Special Issue on Iterative Methods in Numerical Linear Algebra

More than 180 mathematicians from all corners of the world attended the First Copper Mountain Conference Devoted to Iterative Methods. The meeting, held April 1–5, 1990, took place at the Copper Mountain Resort, which is located 70 miles west of Denver. During the four days of the meeting, over 100 talks on current research were presented. Topics included nonsymmetric systems, preconditioning strategies, parallel implementations, applications, software, multigrid methods, domain decomposition, eigenvalue problems, integral equations, nonlinear systems, indefinite problems, discretization techniques, complex matrix problems and common software standards.

There are two special issues devoted to chronicling the presentations made at the Copper Mountain Conference, one in *SISSC* and the other in the *SIAM Journal on Matrix Analysis and Applications* (*SIMAX*). The review process followed the normal SIAM policies for selecting referees and making recommendations. This issue represents a rich mix of papers on a wide variety of topics related to iterative methods. There are two aspects of this collection that we would like to underscore. First, much of the research represented in these articles was motivated or influenced by the need to develop new algorithms for the growing variety of parallel processing computers. Second, the increasing interaction between the multigrid community and the iterative method community is reflected in the many articles that incorporate multigrid and multilevel ideas into the construction of preconditioners and domain decomposition strategies. The papers in this issue are representative of the lively and stimulating interaction that occurred at the meeting.

A special effort was made to bring students to the meeting. The vehicle for this effort was a Student Paper Competition, in which students were asked to submit an original research paper consisting primarily of their own work. Out of ten submissions, three winners were selected. First place went to Barry Smith of the Courant Institute at New York University. Second place was awarded to Doug James of North Carolina State University. Third place honors were shared by Sverker Holmgren and Kurt Otto from Uppsala University in Stockholm, Sweden. Barry Smith's paper appears in this issue; the other two winning papers will appear in the special issue of *SIMAX*, to be published in July 1992.

We would like to thank the members of the program committee for their help in organizing the meeting. They are: Seymour Parter (chair), Loyce Adams, Steve Ashby, Howard Elman, Roland Freund, Gene Golub, Anne Greenbaum, David Kincaid, Steve McCormick, Ahmed Sameh, Paul Saylor, Olof Widlund, and David Young. In particular, we would like to give special thanks to Parter, Elman, Greenbaum, McCormick, Sameh, and Widlund, who, in addition to helping organize the meeting, acted as special *SISSC* editors for this issue.

Through their efforts, the articles contained in this special issue were carefully refereed and brought into print on schedule. We would also like to thank the following persons for their generous support of this meeting: Fred Howes of the Applied Mathematics Program of the National Science Foundation, Don Austin from the Applied Mathematical Sciences Program of the Department of Energy, Andy White from the Advanced Computing Laboratory at Los Alamos National Laboratories, and Bob Huddleston of the Computing Division of Lawrence Livermore National Laboratories. Without their help, this meeting could not have taken place.

As this issue goes to press, planning for the next Copper Mountain Conference on Iterative Methods is in its final stages. It will be held April 9–16, 1992, in Copper Mountain, Colorado. Plans again include special journal issues in *SISSC* and *SIMAX*. It is our hope that the lively interaction and the fine quality of presentations and papers that marked the first meeting can be duplicated at the upcoming meeting.

Thomas A. Manteuffel
Linda R. Petzold

Special Issue Editors

Howard Elman
Anne Greenbaum
Steve McCormick
Seymor Parter
Ahmed Sameh
Olof Widlund

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 1, pp. 1–29, January 1992

A COMPARISON OF ADAPTIVE CHEBYSHEV AND LEAST SQUARES POLYNOMIAL PRECONDITIONING FOR HERMITIAN POSITIVE DEFINITE LINEAR SYSTEMS*

STEVEN F. ASHBY†, THOMAS A. MANTEUFFEL‡, AND JAMES S. OTTO‡

Abstract. This paper explores the use of adaptive polynomial preconditioning for Hermitian positive definite linear systems, $Ax = b$. Such preconditioners are easy to employ and well suited to vector and/or parallel machines. After examining the role of polynomial preconditioning in conjugate gradient methods, the least squares and Chebyshev preconditioning polynomials are discussed. Eigenvalue distributions for which each is well suited are then determined. An *adaptive procedure* for dynamically computing the best Chebyshev polynomial preconditioner is also described. Finally, the effectiveness of adaptive polynomial preconditioning is demonstrated in a variety of numerical experiments on a Cray X-MP/48 and Alliant FX/8. The results suggest that relatively low degree (2–16) polynomials are usually best.

Key words. conjugate gradient methods, polynomial preconditioning, Chebyshev polynomial, least squares polynomial, adaptive procedure

AMS(MOS) subject classifications. 65F10, 41A10

1. Introduction.

This paper examines polynomial preconditioning for Hermitian positive definite (hpd) linear systems of equations, $Ax = b$. Such systems arise in many scientific applications. For example, the matrix resulting from the seven-point finite difference approximation to a three-dimensional self-adjoint elliptic PDE is large, sparse, and hpd. The conjugate gradient (CG) method of Hestenes and Stiefel [22] is a popular and effective solution technique for these linear systems, especially when combined with a preconditioner. The incomplete Cholesky factorization of Meijerink and van der Vorst [26] is often an effective preconditioner for CG, but other choices include Jacobi and SSOR. In this paper, we will consider *polynomial preconditioning* for conjugate gradient methods. That is, we will solve

$$C(A)Ax = C(A)b \tag{1.1}$$

where $C(\lambda)$ is a *preconditioning polynomial* and $C(A)$ is the associated polynomial preconditioner. We will assume that $C(\lambda)$ has real coefficients, in which case both $C(A)$ and $C(A)A$ are Hermitian.

Polynomial preconditioning has several advantages, the first of which is simplicity. Since there are only two intrinsic operations, matrix-vector multiplication (*matvec*) and vector addition (*saxpy*), it is easy to implement. The user need only specify the polynomial degree and initialize a few parameters. Consequently, it is ideally suited to "matrix-free" computations [8] (in which the matrix is known only implicitly). Polynomial preconditioning is also versatile. As discussed in [3], polynomial preconditioners may be used in a variety of CG methods. The best known of these is the

* Received by the editors July 26, 1990; accepted for publication (in revised form) March 7, 1991. This work was supported in part by National Science Foundation grant DMS-8704169 and by the Applied Mathematical Sciences subprogram of the Office of Energy Research, Department of Energy, by Lawrence Livermore National Laboratory under contract W-7405-ENG-48.

† Computing and Mathematics Research Division, Lawrence Livermore National Laboratory, P.O. Box 808, Livermore, California 94551.

‡ Computational Mathematics Group, University of Colorado, Campus Box 170, P.O. Box 173364, Denver, Colorado 80217.

PCG method of Concus, Golub, and O'Leary [13]. However, one may exploit the special properties of polynomial preconditioners to devise new CG methods [7]. The key here is commutativity: a polynomial in A commutes with A. In other words, the preconditioner C commutes with the matrix A, a property generally not shared by other preconditioners.

The main advantage of polynomial preconditioning is its suitability for vector and/or parallel architectures. If the matvec is vectorizable, as when A has a regular sparsity structure, polynomial preconditioning is effective on vector machines [3], [15], [24], [25]. In contrast, incomplete factorizations are difficult to vectorize, especially for the nonexpert. It is also possible to chain the matvecs implicit in the preconditioning, thereby enhancing data locality and reducing memory traffic [11], [12], [30], [31]. Polynomial preconditioning is also effective on parallel machines, especially those on which inner products are a bottleneck. This is so because polynomial preconditioned CG methods converge in fewer steps than unpreconditioned CG, and thus compute fewer inner products, albeit at the cost of several matvecs per step instead of one. However, in many applications the matvec is parallelizable, and so we can expect an overall reduction in CPU time on some architectures by substituting matvecs for inner products. The effectiveness of polynomial preconditioning has been demonstrated on an Alliant FX/8 [27] and on a Connection Machine [9].

A common complaint about polynomial preconditioning is that, unlike incomplete Cholesky (ICCG), it is only marginally better than unpreconditioned CG, which we will call CGHS. This criticism is somewhat misleading, however, because it is based on the number of iterations required for convergence rather than the CPU time. Although ICCG may take fewer iterations than polynomial preconditioned CG, the latter may take less time [15], [25]. Moreover, even when incomplete Cholesky is more effective, it can be further accelerated by using a polynomial preconditioner. Specifically, one applies CG to

$$C(M^{-1}A)M^{-1}Ax = C(M^{-1}A)M^{-1}b \tag{1.2}$$

where M is the matrix representation of the incomplete factorization. If M and A are Hermitian, then so is the preconditioner $C(M^{-1}A)M^{-1}$, and several CG methods are applicable [1], [2], [7]. Finally, we remark that polynomial preconditioning can be implemented automatically, and so it is as easy to use as CGHS.

1.1. Outline of paper.

In the next section we review preconditioned CG methods and discuss the various ways in which a polynomial preconditioner can be used. In § 3 we examine polynomial preconditioning. In particular, we discuss the least squares and Chebyshev preconditioning polynomials, study them in the context of CG methods, and show that the latter minimizes a bound on the condition number of the preconditioned matrix. We compare the two polynomials in § 4. In a variety of numerical experiments we determine those eigenvalue distributions for which each is well suited. In § 5 we describe an *adaptive procedure* for dynamically computing the extreme eigenvalues of A, which are used to determine the optimum endpoints on which to base the Chebyshev preconditioning polynomial. We also present numerical results demonstrating the accuracy, efficiency, and competitiveness of the resulting adaptive algorithm. Finally, in § 6, we summarize some numerical experiments that demonstrate the effectiveness of polynomial preconditioning on a variety of test problems. Our results suggest that relatively low degree (2–16) polynomials are usually best.

2. Preconditioned CG methods. In this section we examine the use of polynomial preconditioners in CG methods. To do this it is useful to first characterize CG methods. In [7], it is shown that any CG method is characterized by three matrices: an hpd inner product matrix B, a left preconditioning matrix C, and the original system matrix A. The resulting method, $CG(B, C, A)$, minimizes $||e_i||_B = \langle Be_i, e_i\rangle^{1/2}$ over $V_i(CA, Cr_0)$, where

$$V_i(CA, Cr_0) = \text{span}\{Cr_0, (CA)Cr_0, (CA)^2Cr_0, ..., (CA)^{i-1}Cr_0\} \tag{2.1}$$

is a Krylov subspace of dimension at most i, e_i is the error in the current iterate, r_0 is the initial residual, and $\langle\cdot,\cdot\rangle$ denotes the usual Euclidean inner product. By specifying the inner product matrix B, we obtain a particular CG method. For example, when A is hpd, one may take $B = A$ and $C = I$, which yields CGHS, the original method of Hestenes and Stiefel [22]. If one takes $B = A^2$ and $C = I$, the conjugate residual (CR) method results. Of course, one must chose B and C so that the method is computable [7].

The most robust implementation of a CG method is the so-called Odir algorithm [33], in which a three-term recursion is used to generate a set of direction vectors that form a B-orthogonal basis for V_i. This algorithm converges to the solution of $Ax = b$ whenever BCA is Hermitian. (For necessary and sufficient conditions, see [7] and [16].) If BCA is hpd, the cheaper and more familiar Omin algorithm [33] is applicable. Instead of using a three-term recursion for the direction vectors, Omin uses a two-term recursion involving the preconditioned residuals. Unfortunately, Omin may "stall" when BCA is indefinite, in which case the more expensive Odir, or an Odir/Omin hybrid, algorithm should be used [7], [10].

When A is hpd and $C = C(A)$, there are several choices for the inner product matrix B. A few of the resulting CG methods are listed in Table 2.1. (See [3] and [7] for a thorough discussion.) Notice that BCA is Hermitian in all cases, and so Odir converges. The Odir restrictions in Table 2.1 are sufficient to insure that B is hpd. The Omin restrictions are sufficient to guarantee that both B and BCA are hpd. The first method is PCG. Like CGHS, it minimizes the A-norm of the error, but does so over a preconditioned Krylov subspace. Although the matrix A must be hpd to define a norm, the preconditioner $C(A)$ only needs to be Hermitian for Odir. If $C(A)$ is hpd, one may use the more efficient Omin algorithm. The method GCGHS, which is CGHS on CA, minimizes in the $B = C(A)A$ norm, and so $C(\lambda)$ must be chosen so that the preconditioned matrix is hpd. As we will see, this is possible. The advantage of GCGHS is this: if $C(A)$ is a good preconditioner, then $C(A)A \approx I$, and so the method more nearly minimizes the Euclidean norm of the error. The next method, PCR, requires that $C(A)$ be hpd, in which case $C(A)A$ is hpd because A is hpd. The last two methods, GPCR and GCR, employ $B = A^2$ and $B = (C(A)A)^2$, respectively. The Odir algorithm will converge for either method; Omin is applicable if $C(A)A$ is hpd. Note that GPCR is possible because $CA = AC$ (which implies that BCA is Hermitian), an advantage of C being a polynomial in A. The last method, GCR, is simply CR applied to the preconditioned matrix, CA. We remark that each method except PCG is applicable to Hermitian indefinite A [6].

Finally, we note that the spectral and B condition numbers of CA are identical for each of the methods in Table 2.1. That is, $\kappa_I(CA) = \kappa_B(CA)$, where $\kappa_B(G) = ||G||_B||G^{-1}||_B$. Thus, estimates for the extreme eigenvalues of CA yield a bound on $\kappa_I(CA)$, which may be used to implement a stopping criterion based on the true

TABLE 2.1
Polynomial preconditioned CG methods for hpd A.

Method	B	CA	Odir Restrictions	Omin Restrictions
PCG	A	$C(A)A$	A hpd	A hpd, $C(A)$ hpd
GCGHS	$C(A)A$	$C(A)A$	$C(A)A$ hpd	C(A)A hpd
PCR	$AC(A)A$	$C(A)A$	A herm, $C(A)$ hpd	A hpd, $C(A)$ hpd
GPCR	A^2	$C(A)A$	A herm	C(A)A hpd
GCR	$(C(A)A)^2$	$C(A)A$	A herm	C(A)A hpd

error, rather than the more usual residual error. Eigenvalue estimates for CA are easily obtained from the CG iteration parameters [7], [13]. This is also the basis for the adaptive procedure discussed in § 5.

3. Polynomial preconditioning.

In this section we examine several choices for $C(\lambda)$. We wish to choose C to accelerate convergence of the CG iteration. One usually chooses C to approximate A^{-1} in some sense, for example, by choosing $C(\lambda) \approx \lambda^{-1}$. Of course, there are several ways of doing this. As we will see, there is no single "best" polynomial; the proper choice of $C(\lambda)$ depends on the eigenvalue distribution of A, which is seldom known a priori.

A simple choice for $C(\lambda)$ is based on the Neumann series. Let $A = M - N$ and consider

$$A^{-1} = (M-N)^{-1} = (I + G + G^2 + G^3 + \cdots)M^{-1} \tag{3.1}$$

where $G = M^{-1}N$. If the spectral radius of G is less than one, the series converges. We obtain our polynomial approximation to A^{-1} by truncating the Neumann series [2], [9], [15], [25]. The advantage of this polynomial is its simplicity: there are no parameters to estimate. Unfortunately, it may yield a poor preconditioner. If one desires a polynomial preconditioner of degree $m-1$, one can do much better than the Neumann series polynomial. For example, Jordan [25] has shown that the Chebyshev polynomial (§ 3.2) is superior. Experiments also suggest that the optimum degree for the Neumann series polynomial is two [2], [15], [25], whereas the optimum Chebyshev or least squares polynomial degree is often higher [3], [27], [30].

To obtain a better preconditioner, recall that $C(A)$ should approximate A^{-1} in some sense. That is, $C(\lambda)$ should be the "best" polynomial approximation to λ^{-1} on some set S containing the spectrum of A, $\sigma(A)$. Since A is hpd, we will take $S = [c, d]$, where $0 \leq c \leq d$. Ideally, $c = \lambda_c$ and $d = \lambda_d$, the smallest and largest eigenvalues of A. We next define the "best" polynomial to be that one which solves the following approximation problem:

$$\min_{C \in \pi_{m-1}} \|1 - C(\lambda)\lambda\| \tag{3.2}$$

where π_{m-1} is the set of polynomials of degree at most $m-1$. All that remains is to specify the norm.

3.1. The least squares polynomial preconditioner.

Let us define the inner product

$$\langle f, g \rangle = \int_c^d f(\lambda)\overline{g(\lambda)}\omega(\lambda)d\lambda \tag{3.3}$$

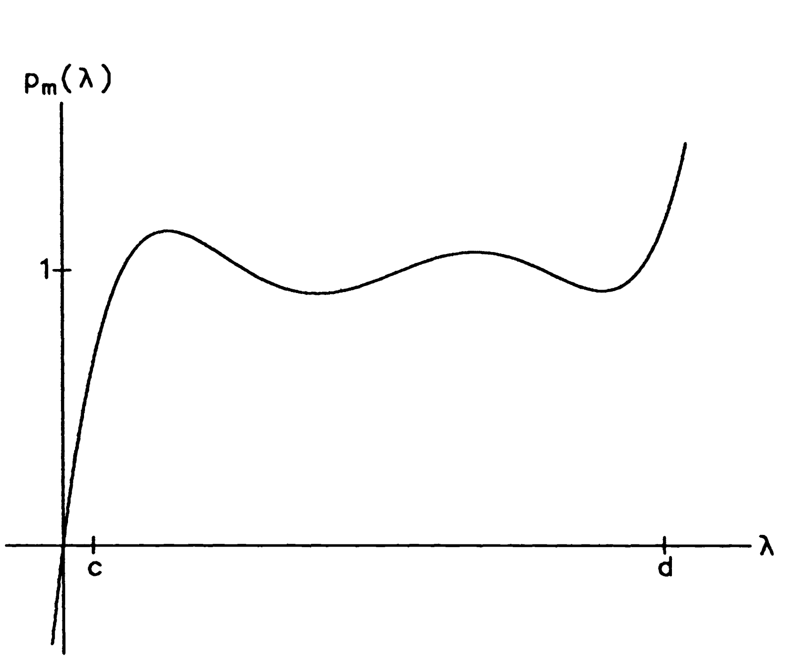


FIG. 3.1. *Least squares preconditioned polynomial* $(m = 5)$ *for* $S = [0, 20]$.

where $\omega(\lambda)$ is a positive weight function on $S = [c, d]$. It induces the following norm:

$$\|f\|_\omega^2 = \int_c^d |f(\lambda)|^2 \omega(\lambda) d\lambda. \tag{3.4}$$

The solution to (3.2) in this norm is called the weighted *least squares* polynomial. The associated *preconditioned* polynomial, $p_m(\lambda) = C(\lambda)\lambda$, is illustrated in Fig. 3.1. (We call $p_m(\lambda)$ the preconditioned polynomial because $p_m(A)$ is the preconditioned matrix.)

The related *residual* polynomials, $r_m(\lambda) = 1 - C(\lambda)\lambda$, are orthogonal with respect to the weight function $\lambda\omega(\lambda)$. This is so because the error in (3.2) is orthogonal to $\lambda\pi_{m-1}$, the subspace from which the best polynomial approximation to 1 is taken. Consequently,

$$\int_c^d r_m(\lambda) r_j(\lambda) \lambda \omega(\lambda) d\lambda = \int_c^d r_m(\lambda) q(\lambda) \omega(\lambda) d\lambda = 0, \qquad j \leq m - 1,$$

where $q(\lambda) = \lambda r_j(\lambda) \in \lambda\pi_{m-1}$. Thus, the least squares residual polynomials are orthogonal in the λ-inner product, $\langle f, g\rangle_\lambda = \langle \lambda f, g\rangle$, and hence satisfy a three-term recursion, which is computationally convenient (§ 3.3). See also [24] and [30].

Unlike the Chebyshev polynomial described below, the least squares polynomial is biased in its suppression of the eigenvalues of A. For example, when $\omega \equiv 1$, the eigenvalues of larger modulus are mapped closer to 1 than those of smaller modulus. If the eigenvalue distribution of A were known, one could choose ω to exploit this bias. In particular, one might consider a *Jacobi* weight function,

$$\omega(\alpha, \beta; \lambda) = (d - \lambda)^\alpha (\lambda - c)^\beta, \qquad \alpha, \beta > -1. \tag{3.5}$$

By appropriately choosing α and β, one portion of $\sigma(A)$ could be emphasized over another.

Saad [30] has noted that the least squares polynomial is relatively insensitive to c, and so one may take $c = 0$. Then, if a Jacobi weight function is used, the least squares polynomial is given by

$$C(\lambda)\lambda = 1 - \frac{P_m(\frac{d+2\lambda}{d})}{P_m(1)} \tag{3.6}$$

where $P_m(x)$ is the Jacobi polynomial [14] on $[-1,1]$ with respect to $\omega(\alpha+1, \beta; \lambda)$. This follows from the λ-orthogonality of the residual polynomials. If it is known that $\sigma(A)$ is clustered near some point $\lambda_0 \in [0,d]$, one might wish to choose α and β with the following result in mind.

THEOREM 3.1. *Let $r(\lambda)$ be the least squares residual polynomial for $[0,d]$ with respect to $\omega(\alpha,\beta;\lambda)$, where $(\alpha,\beta) \in W_1 = \{(\alpha,\beta) : \alpha > -1,\ \beta \geq -\frac{1}{2}\}$. Then the relative extrema of $r(\lambda)$ monotonically decrease in magnitude on $[0,\lambda_0]$ and monotonically increase in magnitude on $[\lambda_0, d]$, where*

$$\lambda_0 = \frac{d}{2}\left(1 - \frac{\beta-\alpha-1}{\beta+\alpha+2}\right) = \frac{d}{2}\left(\frac{2\alpha+3}{\beta+\alpha+2}\right). \tag{3.7}$$

Proof. The result follows from (3.6) and Theorem 7.32.1 of [32]. □

Of course, one should choose the weight function ω so that $p_m(\lambda) = C(\lambda)\lambda$ is positive on $[\lambda_c, \lambda_d]$. This guarantees that $p_m(A)$ is hpd, which makes practicable the Omin implementation of each method in Table 2.1. (Observe that PCG and PCR are applicable because $C(A)$ is hpd whenever $p_m(A) = C(A)A$ is hpd.) We note that $p_m(\lambda)$ may be indefinite if $\omega(\lambda)$ is biased toward the leftmost portion of $[c,d]$. However, if one employs a Jacobi weight function on $[c,d]$, one may show [24, p. 369] that $p_m(\lambda) > 0$ on $[c,d]$ if $(\alpha,\beta) \in W_2 = \{(\alpha,\beta) : \alpha \geq \beta \geq -\frac{1}{2}\}$, which includes the Legendre weight function $\omega \equiv 1$ ($\alpha = \beta = 0$). Moreover, if $c = 0$ and $\alpha = \beta$ in W_2, then $\lambda_0 = \lambda_0(\alpha) : [-\frac{1}{2}, \infty) \rightarrow (d/2, d]$. Thus, if $\sigma(A)$ is clustered near $\lambda_0 \in (d/2, d]$, one might employ the weight function $\omega(\alpha_0, \alpha_0; \lambda)$, where $\alpha_0 = -(\lambda_0 - 3d/4)/(\lambda_0 - d/2)$; cf. Theorem 3.1. Finally, if $(\alpha,\beta) \in W_3 = \{(\alpha,\beta) : \alpha > -1,\ -1 < \beta \leq -\frac{1}{2}\}$, it follows as in Theorem 3.1 that the relative extrema of the least squares polynomial decrease in magnitude on $(0,d]$. This property, which is in stark contrast to the equioscillation property of the Chebyshev preconditioned polynomial (see below), may be used to bias the preconditioner toward the large eigenvalues of A. This property also insures that $p_m(\lambda)$ is positive on $(0,d]$.

