \frac{N}{1 + q^{1/2}} \quad \text{for all } N \geqq 1.$$

3. Upper bounds. We establish upper bounds for the discrepancy $D_N^{(s)}$ of the points $\mathbf{x}_n$ in (3), where the sequence $x_0, x_1, \cdots$ of uniform pseudorandom numbers is gener-

ated by the GFSR algorithm. We distinguish the case $N = \tau$ of the full period and the case $N < \tau$.

The bounds involve the quantity $R^{(s)}(f, p, K)$ which depends on the dimension s, the characteristic polynomial $f = f(x)$ in (5), the prime p, and the m-tuple $K = (k_1, \cdots, k_m)$ of the distinct nonnegative integers chosen in the GFSR algorithm. Using notation introduced in § 2, the quantity $R^{(s)}(f, p, K)$ is defined as follows. To each lattice point $\mathbf{h} = (h_{ij}) \in C_{ms}(p)$ we associated the polynomial

$$g(\mathbf{h}) = \sum_{i=1}^{s} \sum_{j=1}^{m} h_{ij} x^{i-1+k_j}, \tag{17}$$

considered as a polynomial over the finite field F_p. We recall that to each $\mathbf{h}$ there is attached a weight $P(\mathbf{h})$ according to (9), with a simpler form given in (10) for the case $p = 2$. Then

$$R^{(s)}(f, p, K) = \sum_{\substack{\mathbf{h} \neq \mathbf{0} \\ f \mid g(\mathbf{h})}} P(\mathbf{h}), \tag{18}$$

where the sum is extended over all nonzero lattice points $\mathbf{h} \in C_{ms}(p)$ for which f divides $g(\mathbf{h})$, both considered as polynomials over F_p. If there is no $\mathbf{h} \neq \mathbf{0}$ in $C_{ms}(p)$ for which f divides $g(\mathbf{h})$, then in accordance with our convention for empty sums we define $R^{(s)}(f, p, K) = 0$. For the following results the standing hypotheses stated in § 1 are of course operative.

THEOREM 1. *For pseudorandom numbers generated by the* GFSR *algorithm we have for any dimension* $s \geqq 1$,

$$D_\tau^{(s)} \leqq 1 - (1 - p^{-m})^s + \frac{B(\tau)}{\tau} \left(\left(\frac{2}{\pi} m \log p + \frac{7}{5} m - \frac{m-1}{p} \right)^s - 1 \right) + R^{(s)}(f, p, K), \tag{19}$$

where $B(\tau)$ *is defined in Lemma* 4 *with* $q = p^k$. *In the case* $p = 2$ *we have*

$$D_\tau^{(s)} \leqq 1 - (1 - 2^{-m})^s + \frac{B(\tau)}{\tau} \left(\left(\frac{m}{2} + 1 \right)^s - 1 \right) + R^{(s)}(f, 2, K). \tag{20}$$

Proof. We apply Lemma 2 with $b = p$. In the notation of this lemma we have $\mathbf{w}_n = \mathbf{x}_n$, hence

$$w_{ni} = x_{n+i-1} \quad \text{for } 1 \leqq i \leqq s \text{ and } n \geqq 0.$$

From (2) we get the digits of w_{ni} to the base p, namely

$$w_{ni}^{(j)} = y_{n+i-1+k_j} \quad \text{for } 1 \leqq i \leqq s,\ 1 \leqq j \leqq m \text{ and } n \geqq 0.$$

Now take a nonzero lattice point $\mathbf{h} = (h_{ij}) \in C_{ms}(p)$. Since in the exponential sum in Lemma 2 the h_{ij} and $w_{ni}^{(j)}$ can be replaced by their residues modulo p, we may view these integers as elements of F_p. Put

$$z_n = \sum_{i=1}^{s} \sum_{j=1}^{m} h_{ij} w_{ni}^{(j)} = \sum_{i=1}^{s} \sum_{j=1}^{m} h_{ij} y_{n+i-1+k_j} \quad \text{for } n \geqq 0.$$

Then by (6) we get with a suitable $\beta \in F_q^*$,

$$\begin{aligned} z_n &= \sum_{i=1}^{s} \sum_{j=1}^{m} h_{ij} \operatorname{Tr}(\beta \alpha^{n+i-1+k_j}) \\ &= \operatorname{Tr}\left(\beta \sum_{i=1}^{s} \sum_{j=1}^{m} h_{ij} \alpha^{n+i-1+k_j} \right) = \operatorname{Tr}(\gamma \alpha^n) \quad \text{for } n \geqq 0, \end{aligned}$$

with

$$\gamma = \beta \sum_{i=1}^{s} \sum_{j=1}^{m} h_{ij} \alpha^{i-1+k_j} \in F_q.$$

If $\gamma = 0$, then $z_n = 0$ for all $n \geqq 0$, and so

$$\frac{1}{\tau} \sum_{n=0}^{\tau-1} e_p\left(\sum_{i=1}^{s} \sum_{j=1}^{m} h_{ij} w_{ni}^{(j)} \right) = \frac{1}{\tau} \sum_{n=0}^{\tau-1} e_p(z_n) = 1. \tag{21}$$

Note that $\gamma = 0$ if and only if

$$\sum_{i=1}^{s} \sum_{j=1}^{m} h_{ij} \alpha^{i-1+k_j} = 0. \tag{22}$$

Since α is a root of the irreducible polynomial f over F_p, the identity (22) holds if and only if f divides the polynomial $g(\mathbf{h})$ in (17). Therefore we have $\gamma = 0$ if and only if f divides $g(\mathbf{h})$.

If $\gamma \neq 0$, then from the identities above and (16) we get

$$e_p\left(\sum_{i=1}^{s} \sum_{j=1}^{m} h_{ij} w_{ni}^{(j)} \right) = e_p(z_n) = e_p(\mathrm{Tr}(\gamma \alpha^n)) = \chi(\gamma \alpha^n)$$

for all $n \geqq 0$, and since α has order τ in F_q^* we obtain

$$\left| \frac{1}{\tau} \sum_{n=0}^{\tau-1} e_p\left(\sum_{i=1}^{s} \sum_{j=1}^{m} h_{ij} w_{ni}^{(j)} \right) \right| = \left| \frac{1}{\tau} \sum_{n=0}^{\tau-1} \chi(\gamma \alpha^n) \right| \leqq \frac{B(\tau)}{\tau}$$

from Lemma 4. If we take (21) into account, Lemma 2 yields

$$D_\tau^{(s)} \leqq 1 - (1 - p^{-m})^s + \frac{B(\tau)}{\tau} \sum_{\substack{\mathbf{h} \neq \mathbf{0} \\ f \nmid g(\mathbf{h})}} P(\mathbf{h}) + \sum_{\substack{\mathbf{h} \neq \mathbf{0} \\ f \mid g(\mathbf{h})}} P(\mathbf{h})$$

$$\leqq 1 - (1 - p^{-m})^s + \frac{B(\tau)}{\tau} \sum_{\mathbf{h} \neq \mathbf{0}} P(\mathbf{h}) + R^{(s)}(f, p, K).$$

The bounds (19) and (20) follow now from (14) and (15), respectively. □

COROLLARY 1. *If $p = 2$ and f is a primitive polynomial over F_2, then $\tau = 2^k - 1$ and*

$$D_\tau^{(s)} \leqq 1 - (1 - 2^{-m})^s + \frac{1}{\tau} \left(\left(\frac{m}{2} + 1 \right)^s - 1 \right) + R^{(s)}(f, 2, K).$$

Proof. In this case we have $\tau = 2^k - 1 = q - 1$; hence $B(\tau) = 1$ in Lemma 4, and the result follows from (20). □

THEOREM 2. *For pseudorandom numbers generated by the* GFSR *algorithm we have for any dimension $s \geqq 1$,*

$$D_N^{(s)} \leqq 1 - (1 - p^{-m})^s + \frac{B(N, \tau)}{N} \left(\left(\frac{2}{\pi} m \log p + \tfrac{7}{5} m - \frac{m-1}{p} \right)^s - 1 \right) + R^{(s)}(f, p, K)$$

for $1 \leqq N < \tau$, where $B(N, \tau)$ is defined in Lemma 5 with $q = p^k$. In the case $p = 2$ we have

$$D_N^{(s)} \leqq 1 - (1 - 2^{-m})^s + \frac{B(N, \tau)}{N} \left(\left(\frac{m}{2} + 1 \right)^s - 1 \right) + R^{(s)}(f, 2, K) \quad \textit{for } 1 \leqq N < \tau.$$

Proof. We proceed as in the proof of Theorem 1, but we use Lemma 5 instead of Lemma 4. □

COROLLARY 2. *If $p = 2$ and f is a primitive polynomial over F_2, then*

$$D_N^{(s)} \leqq 1 - (1 - 2^{-m})^s + \left(\frac{2^{k/2}}{N} \left(\frac{2}{\pi} \log \tau + \frac{2}{5} \right) + \frac{1}{\tau} \right) \left(\left(\frac{m}{2} + 1 \right)^s - 1 \right) + R^{(s)}(f, 2, K)$$

for $1 \leqq N < \tau = 2^k - 1$.

Proof. In this case we have

$$B(N, \tau) = 2^{k/2}\left(\frac{2}{\pi}\log \tau + \frac{2}{5}\right) + \frac{N}{\tau}$$

in Lemma 5. □

4. Figures of merit. The quantity $R^{(s)}(f, p, K)$ defined in (18) is important since it measures the dependence of $D_N^{(s)}$ on the characteristic polynomial f. According to the results in § 3, a small value of $R^{(s)}(f, p, K)$ guarantees that the pseudorandom numbers pass the s-dimensional serial test. The quantity $R^{(s)}(f, p, K)$ is, however, rather inconvenient numerically. Therefore we introduce a related positive integer, the figure of merit $\rho^{(s)}(f, p, K)$, which is easier to handle. For $\mathbf{h} = (h_{ij}) \in C_{ms}(p)$ we define $D(\mathbf{h})$ as in (10), i.e., $D(\mathbf{h}) = \sum_{i=1}^{s} d(h_{i1}, \cdots, h_{im})$. Then the *figure of merit* is given by

$$\rho^{(s)}(f, p, K) = \min_{\substack{\mathbf{h} \neq \mathbf{0} \\ f \mid g(\mathbf{h})}} D(\mathbf{h}), \tag{23}$$

where the minimum is extended over all nonzero lattice points $\mathbf{h} \in C_{ms}(p)$ for which f divides the polynomial $g(\mathbf{h})$ in (17), both considered as polynomials over F_p. If there is no $\mathbf{h} \neq \mathbf{0}$ in $C_{ms}(p)$ for which f divides $g(\mathbf{h})$, then we define $\rho^{(s)}(f, p, K) = \infty$.

We show now that $R^{(s)}(f, p, K)$ can be bounded in terms of $\rho^{(s)}(f, p, K)$. For an integer $r \geqq 0$ let $E(r)$ be the number of $(d_1, \cdots, d_s) \in \{0, 1, \cdots, m\}^s$ with $\sum_{i=1}^{s} d_i \leqq r$.

THEOREM 3. *We have*

$$R^{(s)}(f, p, K) \leqq (p-1)\left(\frac{1}{\sin(\pi/p)} + 1\right)^s ((m+1)^s - E(\rho - 1))p^{-\rho}, \tag{24}$$

where $\rho = \rho^{(s)}(f, p, K)$. *For* $p = 2$ *we have*

$$R^{(s)}(f, 2, K) \leqq ((m+1)^s - E(\rho-1))2^{-\rho}. \tag{25}$$

Proof. If $\rho = \infty$, then $R^{(s)}(f, p, K) = 0$ by definition, and so (24) and (25) hold trivially. Thus we can assume $\rho < \infty$. Consider a lattice point $\mathbf{h} = (h_{ij}) \in C_{ms}(p)$. From (7) and (8) we get

$$Q(h_{i1}, \cdots, h_{im}) \leqq \left(\frac{1}{\sin(\pi/p)} + 1\right) p^{-d(h_{i1}, \cdots, h_{im})} \quad \text{for } 1 \leqq i \leqq s;$$

hence

$$P(\mathbf{h}) \leqq \left(\frac{1}{\sin(\pi/p)} + 1\right)^s \prod_{i=1}^{s} p^{-d(h_{i1}, \cdots, h_{im})} = \left(\frac{1}{\sin(\pi/p)} + 1\right)^s p^{-D(\mathbf{h})}.$$

It follows from (18) that

$$R^{(s)}(f, p, K) \leqq \left(\frac{1}{\sin(\pi/p)} + 1\right)^s R_1^{(s)}(f, p, K) \tag{26}$$

with

$$R_1^{(s)}(f, p, K) = \sum_{\substack{\mathbf{h} \neq \mathbf{0} \\ f \mid g(\mathbf{h})}} p^{-D(\mathbf{h})}.$$

For $p = 2$ we have

$$R^{(s)}(f, 2, K) = R_1^{(s)}(f, 2, K) \tag{27}$$

in view of (10). We can write

$$R_1^{(s)}(f, p, K) = \sum_{r=\rho}^{ms} M(r)p^{-r}, \tag{28}$$

where $M(r)$ is the number of $\mathbf{h} \in C_{ms}(p)$ (or equivalently $\mathbf{h} \in F_p^{ms}$) with f dividing $g(\mathbf{h})$ and $D(\mathbf{h}) = r$.

We determine now an upper bound for $M(r)$, where $\rho \leqq r \leqq ms$. Let $L(r)$ be the number of $\mathbf{d} = (d_1, \cdots, d_s) \in \{0, 1, \cdots, m\}^s$ with $\sum_{i=1}^s d_i = r$. Fix such a $\mathbf{d}$, and let e_i, $0 \leqq e_i \leqq d_i$, be fixed integers with $\sum_{i=1}^s e_i = \rho - 1$. For those i with $e_i < d_i$, choose elements $g_{ij} \in F_p$, $e_i < j \leqq d_i$, with $g_{id_i} \neq 0$, thus obtaining an $(r-\rho+1)$-tuple $G = (g_{ij})$. For each choice of G consider the set

$$U(\mathbf{d}, G) = \{\mathbf{h} = (h_{ij}) \in F_p^{ms} : h_{ij} = g_{ij} \text{ if } e_i < j \leqq d_i, h_{ij} = 0 \text{ if}$$
$$e_i < d_i \text{ and } d_i < j \leqq m, d(h_{i1}, \cdots, h_{im}) = d_i \text{ if } e_i = d_i\}.$$

For every $\mathbf{h} \in U(\mathbf{d}, G)$ we have $d(h_{i1}, \cdots, h_{im}) = d_i$ for $1 \leqq i \leqq s$; hence $D(\mathbf{h}) = r$. We claim that there is at most one $\mathbf{h} \in U(\mathbf{d}, G)$ with $f \mid g(\mathbf{h})$. Suppose we had $\mathbf{h}^{(1)}, \mathbf{h}^{(2)} \in U(\mathbf{d}, G)$, $\mathbf{h}^{(1)} \neq \mathbf{h}^{(2)}$, and $f \mid g(\mathbf{h}^{(1)})$, $f \mid g(\mathbf{h}^{(2)})$. Consider $\mathbf{h} = \mathbf{h}^{(1)} - \mathbf{h}^{(2)}$; then $\mathbf{h} \neq \mathbf{0}$ and $f \mid g(\mathbf{h})$. In an obvious notation we have for i with $e_i = d_i$,

$$d(h_{i1}, \cdots, h_{im}) = d(h_{i1}^{(1)} - h_{i1}^{(2)}, \cdots, h_{im}^{(1)} - h_{im}^{(2)}) \leqq d_i = e_i.$$

For i with $e_i < d_i$ we have

$$d(h_{i1}, \cdots, h_{im}) = d(h_{i1}^{(1)} - h_{i1}^{(2)}, \cdots, h_{im}^{(1)} - h_{im}^{(2)}) \leqq e_i.$$

Altogether

$$D(\mathbf{h}) = \sum_{i=1}^{s} d(h_{i1}, \cdots, h_{im}) \leqq \sum_{i=1}^{s} e_i = \rho - 1,$$

a contradiction to the definition of ρ in (23). Thus each set $U(\mathbf{d}, G)$ contains at most one $\mathbf{h}$ with $f \mid g(\mathbf{h})$. On the other hand, every $\mathbf{h} \in F_p^{ms}$ with $D(\mathbf{h}) = r$ belongs to a set $U(\mathbf{d}, G)$ for suitable $\mathbf{d}$ and G. Therefore, an upper bound for $M(r)$ is given by the number of sets $U(\mathbf{d}, G)$. For fixed $\mathbf{d}$, the number of sets $U(\mathbf{d}, G)$ is equal to the number of choices for G, which is

$$\prod_{\substack{i=1 \\ e_i < d_i}}^{s} (p-1)p^{d_i - e_i - 1} = p^{r-\rho+1} \prod_{\substack{i=1 \\ e_i < d_i}}^{s} \frac{p-1}{p} \leqq (p-1)p^{r-\rho}.$$

Since there are $L(r)$ possibilities for $\mathbf{d}$, we get

$$M(r) \leqq L(r)(p-1)p^{r-\rho}.$$

We go back to (28), use the fact that $\{0, 1, \cdots, m\}^s$ has $(m+1)^s$ elements and get

$$R_1^{(s)}(f, p, K) \leqq (p-1)p^{-\rho} \sum_{r=\rho}^{ms} L(r) = (p-1)p^{-\rho}\left(\sum_{r=0}^{ms} L(r) - \sum_{r=0}^{\rho-1} L(r)\right)$$
$$= (p-1)p^{-\rho}((m+1)^s - E(\rho-1)).$$

We combine this with (26) to get (24) and with (27) to get (25). □

For most practical purposes it will suffice to work with an obvious consequence of Theorem 3, namely

$$R^{(s)}(f, p, K) \leqq (p-1)\left(\frac{1}{\sin(\pi/p)} + 1\right)^s (m+1)^s p^{-\rho}, \tag{29}$$

and for $p = 2$

$$R^{(s)}(f, 2, K) \leqq (m+1)^s 2^{-\rho}. \tag{30}$$

We note the following elementary bound on the figure of merit.