Because of its bias, the least squares polynomial yields an effective preconditioner in many cases [24], [30]. Since one may take $c = 0$, there is no need to estimate the smallest eigenvalue of A. The right endpoint is often taken to be the Gershgorin estimate for λ_d. However, the least squares polynomial is not always best. For example, if $\sigma(A)$ is dense near the origin, one should instead use the Chebyshev polynomial preconditioner, which we describe next.

3.2. The Chebyshev polynomial preconditioner.

Another interesting norm is the uniform norm:

$$\|f\|_\infty = \max_{\lambda \in S} |f(\lambda)|. \tag{3.8}$$

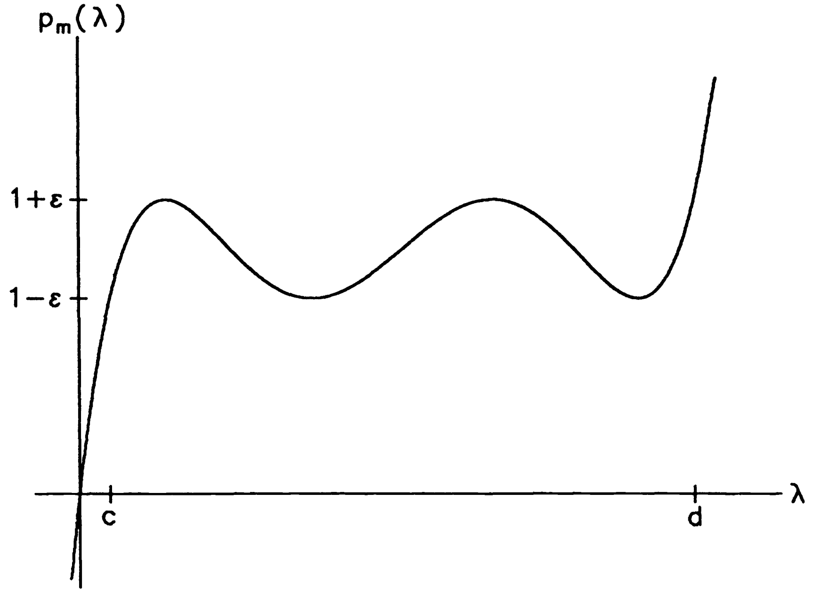


FIG. 3.2. *Chebyshev preconditioned polynomial* ($m = 5$) *for* $S = [1, 20]$.

The solution to (3.2) in this norm is obtained from a shifted and scaled Chebyshev polynomial:

$$C(\lambda)\lambda = 1 - \frac{T_m(\frac{d+c-2\lambda}{d-c})}{T_m(\frac{d+c}{d-c})} \tag{3.9}$$

where $T_m(x)$ is the mth Chebyshev polynomial of the first kind [28]. It is attractive for several reasons. First, like the least squares polynomial, it may be computed from a three-term recursion, which is computationally convenient. Second, since this polynomial is explicitly known, it is much easier to devise an adaptive procedure for dynamically computing the optimal endpoints c and d. Finally, this polynomial is unbiased in its suppression of those eigenvectors constituting the error. In other words, the Chebyshev preconditioning polynomial is well suited to those matrices whose eigenvalues are densely and nearly uniformly distributed throughout the interval $S = [c, d]$. See § 4.

This last fact follows from the Chebyshev minimax property, which states that the preconditioned polynomial, $p_m(\lambda) = C(\lambda)\lambda$, equioscillates about 1; see Fig. 3.2. This equioscillation property has several other implications. For example, if $\sigma(A) \subset [c, d]$, then $\sigma(p_m(A)) \subset [1-\epsilon_m, 1+\epsilon_m]$, where $\epsilon_m = \|1-p_m\|_\infty = |T_m^{-1}(\frac{d+c}{d-c})|$. Since $\epsilon_m < 1$, the preconditioned matrix, $p_m(A)$, is hpd. One may therefore apply CGHS to $p_m(A)$, yielding the method we call GCGHS. PCG is also applicable because $C(A)$ is hpd. Note that the spectral condition number of $p_m(A)$, $\kappa(p_m(A))$, satisfies

$$\kappa(p_m(A)) \leq \frac{1+\epsilon_m}{1-\epsilon_m} \tag{3.10}$$

when $\sigma(A) \subset [c,d]$. Since ϵ_m is a monotonically decreasing function of m, this bound may be made as small as desired by taking m large enough. Specifically, if

$$m \geq \frac{\cosh^{-1}(\frac{\delta+1}{\delta-1})}{\cosh^{-1}(\frac{\kappa(A)+1}{\kappa(A)-1})}, \tag{3.11}$$

then $\kappa(p_m(A)) < \delta$ for any $\delta > 1$. This follows from the definition of $T_m(\lambda)$ for $\lambda > 1$.

The bound (3.10) yields an estimate of the number of CG steps required for convergence. One needs approximately

$$\frac{\ln(\delta/2)}{\ln(cf)} \tag{3.12}$$

steps to reduce the error by an amount δ [21], where

$$cf = cf(p_m(A)) = \frac{\sqrt{\kappa(p_m(A))} - 1}{\sqrt{\kappa(p_m(A))} + 1} \tag{3.13}$$

is the *CG convergence factor* for the hpd matrix $p_m(A)$. If the eigenvalues of $p_m(A)$ are uniformly distributed throughout $[1-\epsilon_m, 1+\epsilon_m]$, then (3.12) is fairly accurate. Since $\kappa(p_m(A)) \leq \kappa(A)$ for $m > 1$, a Chebyshev polynomial preconditioned CG method will usually converge in fewer iterations than the unpreconditioned CGHS method. Of course, each iteration is more expensive, requiring m matvecs instead of one. We remark that $\kappa(p_m(A))$ is minimized when $c = \lambda_c$ and $d = \lambda_d$.

The Chebyshev polynomial preconditioner is also optimum in that it minimizes a bound on $\kappa(C(A)A)$. This is a consequence of the following result.

THEOREM 3.2. *A solution to*

$$\min_{C \in \pi_{m-1}} \frac{\max_{\lambda \in S} |C(\lambda)\lambda|}{\min_{\lambda \in S} |C(\lambda)\lambda|} \tag{3.14}$$

is given by the Chebyshev preconditioning polynomial.

Proof. First observe that (3.14) does not possess a unique solution. In particular, if Q solves (3.14), then so does γQ, where γ is any nonzero constant. We may assume $C(\lambda)\lambda > 0$ for $\lambda \in S$ without loss of generality. (If $C(\lambda)\lambda = 0$ for some $\lambda \in S$, then (3.14) is unbounded.) Thus, we may restrict ourselves to those polynomials $C(\lambda)$ for which

$$1 - \min_{\lambda \in S} (C(\lambda)\lambda) = \max_{\lambda \in S} (C(\lambda)\lambda) - 1 = \epsilon(C) = \epsilon. \tag{3.15}$$

The problem (3.14) is now equivalent to minimizing $\frac{1+\epsilon}{1-\epsilon}$. This is, in turn, equivalent to solving (3.2) in the uniform norm. □

Remark. If p_m is the Chebyshev preconditioned polynomial for S and $\sigma(A) \subset S$, the ratio in (3.14) gives a bound on the condition number of $p_m(A)$. Moreover, this bound is minimized with respect to S when $S = [\lambda_c, \lambda_d]$. The bound is attained when $S = [\lambda_c, \lambda_d]$ and m is odd.

This theorem is similar to Theorem 3 in [24], but our proof is different. It shows the equivalence of the minimax approximation problem (3.2) and the minimization problem (3.14). We also remark that Rutishauser [29] was the first to propose Chebyshev polynomial preconditioning for CGHS; his motive was to mitigate its rounding errors. We advocate polynomial preconditioning because it is well suited to vector and/or parallel architectures.

3.3. Implementation. To implement least squares or Chebyshev polynomial preconditioning, one neither explicitly forms the powers of A nor determines the coefficients of $C(\lambda)$. Instead, one may exploit the three-term recursion underlying the least squares and Chebyshev residual polynomials to effect the polynomial preconditioning. To see how, recall that both polynomials satisfy a recursion of the form

$$Q_{k+1}(\lambda) = (\phi_k \lambda + \psi_k) Q_k(\lambda) - \xi_k Q_{k-1}(\lambda) \tag{3.16}$$

where Q_k is the least squares or Chebyshev residual polynomial; cf. (3.6) and (3.9). The coefficients ϕ_k, ψ_k, and ξ_k may be generated recursively for the Jacobi-weighted least squares and Chebyshev polynomials. We may use (3.16) to formulate a second order Richardson iteration [21]:

$$r_0 = b - Ax_0 \tag{3.17}$$

$$\Delta_0 = -\phi_0 r_0 \tag{3.18}$$

$$x_{i+1} = x_i + \Delta_i \tag{3.19}$$

$$r_{i+1} = b - Ax_{i+1} \tag{3.20}$$

$$\Delta_{i+1} = \xi_{i+1}\Delta_i - \phi_{i+1} r_{i+1}. \tag{3.21}$$

Thus, to implement the polynomial preconditioning, one simply executes m steps of this nonstationary two-step iteration. (In the case of Chebyshev polynomial preconditioning, one uses the Chebyshev iteration [21].) Specifically, one applies the iteration to the linear system $Aw = v$ with $w_0 = 0$, where v is the vector to be preconditioned, usually the residual. One may show [4] that $w_m = C(A)v$. Note that we need only $m - 1$ matrix-vector multiplications because the final residual need not be computed. We also remark that the three-term recursion underlying the least squares and Chebyshev polynomials insures the stable evaluation of the preconditioning polynomial $C(\lambda)$. See also [24] and [30].

4. Chebyshev versus least squares. In this section we compare the least squares and Chebyshev preconditioning polynomials in a variety of numerical experiments. We will qualitatively describe those matrices for which the least squares polynomial yields a better preconditioner than the exact Chebyshev preconditioner. (The exact Chebyshev polynomial is the one based on $[\lambda_c, \lambda_d]$; recall that this polynomial minimizes a bound on the condition number of $C(A)A$.) Moreover, we will explain why this is so. The importance of the stopping criterion will also be discussed.

Let us begin by dispelling a common misconception: the least squares polynomial is *not* universally superior to the exact Chebyshev polynomial. This follows from a result of Greenbaum [20], who established a partial ordering on preconditioners. In brief, her result implies that the least squares preconditioner cannot be best for every initial guess, x_0. However, it is better in certain cases. For although the exact Chebyshev polynomial minimizes the condition number of $p_m(A)$, this does not alone determine the rate of convergence of the preconditioned CG method. The eigenvalue distribution of $p_m(A)$ is also important. Because of its equioscillation property, the Chebyshev polynomial tends to map $[c, d]$ uniformly into $[1-\epsilon_m, 1+\epsilon_m]$, obliterating any favorable clustering of the eigenvalues of A. The unweighted least squares polynomial ($\omega \equiv 1$) tends to map the larger eigenvalues of A most closely about 1, giving less weight to the smaller eigenvalues. This is due to the tendency of its relative extrema to decrease in magnitude on S. (This property can be made strict by an appropriate choice of weight function; see § 3.1.) Thus, if there are relatively few eigenvalues of A near c, these will become isolated eigenvalues of the least squares preconditioned matrix, the majority of whose eigenvalues will be clustered about 1. On the other hand, if the eigenvalues of A are dense near c, there will be no such clustering of eigenvalues. The proper choice of polynomial therefore depends on the spectrum of A. We will now explore this question numerically.

In the experiments below, the test matrices are diagonal with $N = 100,000$ eigenvalues between $\lambda_c = \delta$ and $\lambda_d = 1 + \delta$. The true solution is the vector having 1 in each of its components and $x_0 = 0$. Three eigenvalue distributions are considered. In the first, $\lambda_k = \delta + 1 - 1/k$, $k = 2, \ldots, N-1$, and so the eigenvalues are dense near the right endpoint d, such as with integral operators of the second kind. In the second, $\lambda_k = \delta + 1/(N - k + 1)$, and so the eigenvalues are dense near the left endpoint c, as with integral operators of the first kind. In the third, the eigenvalues are uniformly distributed, as is roughly the case for differential operators. In Figs. 4.1–4.4, we plot the PCG relative error, $\log_{10}(\|e_i\|_2/\|e_0\|_2)$, against i for three polynomials of degree $m = 9$. (Recall that PCG refers to the preconditioned CG method of Concus, Golub, and O'Leary [13].) The first is a least squares polynomial (with Legendre weight $\omega \equiv 1$) based on $[0, 1 + \delta]$; the second is the exact Chebyshev polynomial based on $[\delta, 1+\delta]$; and the third is a Chebyshev polynomial based on $[\zeta, 1+\delta]$, where ζ is chosen so that the related least squares and Chebyshev residual polynomials have the same first root. By choosing ζ in such a manner, we force this LS-Chebyshev polynomial to mimic the behavior of the least squares polynomial on $(0, \zeta)$. Consequently, the two preconditioning polynomials behave alike for matrices with eigenvalues in $(0, \zeta)$.

In Fig. 4.1, we have the dense-right eigenvalue distribution with $\delta = 10^{-3}$. Since the least squares polynomial is small on the large eigenvalues of A, the least squares PCG method converges much more rapidly than the exact Chebyshev PCG method. In Fig. 4.2, the eigenvalues are dense near the left endpoint, and the exact Chebyshev PCG method converges faster. Similar results were observed for other values of δ. In Figs. 4.3–4.4, we have the uniform eigenvalue distribution. When $\delta = 10^{-3}$, the gap between successive eigenvalues is $1/N$, which is smaller than λ_c, and the exact Chebyshev PCG method converges fastest. However, if $\delta = 10^{-5}$, the gap between successive eigenvalues is larger than λ_c, and the least squares PCG method converges faster. We have seen similar behavior in several other experiments. In short, the exact Chebyshev polynomial appears to be superior to the least squares polynomial when the gap between successive eigenvalues is small relative to the size of λ_c. The exact Chebyshev polynomial is also superior to the least squares polynomial when the eigenvalues of A are dense near both endpoints of S or throughout S. In the latter case, for instance, if we fix $\delta = 10^{-3}$ and increase N, the exact Chebyshev polynomial performs best for large N.

As claimed, the LS-Chebyshev polynomial behaves like the least squares polynomial and, depending on the eigenvalue distribution of A, may be superior to the exact Chebyshev polynomial. For example, if the eigenvalues of A are sparse near c and dense near d (recall Fig. 4.1), the LS-Chebyshev PCG method will usually converge in fewer iterations than the exact Chebyshev PCG method. The explanation is similar to that for the superiority of the least squares polynomial: The LS-Chebyshev polynomial maps those eigenvalues in $[\zeta, d]$ more tightly about 1 than does the exact Chebyshev polynomial. Of course, those eigenvalues in $[c, \zeta)$ are mapped further away from one. However, there are relatively few eigenvalues in $[c, \zeta)$; moreover, they be-

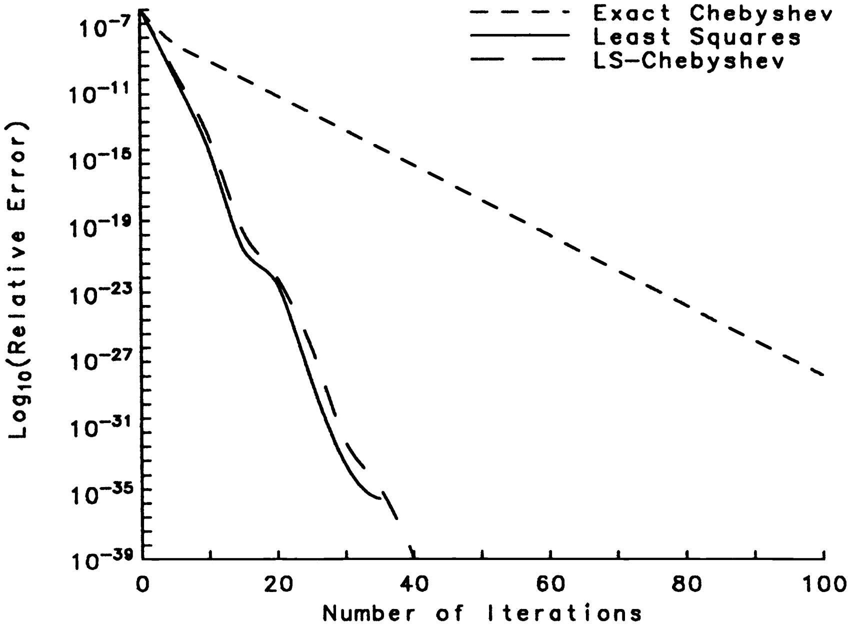


FIG. 4.1. *Dense-right eigenvalue distribution;* $\delta = 10^{-3}$.

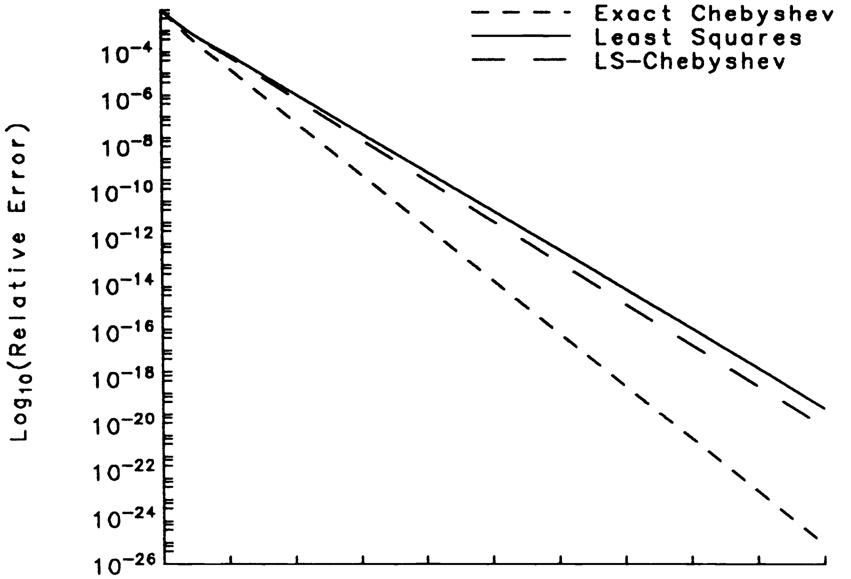


FIG. 4.2. *Dense-left eigenvalue distribution;* $\delta = 10^{-3}$.

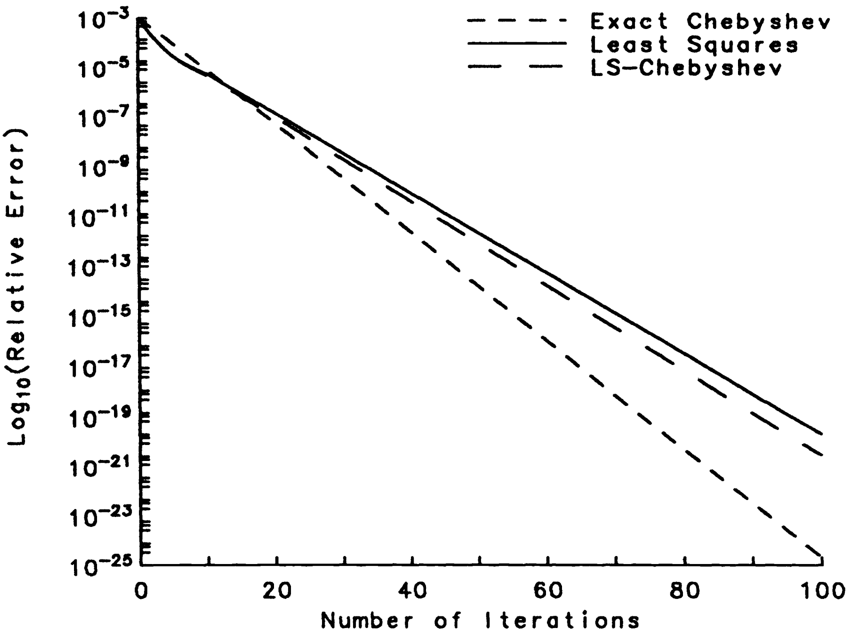


FIG. 4.3. *Uniform eigenvalue distribution;* $\delta = 10^{-3}$.

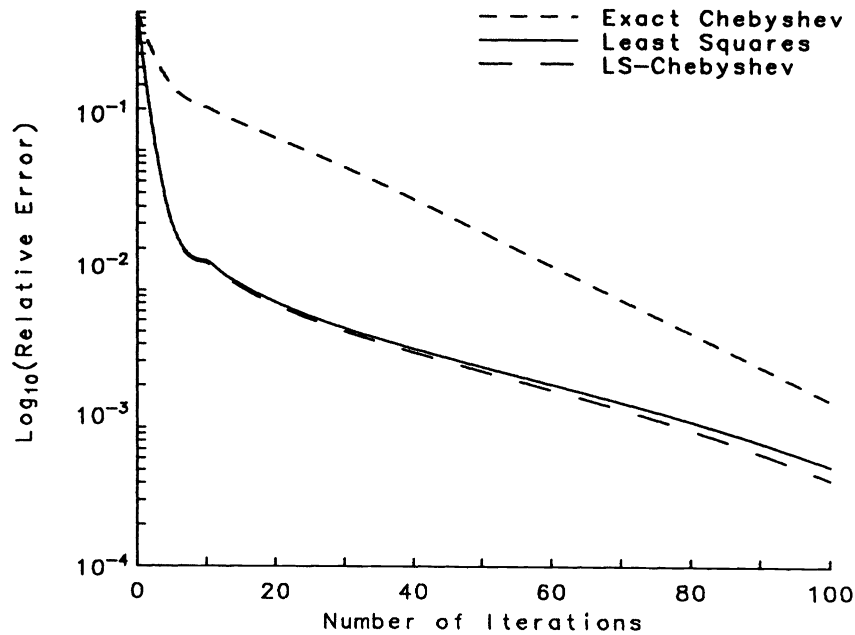


FIG. 4.4. *Uniform eigenvalue distribution;* $\delta = 10^{-5}$.

come isolated eigenvalues of $C(A)A$. It is well known that CG rapidly damps the error in the direction of the corresponding eigenvectors. After doing this, it is able to focus its effort on the dense part of the spectrum where the LS-Chebyshev polynomial does a better job of clustering the eigenvalues about 1. Since one seldom knows how the eigenvalues of A are distributed, one must rely on an adaptive procedure to find the optimum S. If one knew the eigenvalue distribution of A, an appropriately weighted Chebyshev or least squares polynomial could be used to achieve faster convergence. Freund [17] has recently proposed using the Lanczos eigenvalue estimates to obtain such a weight function, and his results are promising. In particular, he has shown that the resulting preconditioned CG method often converges faster than the method based on the exact Chebyshev polynomial. Unfortunately, there is no guarantee that the preconditioned matrix will be hpd for all S, and this can make difficult or impossible the robust implementation of some adaptive CG algorithms.

Finally, we note that the choice of stopping criterion can also affect the choice of polynomial. Since the least squares polynomial is small on the large eigenvalues of A, it is biased toward this part of the spectrum. Those eigenvectors associated with the large eigenvalues of A are consequently damped the most. If one bases the stopping criterion on the relative residual, the eigenvectors corresponding to these large eigenvalues are given greater weight. Thus, this stopping criterion is ideally suited to the least squares polynomial. The exact Chebyshev polynomial, on the other hand, is well suited for use in stopping criteria based on the true error. (Although the true error is unknown, it can be bounded [7].) The difference between these stopping criteria can be as large as $\kappa_B(A)$.

4.1. The need for an adaptive procedure. Recall that the weighted least squares and uniform norms are defined with respect to the positive interval S, which we have assumed contains the spectrum of A. That is, we have assumed that the smallest and largest eigenvalues of A, λ_c and λ_d, are given. Unfortunately, this is seldom true. In the case of the least squares polynomial, one may avoid this difficulty by choosing c and d to be the Gershgorin estimates for λ_c and λ_d. In particular, one may take $c = 0$. The resulting preconditioner is often effective, but there are eigenvalue distributions for which the exact Chebyshev polynomial is better. Here one needs accurate estimates for the extreme eigenvalues of A. Although this might be viewed as a reason for using the Neumann series or least squares polynomial, it is not. As we will see, one may dynamically estimate λ_c and λ_d from the CG iteration parameters. This is equivalent to dynamically determining the exact Chebyshev polynomial preconditioner. However, as we have seen, one can often do much better by basing the Chebyshev preconditioning polynomial on an interval $[c_{opt}, d_{opt}]$, where $\lambda_c \leq c_{opt} \leq d_{opt} \leq \lambda_d$. In the next section, we describe an *adaptive procedure* for finding these optimum endpoints.

5. Adaptive CG algorithms. In this section we discuss *adaptive* CG algorithms. In such an algorithm we apply a given CG method to the preconditioned linear system $C(A)Ax = C(A)b$, where $C(\lambda)$ is a Chebyshev preconditioning polynomial. Information about the spectrum of A is extracted from the CG iteration parameters and used to obtain a better preconditioner, $\tilde{C}(A)$. In this way the adaptive algorithm tries to determine the *optimum* Chebyshev polynomial preconditioner for A. Here the "optimum" polynomial is the one yielding the fastest convergence of the preconditioned CG algorithm. Although we consider only the Chebyshev polynomial, a similar procedure could be devised for the least squares polynomial.

In general, the interval on which the optimum polynomial is based, $[c_{opt}, d_{opt}]$, is a proper subset of $\Sigma(A) = [\lambda_c, \lambda_d]$. Our goal is to determine c_{opt} and d_{opt}. To do this, we start with an initial set $S_0 = [c_0, d_0]$ and expand it using information obtained from the CG iteration. Specifically, given an interval $S \subset \Sigma(A)$, and a Chebyshev preconditioning polynomial $C(\lambda)$ based on S, we apply a CG method to $C(A)Ax = C(A)b$. After a prescribed number of steps, say ℓ, the *adaptive procedure* is called:

(1) Compute eigenvalue estimates for $p_m(A) = C(A)A$.
(2) Extract eigenvalue estimates for A and update S.
(3) Determine the new preconditioning polynomial, $\tilde{C}(\lambda)$.
(4) Resume or restart the CG iteration, whichever is appropriate.

After another ℓ CG steps, the adaptive procedure is called again, and so on until convergence.

Eigenvalue estimates for $p_m(A)$ are easily obtained from the CG iteration parameters by exploiting the equivalence of the CG and Lanczos algorithms [7], [13], [19]. (See [18] for an alternative.) As we will see, it is easy to recover eigenvalue estimates for A when the degree m of the polynomial is odd. Once we have these estimates, we determine the new polynomial (i.e., the new endpoints on which to base it) by comparing several convergence factors. If a new polynomial results, we *restart* the CG iteration. By this we mean that the current iteration is abandoned and the CG method is applied to $\tilde{C}(A)Ax = \tilde{C}(A)b$. The new initial guess is the last iterate of the previous iteration or some linear combination of past iterates.

5.1. Estimating the extreme eigenvalues of A. Suppose we are executing a polynomial preconditioned CG iteration, where

$$p_m(\lambda) = C(\lambda)\lambda = 1 - \frac{T_m(\frac{d+c-2\lambda}{d-c})}{T_m(\frac{d+c}{d-c})} \tag{5.1}$$

is the Chebyshev preconditioned polynomial for $S = [c, d]$. Note that the image of S under p_m is $J_\epsilon = [1 - \epsilon, 1 + \epsilon]$, where $\epsilon = T_m^{-1}(\frac{d+c}{d-c})$; recall § 3.2. Next, assume the adaptive procedure has been called, and let μ be an eigenvalue estimate for $p_m(A)$ such that $\mu \in \Sigma(p_m(A))$, the convex hull of $\sigma(p_m(A))$. (This is true of the estimates we will obtain.) The desired eigenvalue estimate for A is one of the inverse images of μ; the task is to determine which one. It is important to choose an inverse image that lies in $\Sigma(A)$. Otherwise S might be improperly and irrevocably expanded, which would slow the convergence of subsequent CG iterations.

Suppose first that $\mu \in J_\epsilon$. Then there exists an inverse image of μ inside S. Since there is no justification for expanding S, this estimate may be discarded. If every eigenvalue estimate for $p_m(A)$ is in J_ϵ, there is no need to update S, and the CG iteration resumes. The adaptive procedure has yielded no new information.

Now suppose $\mu \notin J_\epsilon$. Since $\mu \in \Sigma(p_m(A))$, μ must have an inverse image in $\Sigma(A)\backslash S$, which means there is an eigenvalue of A outside S. If an estimate, λ, of this eigenvalue can be recovered, S can be expanded, and a new, better preconditioner computed. When m is odd, it is easy to extract λ from μ, as may be seen in Fig. 5.1. For example, let $\mu = \mu_1 < 1-\epsilon$. Since $p_m(\lambda)$ is monotonically increasing for $\lambda \in (0, c)$, there is a unique $\lambda_1 \in (0, c)$ such that $\mu_1 = p_m(\lambda_1)$. Moreover, since $\mu_1 \in \Sigma(p_m(A))$, λ_1 must lie in $\Sigma(A)$. Similarly, if $\mu = \mu_2 > 1+\epsilon$, then the unique inverse image of μ_2, λ_2, lies in $\Sigma(A)$. Note that estimates for only the smallest and largest eigenvalues of

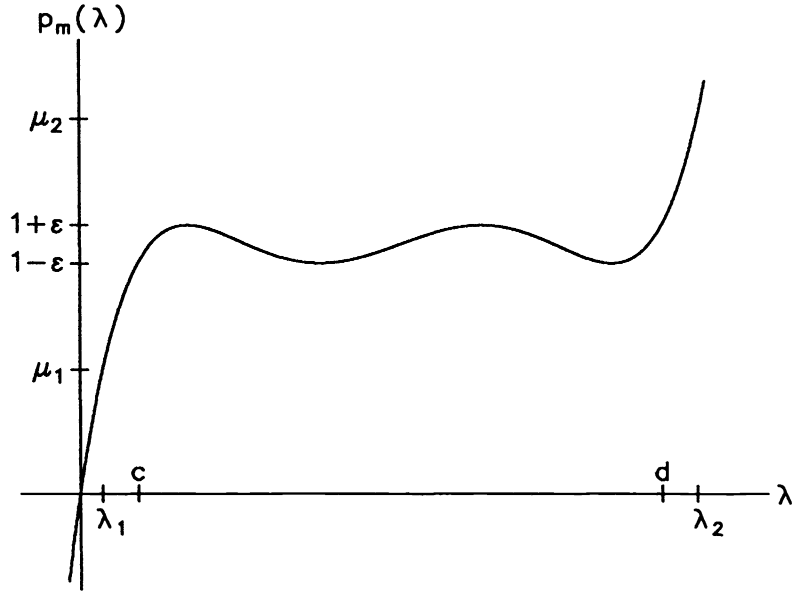


FIG. 5.1. *Chebyshev preconditioned polynomial* ($m = 5$) *for* $S = [1, 10]$.

$p_m(A)$ are needed, for these yield estimates, λ_1 and λ_2, for the extreme eigenvalues of A.

To compute the inverse images of μ_1 and μ_2, a rootfinder could be used, but this is unnecessary because $p_m(\lambda)$ is known explicitly. For m odd and $d \neq c$, one may show

$$\lambda_1 = \frac{1}{2}\left[(d+c) - (d-c)\cosh\left(\frac{1}{m}\cosh^{-1}\left(\frac{1-\mu_1}{\epsilon}\right)\right)\right] \tag{5.2}$$

and

$$\lambda_2 = \frac{1}{2}\left[(d+c) + (d-c)\cosh\left(\frac{1}{m}\cosh^{-1}\left(\frac{\mu_2-1}{\epsilon}\right)\right)\right]. \tag{5.3}$$

If $d = c$ (a common choice for the initial S), then

$$\lambda_1 = d\left(1 - (1-\mu_1)^{1/m}\right) \quad \text{and} \quad \lambda_2 = d\left(1 + (\mu_2 - 1)^{1/m}\right). \tag{5.4}$$

So far we have assumed that m is odd, which is important for two reasons. To see why, consider Fig. 5.2, in which m is even. As before, any eigenvalue estimates for $p_m(A)$ in J_ϵ are discarded. Since both tails of $p_m(\lambda)$ are negative, there can be no estimate $\mu > 1 + \epsilon$, so suppose $\mu < 1 - \epsilon$. There are now two inverse images, λ_1 and λ_2, at least one of which lies in $\Sigma(A)$. If the wrong one is chosen, the set S may be incorrectly enlarged, and the CG method will converge more slowly than necessary. To avoid this *ambiguity*, we shall always choose m odd.

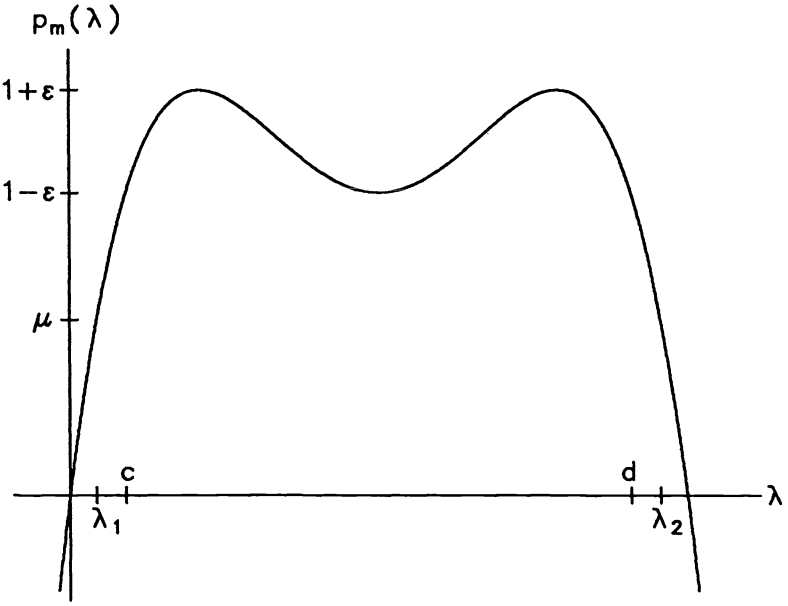


FIG. 5.2. *Chebyshev preconditioned polynomial* ($m = 4$) *for* $S = [1, 10]$.

Another advantage of choosing m odd is that it yields *robust* CG methods. By this we mean that $p_m(A)$ is hpd for any hpd A and for any set S. If this were not true, the CG method might not be defined in the early iterations. For example, if m were even and there were an eigenvalue of A greater than the largest root of $C(\lambda)\lambda$, $p_m(A)$ would be indefinite, in which case GCGHS and PCR are inappropriate, as are the Omin implementations of PCG, GPCR, and GCR. One would have to use the Odir or Odir/Omin implementation of PCG, GPCR, or GCR. When m is odd, on the other hand, $p_m(A)$ is hpd for any set S and the Omin implementation of each method in Table 2.1 is applicable.

5.2. Determining the new endpoints. After eigenvalue estimates for A are computed, we must determine the new endpoints on which to base the Chebyshev preconditioning polynomial. A simple choice is $c = \lambda_1$ and $d = \lambda_2$, which essentially gives the adaptive procedure in [3] and [5]. Since the Lanczos algorithm quickly determines the extreme eigenvalues of $C(A)A$, and hence of A, this procedure quickly finds the exact Chebyshev preconditioning polynomial. If this polynomial were used, $\kappa(C(A)A)$ would be minimized. However, this polynomial does not necessarily minimize the number of CG iterations required for convergence. It has been observed [24], [30] that one often obtains better performance by basing the Chebyshev preconditioning polynomial on some interval $[c_{opt}, d_{opt}]$, where $\lambda_c \leq c_{opt} \leq d_{opt} \leq \lambda_d$. The LS-Chebyshev polynomial (§ 4) is such an example. To understand why, recall that the true rate of convergence of a CG method depends on the distribution of the eigenvalues of A within $\Sigma(A)$. For instance, if λ_c is an isolated eigenvalue of A and $\sigma(A)\backslash\lambda_c \subset [\lambda_\nu, \lambda_d]$ for $\lambda_\nu \gg \lambda_c$, then $S = [\lambda_\nu, \lambda_d]$ is a better choice than $[\lambda_c, \lambda_d]$. The reason is that CG methods pick out isolated eigenvalues and rapidly suppress the error in the direction of corresponding eigenvectors. Unfortunately, one seldom knows such detailed information about $\sigma(A)$. Although the recent idea of Freund [17] to use the Lanczos eigenvalue estimates to approximate the eigenvalue distribution of A is appealing, it is unclear whether this can be done dynamically.

We will now describe an adaptive procedure for finding the optimal left endpoint, c_{opt}, on which to base the Chebyshev preconditioning polynomial. To ease the task, we will assume $d = \lambda_d$ (see below). Although there is a $d_{opt} \leq \lambda_d$, the CG iteration is less sensitive to changes in d than in c. This is so because $\epsilon = \epsilon(c, d)$ is most sensitive to changes in c. Ideally, the adaptive procedure will find $c_{opt} = \lambda_c$ when the exact Chebyshev preconditioning polynomial is best, and will find $c_{opt} = \zeta$ when the least squares polynomial is preferred. (Recall that the Chebyshev polynomial based on $[\zeta, \lambda_d]$ mimics the least squares polynomial.) In § 5.4 we will demonstrate the effectiveness of this adaptive procedure in a variety of numerical experiments.

Let $p_m(\lambda) = p_m(\lambda; c, d)$ be the Chebyshev preconditioned polynomial based on $[c, d]$, $0 < c \leq d$, m odd. Also let 2ϵ be the size of the oscillations in the Chebyshev polynomial, that is,

$$\epsilon = 1 - p_m(c) = p_m(d) - 1 = T_m^{-1}\left(\frac{d+c}{d-c}\right).$$

Since $d = \lambda_d$ is fixed, we will consider ϵ to be a function of c alone. As c increases from 0 to d, ϵ decreases from 1 to 0. If $\sigma(A) \subset [c, d]$, then $\kappa(p_m(A)) \leq \frac{1+\epsilon}{1-\epsilon}$. Suppose, however, that $\sigma(A) \subset [\lambda_o, d]$ for some $0 < \lambda_o \leq c$. That is, assume we have based the preconditioning polynomial on $[c, d]$, but there is an eigenvalue $\lambda_o \leq c$. Then $\kappa(p_m(A)) \leq \frac{1+\epsilon}{1-\delta}$, where $\delta = 1 - p_m(\lambda_o)$, and the associated *asymptotic convergence factor* is

$$cf(\lambda_o, c) = \frac{\sqrt{\frac{1+\epsilon}{1-\delta}} - 1}{\sqrt{\frac{1+\epsilon}{1-\delta}} + 1}. \tag{5.5}$$

(Note that $0 \leq \epsilon \leq \delta \leq 1$.) If $\sigma(A)$ is dense in $[\lambda_o, d]$, this convergence factor may be used to accurately predict the rate of convergence of the CG iteration.

Consider next Fig. 5.3, in which we plot $cf_c = cf(c, c)$ and $cf_o = cf(\lambda_o, c)$ as functions of c. The lower curve plots cf_c, the convergence factor when $\sigma(A) \subset [c, d]$ and $C(\lambda)$ is based on $p_m(\lambda; c, d)$. As c increases from 0 to d, cf_c decreases rapidly from 1 to 0. The upper curve plots cf_o, the convergence factor when $\sigma(A) \subset [\lambda_o, d]$, but $C(\lambda)$ is still based on $p_m(\lambda; c, d)$. Note that the curves intersect at λ_o and coincide for $c \leq \lambda_o$. Next let cf_e be the *empirical* convergence factor, which is given by

$$cf_e = \left(\frac{\|s_k\|}{\|s_0\|}\right)^{1/k} \tag{5.6}$$

if k is large enough. Here $s_k = C(A)r_k$ is the preconditioned residual, where $C(A)$ is the Chebyshev preconditioning polynomial for $[c, d]$. Note that cf_e may be monitored throughout the CG iteration. If $cf_e \leq cf_c$, the iteration is converging as it should, and there is no need to call the adaptive procedure. On the other hand, suppose $cf_e > cf_c$. Then the iteration is converging more slowly than it should because we have missed an eigenvalue of A, and so we need to compute eigenvalue estimates (§ 5.1). Since we have assumed $d = \lambda_d$, we will obtain only an estimate λ_1 for λ_c. Next let $cf_1 = cf(\lambda_1, c)$,

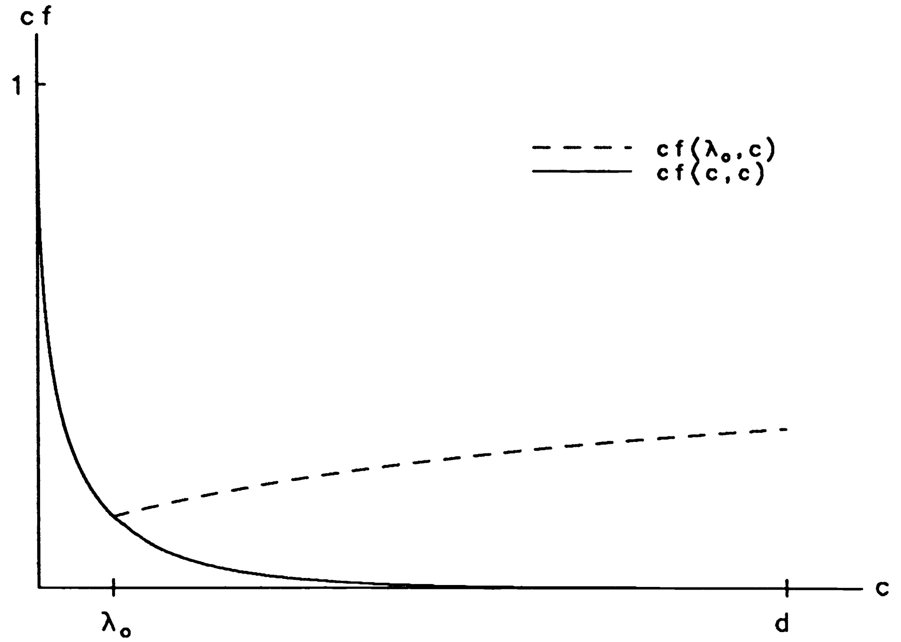


FIG. 5.3. *Convergence factors for $\sigma(A)$.*

which is the convergence factor for $\sigma(A) \subset [\lambda_1, d]$. If $cf_e > cf_1$, we will decrease c to λ_1, as in [3] and [5]. But suppose $cf_c < cf_e < cf_1$. This suggests that we should not decrease c to λ_1, but to λ_e, where $cf_e = cf(\lambda_e, c)$. As in (5.2), λ_e is given by

$$\lambda_e = \frac{1}{2}\left[(d+c) - (d-c)\cosh\left(\frac{1}{m}\cosh^{-1}\left(\frac{\delta}{\epsilon}\right)\right)\right] \tag{5.7}$$

where

$$\delta = 1 - (\epsilon - 1)\left(\frac{1+cf_e}{1-cf_e}\right)^2$$

from (5.5). By decreasing c to λ_e instead of λ_1, we are trying to minimize the *actual* convergence factor (see below). Note that this strategy guarantees that $c \in \Sigma(A)$.

5.3. Summary description. Given an interval $S \subset \Sigma(A)$, fix c and determine λ_d via the CG-Lanczos equivalence (§ 5.1). We will assume convergence when the relative difference in two successive estimates is less than some tolerance, say 10^{-2}. (Of course, if an a priori estimate of λ_d is available, it may be used.) After a prescribed number of steps, say ℓ, call the adaptive procedure:

(1) Compute eigenvalue estimates for $p_m(A) = C(A)A$.
(2) Extract an eigenvalue estimate λ_1 for λ_c.
(3) Compute cf_e, cf_c, and cf_1.
(4) Update c:
- If $cf_e \leq cf_c$, $c \leftarrow c$.
- If $cf_c < cf_e < cf_1$, $c \leftarrow \lambda_e$.