THEOREM 4. *If $s > k/m$, then $\rho = \rho^{(s)}(f, p, K) \leqq k+1$. If $s \leqq k/m$ and $\rho < \infty$, then $\rho \leqq ms \leqq k$.*

Proof. If $ms > k$, then there exist integers d_i, $0 \leqq d_i \leqq m$, with $\sum_{i=1}^{s} d_i = k+1$. The finite field F_q with $q = p^k$ elements can be viewed as a vector space of dimension k over F_p. If α denotes a fixed root of f in F_q, then the $k+1$ elements α^{i-1+k_j}, $1 \leqq j \leqq d_i$, $1 \leqq i \leqq s$, of F_q are linearly dependent over F_p. Thus we have a nontrivial dependence relation

$$\sum_{i=1}^{s} \sum_{j=1}^{d_i} h_{ij} \alpha^{i-1+k_j} = 0 \quad \text{with } h_{ij} \in F_p.$$

Let $\mathbf{h} = (h_{ij}) \in F_p^{ms}$ with $h_{ij} = 0$ for $d_i < j \leqq m$, $1 \leqq i \leqq s$. Then $\mathbf{h} \neq \mathbf{0}$ and

$$\sum_{i=1}^{s} \sum_{j=1}^{m} h_{ij} \alpha^{i-1+k_j} = 0.$$

Consequently, f divides the polynomial $g(\mathbf{h})$ in (17). Identifying $\mathbf{h}$ with an element of $C_{ms}(p)$, we obtain from the definition of ρ in (23),

$$\rho \leqq D(\mathbf{h}) \leqq \sum_{i=1}^{s} d_i = k+1.$$

The second part follows from the fact that for any $\mathbf{h} \in C_{ms}(p)$ we have $D(\mathbf{h}) \leqq ms \leqq k$. □

5. Lower bounds. We show first that the term $1-(1-p^{-m})^s$ appearing in the upper bounds for $D_N^{(s)}$ in § 3 serves also as a universal lower bound.

THEOREM 5. *For pseudorandom numbers generated by the* GFSR *algorithm we have*

$$D_N^{(s)} \geqq 1-(1-p^{-m})^s \quad \textit{for all } s \geqq 1 \textit{ and } N \geqq 1.$$

Proof. We note that all the pseudorandom numbers x_n given by (2) are rational numbers in $[0, 1)$ with denominator p^m. It follows that all the points $\mathbf{x}_n$ in (3) belong to the interval

$$J = [0, 1-p^{-m}]^s.$$

In the notation of (4) we then have $E_N(J) = 1$ for $N \geqq 1$ and $V(J) = (1-p^{-m})^s$, and so the desired lower bound follows immediately from (4). □

Since the lower bound in Theorem 5 results from the fact that all x_n are rationals with a fixed denominator, we may interpret it as the discretization error.

The results in §§ 3 and 4 show that $D_N^{(s)}$ can be bounded from above in terms of the figure of merit. We prove now that $D_N^{(s)}$ can also be bounded from below in a similar manner.

THEOREM 6. *For pseudorandom numbers generated by the* GFSR *algorithm we have*

$$D_N^{(s)} \geqq \frac{p-1}{2} p^{-\rho} \quad \textit{for all } s \geqq 1 \textit{ and } N \geqq 1,$$

where $\rho = \rho^{(s)}(f, p, K)$.

Proof. The result is trivial for $\rho = \infty$, so that we can assume $\rho < \infty$. Then by (23) there exists $\mathbf{h} = (h_{ij}) \in C_{ms}(p)$ with $\mathbf{h} \neq \mathbf{0}$, $f \mid g(\mathbf{h})$, and $D(\mathbf{h}) = \rho$. Let r with $1 \leqq r \leqq s$ be the largest index for which $(h_{r1}, \cdots, h_{rm}) \neq (0, \cdots, 0)$. Put $d_i = d(h_{i1}, \cdots, h_{im})$ for $1 \leqq i \leqq r$; then

$$D(\mathbf{h}) = \sum_{i=1}^{r} d_i.$$

Furthermore

$$g(\mathbf{h}) = \sum_{i=1}^{r} \sum_{j=1}^{d_i} h_{ij} x^{i-1+k_j},$$

which we view as a polynomial over F_p. Now $y_0, y_1, \cdots$, considered as a sequence of elements of F_p, has the irreducible polynomial f over F_p as a characteristic polynomial, and not all initial values of the sequence are 0. Hence [5, Thm. 8.50] implies that f is the minimal polynomial of the sequence. Since f divides $g(\mathbf{h})$, it follows from [5, Thm. 8.42] that $g(\mathbf{h})$ is also a characteristic polynomial of the sequence. Therefore

$$\sum_{i=1}^{r} \sum_{j=1}^{d_i} h_{ij} y_{n+i-1+k_j} = 0 \quad \text{for all } n \geqq 0. \tag{31}$$

Choose a small $\varepsilon > 0$ and define the intervals

$$I_i = [0, p^{-d_i} - \varepsilon] \quad \text{for } 1 \leqq i < r,$$
$$I_r = (p^{-d_r}, p^{-d_r+1} - \varepsilon].$$

The subinterval I of $[0,1]^s$ is then defined by

$$I = I_1 \times \cdots \times I_r \times [0,1]^{s-r}.$$

We claim that no point $\mathbf{x}_n$ in (3) belongs to I. Suppose that for some $n \geqq 0$ we had

$$\mathbf{x}_n = (x_n, x_{n+1}, \cdots, x_{n+s-1}) \in I.$$

Then from (2),

$$x_{n+i-1} = \sum_{j=1}^{m} y_{n+i-1+k_j} p^{-j} \in I_i \quad \text{for } 1 \leqq i \leqq r.$$

For $1 \leqq i < r$ it follows from the definition of I_i that

$$y_{n+i-1+k_j} = 0 \quad \text{for } 1 \leqq j \leqq d_i.$$

For $i = r$ it follows from the definition of I_r that

$$y_{n+r-1+k_j} = 0 \quad \text{for } 1 \leqq j < d_r, \qquad y_{n+r-1+k_j} \neq 0 \quad \text{for } j = d_r.$$

These facts yield a contradiction to (31) since $h_{rd_r} \neq 0$. Thus, indeed, no point $\mathbf{x}_n$ belongs to I. Therefore we have $E_N(I) = 0$ for all $N \geqq 1$ in the notation of (4). Define subintervals J_1 and J_2 of $[0,1]^s$ by

$$J_1 = I_1 \times \cdots \times I_{r-1} \times [0, p^{-d_r+1} - \varepsilon] \times [0,1]^{s-r},$$
$$J_2 = I_1 \times \cdots \times I_{r-1} \times [0, p^{-d_r}] \times [0,1]^{s-r}.$$

Then J_1 is the disjoint union of I and J_2, thus

$$V(J_1) = V(I) + V(J_2), \qquad E_N(J_1) = E_N(I) + E_N(J_2) \quad \text{for all } N \geqq 1.$$

Hence

$$V(I) = |E_N(I) - V(I)| = |(E_N(J_1) - V(J_1)) - (E_N(J_2) - V(J_2))|$$
$$\leqq |E_N(J_1) - V(J_1)| + |E_N(J_2) - V(J_2)| \leqq 2D_N^{(s)}$$

by (4), and so

$$D_N^{(s)} \geqq \frac{1}{2} V(I) = \frac{1}{2} (p^{-d_r+1} - p^{-d_r} - \varepsilon) \prod_{i=1}^{r-1} (p^{-d_i} - \varepsilon).$$

Letting $\varepsilon \to 0+$, we get

$$D_N^{(s)} \geqq \frac{p-1}{2} \prod_{i=1}^{r} p^{-d_i} = \frac{p-1}{2} p^{-D(\mathbf{h})},$$

and since $D(\mathbf{h}) = \rho$, the proof is complete. □

6. Discussion. The discussion of the order of magnitude of our bounds for the discrepancy will be facilitated by the observation that the expression $1-(1-p^{-m})^s$ occurring in various bounds satisfies

$$p^{-m} \leqq 1-(1-p^{-m})^s \leqq sp^{-m}. \tag{32}$$

We consider now the upper bounds for $D_\tau^{(s)}$ in Theorem 1 as to their dependence on the period τ. From the formula for $B(\tau)$ in Lemma 4 it is seen that $B(\tau)/\tau$ is a decreasing function of τ. Therefore the minimal value of $B(\tau)/\tau$ is attained for the maximal value of τ, which for given p and k is $\tau = p^k - 1$ as we have mentioned in § 1. This corresponds to choosing for the characteristic polynomial f a primitive polynomial over F_p. In the following discussion we will concentrate on this case. We note that in this case we have $B(\tau) = 1$.

The most favorable situation arises when $R^{(s)}(f, p, K) = 0$, or equivalently $\rho^{(s)}(f, p, K) = \infty$. In view of Theorem 4 this can only occur if $s \leqq k/m$. The case $R^{(s)}(f, p, K) = 0$ arises if and only if there is no $\mathbf{h} \neq \mathbf{0}$ in $C_{ms}(p)$ for which f divides $g(\mathbf{h})$. By an argument in the proof of Theorem 4, this is equivalent to the property that the ms elements α^{i-1+k_j}, $1 \leqq i \leqq s$, $1 \leqq j \leqq m$, of the finite field F_q, $q = p^k$, are linearly independent over F_p. The exponent $i-1+k_j$ is only relevant modulo τ, and so the element α^{i-1+k_j} can be calculated by the standard "square and multiply" technique using $O(\log \tau)$ operations in F_q (compare with [6, p. 347]).

In the case where $\tau = p^k - 1$ and $R^{(s)}(f, p, K) = 0$, the points $\mathbf{x}_n$, $0 \leqq n < \tau$, in (3) can in fact be described explicitly. As noted above, we have in this case $ms \leqq k$ and the system A of elements α^{i-1+k_j}, $1 \leqq i \leqq s$, $1 \leqq j \leqq m$, is linearly independent over F_p. Let (c_{ij}), $1 \leqq i \leqq s$, $1 \leqq j \leqq m$, be an arbitrary ms-tuple of elements of $\{0, 1, \cdots, p-1\}$. Then

$$L(\alpha^{i-1+k_j}) = c_{ij} \quad \text{for } 1 \leqq i \leqq s, \quad 1 \leqq j \leqq m$$

uniquely determines an F_p-linear transformation L from the subspace of F_q spanned by A into F_p. Extend L in an arbitrary manner to an F_p-linear transformation L' from F_q into F_p. By [5, Thm. 2.24], L' uniquely determines an element $\gamma \in F_q$ such that

$$L'(\eta) = \mathrm{Tr}(\gamma\eta) \quad \text{for all } \eta \in F_q.$$

If L' is not the zero transformation, then $\gamma \neq 0$. Now $\tau = p^k - 1 = q - 1$ implies that α is a generator of the group F_q^*, and so each $\gamma \neq 0$ uniquely determines an n, $0 \leqq n < \tau$, with $\gamma = \beta\alpha^n$, where β is the element in (6). Therefore, each nonzero extension L' of L determines a unique n, $0 \leqq n < \tau$, satisfying

$$y_{n+i-1+k_j} = \mathrm{Tr}(\beta\alpha^{n+i-1+k_j}) = L'(\alpha^{i-1+k_j}) = c_{ij} \quad \text{for } 1 \leqq i \leqq s, \quad 1 \leqq j \leqq m. \tag{33}$$

Hence the number of n, $0 \leqq n < \tau$, satisfying (33) is equal to the number of nonzero extensions L' of L. By vector space theory, this number is equal to p^{k-ms} for $(c_{ij}) \neq \mathbf{0}$ and equal to $p^{k-ms} - 1$ for $(c_{ij}) = \mathbf{0}$. In view of (2) and (3), this can be translated into the following information about the points $\mathbf{x}_n$. Let $\mathbf{t} \in [0, 1)^s$ be a given point all of whose coordinates are rationals with denominator p^m. If $\mathbf{t} \neq \mathbf{0}$, then the number of n,

$0 \leqq n < \tau$, with $\mathbf{x}_n = \mathbf{t}$ is equal to p^{k-ms}, and if $\mathbf{t} = \mathbf{0}$, then the number of n, $0 \leqq n < \tau$, with $\mathbf{x}_n = \mathbf{t}$ is equal to $p^{k-ms} - 1$. An elementary argument similar to that in [14, Thm. 3] shows then that

$$D_\tau^{(s)} = 1 - (1 - p^{-m})^s$$

whenever $\tau = p^k - 1$ and $R^{(s)}(f, p, K) = 0$. Therefore, in this case $D_\tau^{(s)}$ is equal to the discretization error. This shows, in particular, that the lower bound in Theorem 5 is the best possible.

We turn now to the case where $R^{(s)}(f, p, K) > 0$, or equivalently $\rho = \rho^{(s)}(f, p, K) < \infty$. By Theorem 4 this case will always occur if $s > k/m$. We use the symbol $\ll$ to denote an inequality in which constant factors depending only on s and p are suppressed. If $\tau = p^k - 1$, then we have

$$p^{-m} + p^{-\rho} \ll D_\tau^{(s)} \ll p^{-m} + \tau^{-1} m^s + m^s p^{-\rho},$$

where the lower bound stems from Theorems 5 and 6 and the inequality (32), whereas the upper bound is obtained by combining Theorems 1 and 3 with (32). Since $\rho \leqq k+1$ by Theorem 4, we can further simplify to get

$$p^{-m} + p^{-\rho} \ll D_\tau^{(s)} \ll p^{-m} + m^s p^{-\rho}. \tag{34}$$

This demonstrates that if the characteristic polynomial f is restricted to be a primitive polynomial over F_p of degree k, then the dependence of the order of magnitude of $D_\tau^{(s)}$ on f is controlled by the figure of merit ρ. If the parameters s, p, k, m and the m-tuple $K = (k_1, \cdots, k_m)$ are given, then we should choose a primitive polynomial f with $\rho^{(s)}(f, p, K)$ being as large as possible in order to guarantee that the GFSR pseudorandom numbers pass the s-dimensional serial test. An interesting special case occurs if we choose $m = k$, for then (34) reduces to

$$p^{-\rho} \ll D_\tau^{(s)} \ll p^{-\rho} (\log \tau)^s,$$

so that the order of magnitude of $D_\tau^{(s)}$ is, up to a logarithmic factor, given by $p^{-\rho}$. This shows also that the bounds in Theorems 1 and 6 are in general best possible (up to logarithmic factors).

REFERENCES

[1] A. C. ARVILLIAS AND D. G. MARITSAS, *Partitioning the period of a class of m-sequences and application to pseudorandom number generation*, J. Assoc. Comput. Mach., 25 (1978), pp. 675–686.

[2] M. FUSHIMI AND S. TEZUKA, *The k-distribution of the generalized feedback shift register pseudorandom numbers*, Comm. ACM, 26 (1983), pp. 516–523.

[3] S. KIRKPATRICK AND E. P. STOLL, *A very fast shift-register sequence random number generator*, J. Comput. Physics, 40 (1981), pp. 517–526.

[4] T. G. LEWIS AND W. H. PAYNE, *Generalized feedback shift register pseudorandom number algorithm*, J. Assoc. Comput. Mach., 20 (1973), pp. 456–468.

[5] R. LIDL AND H. NIEDERREITER, *Finite Fields*, Encyclopedia of Mathematics and Its Applications, Vol. 20, Addison-Wesley, Reading, MA, 1983.

[6] ———, *Introduction to Finite Fields and Their Applications*, Cambridge Univ. Press, Cambridge, 1986.

[7] G. MARSAGLIA AND L.-H. TSAY, *Matrices and the structure of random number sequences*, Linear Algebra Appl., 67 (1985), pp. 147–156.

[8] H. NIEDERREITER, *On the distribution of pseudorandom numbers generated by the linear congruential method*, III, Math. Comp., 30 (1976), pp. 571–597.

[9] ———, *Quasi-Monte Carlo methods and pseudo-random numbers*, Bull. Amer. Math. Soc., 84 (1978), pp. 957–1041.

[10] ———, *Applications des corps finis aux nombres pseudo-aléatoires*, Sém. Théorie des Nombres 1982–1983, Exp. 38, Univ. de Bordeaux I, Talence, 1983.

[11] ———, *The performance of k-step pseudorandom number generators under the uniformity test*, this Journal, 5 (1984), pp. 798–810.

[12] ———, *Number-theoretic problems in pseudorandom number generation*, Proc. Symp. on Applications of Number Theory to Numerical Analysis, Kyoto, 1984, Lecture Notes No. 537, Research Inst. of Math. Sciences, Kyoto Univ., 1984, pp. 18–28.

[13] ———, *Multidimensional numerical integration using pseudorandom numbers*, Proc. Numerical Methods for Stochastic Optimization, Laxenburg, 1983, Math. Programming Study, Vol. 27, North-Holland, Amsterdam, 1986, pp. 17–38.

[14] ———, *A pseudorandom vector generator based on finite field arithmetic*, Math. Japonica, 31 (1986), pp. 759–774.

[15] ———, *The serial test for digital k-step pseudorandom numbers*, preprint.

[16] A. I. PAVLOV AND B. B. POHODZEĬ, *Pseudorandom numbers generated by linear recurrence relations over a finite field*, Ž. Vyčisl. Mat. i Mat. Fiz., 19 (1979), pp. 836–842. (In Russian.)

[17] R. C. TAUSWORTHE, *Random numbers generated by linear recurrence modulo two*, Math. Comp., 19 (1965), pp. 201–209.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 6, November 1987
© 1987 Society for Industrial and Applied Mathematics
014

A STABLE AND EFFICIENT ALGORITHM FOR NONLINEAR ORTHOGONAL DISTANCE REGRESSION*

PAUL T. BOGGS†, RICHARD H. BYRD‡ AND ROBERT B. SCHNABEL§

Abstract. One of the most widely used methodologies in scientific and engineering research is the fitting of equations to data by least squares. In cases where significant observation errors exist in the independent variables as well as the dependent variables, however, the ordinary least squares (OLS) approach, where all errors are attributed to the dependent variable, is often inappropriate. An alternate approach, suggested by several researchers, involves minimizing the sum of squared orthogonal distances between each data point and the curve described by the model equation. We refer to this as orthogonal distance regression (ODR). This paper describes a method for solving the orthogonal distance regression problem that is a direct analog of the trust region Levenberg-Marquardt algorithm. The number of unknowns involved is the number of model parameters plus the number of data points, often a very large number. By exploiting sparsity, however, our algorithm has a computational effort per step which is of the same order as required for the Levenberg-Marquardt method for ordinary least squares. We prove our algorithm to be globally and locally convergent, and perform computational tests that illustrate some differences between ODR and OLS.

Key words. least squares, trust-region methods, Levenberg–Marquardt methods, errors in variables, total least squares

AMS(MOS) subject classifications. Primary 65K99; secondary 62J99, 65D10

1. Introduction.

The problem of fitting a model to data with errors in the observations has a rich history and a considerable literature. The problem where there are also errors in the independent variables at which these observations are made, however, has only relatively recently been given attention. In this paper, we consider a general form of this extended problem and provide an efficient and stable algorithm for its solution. Several names for this extended problem have been suggested; we prefer orthogonal distance regression (ODR).

Errors in independent variables virtually always occur, but are often ignored in order that classical or ordinary (linear or nonlinear) least squares (OLS) techniques can be applied (see, e.g., [LawH74], [Ste73], [Mor77], [DenGW81]). Also, if these errors are small with respect to those in the observed variables, then ignoring them does not usually seriously degrade the accuracy of the estimates. In some fields, however, measurement techniques are sufficiently accurate that errors in the independent variables are not insignificant compared to those in the observations. Examples at the National Bureau of Standards (NBS) include the calibration of electronic devices, flow-meters and calorimeters. Another class of examples comes from curve and surface fitting problems.

We first develop a formal statement of the ODR estimation problem and briefly discuss its application to statistical estimation and to curve fitting. The main contributions of the paper are the derivation and convergence analysis of a highly efficient algorithm for solving ODR problems (§§2 and 3). In §4, the results of some compu-

* Received by the editors December 24, 1985; accepted for publication (in revised form) March 20, 1987.

† Scientific Computing Division, National Bureau of Standards, Gaithersburg, Maryland 20899.

‡ Computer Science Department, University of Colorado, Boulder, Colorado 80309. The research of this author was supported in part by National Science Foundation grant DCR-8403483.

§ Computer Science Department, University of Colorado, Boulder, Colorado 80309 and Scientific Computing Division, National Bureau of Standards, Boulder, Colorado 80303. The research of this author was supported in part by Army Research Office Contract DAAG 29-84-K-0140.

tations are shown which illustrate the performance of the algorithm and allow some comparisons with ordinary least squares.

Observations in applied science are often thought of as satisfying a mathematical model of the form

$$y^a = f(x, \beta^a) \tag{1.1}$$

where y^a, (a for "actual" or "true") is taken to be the dependent variable that is to be measured as a function of the independent variable x; and $\beta^a \in R^p$ is the true parameter vector to be estimated based on the measured values, or data pairs, $(x_i, y_i), i = 1, \cdots, n$. The function f is not assumed to be linear, but is assumed to be smooth. Typically the number of data points, n, is far greater than the number of parameters, p.

In the classical case, only the observations y_i are assumed to contain errors. If these errors, designated for convenience by $-\epsilon_i^a$, are additive and the mathematical model is exact then

$$y_i = f(x_i, \beta^a) - \epsilon_i^a, \qquad i = 1, \cdots, n. \tag{1.2}$$

If in addition the errors are normally distributed with mean 0 and variance $\sigma^2 I$, then the maximum likelihood estimate, $\hat{\beta}$, is the solution to the least squares problem

$$\min_{\beta} \sum_{i=1}^{n} [f(x_i, \beta) - y_i]^2 . \tag{1.3}$$

If f is a linear function of β then this is a classical linear least squares problem; otherwise it is a classical nonlinear least squares problem. Even when the above assumptions on the model or the errors are not satisfied, problem (1.3) is the most frequently used method for parameter estimation.

In the more general situation, the measurements of the independent variables x_i are also assumed to contain errors. If we assume that y_i has unknown additive error ϵ_i^a and that x_i has unknown additive error δ_i^a, then (1.2) becomes

$$y_i = f(x_i + \delta_i^a; \beta^a) - \epsilon_i^a. \tag{1.4}$$

A reasonable way to select the parameters in this case is to choose the β that causes the sum of the squares of the orthogonal distances from the data points (x_i, y_i) to the curve $f(x, \beta)$ to be minimized. (See Fig. 1.) If r_i is the orthogonal distance from (x_i, y_i) to the curve, then

$$r_i^2 = \epsilon_i^2 + \delta_i^2, \qquad i = 1, \cdots, n$$

where ϵ_i and δ_i solve

$$\begin{aligned} &\min_{\epsilon_i, \delta_i} (\epsilon_i^2 + \delta_i^2) \\ &\text{subject to } f(x_i + \delta_i; \beta) - \epsilon_i = y_i. \end{aligned} \tag{1.5}$$

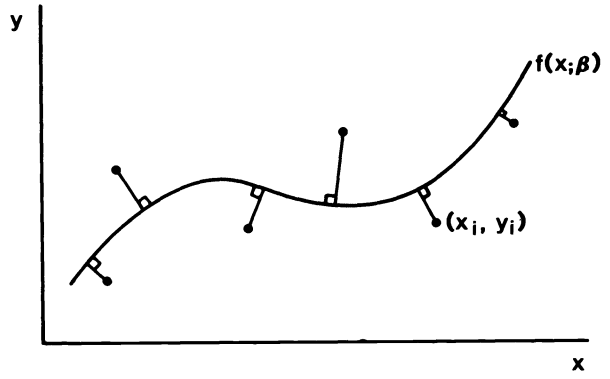


Figure 1

The constraint in (1.5) ensures that the distance r_i connects the point (x_i, y_i) to the curve. The minimization ensures that r_i is the radius of the smallest circle centered at (x_i, y_i) which is tangent to the curve $f(x_i; \beta)$. (See Fig. 2.) Therefore, the parameter estimate $\hat{\beta}$ and the error estimates $\hat{\epsilon}$ and $\hat{\delta}$ that cause the sum of the

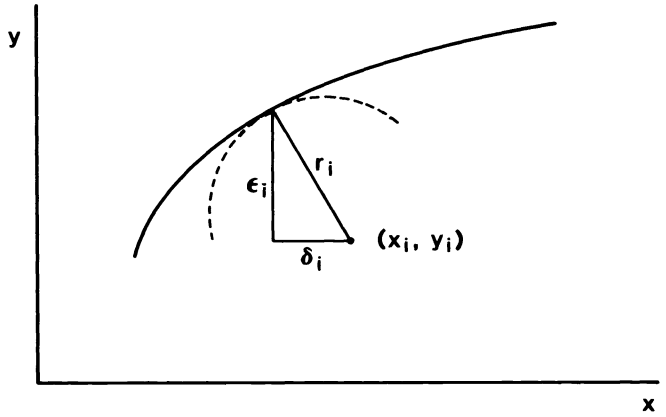


Figure 2

squares of the orthogonal distances from the data points to the curve to be minimized are found by solving

$$\min_{\beta,\epsilon,\delta} \sum_{i=1}^{n} r_i^2 = \min_{\beta,\epsilon,\delta} \sum_{i=1}^{n} (\epsilon_i^2 + \delta_i^2) \tag{1.6}$$

$$\text{subject to } y_i = f(x_i + \delta_i; \beta) - \epsilon_i, \qquad i = 1, \cdots, n.$$

Since the constraints in (1.6) are simple linear constraints in ϵ_i, we solve for ϵ_i and eliminate both the ϵ_i and all of the constraints thereby obtaining

$$\min_{\beta,\delta} \sum_{i=1}^{n} \left[(f(x_i + \delta_i; \beta) - y_i)^2 + \delta_i^2 \right] \tag{1.7}$$

which is now an unconstrained minimization problem.

Two slight extensions to this form constitute the ultimate problem to be considered. The first allows the possibility that $x_i \in R^m$ rather than R^1. Therefore, $\delta_i \in R^m$ and instead of δ_i^2 in (1.7) we have $\delta_i^T \delta_i = \sum_{j=1}^m \delta_{ij}^2$. (The superscript T denotes transpose.) The second extension merely admits a general weighting scheme on the problem. The form we have chosen results in the general nonlinear ODR problem

$$\text{(ODR)} \qquad \min_{\beta,\delta} \sum_{i=1}^{n} w_i^2 \left[(f(x_i + \delta_i; \beta) - y_i)^2 + \delta_i^T D_i^2 \delta_i \right]$$

where $w_i > 0, i = 1, \cdots, n$ and $D_i = \text{diag}\{d_{ij} > 0,\ j = 1, \cdots, m\}$, $i = 1, \cdots, n$, i.e., D_i is a diagonal matrix of order m. It follows that the vectors $y, w \in R^n$ and $x, \delta \in R^{nm}$ and that $\delta_i^T D_i^2 \delta = \sum_{j=1}^m \delta_{ij}^2 d_{ij}^2$.

While we have not assumed that f is linear, it is important to note that (ODR) is a nonlinear optimization problem even if f is the simple linear function

$$y = \beta_1 x + \beta_2$$

since we then have that

$$y_i = \beta_1 (x_i + \delta_i) + \beta_2 - \epsilon_i.$$

Clearly the product of β_1 and δ_i is an unavoidable nonlinearity.

ODR problems have been considered by statisticians, usually under the rubric errors in variables. Most of this effort, however, has been devoted to linear models, i.e., when f is linear in β. (See e.g.,[Mor71], [KenSO83], [Bar74,p.67] and [Ful86].) As in the classical nonlinear least squares case, little theory on the statistical properties of the solution appears to exist. It is known that if both ϵ and δ are normally distributed with mean zero and variances $\sigma_\epsilon^2 I$ and $\sigma_\delta^2 I$, respectively, then the solution of (ODR) with $w_i = 1$ and $D_i = (\sigma_\epsilon / \sigma_\delta) I$, $i = 1, \cdots, n$ is a maximum likelihood estimate of the parameters. Unfortunately, as in the nonlinear classical case, no generally valid, computationally efficient, inferential statistical tests are known.

Independent of statistical considerations, ODR has potentially significant applications in curve and surface fitting. Consider, for example, the problem of finding the parabola which best fits the given set of points (see Fig. 3). (We have seen this problem arise from a dental application.) Here it is clear that ordinary least squares will unduly weight the top data points, while fitting in the horizontal direction would unduly weight the bottom data points. An orthogonal measure of distance alleviates

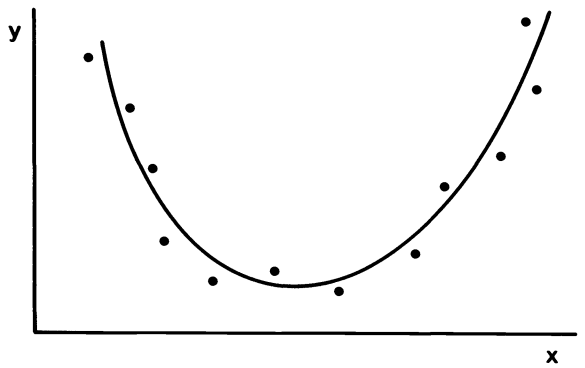


Figure 3

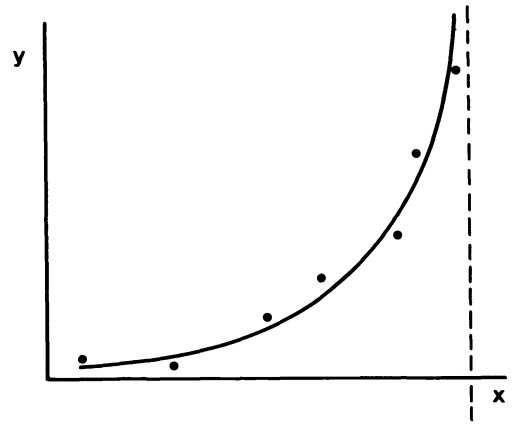


Figure 4

these problems and provides a reasonable fit. A related case is the problem of fitting near an asymptote as illustrated in Fig. 4. Orthogonal distances here prevent the undue influence of points close to the asymptote. This problem is discussed further in §4.

The literature contains several algorithms for solving (ODR) and related problems. For example, Golub and Van Loan [GolV83] give a singular value decomposition procedure for the problem when f is linear. They refer to this problem as total least squares. Britt and Luecke [BriL73] consider the nonlinear case as well as the nonlinear implicit case and present an algorithm. Powell and Macdonald [PowM72] also consider (ODR) and propose an algorithm which ignores the mixed partial derivatives and approximates the other derivatives by (central) finite differences. They report some good results on small problems. More recently, Schwetlick and Tiller [SchT85] propose an algorithm similar to the one here for the nonlinear problem. Our algorithm, however, does not make use of the singular value decomposition and it does incorporate a full trust region strategy. Finally, Watson [Wat85] also proposes a structured approach for the Gauss-Newton step, but does not propose a trust region strategy.

2. The Algorithm. In order to solve the minimization problem (ODR),

$$\min_{\beta,\delta} \sum_{i=1}^{n} w_i^2 \left[(f(x_i + \delta_i; \beta) - y_i)^2 + \delta_i^T D_i^2 \delta_i \right] \tag{2.1}$$

we first express it in a more convenient form and simplify the notation. Next, we give an overview of the iteration which is based on the trust region Levenberg-Marquardt strategy popularized by Moré [Mor77]. (See also [Heb73], [DenS83].) We then show how to modify this technique to obtain an algorithm which requires the same order of work per iteration as these algorithms applied to the same problem without allowing changes to x_i. That is, if the $\delta_i's$ are held fixed at zero, ODR reduces to OLS and trust region methods require $O(np^2)$ operations per iteration. Our algorithm, by exploiting the structure of (ODR), still requires only $O(np^2) + O(nm)$ operations per iteration to solve the problem.

While we have designed and implemented the algorithm to handle the full generality of (2.1), the notation is considerably simplified by assuming $x_i \in R^1$. We

temporarily make this assumption and rewrite (2.1) into the form of an OLS problem by the following device. Let

$$g_i(\beta,\delta) = \begin{cases} w_i\left(f(x_i+\delta_i;\beta)-y_i\right), & i=1,\cdots,n, \\ w_{i-n}d_{i-n}\delta_{i-n}, & i=n+1,\cdots,2n. \end{cases} \tag{2.2}$$

Also let $G: R^{p+n} \to R^{2n}$ have component functions $g_i(\eta)$ where $\eta = \binom{\beta}{\delta}$. Now (ODR) becomes

$$\min_{\eta} \|G(\eta)\|^2 = \min_{\beta,\delta} \sum_{i=1}^{2n} \left(g_i(\beta,\delta)\right)^2 \tag{2.3}$$

which is an OLS problem with $(p+n)$ parameters and $2n$ equations. (In all cases in this paper, $\|\cdot\|$ denotes the ℓ_2 vector or matrix norm.) Direct application of trust region methods to (2.3) would require $O(2n(n+p)^2)$ operations per iteration which rapidly becomes prohibitive if n is large. (Recall that n is usually far greater than p in practice.)

The basic idea of a trust region strategy is to choose as the step that vector which minimizes a linear approximation to G over a region in which the linearization is a "reasonable" approximation to G. Specifically, if $G'(\eta^c) \in R^{2n\times(n+p)}$ is the Jacobian matrix of G evaluated at the current iterate, η^c, then the step z is chosen by solving

$$\begin{aligned} &\min_{z} \|G(\eta^c)+G'(\eta^c)z\|^2 \\ &\text{subject to} \|Zz\| \le \tau \end{aligned} \tag{2.4}$$

where Z is a nonsingular (usually diagonal) scaling matrix and τ is the trust region radius. It is easy to show that the solution to (2.4) is given by the $z(\alpha)$ satisfying

$$\left(G'(\eta^c)^T G'(\eta^c) + \alpha Z^T Z\right) z(\alpha) = -G'(\eta^c)^T G(\eta^c) \tag{2.5}$$

where $\alpha > 0$ is the Lagrange multiplier for the inequality constraint. Note that if $\|Zz(0)\| \le \tau$, $\alpha = 0$ and the constraint is inactive. Otherwise $\alpha > 0$ and the constraint is active. Equation (2.5) is the famous Levenberg-Marquardt formula, but this derivation has given rise to more stable and robust implementations. (See, e.g., [Mor77] and [DenS83]). Clearly (2.5) can be regarded as the"normal equations" for the extended least squares problem,

$$\begin{bmatrix} G'(\eta^c) \\ \alpha^{1/2}Z \end{bmatrix} z =_2 - \begin{bmatrix} G \\ 0 \end{bmatrix} \tag{2.6}$$

where "$=_2$" means "equal in the least squares sense."

Our implementation is based on the careful exploitation of the structure of the extended Jacobian matrix in (2.6). From (2.2) we have that

$$G'(\eta^c) = \begin{pmatrix} J & V \\ 0 & D \end{pmatrix}$$

where

$$J \in R^{n\times p}: \quad J_{ij} = \frac{\partial g_i(\beta,\delta)}{\partial \beta_j} = \frac{w_i \partial f(x_i+\delta_i;\beta)}{\partial \beta_j},$$
$$i = 1,\cdots,n, \qquad j = 1,\cdots,p;$$
$$V \in R^{n\times n}: \quad V_{ij} = \frac{\partial g_i(\beta,\delta)}{\partial \delta_j} = \frac{w_i \partial f(x_i+\delta_i;\beta)}{\partial \delta_j},$$
$$i = 1,\cdots,n, \qquad j = 1,\cdots,n;$$
$$D \in R^{n\times n}: \quad D = \operatorname{diag}\{w_i d_i, i = 1,\cdots,n\}.$$

Here, we have omitted the arguments of J and V for the sake of clarity. Observe that since g_i only depends on δ_i, $i = 1,\cdots,n$,

$$V = \operatorname{diag}\left\{\frac{\partial g_i(\beta,\delta)}{\partial \delta_i}, i = 1,\cdots,n\right\}.$$

Commensurate with this partitioning of $G'(\eta^c)$, η^c is naturally partitioned into components $(\beta^c,\delta^c)^T$ and the step z into a step in β, say s, and a step in δ, say t. Furthermore, we allow for s to be scaled by a nonsingular diagonal scaling matrix S and t by a nonsingular diagonal matrix T. Thus (2.6) becomes

$$\begin{bmatrix} J & V \\ 0 & D \\ \alpha^{1/2}S & 0 \\ 0 & \alpha^{1/2}T \end{bmatrix} \begin{bmatrix} s \\ t \end{bmatrix} =_2 - \begin{bmatrix} G_1 \\ G_2 \\ 0 \\ 0 \end{bmatrix} \tag{2.7}$$

where G_1 is the first n components of G and G_2 is the last n components.