- If $cf_e \geq cf_1$, $c \leftarrow \lambda_1$.

(5) Restart the CG iteration if c has changed; resume if c is unchanged.

The key to this adaptive procedure is the comparison of cf_e with cf_c and cf_1. If $\sigma(A)$ is dense in $[\lambda_1, d]$, then cf_1 is an accurate indicator of the CG rate of convergence, and the comparison is well founded. Otherwise, it is simply an heuristic. If $\sigma(A)$ is not dense in $[\lambda_1, d]$, the adaptive procedure should detect this by computing $cf_e < cf_1$. In this case, the iteration is converging as if $\sigma(A)$ were dense in $[\lambda_e, d]$. However, we know that there is at least one eigenvalue to the left of λ_e because we computed $\lambda_1 \in \Sigma(A)\backslash[\lambda_e, d]$. Since the iteration is converging at a rate determined by cf_e, the dense portion of $\sigma(A)$ is no larger than $[\lambda_e, d]$. Thus, we assume that any eigenvalues to the left of λ_e are isolated, and we choose to base the Chebyshev preconditioning polynomial on $[\lambda_e, d]$. This heuristic is flawed because the dense portion of $\sigma(A)$ is actually some interval $[\lambda_\nu, d]$ for $\lambda_\nu > \lambda_e$. (The isolated eigenvalues gave rise to the convergence factor cf_e rather than cf_ν.) Unfortunately, there seems to be no way of determining λ_ν.

The difference between this adaptive procedure and the one described in [3] and [5] is essentially this: here we determine λ_d first and then refine c. (Of course, we can continue to refine d, but it seems to make little difference in practice.) In general, this new procedure will outperform the one in [3] and [5]. Of course, if the Chebyshev preconditioning polynomial is optimum, the other procedure will find it more quickly since c and d are adapted simultaneously.

Since S is either expanded or unchanged with each call to the adaptive procedure, it is important that the initial set, S_0, be such that $S_0 \subset \Sigma(A)$. If the $N \times N$ matrix A is scaled to have unit diagonal, one may take $S_0 = [1, 1]$. A more general choice is $S_0 = [\tau, \tau]$, where $\tau = \mathit{trace}(A)/N$ or $\tau = \langle Ab, b\rangle / \langle b, b\rangle$, both of which lie in $\Sigma(A)$. If one had an a priori estimate for λ_d, say from Gershgorin's theorem, one could take $c_0 = \zeta$ and $d_0 = \lambda_d$. As discussed in § 4, this would mimic the least squares polynomial initially, but retain the flexibility of the adaptive procedure. Although there is no guarantee that $\zeta \in \Sigma(A)$, this is often the case for moderate m and $\kappa(A) \gg 1$. In the next section we show that the polynomial found by this procedure usually bests the least squares polynomial.

5.4. Performance. Having introduced the theory behind the adaptive procedure, we now consider its performance in practice. We will first show that the adaptive procedure quickly and accurately determines λ_c and λ_d. We will then compare two adaptively determined Chebyshev preconditioning polynomials with the exact Chebyshev and least squares polynomials. We will see that the former are generally competitive with the latter. In § 6 we will demonstrate the effectiveness of adaptive polynomial preconditioned CG algorithms for some large matrices from hydrology.

The tables below summarize the behavior of the adaptive *procedure* for two simple test problems. The results are taken from [5], but the adaptive procedure described in this paper would give similar results. The matrices have order 2500 and result from a five-point and nine-point finite difference approximation to the two-dimensional Laplacian. Although PCG results are given only for a polynomial of degree 7, these results are typical. In each table we list the estimates for λ_c and λ_d computed by the adaptive procedure, which is called every five steps. The adaptive algorithm is initially given $c = d = 1$. In the last column we report the action taken by the adaptive procedure.

Consider Table 5.1. After five steps, the adaptive procedure found new estimates

TABLE 5.1
PCG adaptive procedure for five-point Laplacian.

$N = 2500$			$m = 7$
k	λ_c	λ_d	ACTION
0	0.10000e+01	0.10000e+01	initial
5	0.24762e-01	0.19870e+01	restart
10	**0.27832e-02**	**0.19962e+01**	restart
15	0.27577e-02	0.19972e+01	resume
20	0.19262e-02	0.19981e+01	resume
25	0.18981e-02	0.19981e+01	resume
30	0.18968e-02	0.19981e+01	resume
TRUE	0.18967e-02	0.19981e+01	

TABLE 5.2
PCG adaptive procedure for nine-point Laplacian.

$N = 2500$			$m = 7$
k	λ_c	λ_d	ACTION
0	0.10000e+01	0.10000e+01	initial
5	0.17330e-01	0.14347e+01	restart
10	0.15041e-01	0.15959e+01	restart
15	**0.22899e-02**	**0.15968e+01**	restart
20	0.22899e-02	0.15990e+01	resume
25	0.22899e-02	0.15992e+01	resume
TRUE	0.22753e-02	0.15992e+01	

for λ_c and λ_d and decided to restart the iteration using a new preconditioning polynomial based on these estimates. After another five steps, the adaptive procedure refined its estimates for λ_c and λ_d and again restarted. From here on it continues to improve its estimates for λ_c and λ_d, but opts to resume the iteration using the polynomial determined at step 10. Similar behavior is seen in Table 5.2. We remark that λ_c and λ_d are found more quickly with higher degree polynomials.

The performance described here is typical. Although the estimates for λ_c and λ_d eventually converge to their true values, the adaptive procedure often finds satisfactory estimates early on in the iteration. In other words, the adaptive procedure is able to find a nearly optimum polynomial preconditioner within a few calls. This means that there is little overhead associated with the adaptive procedure. We remark that the resume versus restart decision is an important one: it can make a dramatic difference in the number of steps required for convergence to the solution of the linear system.

In Figs. 5.4–5.9 we compare the performance of four polynomial preconditioned CG (PPCG) algorithms on four eigenvalue distributions. (Here PPCG denotes PCG [13] with a polynomial preconditioner.) The algorithms are: (1) PPCG with the least squares preconditioning polynomial for $[0, \lambda_d]$; (2) PPCG with the exact Chebyshev preconditioning polynomial for $[\lambda_c, \lambda_d]$; (3) adaptive PPCG with $c_0 = \zeta$ and $d_0 = \lambda_d$; and (4) adaptive PPCG with $c_0 = d_0 = trace(A)/N$. We will abbreviate these algorithms LS, EC, AC(LS), and AC(TR), respectively. For these experiments, we employed the adaptive procedure described above. (Since the adaptive procedure in [3], [5] quickly finds λ_c and λ_d, it would behave like EC.) The algorithms AC(LS) and AC(TR) differ only in their initial intervals. The point of the former is to see how the adaptive algorithm compares with the least squares polynomial when given the same information, namely, λ_d. AC(TR) is, in a sense, the "black box" algorithm of choice since it requires no a priori knowledge of $\sigma(A)$.

The first three plots compare the four algorithms on the uniform, dense-left, and dense-right distributions described in § 4 ($\delta = 10^{-5}$, $N = 100,000$). In Fig. 5.4, the least squares polynomial outperforms the exact Chebyshev polynomial, but is inferior to AC(LS). The reason is this: Although AC(LS) initially behaves like the least squares polynomial, the adaptive procedure allows us to refine the left endpoint. In Fig. 5.5, the exact Chebyshev polynomial is best, but the two adaptive algorithms are competitive: Whereas EC starts off with the optimum left endpoint, they must determine it dynamically. Finally, in Fig. 5.6, all four algorithms converge quickly. AC(TR) beats AC(LS) because $c_0 = \zeta$ is already too small. AC(LS) is slightly worse than LS for this dense-right distribution because it erroneously adapts the left endpoint toward the origin.

In the last three plots, the eigenvalue distribution is that of the five-point Laplacian for a 400×400 grid. Since these eigenvalues are somewhat uniformly distributed in $(0, 8)$, Fig. 5.7 is nearly identical to Fig. 5.4, and the explanation for the algorithms' relative performance is the same. In Fig. 5.8, we consider the effect of overestimating λ_d, as is often the case with Gershgorin's theorem. Here LS, EC, and AC(LS) are all given $d_0 = 12$. Consequently, AC(TR) performs a bit better relative to the other three, but the difference is marginal. In Fig. 5.9, we underestimate λ_d with $d_0 = 6$. Note that LS and EC both suffer. The adaptive algorithms, however, are able to find λ_d quickly, before adapting the left endpoint. We remark that overestimating (underestimating) λ_d would have been much more troublesome for the dense-left (dense-right) distribution.

In summary, our results show that adaptive Chebyshev polynomial preconditioning is not only competitive with least squares and exact Chebyshev polynomial preconditioning, but usually beats them. In particular, AC(LS) was superior to the least squares polynomial for all but the dense-right distribution. This is so because, although it initially behaves like the least squares polynomial, it retains the option to adapt the left endpoint. Moreover, we have shown that the "black box" AC(TR) algorithm is always competitive. This is important because one seldom has any a priori knowledge of the eigenvalue distribution of A.

6. Numerical experiments.

In this section we demonstrate the effectiveness of adaptive polynomial preconditioning on a Cray X-MP/48 and Alliant FX/8 for some large matrices arising in hydrology. In particular, we show that polynomial preconditioned PCG (PPCG) can converge in less CPU time than the unpreconditioned CGHS method. Although we do not compare it with other preconditionings, we emphasize that polynomial preconditioning can be used to further accelerate any other preconditioning, for example, an incomplete factorization.

6.1. Description of experiments.

Our test matrices, which arise in the modeling of groundwater flow in a heterogeneous aquifer, result from the seven-point finite difference approximation to a three-dimensional elliptic PDE with variable coefficients. Although several parameters determine the difficulty of the problem, we isolate just two. In the first set of experiments, run on a Cray X-MP/48, the hydraulic conductivity field K is uncorrelated, which makes the problem difficult. In the second set of experiments, run on an Alliant FX/8, the field is correlated. For each machine we vary γ, the standard deviation of the $\ln K$ field. As γ increases, so does the condition number of A. See [27] for details.

In the tables below, m is the degree of the preconditioned polynomial, $p_m(\lambda)$.

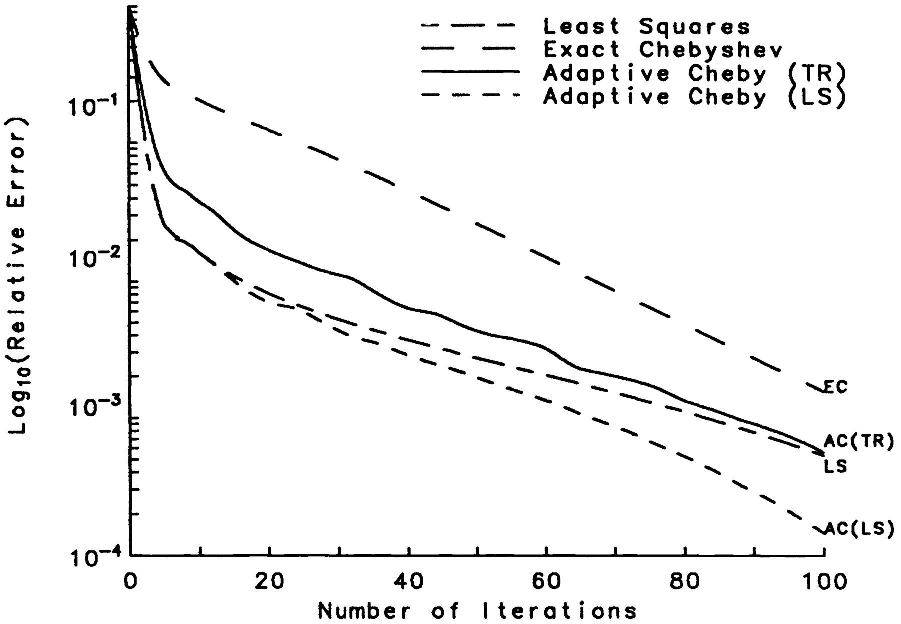


FIG. 5.4. *Uniform eigenvalue distribution;* $\delta = 10^{-5}$.

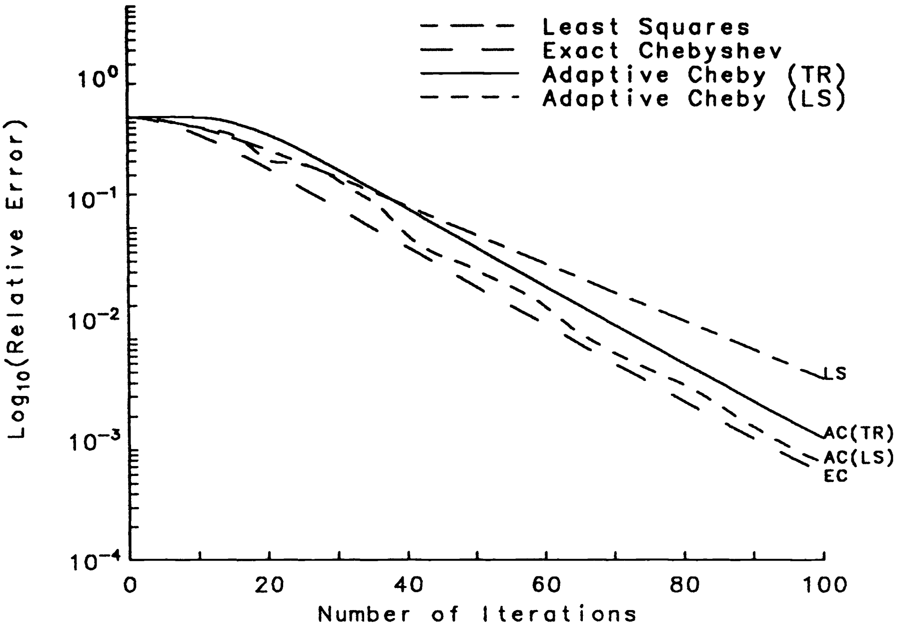


FIG. 5.5. *Dense-left eigenvalue distribution;* $\delta = 10^{-5}$.

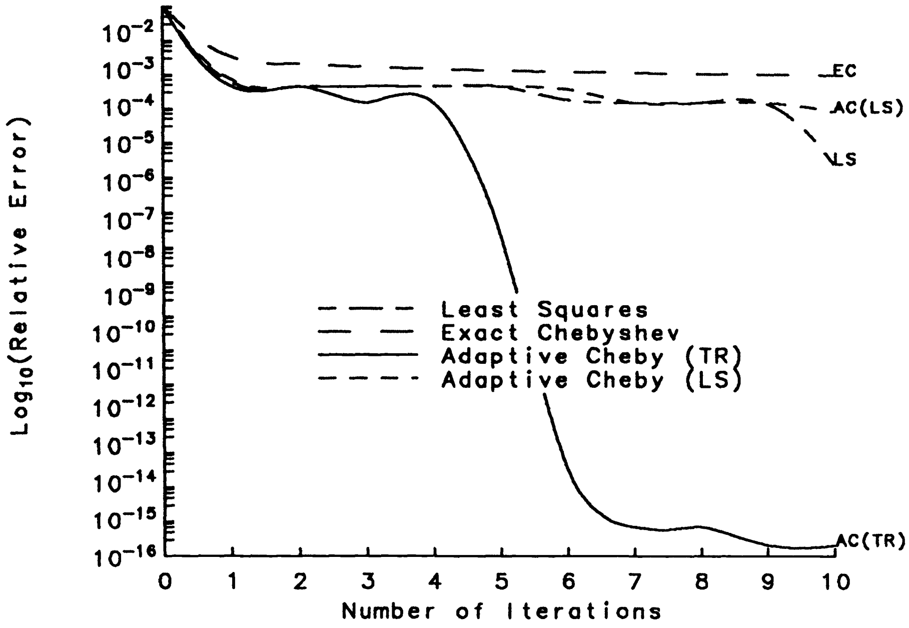


FIG. 5.6. *Dense-right eigenvalue distribution;* $\delta = 10^{-5}$.

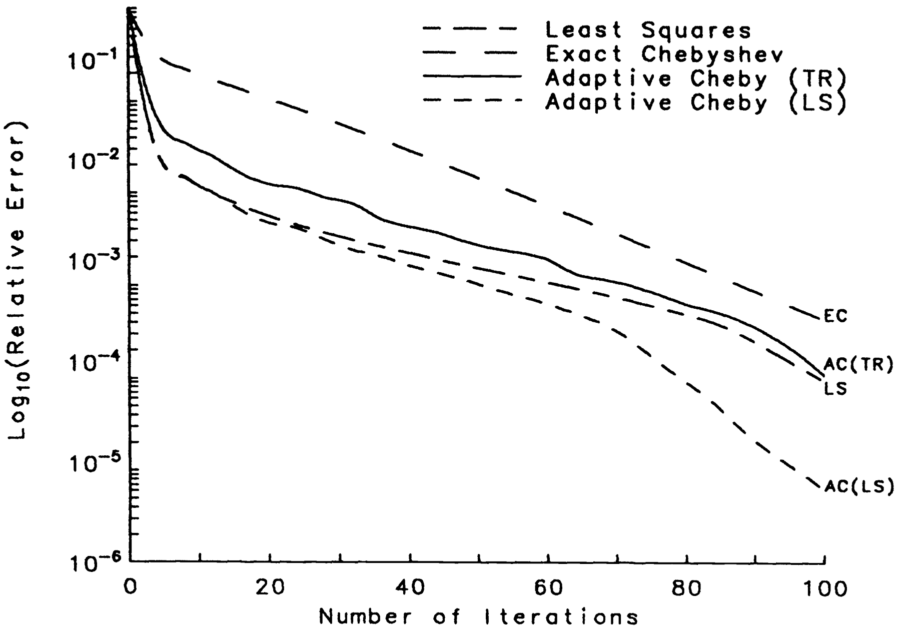


FIG. 5.7. *Laplacian distribution.*

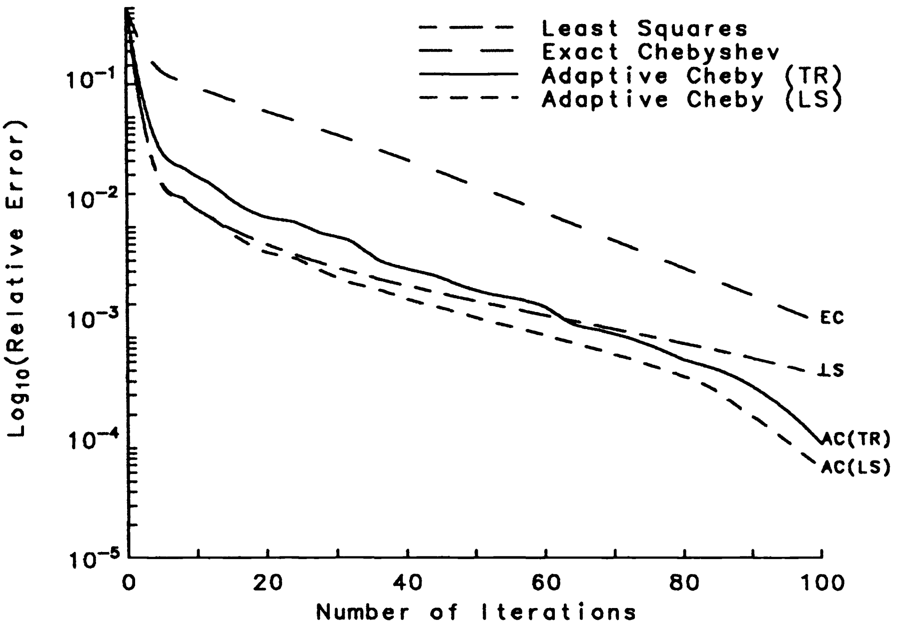


FIG. 5.8. *Overestimated Laplacian distribution.*

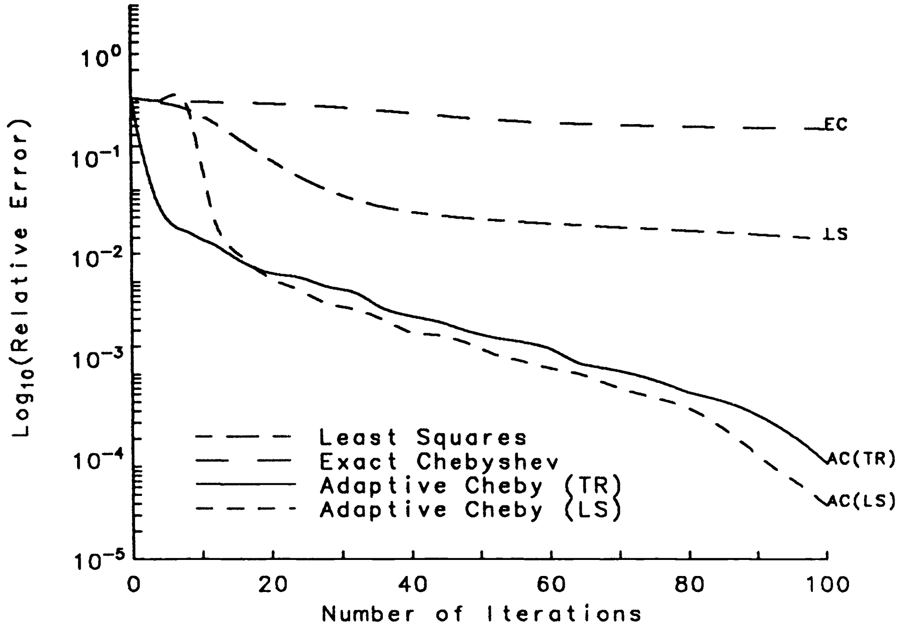


FIG. 5.9. *Underestimated Laplacian distribution.*

The first row of each table, $m = 1$, corresponds to the unpreconditioned CGHS method. We next give the number of CPU seconds required for convergence of the CG iteration, which includes the adaptive procedure. The iteration was halted once the relative error was brought below 10^{-8} on the Cray and 10^{-6} on the Alliant. In the last column of each table we list the ratio of CGHS time to PPCG time. If this ratio is greater than 1, we say that polynomial preconditioning is *effective*. In all the experiments, the right-hand-side vector b was chosen so that the true solution vector has 1 in each component, and the initial guess was the zero vector. The adaptive procedure used is the one described in [3] and [5]; the one presented in this paper would give even better performance. Since the matrix was symmetrically scaled to have unit diagonal, we set $c_0 = d_0 = 1$ in the adaptive procedure. New eigenvalue estimates were computed every ten steps with a maximum of ten calls to the adaptive procedure. Finally, we note that the results below were taken from [3] and [27].