Now, if $x_i \in R^m$, then (2.7) will have the same form except that $V \in R^{n\times nm}$; $T, D \in R^{nm\times nm}$ are still diagonal; and V, instead of being diagonal, has the "staircase" structure which is illustrated for $n = 4$ and $m = 3$ as follows:

$$V = \begin{bmatrix} xxx & & & \\ & xxx & & \\ & & xxx & \\ & & & xxx \end{bmatrix} = \begin{bmatrix} V_1 & & & \\ & V_2 & & \\ & & V_3 & \\ & & & V_4 \end{bmatrix}$$

where $V_i \in R^{1\times m}$. The rest of the development now allows $x_i \in R^m$.

To derive an efficient procedure for solving (2.7), we first form the normal equations associated with (2.7):

$$\begin{pmatrix} J^TJ + \alpha S^2 & J^TV \\ V^TJ & V^TV + D^2 + \alpha T^2 \end{pmatrix} \begin{pmatrix} s \\ t \end{pmatrix} = - \begin{pmatrix} J^TG_1 \\ V^TG_1 + DG_2 \end{pmatrix}. \tag{2.8}$$

Let $P = V^TV + D^2 + \alpha T^2$. Solving the bottom equations of (2.8) for t in terms of s yields

$$t = -P^{-1}(V^TG_1 + DG_2 + V^TJs) \tag{2.9}$$

and then substituting (2.9) into the top equations of (2.8) gives

$$[J^T(I - VP^{-1}V^T)J + \alpha S^2]\, s = -J^T G_1 + J^T V P^{-1}(V^T G_1 + DG_2). \tag{2.10}$$

Thus s is the solution to the least squares problem

$$\begin{pmatrix} (I - VP^{-1}V^T)^{1/2} J \\ \alpha^{1/2} S \end{pmatrix} s \\ =_2 \begin{pmatrix} (I - VP^{-1}V^T)^{-1/2} \left[-G_1 + VP^{-1}(V^T G_1 + DG_2)\right] \\ 0 \end{pmatrix}. \tag{2.11}$$

The following propositions provide the necessary tools for the accurate and efficient solution of (2.9) and (2.11).

PROPOSITION 2.1. *Let* $E = D^2 + \alpha T^2$. *Then,* $VP^{-1}V^T = diag\{\omega_i/(1+\omega_i), i = 1, \cdots n\}$ *where*

$$\omega_i = \sum_{j=1}^{m} \frac{V_{ij}^2}{E_{(i-1)m+j}}, \qquad i = 1, \cdots, n.$$

Proof. We have that E is a nonsingular diagonal matrix and that $V^T V$ is block diagonal where the blocks are $m \times m$ rank 1 matrices. Thus P^{-1} can be calculated by the Sherman-Morrison-Woodbury formula (see, e.g., [DenS83, p. 188] or [OrtR70, p. 50]) as

$$\begin{aligned} P^{-1} &= (V^T V + E)^{-1} \\ &= E^{-1} - E^{-1} V^T (I + VE^{-1}V^T)^{-1} VE^{-1}. \end{aligned} \tag{2.12}$$

By direct calculation, $VE^{-1}V^T = \text{diag}\,\{\omega_i,\ i = 1, \cdots, n\}$ with ω_i as defined above. The calculation of $VP^{-1}V^T$ follows directly from (2.12).

Diagonal matrices involving the ω_i frequently enter into the formulas which follow. We simplify the notation by defining

$$M = \text{diag}\left\{ \left[\frac{1}{1+\omega_i}\right]^{1/2},\ i = 1, \cdots, n \right\}.$$

PROPOSITION 2.2. $(I - VP^{-1}V^T)^{1/2} = M$ *where* M *is as above.*

Proof. The result follows immediately from Proposition 2.1.

PROPOSITION 2.3. *With* M *as above*

$$VP^{-1} = M^2 VE^{-1}$$

and

$$P^{-1}V^T = E^{-1}V^T M^2.$$

Proof. The results follow directly from (2.12) and Propositions 2.1 and 2.2.

PROPOSITION 2.4. *The right-hand side of* (2.11) *is given by*

$$(I - VP^{-1}V^T)^{-1/2} \left[-G_1 + VP^{-1}(V^T G_1 + DG_2)\right] = -M(G_1 - VE^{-1}DG_2).$$

Proof. We have that

$$\begin{aligned} &-G_1 + VP^{-1}(V^T G_1 + DG_2) \\ &\quad = -(I - VP^{-1}V^T)G_1 + VP^{-1}DG_2 \\ &\quad = -M^2 G_1 + VP^{-1}DG_2. \end{aligned}$$

Now the result follows immediately from Propositions 2.2 and 2.3.

With the above formulas, we can solve (2.11) for s by first calculating E and ω in $O(nm)$ operations and then forming

$$\bar{J} = MJ$$

and

$$y = -M(G_1 - VE^{-1}DG_2).$$

Now rewrite the least squares problem (2.11) as

$$\begin{pmatrix} \bar{J} \\ \alpha^{1/2} S \end{pmatrix} s =_2 \begin{pmatrix} y \\ 0 \end{pmatrix} \tag{2.13}$$

which requires $O(np + nm)$ operations. The solution of (2.13) then involves a QR decomposition of $\bar{J}$ (accomplished by Householder transformations with column pivoting) and then a sequence of plane rotations to eliminate $\alpha^{1/2}S$. The cost for this phase is dominated by the $O(np^2)$ operations for the QR decomposition of $\bar{J}$.

Using Proposition 2.3 and (2.12), it is easily verified that

$$t = -E^{-1}\left[V^T M^2(G_1 + Js - VE^{-1}DG_2) + DG_2\right]$$

which is dominated in cost by the $O(np)$ operations needed to form Js and several $O(nm)$ terms. Thus the leading cost of calculating a step for ODR is the same $O(np^2)$ operations needed to do the factorization of an $n \times p$ matrix as in OLS. The only additional costs are a small number of calculations costing $O(nm)$ or $O(np)$ operations.

It may occur to the reader that an efficient QR factorization of the matrix in (2.7) might yield a procedure with the same order of work. By re-ordering the upper 2×2 blocks, one can, indeed, do the factorization of this part in $O(np^2)$ operations. The subsequent elimination of the αS and αT blocks, however, would require $O((nm+p)^2)$ operations for each α. It is for this reason, as well as others, that Schwetlick and Tiller [SchT85] do only a "partial" trust region strategy, i.e., their trust region only applies to the step in the β variables. In some badly scaled problems, however, (e.g., Example 3 in §4) the ability to scale and constrain the step in δ is essential to solve the problem.

The above formulas for s and t are used for each α value in (2.5). Thus in order to complete the specification of the algorithm, we need to provide the procedure for computing the trust region parameter α to satisfy (2.4) and to discuss a few miscellaneous details.

Moré [Mor77], following Hebden [Heb73] (see also [DenS83]), suggested a procedure for computing α in (2.5) so that

$$\|Zz(\alpha)\| \approx \tau$$

when $\|Zz(0)\| > \tau$. This procedure is based on approximating the function

$$\phi(\alpha) = \|Zz(\alpha)\| - \tau$$

by a rational function $\theta(\alpha) = \gamma/(\mu - \alpha)$ where the constants γ and μ are chosen by making $\theta(\alpha^c) = \phi(\alpha^c)$ and $\theta'(\alpha^c) = \phi'(\alpha^c)$. This results in the iteration

$$\alpha^+ = \alpha^c - \frac{\phi(\alpha^c)}{\phi'(\alpha^c)} \times \frac{\phi(\alpha^c) - \tau}{\tau}$$

which is a modified Newton step for the equation $\phi(\alpha) = 0$. In our case, the derivative of $\phi(\alpha)$ is not as simple to compute as in OLS and thus we opt to compute γ and μ by making $\theta(\alpha^c) = \phi(\alpha^c)$ and $\theta(\alpha^-) = \phi(\alpha^-)$. ($\alpha^-$ is the previous estimate.) Then

$$\alpha^+ = \alpha^c - \frac{\phi(\alpha^c)(\alpha^c - \alpha^-)}{\phi(\alpha^c) - \phi(\alpha^-)} \times \frac{\phi(\alpha^c) + \tau}{\tau}$$

which is clearly a modified secant step for the equation $\phi(\alpha) = 0$. Moré found it necessary to safeguard his procedure by computing and updating upper and lower bounds on α. Similarly, we maintain such bounds, but with formulas appropriate to the secant-like method. These will be provided in a subsequent paper.

The trust region bound τ is updated according to well-tested ideas which are in several existing codes. (See e.g., [Mor77], [Gay84], [SchKW86].) A valuable feature in our code has been the "internal doubling" step. For a given η^c and τ, suppose z_τ is generated such that τ restricts z and the reduction in $\|G\|$ predicted by the linear approximation agrees with the actual reduction in $\|G\|$ to a high precision. Normally, one would accept z_τ , set $\eta^+ := \eta^c + z_\tau$ and double τ for the next step. The internal doubling procedure is to remember $\eta^r := \eta^c + z_\tau$, double τ and generate a new z_τ from η^c. Note that this procedure only costs an evaluation of G and, if successful, may save several evaluations of J . In practice, it has been successful often enough to warrant leaving it in. Its main advantage is that it permits rapid and cheap increases in τ based on an overly conservative initial value of τ or when the iterates are moving away from a highly nonlinear region in parameter space.

Since many users will want to compare the results of OLS with ODR, an option to do OLS has been implemented. Enabling this option merely initializes the δ vector to zero and sets V to zero whenever it is computed. It is easily verified that, in this case, (2.11) reduces to the OLS Levenberg-Marquardt step and (2.9) yields $t = 0$ leaving $\delta = 0$. Using this procedure to do OLS, therefore, is equivalent to a standard OLS algorithm with a moderate extra algebraic overhead.

3. Local and global convergence analysis. The global convergence properties of trust region-Levenberg-Marquardt methods applied to the general nonlinear least squares problem (2.3) are well known (see e.g., [Pow75], [Mor77], [MorS81], [SchSB85]). As long as the sequence of Jacobian matrices, $\{G'(\eta_k)\}$, is uniformly bounded, then

$$\lim_{k \to \infty} G'(\eta_k)^T G(\eta_k) = 0,$$

so that any cluster point satisfies the first order necessary conditions for a local minimizer. These results apply to our algorithm and nothing more needs to be said regarding global convergence.

The local convergence behavior of general trust region-Levenberg-Marquardt methods for nonlinear least squares is discussed by Byrd and Schnabel [ByrS87] who show that, if there is a cluster point η_* where $G'(\eta_*)$ is nonsingular, then the iterates converge at least linearly to η_* independent of the size of $G(\eta_*)$. This theory also applies to our algorithms. If, in addition, the residual $G(\eta_*)$ is sufficiently small, Byrd and Schnabel show that asymptotically the trust region constraint becomes inactive, and that the Levenberg-Marquardt algorithm reduces to the Gauss-Newton iteration

$$\eta_{k+1} = \eta_k - \left[G'(\eta_k)^T G'(\eta_k)\right]^{-1} G'(\eta_k)^T G(\eta_k)$$

and is linearly convergent to η_*. The linear convergence analysis of the Gauss-Newton method is well known (see e.g., [OrtR70], [DenS83]). The constant of linear convergence depends upon the smallest singular value of $G'(\eta_*)$, the residual $G(\eta_*)$, and the nonlinearity of $G(\eta)$ near η_*.

The small residual analysis is particularly relevent to ODR because most applications of ODR will have small residuals. This is especially true when ODR is used to consider errors in independent variables in parameter estimation, because errors in the independent variables are most likely to be considered when the model and the dependent variable measurements are accurate, which implies that the residuals will be small.

It turns out that the application of the local Gauss-Newton analysis to ODR is nontrivial, although the expected results can be proven. This is the main contribution of this section. To simplify the algebra here, we consider a version of the ODR problem (2.1) with the simplified weighting scheme $w_i = 1$ and $d_i = \sigma$ for all i, i.e.,

$$\min_{\beta,\ \delta} \sum_{i=1}^{n} \left[(f(x_i + \delta_i; \beta) - y_i)^2 + \sigma^2 \delta_i^T \delta_i\right] \tag{3.1}$$

where $\sigma > 0$. This weighting still allows the metric of distance from the curve $f(x; \beta)$ to the data points (x_i, y_i) to vary from vertical (as $\sigma \to \infty$) to orthogonal ($\sigma = 1$) to horizontal (as $\sigma \to 0$). (We explain this statement more carefully later in this section.) This is all the generality in the weighting that is usually used in practice, and precisely what we use in most of our computational results in §4.

In fact, as we illustrate in §4, in practical ODR applications, the user may wish to solve (3.1) for various values of σ. A second contribution of this section is to produce a bound on the constant of linear convergence of the Gauss-Newton method applied to (3.1) that is independent of the value of σ.

To further simplify notation, we rewrite (3.1) as

$$\min_{\eta} R(\eta)^T R(\eta) + \sigma^2 \delta^T \delta \tag{3.2a}$$

or equivalently,

$$\min_{\eta} G(\eta)^T G(\eta) \tag{3.2b}$$

where $\delta = (\delta_1^T, \delta_2^T, \cdots, \delta_n^T)^T$, $\eta = (\beta^T, \delta^T)^T$, $R(\eta)_i = f(x_i + \delta_i; \beta) - y_i$, $i = 1, \cdots, n$, and $G(\eta) = (R(\eta)^T, \sigma\delta^T)^T$. Our analysis will not depend upon the special form of

$R(\eta)$ in any way. Recall that

$$G'(\eta) = \begin{pmatrix} J(\eta) & V(\eta) \\ 0 & \sigma I \end{pmatrix}$$

where $J(\eta)$ and $V(\eta)$ are, as in §2, the derivatives of $R(\eta)$ with respect to β and δ respectively.

The difficulty in applying standard Gauss-Newton analysis to (3.2) is that $G(\eta)$ and $G'(\eta)$ are functions of σ. In Theorem 3.5 we show that the convergence can be analyzed in terms of the properties of $J(\eta)$, $V(\eta)$, $R(\eta_*)$, and δ_* only, i.e., independent of σ except for its role in determining η_*. Lemmas 3.3 and 3.4 are used in the proof of Theorem 3.5 to bound the linear and quadratic terms in the convergence results, respectively, independent of σ. Lemmas 3.1 and 3.2 are technical results used in the proof of Lemma 3.3. Theorem 3.5 can be applied to the more generally weighted problem (2.1) by making an appropriate change of variables.

LEMMA 3.1. *Let $A \in R^{p\times p}$, $C \in R^{q\times q}$, $B \in R^{p\times q}$, and let*

$$H = \begin{bmatrix} A & B \\ B^T & C \end{bmatrix}$$

be symmetric and positive definite. Then $\|H\| \le \|A\| + \|C\|$.

Proof. Define $a = \|A\|, c = \|C\|$. For any $v \in R^p$ and $w \in R^q$, consider

$$y = \begin{bmatrix} v\|w\|\sqrt{c} \\ -w\|v\|\sqrt{a} \end{bmatrix}.$$

Then, since H is positive definite,

$$0 \le y^T H y = v^T A v \|w\|^2 c - 2v^T B w \|w\| \|v\| \sqrt{ac} + w^T C w \|v\|^2 a$$

and therefore

$$\begin{aligned} 2v^T B w &\le \frac{v^T A v \|w\|^2 c + w^T C w \|v\|^2 a}{\|w\| \|v\| \sqrt{ac}} \\ &\le \frac{\|v\|^2 a \|w\|^2 c + \|w\|^2 c \|v\|^2 a}{\|w\| \|v\| \sqrt{ac}} \\ &= 2\|v\| \|w\| \sqrt{ac} \\ &\le \|v\|^2 c + \|w\|^2 a. \end{aligned}$$

Thus

$$\begin{aligned} \|H\| &= \max_{v\in R^p,\ w\in R^q} \frac{v^T A v + 2v^T B w + w^T C w}{v^T v + w^T w} \\ &\le \max_{v\in R^p,\ w\in R^q} \frac{\|v\|^2 a + (\|v\|^2 c + \|w\|^2 a) + \|w\|^2 c}{v^T v + w^T w} \\ &= a + c. \end{aligned}$$

LEMMA 3.2. *Let $A \in R^{n\times p}$ have full column rank and $B \in R^{n\times n}$ be positive definite. Then*

$$\|(A^T B^{-1} A)^{-1}\| \le \|(A^T A)^{-1}\| \|B\|.$$

Proof.

$$\|(A^TB^{-1}A)^{-1}\| = \max_{v\in R^p} \frac{v^Tv}{v^TA^TB^{-1}Av}$$
$$\leq \max_v \frac{v^Tv\|B\|}{v^TA^TAv} \leq \|(A^TA)^{-1}\|\,\|B\|.$$

LEMMA 3.3. *Let $J \in R^{n\times p}$ have full column rank, $V \in R^{n\times q}$, I the $q \times q$ identity, and σ a positive scalar. Let*

$$M(\sigma) = \begin{pmatrix} J & V \\ 0 & \sigma I \end{pmatrix}, \quad \textit{and} \quad N(\sigma) = M(\sigma)^TM(\sigma).$$

Then $N(\sigma)$ is nonsingular and

$$\|N(\sigma)^{-1}\| \leq \|(J^TJ)^{-1}\| + \sigma^{-2}\left(1 + \|(J^TJ)^{-1}\|\,\|V\|^2\right). \tag{3.3}$$

Proof. Since J has full column rank, $M(\sigma)$ has full column rank. Thus $N(\sigma)$ is positive definite, and it is straightforward to verify that

$$N(\sigma)^{-1} = \begin{pmatrix} A & B \\ B^T & C \end{pmatrix},$$

where

$$A = \left[\sigma^2J^T\left(V^TV + \sigma^2I\right)^{-1}J\right]^{-1},$$
$$C = \left[\sigma^2I + V^T\left(I - J(J^TJ)^{-1}J^T\right)V\right]^{-1},$$
$$B = -(J^TJ)^{-1}J^TVC.$$

From Lemma 3.2,

$$\begin{aligned}\|A\| &\leq -\sigma^{-2}\|(J^TJ)^{-1}\|\,\|V^TV + \sigma^2I\| \\ &= \sigma^{-2}\|(J^TJ)^{-1}\|(\|V^TV\| + \sigma^2) \\ &= \sigma^{-2}\|(J^TJ)^{-1}\|\,\|V\|^2 + \|(J^TJ)^{-1}\|.\end{aligned} \tag{3.4}$$

Since $C^{-1} - \sigma^2I = V^T\left[I - J(J^TJ)^{-1}J^T\right]V$ is positive semi-definite, the smallest eigenvalue of C^{-1} is at least σ^2 which shows

$$\|C\| \leq \sigma^{-2}. \tag{3.5}$$

Thus, since $N(\sigma)^{-1}$ is positive definite, applying Lemma 3.1 and using (3.4) and (3.5) gives (3.3).