6.2. Discussion of results. In Tables 6.1–6.3 we report results for a single vector processor of a Cray X-MP/48. In the first table, the condition number of A is about 60,000, as estimated by the adaptive procedure. Here we obtain a 15 percent improvement over CGHS with a polynomial of degree 5. In the next two tables, $\kappa(A)$ is 160,000 and 360,000, respectively, corresponding to $\gamma = 1.5$ and $\gamma = 1.75$. Notice that polynomial preconditioning is more effective here: it reduces the CPU time required to solve the problem by about 25 percent. Also observe that the optimum m is increasing with $\kappa(A)$.

TABLE 6.1
Cray X-MP/48 CPU times.

$N = 103,823$			$\gamma = 1.0$
m	Iterations	Seconds	CGHS/PPCG
1	1112	18.00	1.00
3	386	15.77	1.14
5	242	15.65	1.15
7	229	20.59	0.87
9	215	24.82	0.73
11	152	21.35	0.84

TABLE 6.2
Cray X-MP/48 CPU times.

$N = 103,823$			$\gamma = 1.5$
m	Iterations	Seconds	CGHS/PPCG
1	2315	37.04	1.00
3	780	31.81	1.16
5	473	30.56	1.21
7	341	30.22	1.23
9	268	30.16	1.23
11	227	31.28	1.18
13	213	34.84	1.06

In Tables 6.4–6.6 we see similar qualitative results for the Alliant FX/8, which is an 8-vector-processor machine. Although the problems are much larger, they are not nearly as ill conditioned. (We estimate the condition numbers to be 8,400, 14,000, and 26,000.) We once again see the best performance on the hardest problem. Moreover,

TABLE 6.3
Cray X-MP/48 CPU times.

$N = 103,823$			$\gamma = 1.75$
m	Iterations	Seconds	CGHS/PPCG
1	4126	66.76	1.00
3	1383	57.30	1.17
5	833	54.67	1.22
7	600	53.72	1.24
9	469	53.82	1.24
11	386	53.60	1.25
13	328	53.55	1.25
15	287	53.55	1.25
17	255	53.36	1.25
19	235	55.10	1.21

TABLE 6.4
Alliant FX/8 CPU times.

$N = 410,625$			$\gamma = 1.0$
m	Iterations	Seconds	CGHS/PPCG
1	468	887.61	1.00
3	183	673.77	1.32
5	103	541.13	1.64
7	82	564.95	1.57
9	70	598.83	1.48
11	57	586.48	1.51

TABLE 6.5
Alliant FX/8 CPU times.

$N = 410,625$			$\gamma = 1.7$
m	Iterations	Seconds	CGHS/PPCG
1	607	1141.40	1.00
3	249	910.98	1.25
5	155	801.22	1.42
7	95	664.83	1.76
9	79	684.41	1.70
11	69	714.39	1.62

notice the much larger CGHS/PPCG ratios: the time required to solve the problem has been nearly cut in half. The computer architecture does indeed make a difference.

These results demonstrate the effectiveness of polynomial preconditioning on a Cray X-MP/48 and an Alliant FX/8. We have seen that polynomial preconditioning is most effective when the matrix A is ill conditioned. Moreover, as $\kappa(A)$ increases, so does the optimum degree m. In general, however, low degree (2–16) preconditioning polynomials are usually best. In contrast, high degree (20–50) polynomials are usually best for Hermitian indefinite matrices [3], [6]. Although we have presented results for only the hydrology problem, our conclusions are supported by a variety of other numerical experiments, including those in [3], [9], [15], [25], [27].

We emphasize that our adaptive CG algorithms are as easy to use as CGHS, yet can reduce the CPU time required to solve the linear system. The amount of reduction depends on the computer architecture. We note that Holst [23] has obtained results similar to those for the Alliant on a Cray 2. In particular, he has reported

TABLE 6.6
Alliant FX/8 CPU times.

$N = 410,625$			$\gamma = 2.3$
m	Iterations	Seconds	CGHS/PPCG
1	839	1584.50	1.00
3	308	1108.60	1.43
5	205	1057.20	1.50
7	134	913.34	1.73
9	103	869.94	1.82
11	88	889.15	1.78

CGHS/PPCG ratios of nearly 2 to 1, which is far better than those achieved on the X-MP. Chan, Kuo, and Tong [9] have shown that polynomial preconditioning is competitive with other preconditioners on the massively parallel CM-2.

7. Summary. In this paper we have explored the use of adaptive polynomial preconditioning for Hermitian positive definite linear systems. Such preconditioners are easy to employ and well suited to vector and/or parallel computer architectures. After reviewing preconditioned CG methods, we showed how one could use a polynomial preconditioner in a variety of different ways. We then discussed the least squares and Chebyshev preconditioning polynomials, studied them in the context of CG methods, and showed that the latter minimizes a bound on the condition number of the preconditioned matrix. We next compared the two polynomials in a variety of numerical experiments. In particular, we sought to determine those eigenvalue distributions for which each is well suited. The least squares polynomial is superior for those matrices whose eigenvalues are dense near the largest eigenvalue, λ_d. In contrast, the Chebyshev preconditioner is superior when the eigenvalues are dense throughout the interval or when the gap between successive eigenvalues is smaller than the smallest eigenvalue, λ_c. We next described an *adaptive procedure* for dynamically computing λ_c and λ_d, which are needed to determine the optimal Chebyshev polynomial preconditioner. Specifically, by comparing various convergence factors, we attempt to find the optimum endpoints on which to base the Chebyshev preconditioning polynomial. The accuracy and efficiency of this adaptive procedure was also demonstrated. In particular, we showed that our adaptive algorithm usually beats algorithms based on the least squares or exact Chebyshev polynomials. Finally, in the previous section, we presented some numerical results that demonstrate the effectiveness of adaptive polynomial preconditioning for some large matrices arising in hydrology. Our results suggest that relatively low degree (2–16) polynomials are usually best. Moreover, the optimum degree m of the polynomial tends to increase with the condition number of A, as does the effectiveness of polynomial preconditioning.

Acknowledgments. We wish to thank Professor Paul Saylor for his many helpful comments and thoughtful questions. We also thank Philip Meyer for providing the hydrology test problems and associated software.

REFERENCES

[1] L. M. ADAMS, *Iterative Algorithms for Large, Sparse Linear Systems on Parallel Computers*, Ph.D. thesis, Dept. of Applied Mathematics, University of Virginia, Charlottesville, VA, 1982.

[2] L. M. ADAMS, *m-step preconditioned conjugate gradient methods*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 452–463.

[3] S. F. ASHBY, *Polynomial Preconditioning for Conjugate Gradient Methods*, Ph.D. thesis, Dept. of Computer Science, University of Illinois, Urbana, IL, December 1987. Available as Tech. Report 1355.

[4] ———, *Minimax polynomial preconditioning for Hermitian linear systems*, SIAM J. Mat. Anal. Appl., 12 (1991), pp. 766–789.

[5] S. F. ASHBY, T. A. MANTEUFFEL, AND J. S. OTTO, *Adaptive polynomial preconditioning for HPD linear systems*, in Proc. Ninth International Conference on Computing Methods in Applied Sciences and Engineering, R. Glowinski and A. Lichnewsky, eds., Paris, January 1990, SIAM, pp. 3–23.

[6] S. F. ASHBY, T. A. MANTEUFFEL, AND P. E. SAYLOR, *Adaptive polynomial preconditioning for Hermitian indefinite linear systems*, BIT, 29 (1989), pp. 583–609.

[7] ———, *A taxonomy for conjugate gradient methods*, SIAM J. Numer. Anal., 27 (1990), pp. 1542–1568.

[8] P. N. BROWN AND A. C. HINDMARSH, *Matrix-free methods for stiff systems of ODE's*, SIAM J. Numer. Anal., 23 (1986), pp. 610–638.

[9] T. F. CHAN, C. J. KUO, AND C. TONG, *Parallel elliptic preconditioners: Fourier analysis and performance on the Connection Machine*, Tech. Report CAM 88-22, Dept. of Mathematics, University of California, Los Angeles, 1988.

[10] R. CHANDRA, S. C. EISENSTAT, AND M. H. SCHULTZ, *The modified conjugate residual method for partial differential equations*, in Advances in Computer Methods for Partial Differential Equations II, R. Vichnevetsky, ed., 1977, pp. 13–19.

[11] A. CHRONOPOULOS, *A Class of Parallel Iterative Methods Implemented on Multiprocessors*, Ph.D. thesis, Dept. of Computer Science, University of Illinois, Urbana, IL, November 1986. Available as Tech. Report 1267.

[12] A. T. CHRONOPOULOS AND C. W. GEAR, *Implementation of s-step methods on parallel vector architectures*, Tech. Report 1346, Dept. of Computer Science, University of Illinois, Urbana, IL, June 1987.

[13] P. CONCUS, G. H. GOLUB, AND D. P. O'LEARY, *A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1976, pp. 309–332.

[14] P. J. DAVIS, *Interpolation and Approximation*, Dover, New York, 1975.

[15] P. F. DUBOIS, A. GREENBAUM, AND G. H. RODRIGUE, *Approximating the inverse of a matrix for use on iterative algorithms on vector processors*, Computing, 22 (1979), pp. 257–268.

[16] V. FABER AND T. A. MANTEUFFEL, *Necessary and sufficient conditions for the existence of a conjugate gradient method*, SIAM J. Numer. Anal., 21 (1984), pp. 352–362.

[17] R. FREUND, *Polynomial Preconditioners for Hermitian and Certain Nonhermitian Matrices*, paper presented at SIAM Annual Meeting, San Diego, CA, July 1989.

[18] G. H. GOLUB AND M. D. KENT, *Estimates of eigenvalues for iterative methods*, Math. Comp., 53 (1989), pp. 619–626.

[19] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 2nd ed., 1989.

[20] A. GREENBAUM, *Comparison of splittings used with the conjugate gradient algorithm*, Numer. Math., 33 (1979), pp. 181–194.

[21] L. A. HAGEMAN AND D. M. YOUNG, *Applied Iterative Methods*, Academic Press, New York, 1981.

[22] M. R. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 409–435.

[23] M. J. HOLST, *private communication*, 1989.

[24] O. G. JOHNSON, C. A. MICCHELLI, AND G. PAUL, *Polynomial preconditioning for conjugate gradient calculations*, SIAM J. Numer. Anal., 20 (1983), pp. 362–376.

[25] T. L. JORDAN, *Conjugate gradient preconditioners for vector and parallel processors*, in Proc. Conference on Elliptic Problem Solvers, G. Birkhoff and A. Schoenstadt, eds., Academic Press, 1984.

[26] 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.

[27] P. D. MEYER, A. J. VALOCCHI, S. F. ASHBY, AND P. E. SAYLOR, *A numerical investigation of the conjugate gradient method as applied to three-dimensional groundwater flow problems in randomly heterogeneous porous media*, Water Resources Res., 25 (1989), pp. 1440–1446.

[28] T. J. Rivlin, *The Chebyshev Polynomials*, John Wiley and Sons, New York, 1974.

[29] H. Rutishauser, *Theory of gradient methods*, in Refined Iterative Methods for Computation of the Solution and the Eigenvalues of Self-Adjoint Boundary Value Problems, Mitt. Inst. angew. Math. ETH Zürich, Nr. 8, Birkhäuser, Basel, 1959, pp. 24–49.

[30] Y. Saad, *Practical use of polynomial preconditionings for the conjugate gradient method*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 865–881.

[31] P. E. Saylor, *Leapfrog variants of iterative methods for linear algebraic equations*, J. Comput. Appl. Math., 24 (1988), pp. 169–193.

[32] G. Szego, *Orthogonal Polynomials*, Colloquium Publications 23, Revised Edition, American Mathematical Society, Providence, RI, 1959.

[33] D. M. Young and K. C. Jea, *Generalized conjugate gradient acceleration of nonsymmetrizable iterative methods*, Linear Algebra Appl., 34 (1980), pp. 159–194.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 1, pp. 30–46, January 1992

002

PRECONDITIONED ITERATIVE METHODS FOR HOMOTOPY CURVE TRACKING*

COLIN DESA[†], KASHMIRA M. IRANI[†], CALVIN J. RIBBENS[†], LAYNE T. WATSON[†], AND HOMER F. WALKER[‡]

Abstract. Homotopy algorithms are a class of methods for solving systems of nonlinear equations that are globally convergent with probability one. All homotopy algorithms are based on the construction of an appropriate homotopy map and then the tracking of a curve in the zero set of this homotopy map. The fundamental linear algebra step in these algorithms is the computation of the kernel of the homotopy Jacobian matrix. Problems with large, sparse Jacobian matrices are considered. The curve-tracking algorithms used here require the solution of a series of very special systems. In particular, each $(n+1) \times (n+1)$ system is in general nonsymmetric but has a leading symmetric indefinite $n \times n$ submatrix (typical of large structural mechanics problems, for example). Furthermore, the last row of each system may by chosen (almost) arbitrarily. The authors seek to take advantage of these special properties. The iterative methods studied here include Craig's variant of the conjugate gradient algorithm and the SYMMLQ algorithm for symmetric indefinite problems. The effectiveness of various preconditioning strategies in this context are also investigated, and several choices for the last row of the systems to be solved are explored.

Key words. globally convergent, homotopy algorithm, nonlinear equations, preconditioned conjugate gradient, homotopy curve tracking, sparse matrix, matrix splitting, bordered matrix

AMS(MOS) subject classifications. 65F10, 65F50, 65H10, 65K10

1. Introduction. The fundamental problem motivating this work is to solve a nonlinear system of equations $F(x) = 0$, where $F : E^n \to E^n$ is a C^2 map defined on real n-dimensional Euclidean space E^n. The homotopy approach to solving $F(x) = 0$ is to construct a continuous map $H(\lambda, x)$, the "homotopy," deforming a simple function $s(x)$ to the given function $F(x)$ as λ varies from 0 to 1. Starting from the easily obtained solution to $H(0,x) = s(x) = 0$, the essence of a homotopy algorithm is to track solutions of $H(\lambda, x) = 0$ until a solution of $H(1,x) = F(x) = 0$ is obtained. The theoretical and implementational details of such algorithms are nontrivial; see Rheinboldt and Burkardt [24] and Watson, Billups, and Morgan [36] for summaries of significant recent progress in this area.

Homotopies are a traditional part of topology and only recently have begun to be used for practical numerical computation. The homotopies considered here are sometimes called "artificial-parameter generic homotopies," in contrast to natural-parameter homotopies, where the homotopy variable is a physically meaningful parameter. In the latter case, the resulting homotopy zero curves must be dealt with as they are, bifurcations, ill-conditioning, and all. Therefore, curve tracking becomes the main focus of the problem-solving effort. Our artificial-parameter generic homotopies require that the homotopy zero curves obey strict smoothness conditions. These conditions generally will not hold if the homotopy parameter represents a physically meaningful quantity, but they can always be obtained via certain generic constructions using an artificial (i.e., nonphysical) homotopy parameter.

* Received by the editors April 5, 1990; accepted for publication (in revised form) December 5, 1990.

[†] Department of Computer Science, Virginia Polytechnic Institute & State University, Blacksburg, Virginia 24061. The work of these authors was supported in part by Department of Energy grant DE-FG05-88ER25068 and Air Force Office of Scientific Research grant 89-0497.

[‡] Department of Mathematics and Statistics, Utah State University, Logan, Utah 84322. The work of this author was supported in part by Department of Energy grant DE-FG02-86ER25018 and National Science Foundation grant DMS-0088995.

The objective is to solve a "parameter-free" system of equations, $F(x) = 0$. What this means is that one can design the homotopy to have paths with nice properties. Thus extra attention is devoted to constructing the homotopy, and the curve-tracking algorithm can be limited to a well-behaved class of curves. The goal of using these artificial-parameter homotopies is to solve fixed-point and zero-finding problems with homotopies whose zero curves do not have bifurcations or other singular or ill-conditioned behavior. The mathematical software package HOMPACK [36] used here for comparative purposes is designed for artificial-parameter generic homotopies.

The theory and algorithms for functions $F(x)$ with small, dense Jacobian matrices $DF(x)$ are well developed [33], [32]. In this paper we focus on large, sparse $DF(x)$, a class of problems about which much less is known. Solving large sparse nonlinear systems of equations via homotopy methods involves sparse rectangular linear systems of equations. The sparsity suggests the use of iterative solution methods. Preconditioning techniques are used to make the iterative methods more efficient. In this paper we are particularly interested in problems where the Jacobian matrix $DF(x)$ is symmetric. Such problems are common, for example, in structural mechanics, where the Jacobian matrix is often the Hessian of a potential energy function.

Section 2 describes the homotopy approach to zero-finding problems and outlines the curve-tracking algorithm used in this paper. Section 3 discusses the linear algebra details of homotopy curve tracking. Several algorithmic possibilities are presented. Section 4 presents numerical results from the application of the various algorithms to three test problems. Some general conclusions from these results are drawn in § 5.

2. Homotopy algorithm.

The philosophy of artificial-parameter homotopy algorithms is to create homotopies whose zero curves are well behaved, with Jacobian matrices that are well conditioned, and that reach a solution for almost all choices of a parameter. These homotopies may be used to solve fixed-point and zero-finding problems.

In this paper we concentrate on the zero-finding problem $F(x) = 0$, where $F : E^n \to E^n$ is a C^2 map. The theoretical basis for the homotopy algorithm can be summarized as follows (see [7] for details). Suppose there exists a C^2 map

$$\rho : E^m \times [0,1) \times E^n \to E^n$$

such that

(a) the $n \times (m + 1 + n)$ Jacobian matrix $D\rho(a, \lambda, x)$ has rank n on the set

$$\rho^{-1}(0) = \{\, (a, \lambda, x) \,|\, a \in E^m, 0 \le \lambda < 1, x \in E^n, \rho(a, \lambda, x) = 0 \,\},$$

and for any fixed $a \in E^m$, letting $\rho_a(\lambda, x) = \rho(a, \lambda, x)$,

(b) $\rho_a(0, x) = \rho(a, 0, x) = 0$ has a unique solution x_0,

(c) $\rho_a(1, x) = F(x)$,

(d) ${\rho_a}^{-1}(0)$ is bounded.

Then for almost all $a \in E^m$ there exists a zero curve γ of ρ_a along which the Jacobian matrix $D\rho_a$ has rank n, emanating from $(0, x_0)$ and reaching a zero $\bar{x}$ of F at $\lambda = 1$. Furthermore, γ does not intersect itself and is disjoint from any other zeros of ρ_a. Thus, with probability one, picking an $a \in E^m$ (which uniquely determines x_0), and following γ from $(0, x_0)$ to $(1, \bar{x})$, leads to a zero $\bar{x}$ of F.

There are many different algorithms for tracking the zero curve γ. HOMPACK supports three such algorithms: ordinary differential equation-based, "normal flow," and "augmented Jacobian matrix." We consider only the normal flow algorithm in this paper. A brief description of the normal flow algorithm is now given.

Consider the homotopy map

$$\rho_a(x,\lambda) = \lambda F(x) + (1-\lambda)(x-a).$$

The matrix $D_x\rho_a(x,\lambda) = \lambda DF(x) + (1-\lambda)I$ is symmetric and sparse. (For the problems of interest, $D_x\rho_a(x,\lambda)$ has a "skyline" structure, and is conveniently stored in a packed skyline format, in which the upper triangle is stored in a one-dimensional indexed array. An auxiliary array of diagonal indices is also required.) Assuming that $F(x)$ is C^2, a is such that the Jacobian matrix $D\rho_a(x,\lambda)$ has full rank along γ, and γ is bounded, the zero curve γ is C^1 and can be parameterized by arc length s. Thus $x = x(s), \lambda = \lambda(s)$ along γ, and

$$\rho_a(x(s),\lambda(s)) = 0$$

identically in s.

The zero curve γ given by $(x(s),\lambda(s))$ is the trajectory of the initial value problem

$$\text{(1)} \qquad \frac{d}{ds}\rho_a\Big(x(s),\lambda(s)\Big) = \Big[D_x\rho_a(x(s),\lambda(s)),\ D_\lambda\rho_a(x(s),\lambda(s))\Big]\begin{pmatrix}\frac{dx}{ds}\\ \frac{d\lambda}{ds}\end{pmatrix} = 0,$$

$$\text{(2)} \qquad \left\|\left(\frac{dx}{ds},\frac{d\lambda}{ds}\right)\right\|_2 = 1,$$

$$\text{(3)} \qquad x(0) = a, \qquad \lambda(0) = 0.$$

Since the Jacobian matrix has rank n along γ, the derivative $(dx/ds, d\lambda/ds)$ is uniquely determined by (1), (2), and continuity, and the initial value problem (1–3) can be solved for $x(s), \lambda(s)$. From (1) it can be seen that the unit tangent $(dx/ds, d\lambda/ds)$ to γ is in the one-dimensional kernel of $D\rho_a$.

The normal flow curve tracking algorithm has four phases: prediction, correction, step size estimation, and computation of the solution at $\lambda = 1$. For the prediction phase, assume that two points $(x(s_1),\lambda(s_1))$ and $(x(s_2),\lambda(s_2))$ on γ with corresponding tangent vectors $(dx/ds(s_1),\ d\lambda/ds(s_1))$, $(dx/ds(s_2),\ d\lambda/ds(s_2))$ 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

$$Z^{(0)} = p(s_2 + h),$$

where $p(s)$ is the Hermite cubic interpolating $(x(s),\lambda(s))$ at s_1 and s_2. Precisely,

$$p(s_1) = (x(s_1),\lambda(s_1)), \qquad p'(s_1) = \left(\frac{dx}{ds}(s_1), \frac{d\lambda}{ds}(s_1)\right),$$

$$p(s_2) = (x(s_2),\lambda(s_2)), \qquad p'(s_2) = \left(\frac{dx}{ds}(s_2), \frac{d\lambda}{ds}(s_2)\right),$$

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

$$\text{(4)} \qquad Z^{(k+1)} = Z^{(k)} - \Big[D\rho_a\big(Z^{(k)}\big)\Big]^+ \rho_a\big(Z^{(k)}\big), \qquad k = 0, 1, \cdots,$$

where $[D\rho_a(Z^{(k)})]^+$ is the Moore–Penrose pseudoinverse of the $n \times (n+1)$ Jacobian matrix $D\rho_a$. Small perturbations of a produce small changes in the trajectory γ, and the family of trajectories γ for varying a is known as the "Davidenko flow." Geometrically, the iterates given by (4) 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

$$[D\rho_a]\Delta Z = -\rho_a. \tag{5}$$

Fortunately, ΔZ can be calculated at the same time as the kernel of $[D\rho_a]$, and with just a little more work. The numerical linear algebra details for solving (5), the optimal step size estimation, and the endgame to obtain the solution at $\lambda = 1$ are described by Watson [35], [34].

The calculation of the implicitly defined derivative $(dx/ds, d\lambda/ds)$ is done by computing the one-dimensional kernel of $D\rho_a$, i.e., by solving the $n \times (n+1)$ linear system

$$[D\rho_a]y = 0. \tag{6}$$

This can be elegantly and efficiently done for small dense matrices, but the large sparse Jacobian matrix presents special difficulties. The difficulty now is that the first n columns of the Jacobian matrix $D\rho_a(x,\lambda)$ involving $DF(x)$ are definitely special, and any attempt to treat all $n+1$ columns uniformly would be disastrous from the point of view of storage allocation. Hence, what is required is a good algorithm for solving nonsquare linear systems of equations (5) and (6), where the leading $n \times n$ submatrix of $D\rho_a$ is symmetric and sparse. This paper considers various iterative methods for solving such linear systems of equations.

3. Numerical linear algebra algorithms. As discussed in §2 for the normal flow curve-tracking algorithm, computing both the corrector step ΔZ and the tangent vector $(dx/ds, d\lambda/ds)$ requires the solution of a rectangular linear system involving the $n \times (n+1)$ matrix $[D\rho_a]$. This section describes various algorithms for the solution of such linear systems. In particular, we first consider various ways to construct an $(n+1) \times (n+1)$ invertible system $Ay = b$ whose solution yields a solution to the rectangular systems (5) and (6). We then describe two general approaches to solving systems involving this augmented matrix A, and within these approaches we consider various iterative solution methods. Finally, several preconditioners are suggested for improving the convergence of the iterative methods.

3.1. Defining an invertible system. Let $(\bar{x}, \bar{\lambda})$ be a point on the zero curve γ, and $\bar{y}$ the unit tangent vector to γ at $(\bar{x}, \bar{\lambda})$ in the direction of increasing arc length s. Then the matrix

$$A = \begin{pmatrix} D_x\rho_a(x,\lambda) & D_\lambda\rho_a(x,\lambda) \\ c^t & d \end{pmatrix},$$

where $(c^t \quad d)$ is any vector outside a set of measure zero (a hyperplane), is invertible at $(\bar{x}, \bar{\lambda})$ and in a neighborhood of $(\bar{x}, \bar{\lambda})$. Thus the kernel of $D\rho_a$ for (x, λ) near $(\bar{x}, \bar{\lambda})$ can be found by solving the linear system of equations

$$Ay = \alpha e_{n+1} = b, \tag{7}$$

where $(c^t \quad d)\bar{y} = \alpha$, and e_{n+1} is the $(n+1)$st standard basis vector. Similarly, the corrector step can be found by solving

$$A\Delta Z = \begin{bmatrix} -\rho_a \\ 0 \end{bmatrix}. \tag{8}$$

The coefficient matrix A in the linear systems of equations (7) and (8) has a very special structure, which can be exploited in several ways. Recall that the leading $n \times n$ submatrix of A is $D_x\rho_a$, which is symmetric and sparse, but possibly indefinite. We shall attempt to exploit this property below. The choice of the last row $(c^t \quad d)$ is considered first.

From an implementation point of view, the easiest choice for $(c^t \quad d)$ is probably the kth standard basis vector e_k^t, where the index k is defined by $|\bar{y}_k| = \max_i |\bar{y}_i|$. This is the choice made in HOMPACK. It is not difficult to show that with this choice for $(c^t \quad d)$, A is invertible, though clearly not symmetric.

A second choice for the last row of A considered here is $(c^t \quad d) = \bar{y}^t$. This choice would seem to be best from a conditioning standpoint, since $\bar{y}$ is orthogonal to the rows of $D\rho_a(\bar{x}, \bar{\lambda})$, and hence one expects $\bar{y}$ to be nearly orthogonal to the rows of $D\rho_a(x,\lambda)$ for (x,λ) near $(\bar{x},\bar{\lambda})$. It is clear that choosing $(c^t \quad d) = \bar{y}^t$ also makes A nonsymmetric, and in addition introduces a dense row into the otherwise sparse matrix A (the last column $D_\lambda\rho_a$ is also dense).

Since symmetry is advantageous for some algorithms, A can be made symmetric and invertible by a third choice $c = D_\lambda\rho_a$. The scalar d must still be chosen so that rank $A = n+1$. It is enough to consider two cases:

1. Suppose rank $D_x\rho_a = n-1$. Then $D_\lambda\rho_a$ is not a linear combination of the columns of $D_x\rho_a$, because rank $\left[D_x\rho_a \; D_\lambda\rho_a\right] = n$ by the homotopy theory. Thus $c^t = \left(D_\lambda\rho_a\right)^t$ is not a linear combination of the rows of the symmetric matrix $D_x\rho_a$, and we have
$$\text{row rank}\begin{bmatrix} D_x\rho_a \\ (D_\lambda\rho_a)^t \end{bmatrix} = n.$$
Finally, $(c^t \quad d)^t$ is not a linear combination of the first n columns of A, for *any* choice of d, so the column rank of A is $n+1$.
2. Now suppose that rank $D_x\rho_a = n$. Then
$$\text{rank}\begin{bmatrix} D_x\rho_a \\ \left(D_\lambda\rho_a\right)^t \end{bmatrix} = n,$$
and it suffices to choose d to make the last column of A independent from the first n columns. $D_\lambda\rho_a$ is a unique linear combination of the columns of $D_x\rho_a$, and any choice of d other than this combination of the components of $\left(D_\lambda\rho_a\right)^t$ will make the $(n+1)$st column independent. Let $\bar{A}$ denote A at $(\bar{x},\bar{\lambda})$. Since dim[ker$(\bar{A})$] ≤ 1, $\bar{A}y = 0$ implies $y = \alpha\bar{y}$, and thus with $\bar{y}^t = \left(\hat{y}^t, \bar{y}_{n+1}\right)$, $\left(D_\lambda\rho_a(\bar{x},\bar{\lambda})\right)^t\hat{y} + d\,\bar{y}_{n+1} = 0$. Choosing any $\beta \neq 0$ and solving $\left(D_\lambda\rho_a(\bar{x},\bar{\lambda})\right)^t\hat{y} + d\,\bar{y}_{n+1} = \beta$ for d ($\bar{y}_{n+1} \neq 0$ since rank $D_x\rho_a(\bar{x},\bar{\lambda}) = n$) gives a d such that rank $A = n+1$ for (x,λ) near $(\bar{x},\bar{\lambda})$.

Observe also that if $D_x\rho_a$ is positive definite, choosing $d > 0$ sufficiently large guarantees that

$$A = \begin{pmatrix} D_x\rho_a & D_\lambda\rho_a \\ \left(D_\lambda\rho_a\right)^t & d \end{pmatrix}$$

is also positive definite.

Proof. Since A is symmetric, by Sylvester's theorem A is positive definite if and only if all its leading principal minors are positive. Since $D_x\rho_a$ is positive definite, the first n leading principal minors are positive, and it suffices to show $\det A > 0$. Expanding $\det A$ along the last column,

$$\det A = d \cdot \det D_x\rho_a + \text{terms not involving } d > 0$$

for $d > 0$ sufficiently large. □

3.2. Splitting vs. direct approaches. Given a choice for the last row $(c^t \quad d)$, we now consider two general approaches to solving systems of the form (7) and (8). The first approach deals with the entire matrix A directly. As we have seen, depending on the choice of last row, it may be that A is nonsymmetric. This immediately eliminates several iterative solvers, at least for these cases. However, we do consider a few versions of this approach where possible in the experiments reported in § 4.

The second general approach is to attack (7) and (8) indirectly as follows. Split A into the sum of a symmetric matrix M and a low rank modification L:

$$A = M + L, \tag{9}$$

where

$$M = \begin{pmatrix} D_x\rho_a & c \\ c^t & d \end{pmatrix}, \tag{10}$$

$$L = ue_{n+1}^t, \qquad u = \begin{pmatrix} D_\lambda\rho_a - c \\ 0 \end{pmatrix}. \tag{11}$$

Observe that for almost all choices of $(c^t \quad d)$ the symmetric part M is also invertible. Then using the Sherman–Morrison formula, the solution y to the original system $Ay = b$ can be obtained from

$$y = \left[I - \frac{M^{-1}ue_{n+1}^t}{(M^{-1}u)^t e_{n+1} + 1} \right] M^{-1}b. \tag{12}$$

Equation (12) requires the solution of two linear systems involving the sparse (except possibly for c), symmetric, invertible matrix M. The scheme (9)–(12) was proposed by Kamat, Watson, and Junkins [20], and further investigated by Chan and Saad [6].

A third general approach, not considered here, would be a block-elimination approach that depends on solving systems involving the $n \times n$ matrix $D_x\rho_a$. Observe that block elimination will frequently fail in the homotopy context, because even though rank $A = n + 1$ and rank $(D_x\rho_a \quad D_\lambda\rho_a) =$ rank $D\rho_a = n$, it may very well happen that $D_x\rho_a$ is singular (i.e., rank $= n - 1$). Block-elimination strategies are considered by Chan [4], [3] and Chan and Resasco [5].

3.3. Iterative solution methods. In § 4 we report results from experiments using two iterative solution methods: Craig's method and SYMMLQ. Craig's method is nearly equivalent to the method of conjugate gradients (CG) [17] applied to the normal equations. (Technically, a different norm is minimized over a different Krylov space than if CG were applied directly to the normal equations.) It has long been known that one way to apply CG to nonsymmetric problems is to solve the normal equations instead of the original system. In particular, given any nonsingular matrix A, the system of linear equations $Ay = b$ can be solved by considering the linear system (normal equations)

$$A^t Ay = A^t b,$$

or the related system

$$AA^t z = b, \qquad y = A^t z.$$

Since the coefficient matrix for the latter system is both symmetric and positive definite, the system can be solved by the CG algorithm. Once a solution vector z is obtained, the vector y from the original system can be computed as $y = A^t z$. A major disadvantage of this technique is that the convergence rate depends on $\mathrm{cond}(AA^t) = (\mathrm{cond}(A))^2$ rather than $\mathrm{cond}(A)$. An implementation of the CG algorithm in which y is computed directly, without reference to z or AA^t, is due to Craig [9] and is described in [13] and [16]. Despite the efficiency of the implementation, the convergence rate still depends on $\mathrm{cond}(AA^T) = (\mathrm{cond}(A))^2$ in general. The cost per iteration of Craig's method is dominated by two matrix-vector products, one involving A and one involving A^t. Preconditioning (see § 3.4) increases the work per iteration substantially.

The second solution method considered here is the SYMMLQ algorithm described in Paige and Saunders [23]. SYMMLQ solves symmetric indefinite systems. It is based on a variant of the Lanczos procedure for tridiagonalizing a symmetric matrix. In [23] it is shown that for symmetric positive definite systems, SYMMLQ is mathematically equivalent to CG. However, unlike CG, which can break down when A is indefinite, SYMMLQ is well defined and numerically stable in this case. Like CG, the cost of one iteration of unpreconditioned SYMMLQ is primarily in a single matrix-vector multiplication.

There are many other CG-like methods (i.e., Krylov subspace methods) for solving nonsymmetric or indefinite problems, but most are not satisfactory in our context. The generalized conjugate gradient method of Concus and Golub [8] and Widlund [37] applies only to matrices with positive definite symmetric part (i.e., the matrix must be positive definite, but not necessarily symmetric), although with preconditioning it can sometimes be used to solve more general problems. The generalized conjugate residual method [12] and ORTHOMIN(k) [29] also may break down if the coefficient matrix is not positive definite. More general systems $Ax = b$, where A is not positive definite, can sometimes be solved by ORTHOMIN(k) if a nonsingular matrix Z is known such that ZA is positive definite. ORTHOMIN(k) is then applied to the transformed system $ZAx = Zb$. A related method known as ORTHODIR [38] does not break down in case A is indefinite, but it is observed to have stability problems [26]. ORTHORES is another method with similar properties. GMRES(k) [28], [31], like ORTHOMIN(k), is guaranteed to converge when the coefficient matrix is positive definite. However, for an indefinite coefficient matrix, GMRES(k), while it does not break down, may fail because the residual norms at each step, although nonincreasing, do not converge to zero. Other Krylov subspace methods are studied by Axelsson [1]; Dennis and Turner [10]; Eisenstat, Elman, and Schultz [11]; Jea [19]; Saad [25]; and Saad and Schultz [27].

Finally, there are other efficient iterative methods for solving sparse linear systems based on matrix splittings (see Hageman and Young [15]). Typical of these is the SSOR method, defined in terms of the splitting $A = D - L - U$, where D is the diagonal of A, L is the strict lower triangle of A, and U is the strict upper triangle of A. The method requires computations involving D^{-1}. In the homotopy context, D^{-1} frequently does not exist, and a diagonal matrix Σ such that $\left[\text{diag }(A+\Sigma)\right]^{-1}$ does exist may not be of low rank. Consequently SSOR and methods based on similar splittings are of limited utility in the present context.

3.4. Preconditioners. It is widely known that "preconditioning" can dramatically improve the performance of many iterative methods. For example, the solution to $Ax = b$ can also be obtained by solving the system

$$\tilde{A}x = (Q^{-1}A)x = Q^{-1}b = \tilde{b},$$

where Q is the so-called *preconditioner.* The goal of preconditioning is to decrease the computational effort required to solve systems of linear equations by increasing the rate of convergence of an iterative method. For preconditioning to be effective, faster convergence must outweigh the costs of applying the preconditioner, so that the total cost of solving the linear system is lower. The preconditioned coefficient matrix $\tilde{A}$ usually is not explicitly computed or stored, since although A is sparse, $\tilde{A}$ may not be. The extra work of preconditioning, then, occurs in solving systems involving the matrix Q. The main storage cost for preconditioning is usually for an extra array to hold a factorization of Q. In this paper we consider two preconditioners:

Gill–Murray (GM). The preconditioner is taken as the modified Choleksy factorization GG^t of a symmetric matrix A (see Gill and Murray [14]). In particular, if A is "sufficiently" positive definite, then $GG^t = A$. Otherwise $GG^t = A + \Sigma$, where Σ is diagonal with nonnegative diagonal entries. In the matrix splitting approach described in § 3.2, we apply GM preconditioning to the symmetric matrix M; in the direct approach we apply GM to the entire matrix A (necessitating the choice $c = D_\lambda \rho_a$ to make A symmetric). We apply the GM preconditioner on the left when Craig's method is used (i.e., if A is the original matrix, $(GG^t)^{-1}A$ is the preconditioned matrix). In the case of SYMMLQ, the preconditioned system is $G^{-1}A(G^t)^{-1}$, since SYMMLQ requires symmetry.

ILU. The incomplete LU factorization described in [22] computes a lower triangular matrix L and unit upper triangular matrix U satisfying

$$\begin{cases} L_{ij} = U_{ij} = 0, & (i,j) \in Z, \\ (LU)_{ij} = A_{ij}, & (i,j) \notin Z. \end{cases}$$

Here, Z is the set of indices where A is known to be zero off the diagonal. (This method is often referred to as ILU(0) to indicate that 0 fill-in is allowed.) It is possible that $L_{ii} = 0$ in this algorithm. In this case L_{ii} is set to a small positive number, in which case $(LU)_{ii} \neq A_{ii}$. The ILU factorization is modified slightly so that it may be used with SYMMLQ. In this case we compute an incomplete LDL^T factorization, and apply $LD^{1/2}$ symmetrically as a preconditioner as described above. If during the factorization procedure, an element of the diagonal matrix D is negative, we simply take its absolute value. This is very similar to the GM factorization, except with ILU we do no extra work to ensure that the factorization is well conditioned.

4. Numerical experiments. Of the various algorithmic possibilities mentioned in the previous section, we consider further 22 distinct combinations. Some possibilities do not make sense or are impractical in the homotopy context, and thus are not considered. Ignoring the choice for the last row of A, and also ignoring the question of

TABLE 1
Execution time in seconds for turning point problem.

	CR			CRILU			CRGM
n	e_k	$\bar{y}$	$D_\lambda\rho_a$	e_k	$\bar{y}$	$D_\lambda\rho_a$	$D_\lambda\rho_a$
20	17	19	21	4	7	4	5
60	167	176	186	13	22	13	22
125	1117	1132	1384	38	64	42	85
250	2296	1925	3873	66	110	74	134
500	4741	3899	8352	129	210	148	260
1000	11577	9335	20375	323	493	353	617

TABLE 2
Execution time in seconds for turning point problem.

	CR-S		CRILU-S		CRGM-S	
n	e_k	$\bar{y}$	e_k	$\bar{y}$	e_k	$\bar{y}$
20	28	36	6	6	12	13
60	266	356	20	22	41	50
125	1635	2310	54	65	127	170
250	3026	3767	95	109	228	267
500	6279	7783	189	207	448	501
1000	14150	17768	434	490	1077	1174

splitting, we have six basic methods: unpreconditioned Craig's method (CR), Craig's method with ILU preconditioning (CRILU), Craig's method with GM preconditioning (CRGM), unpreconditioned SYMMLQ (SY), SYMMLQ with ILU preconditioning (SYILU), and SYMMLQ with GM preconditioning (SYGM). Data for all of these methods, with both the splitting and direct approaches, and for various choices of the last row, are given in Tables 1–24. In the splitting cases, the method names have an -S appended (e.g., CR-S, SY-S, ...). We report results from all of these methods on three test problems, which are now briefly described.

Turning point problem. The turning point problem is a relatively simple (and artificial) example derived from the system of equations

$$F(\mathbf{x}) = \left(F_1(\mathbf{x}), F_2(\mathbf{x}), \cdots, F_N(\mathbf{x})\right)^t = 0$$

where

$$F_i(\mathbf{x}) = \tan^{-1}\left(\sin[x_i(i \bmod 100)]\right) - \frac{(x_{i-1} + x_i + x_{i+1})}{20}, \qquad i = 1, \cdots, N,$$

and $x_0 = x_{N+1} = 0$. The zero curve γ tracked from $\lambda = 0$ to $\lambda = 1$ corresponds to $\rho_a(x, \lambda) = (1 - .8\lambda)(x - a) + .8\lambda F(x)$, where a is chosen artificially to produce turning points in γ. The Jacobian matrix $D_x\rho_a$ for the turning point problem is tridiagonal. Tables 1–4 and 13–16 contain the data for this problem.

Shallow arch problem. This is a relatively small but quite difficult problem from structural mechanics. It results from solving the equilibrium equations for a discretization of a shallow arch under an externally applied load. See [21] and [18] for a more complete description. Although this problem is small, it is included in this study because it is a good test of the accuracy of our methods. To go through the limit point and along the unloading portion of the equilibrium curve requires very accurate Jacobian matrices and numerical linear algebra. In fact, the standard iterative linear equation solver used in HOMPACK is unable to go past the limit point without tweaking the HOMPACK step size control parameters. $D_x\rho_a$ for the

TABLE 3
Execution time in seconds for turning point problem.

n	SY	SYILU	SYGM
	$D_\lambda\rho_a$	$D_\lambda\rho_a$	$D_\lambda\rho_a$
20	12	5	5
60	87	17	20
125	405	54	61
250	701	89	104
500	1376	174	199
1000	3270	400	457

TABLE 4
Execution time in seconds for turning point problem.

n	SY-S		SYILU-S		SYGM-S	
	e_k	$\bar{y}$	e_k	$\bar{y}$	e_k	$\bar{y}$
20	20	23	9	9	9	9
60	134	165	31	32	33	34
125	594	738	98	101	105	108
250	1030	1202	164	165	175	175
500	2109	2421	315	320	332	337
1000	4872	5500	736	738	765	787

shallow arch problem has bandwidth 5. Tables 5–8 and 17–20 contain the data for this problem.

Shallow dome problem. This is another realistic problem from structural mechanics, in which the equations of equilibrium for a model of a shallow dome must be solved. See [18] for a more complete description. $D_x\rho_a$ for the shallow dome problem is block diagonal, with dense 21×21 blocks. Tables 9–12 and 21–24 contain the data for this problem.

The times reported in Tables 1–12 are for tracking the entire zero curve γ and thus represent the solution of many linear systems of varying degrees of difficulty. The average, maximum, and minimum number of iterations for each method (Craig's and SYMMLQ) are reported in Tables 13–24. The experiments are done in double precision using a single processor of a Sequent Symmetry S81 multiprocessor. The major headings are the acronyms for the algorithms, and the subheadings denote the choice $(c^t \quad d)$ for the last row of A. There is asymmetry in the tables because some possibilities do not make sense. For instance, there is no CRGM with e_k because the Gill–Murray preconditioner requires a symmetric matrix; and there are no methods based on splitting when $c = D_\lambda\rho_a$, since this choice makes A symmetric, so there is no need to split A into the sum of a symmetric matrix and a low rank modification.

5. Discussion and conclusions. Regarding the choice of last row $(c^t \quad d)$, Tables 1–12 show that there is no clear winner between e_k, $\bar{y}$, and $D_\lambda\rho_a$. Furthermore, there seems to be little correlation between the algorithm and the best choice for c. If anything, a weak conclusion—that all other things being equal, the best choice is e_k—seems to be indicated by the data. Apparently better conditioning (from $\bar{y}$) or symmetry (from $D_\lambda\rho_a$) does not compensate for the extra work involved in these choices compared with e_k.

A comparison of the direct with the splitting approach results in a slight preference for the direct approach. However, the advantage is not a strong one in most cases. In fact, there are cases where the splitting approach is better; and on the shallow arch problem there is virtually no difference. The average number of iterations is

TABLE 5
Execution time in seconds for shallow arch problem.

	CR			CRILU			CRGM
n	e_k	$\bar{y}$	$D_\lambda \rho_a$	e_k	$\bar{y}$	$D_\lambda \rho_a$	$D_\lambda \rho_a$
29	856	884	919	533	458	443	464
47	14205	13591	14606	5794	5807	6776	6921

TABLE 6
Execution time in seconds for shallow arch problem.

	CR-S		CRILU-S		CRGM-S	
n	e_k	$\bar{y}$	e_k	$\bar{y}$	e_k	$\bar{y}$
29	1108	947	599	470	468	818
47	16904	17593	5674	5957	7322	10105

considerably lower for many of the splitting cases than for the corresponding direct case. This helps explain why the direct methods are often not significantly faster, despite the fact that they solve half as many linear systems.