LEMMA 3.4. *Let the assumptions of Lemma 3.3 hold, and let $u \in R^n$. Let z be the solution to*

$$\min_{z\in R^{p+q}} \left\| \begin{pmatrix} J & V \\ 0 & \sigma I \end{pmatrix} z - \begin{pmatrix} u \\ 0 \end{pmatrix} \right\|. \tag{3.6}$$

Let $J^+ = (J^TJ)^{-1}J^T$, let $Z \in R^{n\times(n-p)}$ satisfy $Z^TZ = I$ and $Z^TJ = 0$, let $\tilde{V} = Z^TV$, and let $\tilde{V}^+$ denote the pseudoinverse of $\tilde{V}$. Then

$$\|z\| \leq \left[\|\tilde{V}^+\| + \|J^+\|\left(1 + \|V\|\,\|\tilde{V}^+\|\right)\right]\|u\|.$$

Proof. Define $y \in R^{p+q}$ by

$$z = \begin{pmatrix} I & -J^+V \\ 0 & I \end{pmatrix} y$$

and let

$$z = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}, \qquad y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

where $z_1, y_1 \in R^p$, and $z_2, y_2 \in R^q$. Then (3.6) is equivalent to

$$\min_{y \in R^{p+q}} \left\| \begin{pmatrix} J & ZZ^TV \\ 0 & \sigma I \end{pmatrix} y - \begin{pmatrix} u \\ 0 \end{pmatrix} \right\| \tag{3.7}$$

since $(I - JJ^+) = ZZ^T$. From the normal equations and using $J^TZ = 0$, the solution to (3.7) is

$$\begin{aligned} y_1 &= J^+u, \\ y_2 &= (V^TZZ^TV + \sigma^2 I)^{-1}V^TZZ^Tu \\ &= (\tilde{V}^T\tilde{V} + \sigma^2 I)^{-1}\tilde{V}^TZ^Tu. \end{aligned}$$

It is well known, for instance from the theory of Levenberg-Marquardt methods, that

$$\|y_2\| \le \left\|\tilde{V}^+Z^Tu\right\|,$$

and since $\left\|Z^Tu\right\| \le \|u\|$,

$$\|y_2\| \le \left\|V^+\right\| \|u\|, \qquad \|y_1\| \le \left\|J^+\right\| \|u\|.$$

Finally since $z_2 = y_2$ and $z_1 = y_1 - J^+Vy_2$,

$$\|z_2\| \le \left\|\tilde{V}^+\right\| \|u\|$$

and

$$\begin{aligned} \|z_1\| &\le \|y_1\| + \left\|J^+\right\| \|V\| \|y_2\| \\ &\le \left\|J^+\right\| \left(1 + \|V\| \left\|\tilde{V}^+\right\|\right) \|u\|. \end{aligned}$$

Using $\|z\| \le \|z_1\| + \|z_2\|$ gives the desired result.

Throughout the statement and proof of Theorem 3.5, we will often omit the argument η; i.e., we will denote $G(\eta_*)$ and $G(\eta_0)$ by G_* and G_0, respectively, and likewise for other symbols in place of G. Also for J having full column rank, J^+ will denote $(J^TJ)^{-1}J^T$, and for $\tilde{V}$ having full row rank, $\tilde{V}^+$ will denote $\tilde{V}^T(\tilde{V}\tilde{V}^T)^{-1}$. Note that $\|J^+\|^2 = \left\|(J^TJ)^{-1}\right\|$.

THEOREM 3.5. *Let $R(\eta) : R^t \to R^n$ be continuously differentiable in an open convex set $D \subset R^t$. Let $\eta^T = (\beta^T, \delta^T)$, $\beta \in R^p$, $\delta \in R^q$, let σ be a positive scalar, and let*

$$G(\eta) = \begin{pmatrix} R(\eta) \\ \sigma\delta \end{pmatrix}.$$

Assume there exists $\eta_* \in D$ such that $G'(\eta_*)^T G(\eta_*) = 0$, and that there exists $\gamma \geq 0$ for which

$$\|R'(\eta) - R'(\eta^*)\| \leq \gamma \|\eta - \eta_*\| \tag{3.8}$$

for all $\eta \in D$. Assume that $J(\eta_*)$ and $R'(\eta_*)$ have full column and row rank, respectively. Let $Z_* \in R^{n\times(n-p)}$ satisfy $Z_*^T Z_* = I$, $Z_*^T J_* = 0$, and define $\tilde{V}_* = Z_*^T V_*$. Define

$$c_1 = \gamma \left[\left\| (J_*^T J_*)^{-1} \right\| \|R_*\| + \left(1 + \left\| (J_*^T J_*)^{-1} \right\| \|V_*\|^2\right) \left\| \tilde{V}_*^+ \right\| \|\delta_*\| \right],$$
$$c_2 = (\gamma/2) \left[\left\| \tilde{V}_*^+ \right\| + \left\| J_*^+ \right\| \left(1 + \|V_*\| \left\| \tilde{V}_*^+ \right\|\right)\right].$$

If $c_1 < 1$, then for any $c \in (1, 1/c_1)$, there exists $\epsilon > 0$ such that for all η_0 for which $\|\eta_0 - \eta_*\| \leq \epsilon$, the sequence generated by the Gauss-Newton method

$$\eta_{k+1} = \eta_k - \left(G'(\eta_k)^T G'(\eta_k)\right)^{-1} G'(\eta_k)^T G(\eta_k)$$

is well defined, converges to η_* and obeys

$$\|\eta_{k+1} - \eta_*\| \leq c\,(c_1 + c_2 \|\eta_k - \eta_*\|) \|\eta_k - \eta_*\|. \tag{3.9}$$

Proof. The crucial remaining part of the proof is to note that since the optimality condition $G_*' G_* = 0$ gives $V_*^T R_* + \sigma^2 \delta_* = 0$, and since $R_* = Z_* Z_*^T R_*$ because $J_*^T R_* = 0$, we have

$$V_*^T Z_* Z_*^T R_* = \tilde{V}_*^T Z_*^T R_* = -\sigma^2 \delta_*.$$

Since R_*' has full row rank, $\tilde{V}_*$ has full row rank so that

$$Z_*^T R_* = \sigma_*^2 (\tilde{V}_* \tilde{V}_*^T)^{-1} \tilde{V}_*^T \delta_* = -\sigma_*^2 (\tilde{V}_*^+)^T \delta_*$$

and

$$\|R_*\| = \left\| Z_* Z_*^T R_* \right\| \leq \left\| Z_*^T R_* \right\| \leq \sigma_*^2 \left\| \tilde{V}_*^+ \right\| \|\delta_*\|.$$

Thus from (3.3) of Lemma 3.2,

$$\begin{aligned} &\left\| \left((G_*')^T G_*'\right)^{-1} \right\| \|R_*\| \\ &\quad \leq \left\| \left(J_*^T J_*\right)^{-1} \right\| \|R_*\| + \sigma_*^{-2} \left(1 + \left\| \left(J_*^T J_*\right)^{-1} \right\| \|V_*\|^2\right) \|R_*\| \\ &\quad \leq \left\| \left(J_*^T J_*\right)^{-1} \right\| \|R_*\| + \left(1 + \left\| \left(J_*^T J_*\right)^{-1} \right\| \|V_*\|^2\right) \left\| \tilde{V}_*^+ \right\| \|\delta_*\| \\ &\quad = c_1/\gamma. \end{aligned} \tag{3.10}$$

The remainder of the proof is by induction. Let c be a fixed constant in $(1, 1/c_1)$. Then there exists $\epsilon_1 > 0$ such that for all η for which $\|\eta - \eta_*\| \leq \epsilon_1$, $J(\eta)$and$V(\eta)$ have full column and row rank, respectively, and

$$\left\| \left(G'(\eta)^T G'(\eta)\right)^{-1} \right\| \leq c \left\| \left((G_*')^T G_*'\right)^{-1} \right\|, \tag{3.11}$$

and

$$\begin{aligned}(3.12)\qquad &\left\|\tilde{V}(\eta)^{+}\right\| + \|J(\eta)^{+}\|\left(1+\|V(\eta)\|\ \left\|\tilde{V}(\eta)^{+}\right\|\right)\\ &\quad\le c\left[\left\|\tilde{V}_{*}^{+}\right\| + \|J_{*}^{+}\|\left(1+\|V_{*}\|\ \left\|\tilde{V}_{*}^{+}\right\|\right)\right].\end{aligned}$$

Let $\epsilon = \min\{\epsilon_1,\ (1-cc_1)/(2cc_2)\}$. Then at the first iteration $\left((G_0')^T G_0'\right)$ is nonsingular, and

$$\begin{aligned}(3.13)\qquad \eta_1 - \eta_* &= \eta_0 - \eta_* - \left((G_0')^T G'_0\right)^{-1}(G_0')^T G_0\\ &= a + b\end{aligned}$$

where

$$\begin{aligned}a &= -\left((G_0')^T G_0'\right)^{-1}(G_0')^T G_*,\\ b &= \left((G_0')^T G_0'\right)^{-1}(G_0')^T\left(G_* - G_0 - G_0'(\eta_* - \eta_0)\right).\end{aligned}$$

Since $(G_*')^T G_* = 0$,

$$(3.14)\qquad (G_0')^T G_* = (G_0' - G_*')^T G_* = (R_0' - R_*')^T R_*$$

due to the linearity of the final q components of $G(\eta)$. Using (3.14) plus (3.8), (3.10), and (3.11), we have

$$\begin{aligned}(3.15)\qquad \|a\| &\le \left\|\left((G_0')^T G'_0\right)^{-1}\right\|\ \|R_0' - R_*'\|\ \|R_*\|\\ &\le c\gamma\left\|\left((G_*')^T G_*'\right)^{-1}\right\|\ \|R_*\|\ \|\eta_0 - \eta_*\|\\ &\le cc_1\|\eta_0 - \eta_*\|.\end{aligned}$$

Also note that b is the solution to

$$\min\|G_0' b - [G_* - G_0 - G_0'(\eta_* - \eta_0)]\|,$$

and that

$$(3.16)\qquad G_* - G_0 - G_0'(\eta_* - \eta_0) = \begin{pmatrix} R_* - R_0 - R_0'(\eta_* - \eta_0)\\ 0\end{pmatrix}.$$

From (3.8) and standard results

$$(3.17)\qquad \|R_* - R_0 - R_0'(\eta_* - \eta_0)\| \le (\gamma/2)\|\eta_0 - \eta_*\|^2.$$

Thus using (3.16), (3.17), Lemma 3.4, and (3.12),

$$\begin{aligned}(3.18)\qquad \|b\| &\le \left[\left\|\tilde{V}_0^{+}\right\| + \|J_0^{+}\|\left(1+\|V_0\|\ \left\|\tilde{V}_0^{+}\right\|\right)\right](\gamma/2)\|\eta_0 - \eta_*\|^2\\ &\le cc_2\|\eta_0 - \eta_*\|^2.\end{aligned}$$

Substituting (3.15) and (3.18) into (3.13) and recalling $\|\eta_0 - \eta_*\| \le (1-cc_1)/2cc_2$,

$$\begin{aligned}\|\eta_1 - \eta_*\| &\le c(c_1 + c_2\|\eta_0 - \eta_*\|)\|\eta_0 - \eta_*\|\\ &\le [(1+cc_1)/2]\|\eta_0 - \eta_*\|\end{aligned}$$

which proves (3.9) in the case $k = 0$ and shows that $\|\eta_1 - \eta_*\| < \|\eta_0 - \eta_*\|$. The proof of the induction step is identical.

Byrd and Schnabel [ByrS87] show that the trust region-Levenberg-Marquardt algorithm described in §2 will reduce asymptotically to Gauss-Newton if the constant of linear convergence, c_1, in Theorem 3.5 is sufficiently small. In particular, our computer code will accept the trial step if the ratio of actual to "predicted" reduction,

$$\frac{\|G(\eta_c + z_T)\| - \|G(\eta_c)\|}{\|G(\eta_c) + G'(\eta_c)z_T\| - \|G(\eta_c)\|}$$

is at least 0.001. In this case, as long as $c_1 < .9999$, the trust region in the Levenberg-Marquardt algorithm is inactive asymptotically, so that the algorithm becomes Gauss-Newton.

Now we consider the behavior of the ODR problem (3.2) as the parameter σ is varied. For this purpose, let us denote the global minimizer to (3.2) by $\eta_*(\sigma)$. Then by standard analyses of barrier function methods, (see e.g., [FiaM68] or [Lue73]) we know that the limit of $\eta_*(\sigma)$ as $\sigma \to \infty$ is the solution to

$$\min_{\eta} \|R(\eta)\|^2 \quad \text{subject to } \delta = 0,$$

i.e., the standard OLS problem

$$\min_{\beta} \|R(\beta, 0)\|^2. \tag{3.19}$$

Similarly, the limit of $\eta_*(\sigma)$ as $\sigma \to 0$ is the solution to the implicit least squares (ILS) problem

$$\min_{\eta} \|\delta\|^2 \quad \text{subject to } R(\eta) = 0. \tag{3.20}$$

In the data fitting context where $R(\eta)_i = f(x_i + \delta_i; \beta) - y_i$, (3.19) is the standard problem where the independent variables x_i are assumed exact so that the metric of distance is in the y (vertical) direction only. In constast (3.20) is the case where the dependent variables y_i are assumed exact and the independent variables x_i inexact, so that the metric is entirely in the x (horizontal) direction.

The standard analysis of barrier function methods also shows that $\|R(\eta_*(\sigma))\|$ is a monotonically increasing function of σ, and that $\|\delta_*(\sigma)\|$ is a monotonically decreasing function of σ. This means that for all $\sigma \in (0, \infty)$, the values of $\|R(\eta_*(\sigma))\|$ and $\|\delta_*(\sigma)\|$ are bounded above by the optimal objective function values for problems (3.19) and (3.20), respectively. In data fitting terms, for any σ, the norm of the optimal vertical residuals in ODR is bounded above by the norm of the optimal residuals in OLS, and the norm of the optimal horizontal residuals in ODR is bounded above by the norm of the optimal residual for the ILS problem. The computational results of Section 4 demonstrate these relationships.

Combining the above facts with Theorem 3.5 shows that, if the optimal objective function values for problems (3.19) and (3.20) are sufficiently small, and if $J(\eta_*(\sigma))$ and $V(\eta_*(\sigma))$ are sufficiently well-conditioned for all $\sigma \in (0, \infty)$, then the Gauss-Newton algorithm applied to (3.2) is linearly convergent for any $\sigma \in (0, \infty)$.

COROLLARY 3.6. *Let η, β, δ, $R(\eta)$, $G(\eta)$, $J(\eta)$, and $V(\eta)$ be defined as in Theorem 3.5 For any $\sigma \in (0, \infty)$, let $\eta_*(\sigma) = (\beta_*(\sigma)^T, \delta_*(\sigma)^T)^T$ denote the global solution to*

$$\min_{\beta, \delta} \|R(\beta, \delta)\|^2 + \sigma^2 \|\delta\|^2 . \tag{3.21}$$

Also let β_{OLS} denote the global solution to the ordinary least squares problem

$$\min_{\beta} \|R(\beta, 0)\|^2$$

and let $(\beta_{\mathrm{ILS}}, \delta_{\mathrm{ILS}})$ denote the global solutions to the implicit least squares problem

$$\min_{\beta, \delta} \|\delta\|^2 \text{ subject to } R(\beta, \delta) = 0.$$

Let $R_{\mathrm{OLS}} = R(\beta_{\mathrm{OLS}})$. Assume that there exist $\hat{\epsilon} > 0$, $\hat{\gamma} \geq 0$ such that for each $\sigma \in (0, \infty)$,

$$\|R'(\eta) - R'(\eta_*(\sigma))\| \leq \hat{\gamma} \|\eta - \eta_*(\sigma)\|$$

for all η for which $\|\eta - \eta_*(\sigma)\| \leq \hat{\epsilon}$. Assume also that for all $\sigma \in (0, \infty)$, $J(\eta_*(\sigma))$ and $R'(\eta_*(\sigma))$ have full column and row rank, respectively, and let $\hat{J}$, $\hat{J}^+$, $\hat{V}$, and $\hat{V}^+$ be uniform bounds on $\|J(\eta_*(\sigma))\|$, $\|J(\eta_*(\sigma))^+\|$, $\|V(\eta_*(\sigma))\|$, and $\left\|\tilde{V}(\eta_*(\sigma))^+\right\|$, respectively, over all $\sigma \in (0, \infty)$ where $\tilde{V}$ is defined analogously to the definition in Theorem 3.5. Define

$$\hat{c}_1 = \hat{\gamma}\left[\left(\hat{J}^+\right)^2 R_{\mathrm{OLS}} + \left(1 + \left(\hat{J}^+\right)^2 \hat{V}^2\right) \hat{V} \delta_{\mathrm{ILS}}\right],$$

$$\hat{c}_2 = (\hat{\gamma}/2)\left[\hat{V}^+ + \hat{J}^+\left(1 + \hat{V}\hat{V}^+\right)\right].$$

If $\hat{c}_1 < 1$, then for any $c \in (1, 1/\hat{c}_1)$, there exists $\hat{\epsilon} > 0$ such that for any $\sigma \in (0, \infty)$, the sequence $\{\eta_k\}$ generated by the Gauss-Newton method applied to (3.2) starting from any η_0 for which $\|\eta_0 - \eta_*(\sigma)\| \leq \hat{\epsilon}$ is well-defined, converges to $\eta_*(\sigma)$, and obeys

$$\|\eta_{k+1} - \eta_*(\sigma)\| \leq c\left[\hat{c}_1 + \hat{c}_2 \|\eta_k - \eta_*(\sigma)\|\right] \|\eta_k - \eta_*(\sigma)\| .$$

4. Computational testing. In this section we report the results of preliminary computational testing. These tests, consisting of two contrived problems and one real problem, were selected in order to illustrate the effectiveness of the implementation and to demonstrate the performance of the basic algorithm. They also point out the need for good starting values in δ as well as in β.

The tests allow a contrast between OLS and ODR which is best brought out in terms of the parameter σ and the function $\beta(\sigma)$ from §3. (Recall that $\beta(\infty)$ corresponds to the OLS solution.) Since, in practice, the correct value of σ may not be known exactly, it is of interest to compute $\beta(\sigma)$ for various values of σ. In each example below we have computed several points on the trajectories $\beta(\sigma)$. Note that the solutions for subsequent values of σ are always obtained using the previous solution as the initial condition. Thus only the initial condition for the first problem in each sequence is important. In the first two examples, more than one trajectory is located.

The algorithm was coded in Fortran 77 and run in single precision on the CDC Cyber 855 at the National Bureau of Standards (NBS). The graphics were done on

the Evans and Sutherland PS-300, also located at NBS. For all of the examples, we used $.84 \times 10^{-7}$ for the f-convergence test and $.37 \times 10^{-9}$ for the x-convergence test. These are $\epsilon_{\text{mach}}^{2/3}$ and $\epsilon_{\text{mach}}^{1/2}$, respectively, where ϵ_{mach} is machine presision. The first two examples were run without scaling, i.e., $S = I$ and $T = I$; the third example was scaled as described below.

Example 1. Consider

$$y = \frac{1}{x - 1}$$

and define $x_i = .01 + (i - 1) * .05, i = 1, \cdots, 40$. Next let

$$y_i = \frac{1}{x_i - 1}, \qquad i = 1, \cdots, 40.$$

Now we perturb the data points as follows:

$$x_i := x_i + rx, \qquad y_i := y_i + ry$$

where the rx are uniformly distributed on $(-.05, .05)$ and the ry are uniformly distributed on $(-.25, .25)$. The model for the data was taken to be

$$y = \frac{\beta_1}{x - \beta_2}$$

and the ODR program was run with several values of σ.

The results are reported in Tables 1 and 2, and Figs. 5, 6 and 7. Table 1 was generated by taking $\beta^0 = (1, 1)^T$ and $\delta^0 = 0$. The second trajectory, reported in

TABLE 1

σ	$\beta_1(\sigma)$	$\beta_2(\sigma)$	Evaluations of G	Evaluations of G'	Final Value of $\|G_1(\eta(\sigma))\|$	Final Value of $\|G_2(\eta(\sigma))\|$
∞	0.6866	0.9909	52	15	21.773	0.0
1000	0.8248	0.9936	10	8	18.881	6.980
500	0.9486	0.9953	12	10	15.524	8.775
300	0.9881	0.9927	1	11	10.409	10.611
100	0.9246	0.9971	8	6	3.203	6.111
25	0.9847	1.001	6	4	1.280	2.003
5	1.015	1.003	6	4	0.711	0.637
2	1.021	1.005	6	4	0.454	0.446
1	1.022	1.005	6	4	0.228	0.354

TABLE 2

σ	$\beta_1(\sigma)$	$\beta_2(\sigma)$	Evaluations of G	Evaluations of G'	Final Value of $\|G_1(\eta(\sigma))\|$	Final Value of $\|G_2(\eta(\sigma))\|$
∞	-0.3172	1.099	20	9	104.708	0.0
1000	-0.3355	1.095	21	13	104.233	6.947
700	-0.3840	1.092	25	19	103.661	10.768
500	0.9486	0.9953	30	16	15.524	8.775

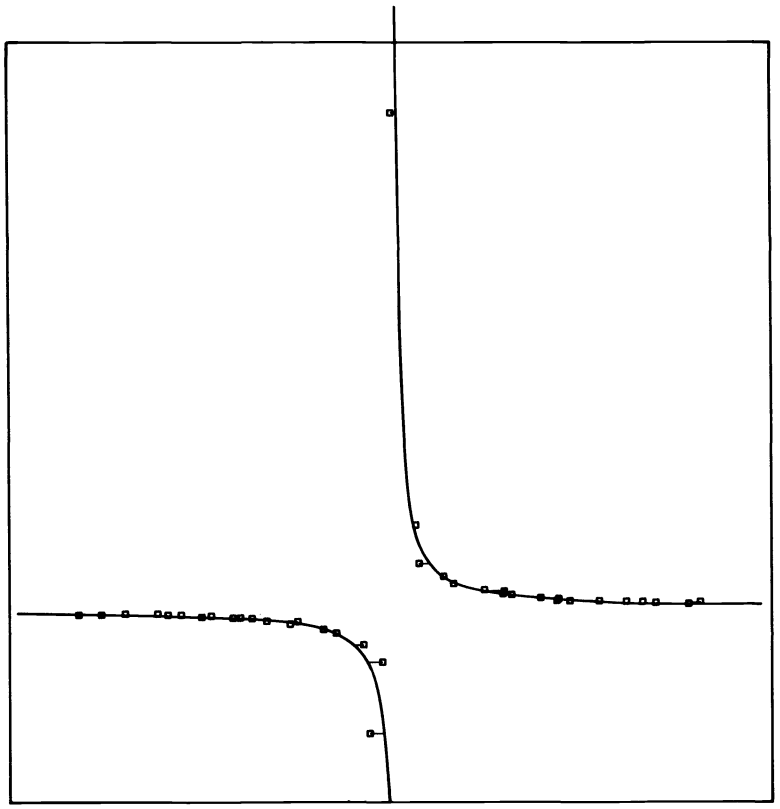


Figure 5
$\sigma = 5$

Table 2, was generated by taking $\beta^0 = (1, 1.1)^T$ and $\delta^0 = 0$. In addition to the values of $\beta(\sigma)$, Tables 1 and 2 contain the number of evaluations of the extended residual function G (cf (2.3)) and its Jacobian, and the optimal values of $\|G_1(\eta(\sigma))\|$ and $\|G_2(\eta(\sigma))\|$ for each value of σ. (Note that since $\|G_2(\eta(\sigma))\| = \sigma \|\delta(\sigma)\|$, dividing the entries in this column by the corresponding value of σ shows the monotonic behavior of $\|\delta(\sigma)\|$ as predicted in §3.) The graphs are as follows: Fig. 5 corresponds to the $\sigma = 2$ fit; Fig. 6 corresponds to the OLS fit from Table 1; and Fig. 7 to the OLS fit from Table 2. Note that on all of these plots, the y-axis has been scaled by a factor of approximately 100 in order to get all of the points on the plot. Because of this scaling, the error pegs (connecting the data points to their predicted values) appear to be much more horizontal than they really are.

Obviously, Tables 1 and 2 exhibit a nonuniqueness of the solutions. It appears that there are two local solutions for the OLS problem corresponding to the asymptote to $+\infty$ being on the left or right half of the curve, and that either the trajectories emanating from these solutions come together near $\sigma = 600$ or the trajectory represented in Table 2 fails to be continuous near $\sigma = 600$. A possible means of investigating this phenomenon is to write the differential equation describing the trajectory $\beta(\sigma)$ and to study possible bifurcation points. This is not pursued here.

Observe that β_2 determines the position of the asymptote and thus the data locate this parameter very well. Note, however, that the data point nearest the asymptote, corresponding to $(x_i^a, y_i^a) = (1.01, 100)$, completely dominates the fitting process for

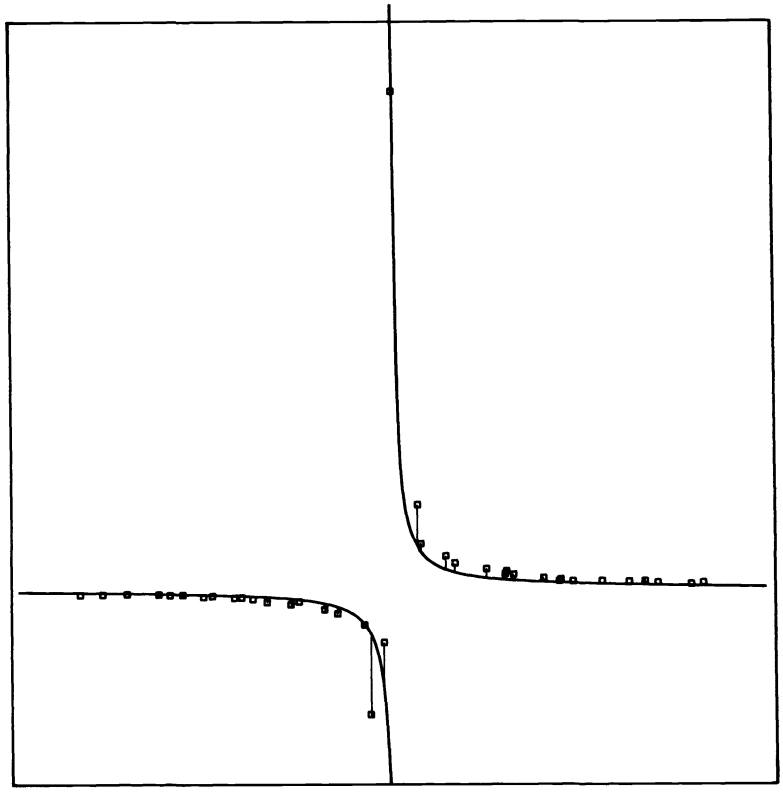


Figure 6
OLS from Table 1

large values of σ, including OLS ($\sigma = \infty$). For smaller values of σ, the ODR fit is not nearly so influenced by this data point and, for a broad range of σ, does a very good job of fitting the data. This last point is important, namely that the parameter values do not vary much as a function of σ, which means that the true value of σ may not need to be known with much accuracy. The stability of $\beta(\sigma)$ has been noticed on all of our examples and on problems not reported here. This is not, of course, a proof that this phenomenon holds more generally, but see Lakshminarayanan and Gunst [LakG84] who investigate this question in linear models. Finally, although the fit "looks" better for ODR, we stress that only one set of data was generated so that no statements on the statistical effectiveness of this procedure can be made. This aspect is currently under investigation.

An examination of the results reveals that, with proper initial values, none of the problems was difficult to solve and that subsequent problems were in fact quite easy. We do point out that the problems tended to be a little more difficult for large values of σ.

While it seems intuitive to use the initial value $\delta^0 = 0$, this is not always adequate. A glance at Fig. 5 reveals that the point nearest the asymptote, corresponding to (x_{21}, y_{21}), should correspond to the right half of the curve if the sign of β_1 is positive. Using $\sigma = 1$, $\beta^0 = (1,1)^T$, and $\delta^0 = 0$ implies the need to cross a discontinuity in δ_{21} in order to find the solution reported here. Initializing δ_{21} to the horizontal distance to

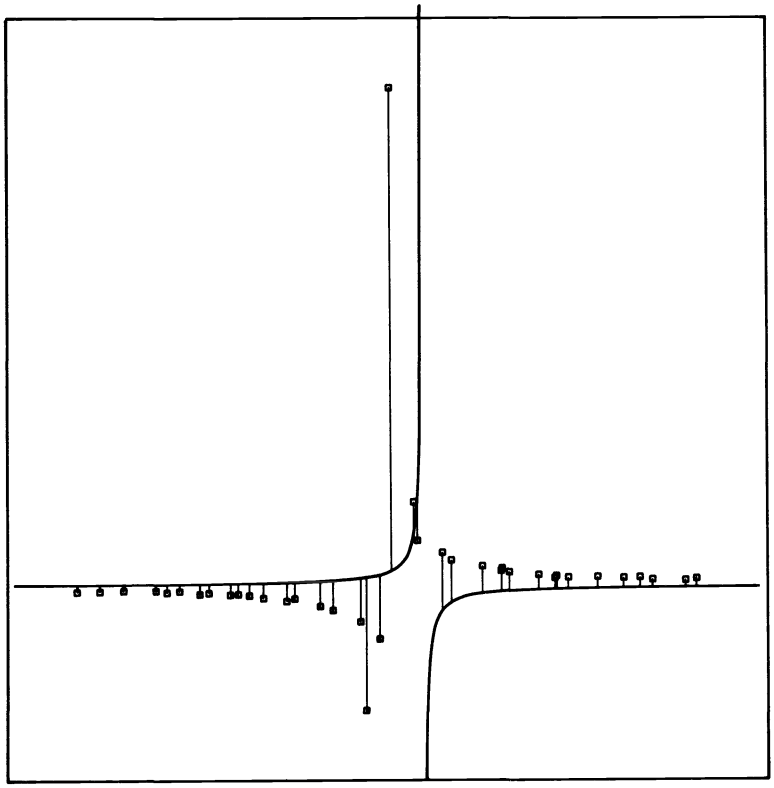


Figure 7
OLS from Table 2

the curve (easily accomplished in this case by algebraically solving for the appropriate values) alleviates this problem.

A final comment on the initial value of δ^0 is in order. Even without a discontinuity, a poor choice of δ^0 can cause the algorithm to converge very slowly. We are indebted to one of the referees for the following simple example. Let $y = e^{\beta x}$. Choose $\beta = 2$ and generate data and errors as for Example 1. Then choose $\beta^0 = 1$ and initialize δ^0 to the horizontal distances. Set $\sigma^2 = .001$ and solve the problem. The algorithm will converge very slowly since $G(\delta, \beta)$ must always remain smaller than $\sigma^2 \sum(\delta_i^0)^2$. With the starting value of $\delta^0 = 0$, however, convergence is very rapid.

Example 2. This example is a two-dimensional version of Example 1. Here we take $x \in R^2$ and

$$y = \frac{-1}{x_1 + x_2 - 1}.$$

This function has a line of singularities along $x_1 + x_2 = 1$. We take the data to be on the rectangular grid of width .1 in the x_1 direction and width .2 in the x_2 direction. The first point is $(.01, .01)^T$ and there are 10 points in the x_1 direction and 5 points in the x_2 direction. y is the evaluated at these points and the data are then perturbed

TABLE 3

σ	$\beta_1(\sigma)$	$\beta_2(\sigma)$	$\beta_3(\sigma)$	Evaluations of G	Evaluations of G'	Final Value of $\|G_1(\eta(\sigma))\|$	Final Value of $\|G_2(\eta(\sigma))\|$
1	0.8988	0.9482	1.014	16	9	0.183	0.669
2	0.9222	0.9478	1.018	8	6	0.427	1.236
4	0.9344	0.9505	1.027	8	6	0.988	2.160
10	0.9048	0.9510	1.047	9	7	2.378	4.289
40	0.7147	0.9568	1.043	10	8	6.410	12.618
100	0.3645	0.9343	0.9893	17	14	19.933	14.422
500	0.09138	0.8830	0.9675	24	15	30.423	19.464
∞	0.1191	0.8882	0.9338	10	6	77.443	0.0

according to the following:

$$(x_1)_i := (x_1)_i + rx,$$
$$(x_2)_i := (x_2)_i + rx,$$
$$y_i := y_i + ry$$

where rx are normally distributed with mean 0 and standard deviation .1 and the ry are distributed normally with mean 0 and standard deviation .2.

The form of the model is

$$y = \frac{-\beta_1}{\beta_2 x_1 + \beta_3 x_2 - 1}.$$

The results are given in Table 3 which is organized as Tables 1 and 2. The initial starting value for the function parameters is $\beta_0 = (1, 1, 1)^T$, and δ^0 is taken to be 0 except for values near the asymptote which were initialized to the horizontal distance as described above. Again we observe that the values of $\beta(\sigma)$ do not vary quickly and that as σ increases the fits depend more and more on the points near the asymptote. Note that the location of the asymptote is well-determined by the data and that only β_1 changes much as σ increases. Here, as in Example 1, the insistence on near vertical measures of the error forces β_1 to assume smaller values which has the effect of flattening the function near the asymptote. This, of course, tends to minimize the vertical component of the error.

The nonuniqueness observed in Example 1 was again observed here. The details are not reported, but we found a second OLS solution which led to a trajectory of solutions that finally joined the above trajectory at $\sigma = 2$.

Example 3. The data here are actual measurements from a calibration run on an electronic device which was intended to give a flat response over a wide range of frequencies. In the (x, y)-data, the x-values are in units of frequency squared and the y-data are the gain. The x-values are scaled to the interval $(0, 1)$ with several measurements made in each decade from 10^{-8} to 1. More measurements were taken at the higher frequencies since most of the important information is obtained there. The data are plotted in Fig. 8 with a log scale on the x-axis in order to see the situation better. The y-scale on these graphs has been magnified to accentuate the differences. The peak at the right side of the data is at 1.001 while the low point at the extreme right end is at .9473. The flat part at the left side is all near .9882.

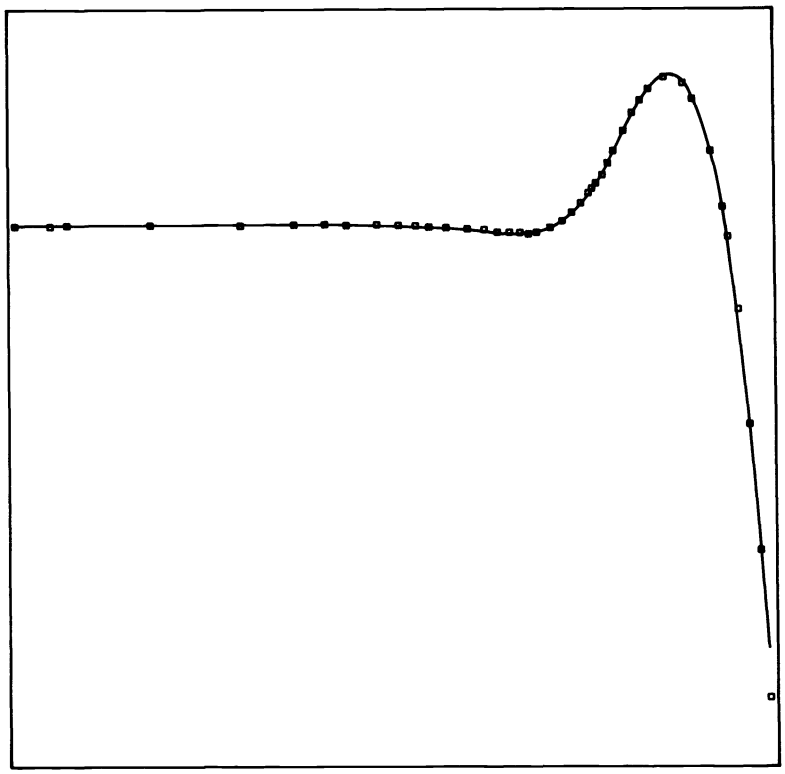


Figure 8
$\sigma = 0.01$

The model for this data was obtained from theoretical considerations and has the form

$$y_i = \sum_{j=1}^{4} \frac{\alpha_j}{x_i + \gamma_j} + \mu, \qquad i = 1, \cdots, 44$$

where the parameters to be determined are

$$\beta = (\alpha_1, \cdots, \alpha_4, \mu, \gamma_1, \cdots, \gamma_4)^T.$$

Initial estimates of the pole locations—the negative γ-values—were obtained quite accurately from other analyses. These γ-values are approximately

$$\gamma_1 = 1.38 \times 10^{-3}, \qquad \gamma_3 = 6.71 \times 10^{1},$$
$$\gamma_2 = 5.96 \times 10^{-2}, \qquad \gamma_4 = 1.07 \times 10^{9}.$$

Starting values for the α-values and μ were obtained by fixing the γ-values at their initial values and then solving for these linear terms using OLS. A feature of the ODR software allowed this step to be accomplished trivially. δ^0 was initialized to 0.