Regarding a comparison between the two basic iterative schemes, the data indicate that unpreconditioned SYMMLQ is faster than unpreconditioned Craig's method, often by a significant amount. The advantage of SYMMLQ seems to be both in fewer iterations and in less work per iteration (one less matrix-vector product). With GM preconditioning, SYMMLQ is still a bit faster than Craig's method for the turning point and shallow dome problems; on the shallow arch problem the two perform roughly the same. When ILU preconditioning is used, Craig's method appears to be superior. It performs slightly better on the turning point and shallow arch problems; and ILU preconditioning combined with SYMMLQ fails completely on the shallow dome problem (SYMMLQ is not converging to a solution of some of the linear systems, and consequently the curve-tracking algorithm does not make progress).

It is tempting to conclude from the data that the best method overall is CRILU. However, it must be pointed out that the ILU factorization fails to exist at turning points and is unstable whenever A is indefinite (as is illustrated by SYILU on the shallow dome problem). We encountered other homotopy curve-tracking runs, on slightly different problems, which failed because the ILU preconditioner failed to exist or generated an overflow, or because of the difficulty caused HOMPACK by inaccurate tangents resulting from ILU. Because of this potential catastrophic failure or instability, it is difficult to seriously consider the ILU preconditioner for use in robust homotopy software. Still, the data dó show why the concern of numerical analysts about unstable algorithms is not always shared by others.

The GM preconditioner, meanwhile, is fairly competitive with ILU on the turning point and shallow arch problems. Furthermore, it is more robust in the presence of turning points and when A becomes indefinite. However, the data for the shallow dome problem show that the GM preconditioner may do a very poor job indeed at a few points on the curve. Tables 21–24 indicate that while the average number of iterations is reduced by using the GM preconditioner, the maximum number can actually increase. Thus the net improvement in efficiency is not at all impressive.

The algorithms SSOR and ORTHOMIN(k), discussed earlier, are not shown in the tables because they totally fail at turning points and along unloading portions of equilibrium curves (for reasons stated in § 3). When these methods do work, they can be very efficient (e.g., ORTHOMIN(1) on A with $c = D_\lambda \rho_a$ took 443 (6092) seconds for the shallow arch problem with $n = 29$ (47)), but that is no consolation for homo-

TABLE 7
Execution time in seconds for shallow arch problem.

	SY	SYILU	SYGM
n	$D_\lambda \rho_a$	$D_\lambda \rho_a$	$D_\lambda \rho_a$
29	635	488	463
47	7343	5350	6277

TABLE 8
Execution time in seconds for shallow arch problem.

	SY-S		SYILU-S		SYGM-S	
n	e_k	$\bar{y}$	e_k	$\bar{y}$	e_k	$\bar{y}$
29	615	664	464	499	500	537
47	8992	8362	5593	5683	5760	6506

TABLE 9
Execution time in seconds for shallow dome problem.

	CR			CRILU			CRGM
n	e_k	$\bar{y}$	$D_\lambda \rho_a$	e_k	$\bar{y}$	$D_\lambda \rho_a$	$D_\lambda \rho_a$
21	46	47	47	16	16	16	89
546	2495	2545	2573	355	369	365	2233
1050	4504	4691	4690	632	665	651	4313

TABLE 10
Execution time in seconds for shallow dome problem.

	CR-S		CRILU-S		CRGM-S	
n	e_k	$\bar{y}$	e_k	$\bar{y}$	e_k	$\bar{y}$
21	57	86	21	25	108	57
546	3127	4803	492	630	2710	1787
1050	5615	8553	887	1133	5107	3177

TABLE 11
Execution time in seconds for shallow dome problem.

	SY	SYILU	SYGM
n	$D_\lambda \rho_a$	$D_\lambda \rho_a$	$D_\lambda \rho_a$
21	22	∞	29
546	957	∞	693
1050	1743	∞	1276

TABLE 12
Execution time in seconds for shallow dome problem.

	SY-S		SYILU-S		SYGM-S	
n	e_k	$\bar{y}$	e_k	$\bar{y}$	e_k	$\bar{y}$
21	35	44	∞	∞	41	35
546	1420	2027	∞	∞	1052	928
1050	2529	3690	∞	∞	1902	1629

topy curve tracking.

GMRES(k) has a solid theoretical justification and has been used very successfully in a variety of contexts [28], [2], [31], [30]. Nevertheless, GMRES(k) with $k < n$ performed unacceptably on the test problems here, at least without preconditioning. For the shallow arch problem with $n = 29$ and tol $= 10^{-12}$, GMRES(29) on A with $c = D_\lambda \rho_a$ took 591 seconds, comparable to CRGM and CRGM-S. For $k = 1$, 3, 25, GMRES(k) took over a day of CPU time. Relaxing the tolerance to 10^{-6}, GMRES(25) took 18,330 seconds. This is especially noteworthy because the A matrices are (theoretically) symmetric and positive definite. For the turning point problem with $n = 20$, tol $= 10^{-12}$, $c = D_\lambda \rho_a$, the performance degradation from the full GMRES to GMRES(k) was dramatic. With $k = 20$, 19, 18, 15, 10, 8, GMRES(k) took, respectively, 19, 117, 154, 375, 338, 420 seconds. Thus for these problems, without preconditioning, only the full GMRES method is competitive. Consequently GMRES(k) was not included in the tables.

There are some theoretical results concerning the convergence of GMRES(k) given in [28, §3.4]. These results give worst-case bounds on the rate of residual norm reduction which are determined by the distribution of eigenvalues of A. For the shallow arch and turning point problems, the eigenvalues of A were determined numerically along the homotopy curve, and the resulting bounds were often (although not in every case) found to guarantee only hopelessly slow residual norm reduction, indeed, often to guarantee no residual norm reduction at all even when $k = n$.

Actually, it is apparent from the data that a crucial advantage of Craig's method over methods such as GMRES(k) and ORTHOMIN(k) is that it can iterate indefinitely, if necessary long after the solution would have been reached in exact arithmetic, without incurring increasing costs per iteration and without restarting or otherwise losing information from earlier iterations. Furthermore, there are theoretical guarantees that Craig's method will make progress at each iteration, whereas GMRES(k) may fail to make any progress at all if A is indefinite. It is possible that if the number of iterations necessary to meet the stopping tolerance could be kept small through preconditioning, then GMRES(k) would be competitive for $k < n$. A complete study, similar to that done here for Craig's method, of GMRES(k) with preconditioning and polynomial acceleration would be interesting and will be the topic of a future paper.

Tables 13–24 show the average, maximum, and minimum number of iterations per linear system solution along the homotopy zero curve γ for the three problems, using the same algorithms as in Tables 1–12. Such iteration statistics give an intuitive feel for how the algorithms behave and are sometimes very revealing. For example, Tables 13 and 14 show that symmetry does improve the algorithms' efficiency (compare CR and CR-S with last row e_k^t), and that, all other things being equal, achieving symmetric coefficient matrices is worthwhile. (The algorithms based on splitting to achieve symmetry are not uniformly better, because all other things are not equal.) Note that in all cases (except for the shallow dome problem with GM preconditioning) the maximum number of iterations is less than or equal to four times the average, which says that the convergence behavior is fairly consistent. On the other hand the range between the minimum and maximum is as great as 3 to 536, showing that there is a wide variation in the difficulty of the linear systems encountered along γ.

A succinct, albeit oversimplified, summary of the discussion is that ILU preconditioning is the most efficient, but it may completely fail for some cases, while the Gill–Murray preconditioner rarely fails but may be considerably slower on extremely difficult problems.

TABLE 13

Average, maximum, and minimum number of iterations per linear system along homotopy curve for turning point problem.

	CR			CRILU			CRGM
n	e_k	$\bar{y}$	$D_\lambda \rho_a$	e_k	$\bar{y}$	$D_\lambda \rho_a$	$D_\lambda \rho_a$
20	24,29,1	24,28,1	26,31,1	2,2,1	4,5,1	2,3,1	3,9,2
60	70,86,1	69,84,1	74,91,2	2,3,1	4,7,1	2,3,2	3,12,1
125	159,292,1	151,232,1	179,328,3	2,3,1	4,5,1	2,3,2	5,15,2
250	196,404,1	150,246,1	231,407,3	2,3,1	4,5,1	2,3,2	4,15,2
500	216,427,1	165,337,1	268,489,3	2,3,1	4,6,1	2,3,2	5,16,2
1000	224,446,1	164,323,1	285,536,3	2,3,1	4,5,1	3,3,2	5,16,2

TABLE 14

Average, maximum, and minimum number of iterations per linear system along homotopy curve for turning point problem.

	CR-S		CRILU-S		CRGM-S	
n	e_k	$\bar{y}$	e_k	$\bar{y}$	e_k	$\bar{y}$
20	21,28,1	24,29,1	2,2,1	2,2,1	4,6,1	5,7,1
60	60,100,1	69,87,1	2,3,1	2,3,1	4,8,1	5,8,1
125	127,261,1	154,264,1	2,3,1	2,3,1	5,9,1	6,11,1
250	139,302,1	150,246,1	2,2,1	2,3,1	5,11,1	5,10,1
500	149,314,1	164,281,1	2,2,1	2,3,1	5,11,1	5,10,1
1000	151,312,1	162,289,1	2,2,1	2,3,1	5,11,1	5,11,1

TABLE 15

Average, maximum, and minimum number of iterations per linear system along homotopy curve for turning point problem.

	SY	SYILU	SYGM
n	$D_\lambda \rho_a$	$D_\lambda \rho_a$	$D_\lambda \rho_a$
20	22,28,2	2,5,1	2,7,2
60	48,70,2	2,7,2	2,9,2
125	77,123,3	3,9,1	3,11,2
250	75,131,3	2,9,1	3,11,2
500	80,146,3	2,8,1	3,11,2
1000	83,156,3	3,8,1	3,11,2

TABLE 16

Average, maximum, and minimum number of iterations per linear system along homotopy curve for turning point problem.

	SY-S		SYILU-S		SYGM-S	
n	e_k	$\bar{y}$	e_k	$\bar{y}$	e_k	$\bar{y}$
20	18,25,0	21,25,0	2,5,0	2,6,0	2,5,0	2,5,0
60	37,75,0	46,61,0	2,6,0	3,7,0	2,5,0	3,7,0
125	56,116,0	71,108,0	3,6,0	3,9,0	3,7,0	3,9,0
250	58,118,0	68,100,0	3,8,0	3,8,0	3,7,0	3,8,0
500	61,122,0	72,107,0	3,8,0	3,7,0	3,7,0	3,8,0
1000	62,127,0	71,106,0	3,9,0	3,8,0	3,8,0	3,8,0

TABLE 17

Average, maximum, and minimum number of iterations per linear system along homotopy curve for shallow arch problem.

	CR			CRILU			CRGM
n	e_k	$\bar{y}$	$D_\lambda \rho_a$	e_k	$\bar{y}$	$D_\lambda \rho_a$	$D_\lambda \rho_a$
29	99,127,51	91,107,38	98,120,52	3,3,2	4,5,2	3,3,2	6,7,2
47	265,360,109	239,305,133	265,355,105	3,3,2	4,4,2	3,3,2	6,7,2

TABLE 18

Average, maximum, and minimum number of iterations per linear system along homotopy curve for shallow arch problem.

	CR-S		CRILU-S		CRGM-S	
n	e_k	$\bar{y}$	e_k	$\bar{y}$	e_k	$\bar{y}$
29	66,109,1	66,101,1	2,3,1	3,3,1	4,10,1	28,40,1
47	190,313,1	194,291,1	2,3,1	3,3,1	5,10,1	37,53,1

TABLE 19

Average, maximum, and minimum number of iterations per linear system along homotopy curve for shallow arch problem.

	SY	SYILU	SYGM
n	$D_\lambda \rho_a$	$D_\lambda \rho_a$	$D_\lambda \rho_a$
29	58,79,37	2,5,2	2,4,2
47	115,152,72	3,5,2	3,5,2

TABLE 20

Average, maximum, and minimum number of iterations per linear system along homotopy curve for shallow arch problem.

	SY-S		SYILU-S		SYGM-S	
n	e_k	$\bar{y}$	e_k	$\bar{y}$	e_k	$\bar{y}$
29	39,78,0	42,74,0	2,7,0	4,7,0	2,5,0	10,12,0
47	91,150,0	82,147,0	2,7,0	4,7,0	2,7,0	12,16,0

TABLE 21

Average, maximum, and minimum number of iterations per linear system along homotopy curve for shallow dome problem.

	CR			CRILU			CRGM
n	e_k	$\bar{y}$	$D_\lambda \rho_a$	e_k	$\bar{y}$	$D_\lambda \rho_a$	$D_\lambda \rho_a$
21	26,36,14	26,36,14	26,36,14	2,3,2	2,3,2	2,3,2	23,113,2
546	58,81,17	57,82,17	58,82,18	2,3,2	2,3,2	2,3,2	23,111,2
1050	58,87,18	59,91,18	58,83,18	2,3,2	2,3,2	2,3,2	23,113,2

TABLE 22

Average, maximum, and minimum number of iterations per linear system along homotopy curve for shallow dome problem.

	CR-S		CRILU-S		CRGM-S	
n	e_k	$\bar{y}$	e_k	$\bar{y}$	e_k	$\bar{y}$
21	17,31,1	24,36,1	2,3,1	2,3,1	17,118,1	7,46,1
546	38,75,1	54,87,1	2,3,1	3,3,1	15,113,1	9,63,1
1050	38,76,1	53,91,1	2,3,1	3,3,1	16,114,1	8,101,1

TABLE 23

Average, maximum, and minimum number of iterations per linear system along homotopy curve for shallow dome problem.

	SY	SYILU	SYGM
n	$D_\lambda\rho_a$	$D_\lambda\rho_a$	$D_\lambda\rho_a$
21	17,32,10	∞	7,23,2
546	34,55,11	∞	6,33,2
1050	34,52,11	∞	7,35,2

TABLE 24

Average, maximum, and minimum number of iterations per linear system along homotopy curve for shallow dome problem.

	SY-S		SYILU-S		SYGM-S	
n	e_k	$\bar{y}$	e_k	$\bar{y}$	e_k	$\bar{y}$
21	14,34,0	19,32,0	∞	∞	5,33,0	4,17,0
546	25,58,0	37,69,0	∞	∞	6,40,0	4,18,0
1050	24,54,0	36,64,0	∞	∞	6,40,0	4,23,0

REFERENCES

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

[2] P. N. BROWN AND A. C. HINDMARSH, *Reduced storage matrix methods in stiff ode systems*, J. Appl. Math. Comp., 31 (1989), pp. 40–91.

[3] T. F. CHAN, *Deflated decomposition of solutions of nearly singular systems*, Tech. Report 225, Department of Computer Science, Yale University, New Haven, CT, 1982.

[4] ———, *Deflation techniques and block-elimination algorithms for solving bordered singular systems*, Tech. Report 226, Department of Computer Science, Yale University, New Haven, CT, 1982.

[5] T. F. CHAN AND D. C. RESASCO, *Generalized deflated block-elimination*, Tech. Report 337, Department of Computer Science, Yale University, New Haven, CT, 1985.

[6] T. F. CHAN AND Y. SAAD, *Iterative methods for solving bordered systems with applications to continuation methods*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 438–451.

[7] 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.

[8] P. CONCUS AND G. H. GOLUB, *A generalised conjugate gradient method for nonsymmetric systems of linear equations*, in Lecture Notes in Economics and Mathematical Systems, 134, R. Glowinski and J. L. Lions, eds., Springer-Verlag, Berlin, 1976, pp. 56–65.

[9] E. J. CRAIG, *Iteration procedures for simultaneous equations*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, 1954.

[10] J. E. DENNIS, JR. AND K. TURNER, *Generalized conjugate directions*, Linear Algebra Appl., 88/89 (1987), pp. 187–209.

[11] S. C. EISENSTAT, H. C. ELMAN, AND M. H. SCHULTZ, *Variational iterative methods for nonsymmetric systems of linear equations*, SIAM J. Numer. Anal., 5 (1983), pp. 345–357.

[12] H. C. ELMAN, *Iterative methods for large, sparse, nonsymmetric systems of linear equations*, Ph.D. thesis, Yale University, New Haven, CT, 1982.

[13] D. K. FADEEV AND V. N. FADEEVA, *Computational Methods of Linear Algebra*, Freeman, London, 1963.

[14] P. E. GILL AND W. MURRAY, *Newton-type methods for unconstrained and linearly constrained optimization*, Math. Programming, 28 (1974), pp. 311–350.

[15] L. A. HAGEMAN AND D. M. YOUNG, *Applied Iterative Methods*, Academic Press, New York, 1981.

[16] M. R. HESTENES, *The conjugate-gradient method for solving linear equations*, Proc. Sympos. Appl. Math., 6 (1956), pp. 83–102.

[17] M. R. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, J. Res. National Bureau of Standards, 49 (1952), pp. 409–435.

[18] K. M. IRANI, M. P. KAMAT, C. J. RIBBENS, H. F. WALKER, AND L. T. WATSON, *Experiments with conjugate gradient algorithms for homotopy curve tracking*, SIAM J. Optimization, 1 (1991), pp. 222–251.

[19] K. C. Jea, *Generalised conjugate gradient acceleration of iterative methods*, Ph.D. thesis, University of Texas at Austin, Austin, TX, 1982.

[20] M. P. Kamat, L. T. Watson, and J. L. Junkins, *A robust and efficient hybrid method for finding multiple equilibrium solutions*, in Proc. Third Internat. Symposium on Numerical Methods in Engineering, Paris, 1983, pp. 799–808.

[21] H. H. Kwok, M. P. Kamat, and L. T. Watson, *Location of stable and unstable equilibrium configurations using a model trust region quasi-Newton method and tunnelling*, Comput. & Structures, 21 (1985), pp. 909–916.

[22] 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.

[23] C. C. Paige and M. A. Saunders, *Solution of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal., 12 (1975), pp. 617–629.

[24] 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.

[25] Y. Saad, *Krylov subspace methods for solving large unsymmetric linear systems*, Math. Comp., 37 (1981), pp. 105–126.

[26] ———, *Practical use of some Krylov subspace methods for solving indefinite and unsymmetric linear systems*, Tech. Report 214, Department of Computer Science, Yale University, New Haven, CT, 1982.

[27] Y. Saad and M. H. Schultz, *Conjugate gradient-like algorithm for solving nonsymmetric linear systems*, Math. Comp., 44 (1985), pp. 417–424.

[28] ———, GMRES*: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.

[29] P. K. W. Vinsome, *Orthomin, an iterative method for solving sparse sets of simultaneous linear equations*, in Proc. Fourth Symposium on Reservoir Simulation, Society of Petroleum Engineers of the AIME, 1976, pp. 149–159.

[30] H. F. Walker, *Implementation of the* GMRES *method using Householder transformations*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 152–163.

[31] ———, *Implementations of the* GMRES *method*, Comput. Phys. Comm., 53 (1989), pp. 311–320.

[32] L. T. Watson, *An algorithm that is globally convergent with probability one for a class of nonlinear two-point boundary value problems*, SIAM J. Numer. Anal., 16 (1979), pp. 394–401.

[33] ———, *A globally convergent algorithm for computing fixed points of C^2 maps*, Appl. Math. Comp., 5 (1979), pp. 297–311.

[34] ———, *Globally convergent homotopy methods: A tutorial*, Tech. Report 87–13, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI, 1985.

[35] ———, *Numerical linear algebra aspects of globally convergent homotopy methods*, SIAM Rev., 28 (1986), pp. 529–545.

[36] L. T. Watson, S. C. Billups, and A. P. Morgan, HOMPACK*: A suite of codes for globally convergent homotopy algorithms*, ACM Trans. Math. Software, 13 (1987), pp. 281–310.

[37] O. Widlund, *A Lanczos method of a class of nonsymmetric systems of linear equations*, SIAM J. Numer. Anal., 15 (1978), pp. 801–812.

[38] D. M. Young and K. C. Jea, *Generalised conjugate gradient acceleration of nonsymmetrizable iterative methods*, Linear Algebra Appl., 34 (1980), pp. 159–194.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 1, pp. 47-70, January 1992

A BLOCK PROJECTION METHOD FOR SPARSE MATRICES*

MARIO ARIOLI†¶, IAIN DUFF‡¶, JOSEPH NOAILLES§, AND DANIEL RUIZ¶

Abstract. A block version of Cimmino's algorithm for solving general sets of consistent sparse linear equations is described. The case of matrices in block tridiagonal form is emphasized because it is assumed that the general case can be reduced to this form by permutations. It is shown how the basic method can be accelerated by using the conjugate gradient (CG) algorithm. This acceleration is very dependent on a partitioning of the original system and several possible partitionings are discussed. Underdetermined systems corresponding to the subproblems of the partitioned system are solved using the Harwell sparse symmetric indefinite solver MA27 on an augmented system. These systems are independent and can be solved in parallel. An analysis of the iteration matrix for the conjugate gradient acceleration leads to the consideration of rather unusual and novel scalings of the matrix that alter the spectrum of the iteration matrix to reduce the number of CG iterations.

The various aspects of this algorithm have been tested by runs on an eight-processor Alliant FX/80 on four block tridiagonal systems, two from fluid dynamics simulations and two from the literature. The effect of partitioning and scaling on the number of iterations and overall elapsed time for solution is studied. In all cases, an accurate solution with rapid convergence can be obtained.

Key words. sparse matrices, block iterative methods, projection methods, partitioning, augmented systems, parallel processing, block Cimmino method, conjugate gradient preconditioning

AMS(MOS) subject classifications. 65F50, 65F10, 65F20, 65Y05

1. Introduction.

Consider the solution of the system

$$\mathbf{A}\mathbf{x}=\mathbf{b} \tag{1.1}$$

where $\mathbf{A}$ is an $m \times n$ sparse matrix, $\mathbf{x}$ is an n-vector, and $\mathbf{b}$ is an m-vector. Although the experiments will examine the case when m is equal to n, the method is applicable for any m and n. The analysis will be principally concerned with the case $m \leqq n$ where the system is consistent, that is, there exists a vector $\mathbf{x}$ satisfying (1.1). In the following, we assume for simplicity that $\mathbf{A}$ has full row rank, that is, $\dim(\mathbf{A}) = m$.