Since all of the poles are negative and all of the data have positive x-values, there is no problem with being close to the asymptotes. The range of the x-values, however,

TABLE 4

σ	Evaluations of G	Evaluations of G'	Final Value of $\|\|G_1(\eta(\sigma))\|\|$	Final Value of $\|\|G_2(\eta(\sigma))\|\|$
0.01	19	13	0.000505	0.000594
0.10	11	6	0.00170	0.000572
1.00	6	4	0.00190	0.000068
∞	4	2	0.00190	0.0

implies the need to scale the trust region. We used for the diagonal scaling matrices S and T the following:

$$S_i = \frac{1}{|(\beta^0)_i|}, \qquad T_i = \frac{1}{|x_i|}.$$

It turns out that the measurements are proportionately more accurate at the lower frequencies and we therefore took the d-weights to be the same as the t-weights.

While the data were measured quite accurately, there were simply no data at a sufficiently high frequency to warrent including the term corresponding to $j = 4$, and to allow γ_3 to be estimated. This situation was evidenced by the fact that the Jacobian J had five almost identical columns.

With these parameters removed, the resulting problem was easily solved. The correct σ-value is estimated to be 0.01 since the gain measurements in this data set were 100 times more accurate than the frequency measurements. Other values of σ were subsequently used for comparison. The results are in Table 4. Figure 8 depicts the ODR fit for $\sigma = .01$. Virtually no difference appears between the ODR and OLS fits at the lower frequencies, but some differences occur at the higher frequencies. In the enlargements (Figs. 9 and 10) one can easily see that the contribution of the

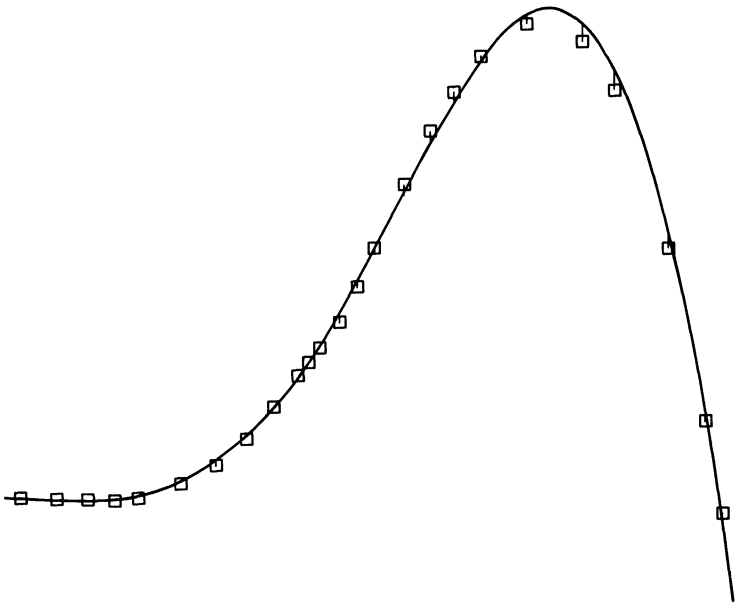


Figure 9
OLS

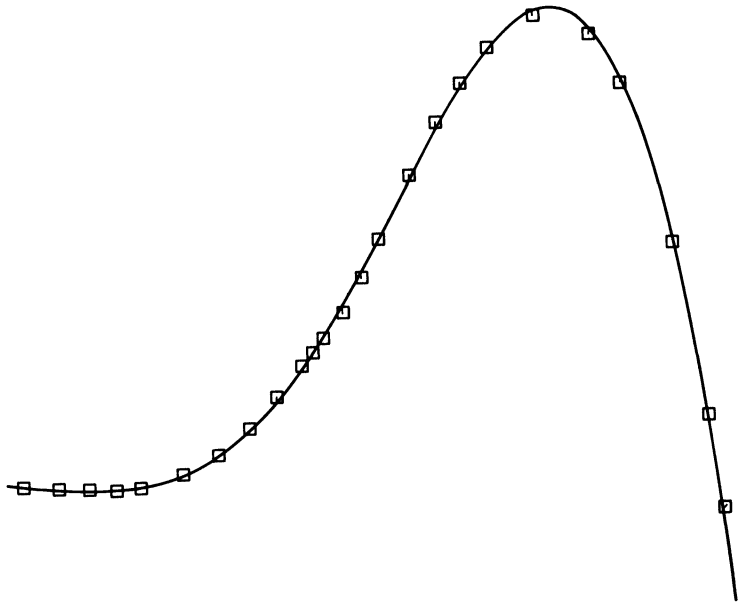


Figure 10
$\sigma = 0.01$

error in the x-values causes ODR to get a significantly better fit than OLS. While the β-values are not reported here, there were, again, very slow changes in $\beta(\sigma)$.

In this section we have shown that our algorithm is effective on highly nonlinear problems, but that these problems themselves often have multiple solutions and other difficulties which imply that potential solutions need to be studied carefully. In subsequent papers, we will provide a more complete description of our implementation and further results on its performance.

Acknowledgments. The authors wish to thank C. Spiegelman and R. Carroll (University of North Carolina) for many useful discussions of the statistical aspects of ODR; J. Donaldson for her help in enhancing the software and conducting the numerical tests; E. Bromberg and T. Griffin for their help in the graphics aspects of these experiments; and T. Griffin for his help in the writing of the data handling and driver routines. All of the above except Carroll are of the Center for Applied Mathematics, National Bureau of Standards.

REFERENCES

[Bar74] Y. BARD, *Nonlinear Parameter Estimation,* Academic Press, New York, 1974.

[BriL73] H. I. BRITT AND R. H. LUECKE, *The estimation of parameters in nonlinear, implicit models,* Technometrics, 15, (1973), pp. 233-247.

[ByrS87] R. H. BYRD AND R. B. SCHNABEL, *A unified local and global convergence analysis of Levenberg-Marquardt methods,* (in preparation).

[DenGW81] J. E. DENNIS, JR., D. M. GAY AND R. E. WELSCH, *An adaptive nonlinear least-squares algorithm,* TOMS, 7 (1981), pp. 348-368.

[DenS83] J. E. DENNIS, JR. AND R. B. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations,* Prentice-Hall, Englewood Cliffs, NJ, 1983.

[FiaM68] A. V. FIACCO AND G. P. MCCORMICK, *Nonlinear Programming: Sequential Unconstrained Minimization Techniques,* John Wiley, New York, 1968.

[Ful86] W. A. FULLER, *Measurement Error Models*, John Wiley, New York, 1986.

[Gay84] D. M. GAY, *Algorithm* 611. *Subroutines for unconstrained minimization using a model/trust-region approach*, TOMS, 9 (1983), pp. 503-524.

[GolV83] G. H. GOLUB AND C. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[Heb73] M. D. HEBDEN, *An algorithm for minimization using exact second derivatives*, Rept. TP515m, AERE, Harwell, England, 1973.

[KenSO83] M. G. KENDALL, A. STEWART AND J. K. ORD, *The Advanced Theory of Statistics*, Fourth Edition, MacMillan, New York, 1983.

[LakG84] M. LAKSHMINARAYANAN AND R. F. GUNST, *Estimation in linear structural relationships: sensitivity to the choice of the ratio of error variances*, Biometrika, 71 (1984), pp. 569-573.

[LawH74] C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[Lue73] D. G. LUENBERGER, *Introduction to Linear and Nonlinear Programming*, Addison-Wesley, Reading, MA, 1973.

[Mor71] T. A. P. MORAN, *Estimating structural and functional relationships*, J. Mult. Anal., 1 (1971), pp. 232-255.

[Mor77] J. J. MORÉ, *The Levenberg-Marquardt algorithm: implementation and theory*, in Numerical Analysis, G. A. Watson, ed., Lecture Notes in Mathematics 630, Springer-Verlag, Berlin, 1977, pp. 105-116.

[MorS81] J. J. MORÉ AND D. C. SORENSON, *Computing a trust region step*, this Journal, 4 (1981), pp. 553-572.

[OrtR70] J. M. ORTEGA AND W. C. RHEINBOLDT, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.

[PowM72] D. R. POWELL AND J. R. MACDONALD, *A rapidly convergent iterative method for the solution of the generalised nonlinear least squares problem*, Computer J., 15 (1972), pp. 148-155.

[Pow75] M. J. D. POWELL, *Converging properties of a class of minimization algorithms*, in Nonlinear Programming 2, O. Mangasarian, R. Meyer and S. Robinson, eds., Academic Press, New York, 1975, pp. 1-27.

[SchKW86] R. B. SCHNABEL, J. E. KOONTZ AND B. E. WEISS, *A modular system of algorithms for unconstrained minimization*, TOMS, (1986).

[SchSB85] G. A. SCHULTZ, R. B. SCHNABEL AND R. H. BYRD, *A family of trust-region algorithms for unconstrained minimization with strong global convergence properties*, SIAM J. Numer. Anal., 22 (1985), pp. 47-67.

[SchT85] H. SCHWETLICK AND V. TILLER, *Numerical methods for estimating parameters in nonlinear models with errors in the variables*, Technometrics, 27 (1985), pp. 17-24.

[Ste73] G. W. STEWART, III, *Introduction to Matrix Computations*, Academic Press, New York, 1973.

[Wat85] G. A. WATSON, *The solution of generalized least squares problems*, International Ser. Numer. Math., 75 (1985), pp. 388-400.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 6, November 1987
© 1987 Society for Industrial and Applied Mathematics

TIMELY COMMUNICATIONS

Under the "timely communications" policy for the SIAM Journal of 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.

THE USE OF BLAS3 IN LINEAR ALGEBRA ON A PARALLEL PROCESSOR WITH A HIERARCHICAL MEMORY*

KYLE GALLIVAN†, WILLIAM JALBY†‡ AND ULRIKE MEIER†§

Abstract. This note describes work at CSRD which shows that a third level of the BLAS (BLAS3) is needed to achieve high-performance on multivector processors with a shared hierarchical memory.

Key words. BLAS3 or third-level BLAS, numerical linear algebra, numerical software, parallel computing, cache management

AMS(MOS) subject classifications. Primary 68B99; secondary 65F05, 65F25, 65F30

1. Introduction. The design of efficient numerical linear algebra codes which are also reasonably portable for vector machines such as the CRAY or CYBER 205 has been simplified considerably by the use of standard kernel subroutines known as the BLAS [3]. These routines are primitives based on vector-vector operations such as dotproduct and DAXPY ($v = a + \gamma b$). The success of these routines depends upon the fact that many linear algebra algorithms are easily expressed in terms of vector-vector operations and that for high-performance pipelined architectures the optimization of performance is straightforward (the longer the vectors the better). Further consideration of algorithm performance has led to the development of the extended BLAS (called BLAS2 below) [4]. These routines are primitives based on matrix-vector operations. BLAS2 retains all of the positive aspects of the BLAS and possesses some unique advantages. On register-to-register vector machines the matrix-vector operations provide much more efficient register management and are able to more efficiently exploit memory bandwidth. Further, the presence of two-dimensional parallelism allows the BLAS2 primitives to be efficiently implemented on the newer supercomputers which provide concurrency as well as vectorization.

Unfortunately, most of the multivector processors make use of a hierarchical memory system, typically a small fast cache and a slower large main memory, to provide data at the required rates (ST-100, CRAY 2, ETA 10 and CEDAR). For these machines a third factor which influences algorithm performance is added to concurrency and vectorization. Algorithms must contain a reasonable amount of data locality in order to make full use of the resources of a multivector processor with a shared hierarchical memory. The data locality present in the BLAS and BLAS2 primitives and algorithms expressed in terms of these primitives is minimal. Hence, neither of

* Received by the editors October 20, 1986; accepted for publication January 21, 1987. This work was supported in part by the National Science Foundation under grants DCR84-10110 and DCR85-09970, the U.S. Department of Energy under grant US DOE DE-FG02-85ER25001, and the IBM Donation.

† Center for Supercomputing Research and Development, University of Illinois, Urbana, Illinois 61801.

‡ This author was on leave from Institut National de Recherche en Informatique et en Automatique, domaine de Voluceau, Rocquencourt, B.P.105 78150, Le Chesnay, France.

§ This author was on leave from ZAM, KFA Juelich GmBH, Postfach 1913, D-5170 Juelich, West Germany.

the first two levels of the BLAS is a suitable basis for algorithm development on such architectures.

Recent work at CSRD has demonstrated that a third level of the BLAS is needed in order to design algorithms which efficiently exploit the resources of a multivector processor with a hierarchical shared memory system [1], [5], [6]. This third level of the BLAS (called BLAS3 below) comprises primitives based on matrix-matrix operations. These primitives possess as much exploitable concurrency and vectorization as BLAS and BLAS2 primitives, if not more, and their data locality is far better. The numerical library under development for the CEDAR machine uses "block" methods implemented via BLAS3 primitives for many of the basic linear algebra tasks such as matrix factorization.

The effect of data locality, concurrency and vectorization on the performance of an algorithm can be investigated by writing the total time required to perform the algorithm as

$$T = T_a + T_l = n_a\tau_a + n_l\tau_l$$

where T_a, T_l, n_a and n_l are the arithmetic time (the time spent computing assuming all operands are in cache), data load time (the time spent transferring data from main memory to cache), number of operations and number of data loads, respectively. The quantities τ_a and τ_l defined by this expression can be viewed as the "average" times for arithmetic and data load operations. The component T_a is largely the result of satisfying the aspects of concurrency and vectorization while data locality dictates the size of T_l. The effect of data locality on performance can be investigated by expressing the relative amount of time spent loading data in terms of a cache-miss ratio and a cost ratio which are $\lambda = \tau_l/\tau_a$ and $\mu = n_l/n_a$, respectively. The relative amount of time spent loading data is then $T_l/T = \mu/(\lambda^{-1}+\mu)$. In [5] and [6] it is shown that by analyzing the data locality of a linear algebra algorithm via these two parameters the influence of the memory system can be reduced to a negligible amount for a large number of machines. The algorithm designer is then free to concentrate on the characteristics of his particular machine and their influence on the arithmetic time of the algorithm under consideration. For many algorithms this technique of optimizing the components T_a and T_l separately and then choosing parameters of the final algorithm from within the intersection of regions of near-optimal behavior for each of the terms yields remarkable results. This is the basic technique that has been used at CSRD in analyzing the performance of algorithms in the numerical library being developed for the CEDAR project.

This correspondence contains a brief survey of the results of some of the work done at CSRD which led to the recommendation of the use of the BLAS3 in the design of algorithms for the numerical library on the CEDAR machine and shared hierarchical memory machines in general. First, the results of the analysis of the basic BLAS3 primitive of $C \leftarrow C + AB$ are discussed in § 2 and experimental results clearly demonstrating the superiority of BLAS3-based algorithms are presented in § 3.

2. Analysis of a BLAS3 primitive. The first task which results from recommending the use of the BLAS3 primitives in algorithm design is that of generating efficient algorithms for the primitives themselves. In this section the results of analyzing the BLAS3 primitive $C \leftarrow C + AB$ presented in [5] are discussed. If A and B are vectors then this primitive is the rank-1 update of BLAS2. In BLAS3 usage, this primitive is usually a rank-k update but it can, of course, be used as a dense matrix multiplication primitive when A, B and C are all large. One of the most commonly used instances

of this primitive is the case where C is n_1 by n_3, A is n_1 by n_2 and B is n_2 by n_3. The values of n_1 and n_3 are assumed large while n_2 may or may not be. It is assumed that the multivector processor of interest contains p vector processors which share a small high-speed cache and a large slower main memory (for example an ALLIANT FX/8 which is used as a single CEDAR cluster).

Let A, B and C be partitioned into submatrices of dimension m_1 by m_2, m_2 by m_3 and m_1 by m_3, respectively. The goal of the analysis is to determine block sizes which deliver near-optimal performance of this BLAS3 primitive on a machine with the assumed architecture.

The algorithm for obtaining the matrix C is as follows:

```
do i = 1, k1
  do k = 1, k2
    do j = 1, k3
      C_ij = C_ij + A_ik * B_kj
end do
```

where $n_1 = k_1 m_1$, $n_2 = k_2 m_2$ and $n_3 = k_3 m_3$ with k_1, k_2 and k_3 are integers greater than 1. The block operations $C_{i,j} = C_{i,j} + A_{i,k} * B_{k,j}$ contain a reasonable amount of potential parallelism, so the algorithm proceeds by first partitioning the matrices and then dedicating the full resources of the machine to each of the block operations in turn.

The kernel algorithm for the block operation uses m_2-adic operations on each processor to accumulate p columns of C_{ij} at a time. (This should be a fairly portable approach since the multiply-add vector instruction tends to be the highest performance instruction on register-to-register vector processors.) Hence, the minimization of the T_a component of the algorithm time requires that: m_1 is a multiple of the length of the vector registers (32 for a CEDAR cluster); m_2 is large enough so that a significant fraction of the peak performance of the m_2-adic instruction is achieved (greater than 24 or so on a CEDAR cluster); and m_3 is either large or a multiple of p ($p = 8$ on a CEDAR cluster).

The minimization of the component T_l requires more effort. If it is assumed that A_{ik} is fetched from memory once and B_{kj} and C_{ij} are fetched as needed then the minimization of the number of loads is equivalent to minimizing $\rho = m_1^{-1} + m_2^{-1}$ subject to the constraints $m_2(p + m_1) \leqq CS$, $1 \leqq m_1 \leqq n_1$ and $1 \leqq m_2 \leqq n_2$ where CS denotes the cache size. These constraints trace a rectangle and a hyperbola in the parameter plane. In [5] it is shown that the solution to the minimization problem, under reasonable assumptions including $n_1 > \sqrt{CS}$, is divided into two cases:

(1) If $n_2(p + n_1) \leqq CS$ then $m_1 = n_1$ and $m_2 = n_2$. This yields the theoretical minimum for the number of loads for this primitive.

(2) If $n_2(p + n_1) \geqq CS$ then $m_2 = \min(n_2, CS[\sqrt{CS} + p]^{-1})$ and $m_1 = (CS/m_2) - p$.

When this primitive is used as a rank-n_2 update it is reasonable to assume that $n_2 \leqq \sqrt{CS}$ and that n_1 and n_3 are large enough for the second case above to be applicable. The cache-miss ratio for this primitive is $\mu \approx 1/2n_2$. This implies that the relative cost of data loading is $\lambda/(2n_2 + \lambda)$. Clearly for BLAS2 where $n_2 = 1$ the cost is too high. For a single CEDAR cluster it is reasonable to assume that $\lambda \approx 3$ implying that 60 percent of the total time of the algorithm is spent loading data for the BLAS2 primitive while if, say, rank-32 updates are used this falls to an acceptable 4.5 percent. The difference in the performance of the basic primitives in the three levels of the BLAS on a single CEDAR cluster is dramatic. For example, experiments run at CSRD show that DAXPY for vectors of length n and a rank-1 update to an n by n matrix run at about 5.5 and 6.5 Megaflops respectively for n ranging from 256 to 1024. A rank-32

update on an n by n matrix runs at approximately 42 Megaflops for the same range of n.

3. Matrix factorizations. Many classical linear algebra algorithms can be expressed in terms of BLAS2 and BLAS3 primitives [1], [5]. In this section the LU decomposition and the modified Gram–Schmidt algorithm are considered as examples of the superiority of BLAS3 over BLAS2 on multivector processors with a hierarchical memory. In the following subsections the block algorithms are described and experimental results obtained on a single CEDAR cluster are presented.

3.1. The LU decomposition. The goal of the LU decomposition is to factor an $n \times n$-matrix A into the product of a lower triangular matrix L and an upper triangular matrix U. The classical LU factorization consists essentially of dotproducts, i.e., the classical BLAS or BLAS1, it can however also be expressed in terms of rank-1 updates, i.e., BLAS2 (see [3]).

In order to use the efficient BLAS3, a block LU factorization must be performed. This algorithm decomposes A into the products of block lower triangular matrix L_k and an block upper triangular matrix U_k with blocks of the size $k \times k$ (it is assumed for simplicity that n is divisible by k). Let A be a diagonally dominant matrix partitioned in the following way:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} I & 0 \\ L_{21} & I \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & B \end{pmatrix}$$

where A_{11} is of order k.

The first step of the algorithm is

$$A_{11} \leftarrow A_{11}^{-1}, \qquad A_{21} \leftarrow L_{21} = A_{21}A_{11}, \qquad A_{22} \leftarrow B = A_{22} - L_{21}A_{12}.$$

The above computations are then performed recursively on the smaller matrix B. Note that the BLAS2 version of the classical LU decomposition is a special case of this algorithm for $k = 1$.

This block LU algorithm consists mainly of matrix-matrix operations. The only difficulty lies in inverting the $k \times k$ blocks on the diagonal. This can be done by using the Gauss–Jordan algorithm which can be implemented efficiently on a CEDAR cluster. Such an algorithm is numerically stable for the diagonally dominant case [7]. Pivoting can be added to the block algorithm in a straightforward manner while maintaining performance superior to that of a BLAS2 implementation with pivoting [2]. This is due to the fact that the number of data loads required is not significantly affected by the addition of pivoting [5].

The block algorithm increases the number of operations by a factor of approximately $(1+2k^2/n^2)$ over the classical LU factorization which requires about $2n^3/3$ operations.

3.2. A block Gram–Schmidt algorithm. The goal of this algorithm is to factor an $m \times n$-matrix A into the product of an orthonormal $m \times n$-matrix Q and an upper rectangular $n \times n$-matrix R where $m > n$ and A is of maximal rank. A is partitioned into two blocks A_1 and B where A_1 consists of s columns of order m, with Q and R partitioned accordingly.

$$(A_1, B) = (Q_1, P) \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}.$$

The first step of the algorithm comprises the computations

$$A_1 = Q_1 R_{11}, \qquad R_{12} = Q_1^T B, \qquad B \leftarrow B_1 = B - Q_1 R_{12}.$$

The above computations are then performed recursively on the smaller problem $B_1 = PR_{22}$, etc. It is easily seen that this algorithm consists mainly of matrix operations. The second and third statements require approximately $4ms(n-s)$ floating point operations while the first, the decomposition of A_1 into the orthonormal matrix Q_1 and the upper triangular matrix R_{11}, for which the modified Gram-Schmidt algorithm is used, requires only $2ms^2$. The BLAS2 version of the modified Gram-Schmidt algorithm is obtained by setting s to 1.

3.3. Experimental results. The classical method and the block variant of the LU factorization were implemented on a single CEDAR cluster using BLAS2 and BLAS3 primitives, respectively. Figure 1 illustrates the performance of both methods for systems of varying sizes. The block algorithm is about 3.5 times as fast as the classical LU factorization if $n > 500$.

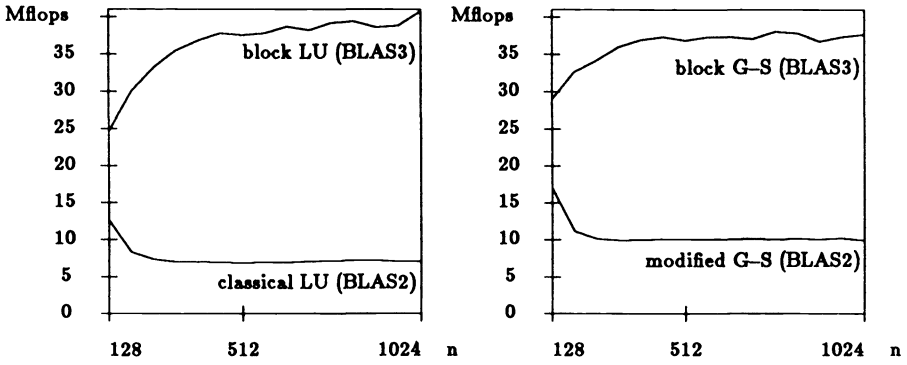


FIG. 1. *Performance results.*

Similar experiments were performed for the Gram-Schmidt orthogonal factorization algorithm. A BLAS2 version as well as a BLAS3 version of the algorithm with $s = 32$ have been implemented on a single CEDAR cluster. The resulting performance is shown in Fig. 1. Increasing the order of A obviously improves the performance for the BLAS3 version as the matrix-matrix operations dominate the computation.

4. Conclusion. This correspondence has surveyed recent work at CSRD which has shown that a third level of the BLAS (BLAS3) based on matrix-matrix primitives is required in order to achieve high performance on multivector processors with a shared memory hierarchy. In this work the relative cost of data loading from memory to cache was expressed in terms of two generic systems parameters, the cache-miss and cost ratios. This model provides an intuitive and theoretically sound basis for analyzing the influence of the memory system on the performance of algorithms implemented on such architectures.

A numerical library based on the BLAS3 is currently under development at CSRD. Numerical experiments based on these algorithms have clearly shown the superiority of designing with BLAS3 primitives on systems such as CEDAR. The definition of BLAS3 is evolving as this work progresses, but at present some of the primitives are: $C \leftarrow C \pm AB$; $C \leftarrow C \pm A^TB$; triangular solvers for matrix equations $LX = B$ and $UX = B$; an inversion primitive using Gauss-Jordan reduction; and, of course, matrix decomposition primitives such as $A = LU$.

REFERENCES

[1] M. BERRY, K. GALLIVAN, W. HARROD, W. JALBY, S. LO, U. MEIER, B. PHILLIPE AND A. SAMEH, *Parallel numerical algorithms on the* CEDAR *system*, CSRD Report, CSRD University of Illinois at Urbana-Champaign, Urbana, IL, 1986.

[2] D. CALAHAN, *Block-oriented, local-memory-based linear equation solution on the* CRAY-2: *uniprocessor algorithms, Proc.* ICPP 1986, IEEE Computer Society Press, Washington D.C., August 1986.

[3] J. DONGARRA, J. BUNCH, C. MOLER AND G. W. STEWART, LINPACK *User's Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979.

[4] J. DONGARRA, J. DUCROZ, S. HAMMARLING AND R. HANSON, *A proposal for an extended set of Fortran basic linear algebra subprograms*, Technical Memo #41, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, December 1984.

[5] K. GALLIVAN, W. JALBY, U. MEIER AND A. SAMEH, *The impact of hierarchical memory systems on linear algebra algorithm design*, CSRD Report, CSRD University of Illinois at Urbana-Champaign, Urbana, IL, 1986.

[6] W. JALBY AND U. MEIER, *Optimizing matrix operations on a parallel multiprocessor with a two-level memory hierarchy*, CSRD Report, CSRD University of Illinois at Urbana-Champaign, Urbana, IL, 1986.

[7] G. PETERS AND J. WILKINSON, *On the stability of Gauss-Jordan elimination with pivoting*, Comm. ACM, 18 (1975), pp. 20-24.

SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 6, November 1987
© 1987 Society for Industrial and Applied Mathematics
016

A NOTE ON SPARSE FACTORIZATION IN A PAGING ENVIRONMENT*

JOSEPH W. H. LIU†

Abstract. The impact of reordering on the Cholesky factorization of a sparse matrix in a paging environment is examined. We show experimentally that an equivalent reordering, if appropriately chosen, can reduce the CPU time and elapsed time for sparse factorization.

Key words. sparse matrix, Cholesky factorization, paging, equivalent reordering, elimination tree

AMS(MOS) subject classifications. 65F50, 65F25

1. Introduction. The solution of linear systems on virtual memory paging systems has been considered by Du Croz, Nugent, Reid and Taylor [1], Moler [5], and others. Perry [6] studies the case when the matrix is banded.

In this paper, we examine the Cholesky factorization of large sparse symmetric positive definite matrices in a paging environment. Let A be such a given n-by-n sparse matrix. We assume that the matrix has already been permuted by some fill-reducing ordering, such as the nested dissection or minimum degree ordering [2]. Consider an equivalent ordering P for A, that is, A and PAP^T are equivalent in terms of fills and arithmetic operations in their Cholesky factorization. We demonstrate that the factorization of A and PAP^T in a virtual memory paging environment may require drastically different amount of processor time. An equivalent reordering based on the so-called *elimination tree* of the sparse matrix is recommended. Experimental results are provided to illustrate reductions in factorization time for a paging environment when this equivalent ordering is used.

The readers are assumed to be familiar with the characteristics of paging systems. We briefly describe some of the relevant terminology used here. The data array is assumed to be partitioned into a number of *pages* of fixed size by the virtual memory system. Those pages that are resident in primary memory at a given time are referred to collectively as the *working set.* In the course of the computation, if a page referenced is not in the working set, a *page fault* occurs. An active page in the working set will be replaced by the page being referenced. The maximum number of pages allowed in primary memory is called the *working set size.* It is clear that the number of page faults depends on the working set size. For a fixed working set size, the paging cost depends on the way in which data is being stored and accessed in the (virtual) storage array.

2. Matrix reordering using elimination tree. For a given sparse matrix A, it is well known that there is a class of orderings equivalent in terms of fills and arithmetic operations for Cholesky factorization. The *elimination tree* structure [4], [7] can be used to define an important subclass of such equivalent orderings.

Let L be the Cholesky factor of A. The elimination tree for A has n nodes $\{1, 2, \cdots, n\}$, and node i is the parent node of j if and only if

$$i = \min \{k | l_{kj} \neq 0\}.$$

In general, this structure is a forest. If A is an irreducible matrix, the structure is a tree rooted at the node n. For the purpose of this paper, we assume that A is irreducible.

* Received by the editors December 8, 1986; accepted for publication February 24, 1987. This research was supported in part by the Natural Sciences and Engineering Research Council of Canada under grant A5509, by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy under contract DE-ACO5-84OR21400 with Martin Marietta Energy Systems Inc., and by the U.S. Air Force of Scientific Research under contract AFOSR-ISSA-84-00056.

† Department of Computer Science, York University, North York, Ontario, Canada M3J 1P3.

Consider a *topological ordering* on the elimination tree, that is, one that numbers nodes before their parent nodes. It is known that any such induced ordering of A is equivalent to the original one in terms of fills and operations in the sparse factorization [7]. On factorization in a paging system, we are interested in finding an equivalent ordering based on the elimination tree so that the correspondingly permuted matrix can be factored most effectively.

In this context, it is desirable to have columns of the factor matrix organized in such a way as to reduce the number of page faults. In the factorization algorithm, each column of the factor matrix is formed by a number of column modifications, governed by the locations of the nonzeros in its corresponding row. In [4], it is shown that the nonzeros in a row of the factor is a partial (or pruned) subtree of the elimination tree. Therefore, it is reasonable to order nodes in each subtree consecutively, so that columns for each subtree will be stored in contiguous (virtual) memory. A *postordering* on the elimination tree, which is known to be an equivalent reordering, is an appropriate choice.

3. Experimental results. The experimental results provided will be on the regular grid problem ordered by the popular minimum degree algorithm. Results on this example are quite representative of runs on other sparse matrix problems. The actual ordering used is the minimum degree implementation with multiple elimination in [3].

We shall provide results on the following three different topological orderings on the given elimination tree:

(a) a postordering of the tree, that is, nodes in each subtree renumbered consecutively,

(b) the original minimum degree ordering,

(c) a level-by-level reordering of the nodes from the last level up in the elimination tree.

The last ordering is included here as an equivalent reordering inappropriate for a paging system. To illustrate the three reorderings and their elimination trees, we present them in Fig. 3.1 for the 4-by-4 grid. In the trees, a double edge is used to indicate a chain of consecutively numbered nodes.

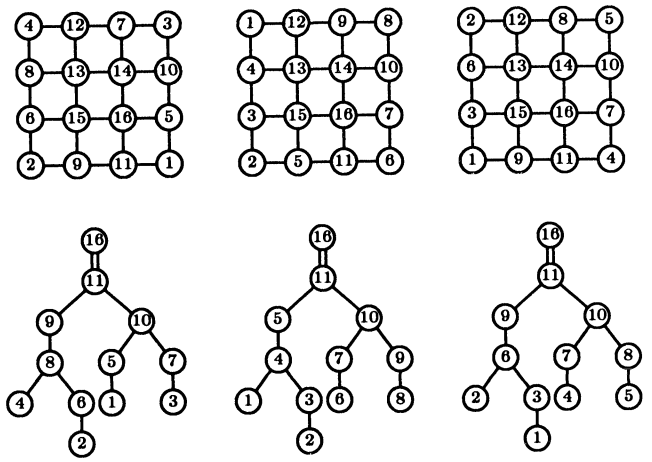


FIG. 3.1. *Three equivalent reorderings and their elimination trees on the* 4*-by-*4 *grid.*

Tables 3.1 and 3.2 contain results on the 180-by-180 grid. "MDeg" is the original minimum degree ordering; "+Post" corresponds to the same ordering followed by a postordering on the elimination tree; and "+Level" is the ordering with the level-by-level order. This matrix problem has 32,400 unknowns with 257,044 off-diagonal nonzeros. With the minimum degree ordering, the resulting Cholesky factor has 1,180,771 number of off-diagonal nonzeros. The sparse Cholesky factorization used is the routine from [2].

TABLE 3.1

Ordering, factorization and solve time on 180×180 *grid with different working set sizes.*

Work set size	Ordering time		Factorization CPU time			Solve CPU time		
	MDeg	+Post	MDeg	+Post	+Level	MDeg	+Post	+Level
512	17.22	21.45	280.79	264.34	553.04	10.38	10.22	11.52
1024	14.60	17.92	243.32	230.76	450.11	9.59	9.32	10.46
2048	11.68	13.79	216.73	209.09	338.45	9.35	9.20	10.17

TABLE 3.2

Factorization elapsed time and page faults on 180×180 *grid with different working set sizes.*

Work set size	Factorization elapsed time			Factorization page faults		
	MDeg	+Post	+Level	MDeg	+Post	+Level
512	303.91	287.64	576.21	530427	422595	2277654
1024	257.69	240.27	474.61	261632	192425	1647193
2048	225.69	218.34	352.57	63485	21300	859791

The experiments were performed on a VAX 8600 running VMS in a *single-user* mode. This is to ensure that the programs were run in the same system environment. In the tables, the "Working Set Size" is the maximum number of pages used in the working set. Each page on a VAX 8600 is a block of 512 contiguous byte locations.

The result shows clearly the importance of reordering. Although the level-by-level order suffers the same fill and requires the same amount of arithmetic operations, it takes much more CPU time and elapsed time for the factorization. This can be attributed solely to the drastic difference in the number of page faults. When the working set size is 2048, its page fault count is more than 40 times that of the postordering sequence. The difference would be even greater if the programs were executed in normal system load.

From this experiment, the original minimum degree ordering as obtained from [3] is quite suitable for paging systems. But the small investment to do a postordering still seems to be worthwhile. It always results in a net decrease in total ordering, factorization, and solve time.

4. Concluding remarks. For solving large sparse linear systems in a paging system, we have demonstrated the impact of reordering on factorization time. The reordering recommended is based on a postordering on the elimination tree associated with the

given sparse matrix. It is shown to be quite effective from the experimental results. On the other hand, using an inappropriate equivalent ordering can significantly degrade the overall factorization performance. From a theoretical viewpoint, it would be useful to introduce a measure of *locality* for a given sparse matrix based only on its nonzero structure. This would allow a comparison of different reorderings on their suitability for paging systems, without having to run the different algorithms on actual systems.

REFERENCES

[1] J. J. DU CROZ, S. M. NUGENT, J. K. REID AND D. B. TAYLOR, *Solving large full set of linear equations in a paged virtual store*, ACM Trans. Math. Software, 7 (1981), pp. 527-536.

[2] J. A. GEORGE AND J. W. H. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[3] J. W. H. LIU, *Modification of the minimum degree algorithm by multiple elimination*, ACM Trans. Math. Software, 11 (1985), pp. 141-153.

[4] ———, *A compact row storage scheme for Cholesky factors using elimination trees*, ACM Trans. Math. Software, 12 (1986), pp. 127-148.

[5] C. B. MOLER, *Matrix computations with Fortran and paging*, Comm. ACM, 15 (1972), pp. 268-270.

[6] J. R. PERRY, *Secondary storage methods for solving symmetric, positive definite, banded linear systems*, Research Report 201, Dept. of Computer Science, Yale University, New Haven, CT, 1981.

[7] R. SCHREIBER, *A new implementation of sparse Gaussian elimination*, ACM Trans. Math. Software, 8 (1982), pp. 256-276.