The general solution to (1.1) can be expressed as the sum

$$\xi+\eta \tag{1.2}$$

where ξ is in the range of $\mathbf{A}^T$ and η is in the nullspace of $\mathbf{A}$. Thus

$$\xi=\mathbf{A}^T\mathbf{z}$$

and, from (1.1), we have

$$\mathbf{A}\mathbf{A}^T\mathbf{z}=\mathbf{b}$$

so that ξ is given by

$$\mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{b} \tag{1.3}$$

* Received by the editors December 11, 1989; accepted for publication (in revised form) November 12, 1990.

† Istituto di Elaborazione dell'Informazione, Consiglio Nazionale delle Ricerche, via S. Maria 46, 56100 Pisa, Italy.

‡ Rutherford Appleton Laboratory, Didcot, Oxon OX11 0QX, England.

§ Ecole Nationale Supérieure d'Electrotechnique, d'Electronique, d'Informatique, et d'Hydraulique de Toulouse, Institut de Recherche en Informatique de Toulouse, 2 rue Camichel, 31071 Toulouse Cedex, France.

¶ Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique, 42 av. G. Coriolis, 31057 Toulouse Cedex, France.

or

$$\mathbf{A}^{+}\mathbf{b}$$

where $\mathbf{A}^{+}$ is the Moore-Penrose pseudoinverse of $\mathbf{A}$. Because, usually, the solution to (1.1) of minimum norm is required, the component η is taken to be zero, so the solution of (1.1) is given by (1.3).

Sometimes the solution needed is the one closest to a vector $\mathbf{y}$, that is, an $\mathbf{x}$ satisfying (1.1) such that

$$\|\mathbf{x}-\mathbf{y}\|_2 \tag{1.4}$$

is minimized. This is given by

$$\mathbf{x}=\mathbf{P}_{\mathcal{N}(\mathbf{A})}\mathbf{y}+\mathbf{A}^{+}\mathbf{b}$$

where $\mathbf{P}_{\mathcal{N}(\mathbf{A})}$ is the orthogonal projector onto the nullspace of $\mathbf{A}$ given by

$$\mathbf{I}-\mathbf{P}_{\mathcal{R}(\mathbf{A}^T)}$$

where $\mathbf{P}_{\mathcal{R}(\mathbf{A}^T)}$, the orthogonal projector on the range of $\mathbf{A}^T$, is given by

$$\mathbf{P}_{\mathcal{R}(\mathbf{A}^T)}=\mathbf{A}^{+}\mathbf{A}.$$

In this paper we will analyse parallel implementations of block iterative methods for solving (1.1) and (1.4). We describe the general framework in § 2, the extension to norms other than the Euclidean norm (1.4) in § 3. In § 4 we identify the iteration matrix associated with the algorithm and introduce in §§ 5 and 6 conjugate gradient acceleration and the augmented system, which is used in the implementation of the algorithm.

In § 7 the numerical properties of this algorithm will be studied, illustrating its performance on block tridiagonal matrices. We introduce in § 8 the block SSOR method of Kamath and Sameh (1988) and Bramley and Sameh (1990), which can be seen as an alternative to the block Cimmino algorithm. In § 9 some ellipsoidal norms, which will be used to improve the performance on some of our test problems, are introduced for block tridiagonal matrices.

In § 10 we then describe some numerical experiments performed on the Alliant FX/80, and we present some concluding remarks in § 11.

2. Block iterative methods. The method which will be introduced can be considered as a generalization of the method of Cimmino (see Sloboda (1988)), and will therefore be called the block Cimmino method.

The blocks are obtained by partitioning the system (1.1) as

$$\begin{pmatrix}\mathbf{A}^1\\ \mathbf{A}^2\\ \vdots\\ \mathbf{A}^p\end{pmatrix}\mathbf{x}=\begin{pmatrix}\mathbf{b}^1\\ \mathbf{b}^2\\ \vdots\\ \mathbf{b}^p\end{pmatrix} \tag{2.1}$$

where $1\leqq p\leqq m$.

If we define by $\mathbf{P}_{\mathcal{R}(\mathbf{A}^{i^T})}$ the projector onto the range of $\mathbf{A}^{i^T}$, and the pseudoinverse of $\mathbf{A}^i$ as $\mathbf{A}^{i^+}$, the block Cimmino algorithm can be described in the following way.

ALGORITHM 2.1 (block Cimmino method).

```
Choose $\mathbf{x}^{(0)}$, set $k=0$
repeat until convergence
    begin
        do in parallel $i=1,\cdots,p$
(2.2)           $\delta^{i^{(k)}} = \mathbf{A}^{i^+}\mathbf{b}^i - \mathbf{P}_{\mathcal{R}(\mathbf{A}^{i^T})}\mathbf{x}^{(k)}$
                $= \mathbf{A}^{i^+}(\mathbf{b}^i - \mathbf{A}^i\mathbf{x}^{(k)})$
        end parallel
        $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \omega \sum_{i=1}^{p} \delta^{i^{(k)}}$
        set $k=k+1$
    end
```

This gives rise to a general purpose iterative solver well suited for both shared memory and distributed memory computers. If we take $p=m$, which means that each manifold from (2.1) is a hyperplane defined by one equation in (1.1), then Algorithm 2.1 becomes the algorithm of Cimmino (see Sloboda (1988)).

The iterative scheme (2.2) will converge for any pseudoinverse of $\mathbf{A}^i$ (see Campbell and Meyer (1979)) and, in particular, does not require $\mathbf{A}^i$ to be full rank. However, with the use of the explicit formulation of $\mathbf{A}^{i^+}$ introduced in (1.3), the partitioning of the system has to be chosen so that the rows in each $\mathbf{A}^i$ are linearly independent. This is possible even if $\mathbf{A}$ is not full rank.

A general study of this block-row method has been performed by Elfving (1980). In this paper, he calls this the "block-row Jacobi" method. He shows the effect of row and column partitioning, makes some comparisons with block SOR methods, and studies the effect of the parameter ω on the convergence of these methods.

First let us recall some results from Elfving (1980). Using the notation:

$$\mathbf{E}_{RJ} = \sum_{i=1}^{p} \mathbf{P}_{\mathcal{R}(\mathbf{A}^{i^T})} = \sum_{i=1}^{p} \mathbf{A}^{i^T}(\mathbf{A}^i\mathbf{A}^{i^T})^{-1}\mathbf{A}^i,$$

$$\mathbf{Q}_{RJ} = \mathbf{I} - \omega\mathbf{E}_{RJ}, \text{ for the iteration matrix,}$$

$$\rho(\mathbf{E}_{RJ}) = \text{the spectral radius of } \mathbf{E}_{\mathrm{RJ}}, \tag{2.3}$$

$$\mu_{\max} = \text{the largest nonzero eigenvalue of } \mathbf{E}_{RJ},$$

$$\mu_{\min} = \text{the smallest nonzero eigenvalue of } \mathbf{E}_{RJ},$$

we have the following:

(1) Suppose $\mathbf{b} \in \mathcal{R}(\mathbf{A})$ and $\mathbf{x}^{(0)} \in \mathcal{R}(\mathbf{A}^T)$, then the block-row method converges towards the minimum norm solution if and only if $0 < \omega < \min(2, (2/\rho(\mathbf{E}_{RJ})))$.

(2) The ω giving the optimal asymptotic rate of convergence is $\bar{\omega} = 2/(\mu_{\max} + \mu_{\min})$.

3. The use of other norms. The solution of (1.1), (1.4) can be determined in any ellipsoidal norm defined by

$$\|\mathbf{x}\|_{\mathrm{G}}^2 = \mathbf{x}^T\mathbf{G}\mathbf{x} \tag{3.1}$$

where $\mathbf{G}$ is an $n \times n$ symmetric positive definite (SPD) matrix.

When doing this, the arguments in § 1 still hold, except that a generalized pseudoinverse (see Rao and Mitra (1971))

$$\mathbf{A}_{\mathrm{G}^{-1}}^{-} = \mathbf{G}^{-1}\mathbf{A}^T(\mathbf{A}\mathbf{G}^{-1}\mathbf{A}^T)^{-1}, \tag{3.2}$$

must be defined in addition to corresponding oblique projectors

$$\mathbf{P}^{(\mathbf{G})}_{\mathcal{R}(\mathbf{G}^{-1}\mathbf{A}^T)} = \mathbf{A}^{-}_{\mathbf{G}^{-1}}\mathbf{A} \quad \text{and} \quad \mathbf{P}^{(\mathbf{G})}_{\mathcal{N}(\mathbf{A})} = \mathbf{I} - \mathbf{P}^{(\mathbf{G})}_{\mathcal{R}(\mathbf{G}^{-1}\mathbf{A}^T)}.$$

The solution to

$$\min_{\mathbf{x}} \|\mathbf{y}-\mathbf{x}\|_{\mathbf{G}} \quad \text{such that } \mathbf{x} \in \{\mathbf{x} \mid \|\mathbf{b}-\mathbf{A}\mathbf{x}\|_{\mathbf{G}} = \min\}$$

is then given by

$$\mathbf{x} = \mathbf{P}^{(\mathbf{G})}_{\mathcal{N}(\mathbf{A})}\mathbf{y} + \mathbf{A}^{-}_{\mathbf{G}^{-1}}\mathbf{b}. \tag{3.3}$$

The block Cimmino algorithm in § 2 can be defined in terms of this general norm.

We will now show that the use of $\mathbf{G} \neq \mathbf{I}$ can be viewed as a right-hand side preconditioning of the matrix **A**.

From (3.1), (3.2), and (3.3), and knowing that any SPD matrix **G** can be decomposed as $\mathbf{G}^{1/2}\mathbf{G}^{1/2}$, with $\mathbf{G}^{1/2}$ SPD, it follows that:

$$\begin{aligned}\mathbf{P}^{(\mathbf{G})}_{\mathcal{R}(\mathbf{G}^{-1}\mathbf{A}^{i^T})} &= \mathbf{G}^{-1/2}(\mathbf{G}^{-1/2}\mathbf{A}^{i^T})[(\mathbf{A}^i\mathbf{G}^{-1/2})(\mathbf{G}^{-1/2}\mathbf{A}^{i^T})]^{-1}(\mathbf{A}^i\mathbf{G}^{-1/2})\mathbf{G}^{1/2} \\ &= \mathbf{G}^{-1/2}\{(\mathbf{A}^i\mathbf{G}^{-1/2})^T[(\mathbf{A}^i\mathbf{G}^{-1/2})(\mathbf{A}^i\mathbf{G}^{-1/2})^T]^{-1}(\mathbf{A}^i\mathbf{G}^{-1/2})\}\mathbf{G}^{1/2} \\ &= \mathbf{G}^{-1/2}\mathbf{P}_{\mathcal{R}((\mathbf{A}^i\mathbf{G}^{-1/2})^T)}\mathbf{G}^{1/2}.\end{aligned} \tag{3.4}$$

Then the iteration matrix becomes:

$$\begin{aligned}\mathbf{Q}^{(\mathbf{G})}_{RJ} &= \mathbf{I} - \omega\mathbf{E}^{(\mathbf{G})}_{RJ} \\ &= \mathbf{I} - \omega\mathbf{G}^{-1/2}\sum_{i=1}^{p}\mathbf{P}_{\mathcal{R}((\mathbf{A}^i\mathbf{G}^{-1/2})^T)}\mathbf{G}^{1/2} \\ &= \mathbf{G}^{-1/2}\mathbf{Q}^*_{RJ}\mathbf{G}^{1/2}\end{aligned} \tag{3.5}$$

where

$$\mathbf{Q}^*_{RJ} = \mathbf{I} - \omega\mathbf{E}^*_{RJ} = \mathbf{I} - \omega\sum_{i=1}^{p}\mathbf{P}_{\mathcal{R}((\mathbf{A}^i\mathbf{G}^{-1/2})^T)}. \tag{3.6}$$

Thus the iteration matrix associated with the matrix **A** and using the ellipsoidal norm, and the one associated with the matrix $\mathbf{A}\mathbf{G}^{-1/2}$ and using the two-norm, are similar. However, in the case of the **G**-norm, all the convergence properties (which are determined only by the spectrum of the iteration matrix) can be written in a similar way to the previous ones (2.3), where block-rows $\mathbf{A}^i\mathbf{G}^{-1/2}$ are considered instead of block-rows $\mathbf{A}^i$.

So from now on, to simplify notation and formulae, we will, as in §§ 1 and 2, usually consider $\mathbf{G} = \mathbf{I}$, knowing that other choices for **G** can easily be incorporated.

4. The iteration matrix. Let the **QR** decomposition of the blocks $\mathbf{A}^{i^T}$ be given by

$\mathbf{A}^{i^T} = \mathbf{Q}^i\mathbf{R}^i$, $\quad i = 1, \cdots, p$ where $\mathbf{A}^i$ is an $m_i \times n$ matrix of full row rank,

$\mathbf{Q}^i$ $n \times m_i$, $\quad \mathbf{Q}^{i^T}\mathbf{Q}^i = \mathbf{I}_{m_i \times m_i}$,

$\mathbf{R}^i$ $m_i \times m_i$, $\quad \mathbf{R}^i$ nonsingular ''nper triangular matrix;

then:

$$
\begin{aligned}
\mathbf{E}_{RJ} &= \sum_{i=1}^{p} \mathbf{A}^{i^T}(\mathbf{A}^i\mathbf{A}^{i^T})^{-1}\mathbf{A}^i \\
&= \sum_{i=1}^{p} \mathbf{Q}^i\mathbf{R}^i(\mathbf{R}^{i^T}\mathbf{Q}^{i^T}\mathbf{Q}^i\mathbf{R}^i)^{-1}\mathbf{R}^{i^T}\mathbf{Q}^{i^T} \\
&= \sum_{i=1}^{p} \mathbf{Q}^i\mathbf{R}^i(\mathbf{R}^{i^T}\mathbf{R}^i)^{-1}\mathbf{R}^{i^T}\mathbf{Q}^{i^T} \qquad (4.1)\\
&= \sum_{i=1}^{p} \mathbf{Q}^i\mathbf{Q}^{i^T} \\
&= (\mathbf{Q}^1 \cdots \mathbf{Q}^p)(\mathbf{Q}^1 \cdots \mathbf{Q}^p)^T.
\end{aligned}
$$

But from the theory of the singular value decomposition (see Golub and Kahan (1965) and Golub and Van Loan (1989)), the nonzero eigenvalues of $(\mathbf{Q}^1 \cdots \mathbf{Q}^p)(\mathbf{Q}^1 \cdots \mathbf{Q}^p)^T$ are also the nonzero eigenvalues of $(\mathbf{Q}^1 \cdots \mathbf{Q}^p)^T(\mathbf{Q}^1 \cdots \mathbf{Q}^p)$.

Thus the spectrum of the matrix $\mathbf{E}_{RJ}$ is the same as that of the matrix

$$
(4.2) \qquad \begin{pmatrix}
\mathbf{I}_{m_1\times m_1} & \mathbf{Q}^{1^T}\mathbf{Q}^2 & \cdots & \cdots & \mathbf{Q}^{1^T}\mathbf{Q}^p \\
\mathbf{Q}^{2^T}\mathbf{Q}^1 & \mathbf{I}_{m_2\times m_2} & \mathbf{Q}^{2^T}\mathbf{Q}^3 & \cdots & \mathbf{Q}^{2^T}\mathbf{Q}^p \\
\vdots & & & & \vdots \\
\mathbf{Q}^{p^T}\mathbf{Q}^1 & & \cdots & & \mathbf{I}_{m_p\times m_p}
\end{pmatrix}
$$

where the $\mathbf{Q}^{i^T}\mathbf{Q}^j$ are matrices whose singular values represent the cosines of the principal angles between the subspaces $\mathcal{R}(\mathbf{A}^{i^T})$ and $\mathcal{R}(\mathbf{A}^{j^T})$ (see Björck and Golub (1973)). This implies that a preconditioner, to be successful, should modify the principal angles between the subspaces $\mathcal{R}(\mathbf{A}^{i^T})$, in order to make these subspaces as much orthogonal between each other as possible.

Another implication of the previous discussion concerns the choice of the partitioning (2.1). As stated in Bramley and Sameh (1990), an ill-conditioned matrix $\mathbf{A}$ has some linear combination of rows almost equal to the zero vector. After row partitioning, these may occur within the blocks or across the blocks. If, in the block Cimmino algorithm, we assume that we compute the projections on the subspaces exactly, the rate of convergence of the method will depend only on the conditioning across the blocks. However, if the method used for solving the subproblems is sensitive to ill-conditioning within the blocks, we can quickly converge to the wrong solution. Thus, for a robust algorithm, we must use the most stable algorithm for computing the projections combined with a partitioning that minimizes the ill-conditioning across the blocks.

5. Conjugate gradient acceleration. Even when using the optimal ω introduced before, the convergence can still be slow. Thus it is natural to try to accelerate the iterative scheme.

The equalities in (4.1) show that $(\mathbf{I}-\mathbf{Q}_{RJ})=\omega\mathbf{E}_{RJ}$ is symmetric positive semi-definite for any $\omega>0$, and is definite if and only if $\mathbf{A}$ is square and of full rank. Now, in the case of ellipsoidal norms as defined in § 3, (3.5) and (3.6) ensure that the matrix $\mathbf{G}^{1/2}(\mathbf{I}-\mathbf{Q}_{RJ}^{(\mathbf{G})})\mathbf{G}^{-1/2}$ will have these properties. Thus $\mathbf{G}^{1/2}$ can be seen as a symmetrization matrix for the block Cimmino method (see Hageman and Young (1981)), and so the CG acceleration procedure can be applied to the preconditioned case in the following manner:

ALGORITHM 5.1 (conjugate gradient acceleration).

$\mathbf{x}^{(0)}$ is arbitrary, $\mathbf{p}^{(0)} = \delta^{(0)}$,

and $\delta^{(k)}$ is the pseudoresidual vector defined, for $k = 0, 1, \cdots$, by

$$\delta^{(k)} = \mathbf{Q}_{RJ}^{(\mathbf{G})}\mathbf{x}^{(k)} - \mathbf{x}^{(k)} + \omega \sum_{i=1}^{p} \mathbf{A}^{i-}_{\mathbf{G}^{-1}}\mathbf{b}^i$$

$$= \omega\left(-\mathbf{E}_{RJ}^{(\mathbf{G})}\mathbf{x}^{(k)} + \sum_{i=1}^{p} \mathbf{A}^{i-}_{\mathbf{G}^{-1}}\mathbf{b}^i \right),$$

for $k = 1, 2, \cdots$, until convergence do:

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \lambda_{k-1}\mathbf{p}^{(k-1)}$$

$$\mathbf{p}^{(k)} = \delta^{(k)} + \alpha_k \mathbf{p}^{(k-1)},$$

$$\alpha_k = \frac{(\mathbf{G}^{1/2}\delta^{(k)}, \mathbf{G}^{1/2}\delta^{(k)})}{(\mathbf{G}^{1/2}\delta^{(k-1)}, \mathbf{G}^{1/2}\delta^{(k-1)})}$$

$$= \frac{(\delta^{(k)}, \delta^{(k)})_{\mathbf{G}}}{(\delta^{(k-1)}, \delta^{(k-1)})_{\mathbf{G}}},$$

$$\lambda_{k-1} = \frac{(\mathbf{G}^{1/2}\delta^{(k-1)}, \mathbf{G}^{1/2}\delta^{(k-1)})}{(\mathbf{G}^{1/2}\mathbf{p}^{(k-1)}, \mathbf{G}^{1/2}(\mathbf{I} - \mathbf{Q}_{RJ}^{(\mathbf{G})})\mathbf{p}^{(k-1)})}$$

$$= \frac{(\delta^{(k-1)}, \delta^{(k-1)})_{\mathbf{G}}}{(\mathbf{p}^{(k-1)}, (\mathbf{I} - \mathbf{Q}_{RJ}^{(\mathbf{G})})\mathbf{p}^{(k-1)})_{\mathbf{G}}}$$

$$= \frac{(\delta^{(k-1)}, \delta^{(k-1)})_{\mathbf{G}}}{(\mathbf{p}^{(k-1)}, \omega\mathbf{E}_{RJ}^{(\mathbf{G})}\mathbf{p}^{(k-1)})_{\mathbf{G}}},$$

where $(\cdot, \cdot)_{\mathbf{G}}$ denotes the dot product in the **G** norm viz.

$$(\mathbf{x}, \mathbf{x})_{\mathbf{G}} = \mathbf{x}^T\mathbf{G}\mathbf{x}.$$

The two main reasons for our choice of CG acceleration are:

(1) It can be seen from the above equations that the CG acceleration is independent of the choice of the relaxation parameter ω, in the sense that the sequence of iterates $\mathbf{x}^{(k)}$ is the same for any nonzero ω (provided the starting point $\mathbf{x}^{(0)}$ is the same). This is the reason why we did not emphasize the use of ω too much in the previous sections.

(2) It is very simple to use ellipsoidal norms in the CG acceleration, because the only requirement is the ability to compute the matrix-vector product relevant to working with the dot product in the **G** norm. The matrix **G** can thus be generated implicitly or stored in any appropriate way.

6. Use of augmented systems in solving subproblems. When solving the subproblems (2.2), the use of the augmented system approach has been chosen for two main reasons. First, it is more stable than the normal equations approach (see Arioli, Duff, and de Rijk (1989)) because it avoids building and storing the normal equations for each block row $\mathbf{A}^i$. This can be useful for two reasons. It is less sensitive to ill-conditioning within the blocks caused by partitionings where rows within a block are nearly linearly dependent. Also, the kind of preconditioners (ellipsoidal norms) that will be used in the experiments introduce some ill-conditioning within the blocks. Second, the use of ellipsoidal norms and the computation of the corresponding oblique projectors can be trivially accommodated, as we will see in the following.

In the augmented system approach, the system

$$\begin{bmatrix} \mathbf{G} & \mathbf{A}^{i^T} \\ \mathbf{A}^i & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u}^i \\ \mathbf{v}^i \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{b}^i - \mathbf{A}^i\mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{r}^i \end{bmatrix},$$

whose solution is

$$\mathbf{v}^i = -(\mathbf{A}^i\mathbf{G}^{-1}\mathbf{A}^{i^T})^{-1}\mathbf{r}^i$$

$$\mathbf{u}^i = \mathbf{G}^{-1}\mathbf{A}^{i^T}(\mathbf{A}^i\mathbf{G}^{-1}\mathbf{A}^{i^T})^{-1}\mathbf{r}^i,$$

is considered.

We solve the augmented systems with the sparse symmetric linear solver MA27 from the Harwell Subroutine Library (see Duff and Reid (1983)). The solver MA27 computes the $\mathbf{LDL}^T$ decomposition of a permutation of the augmented matrix, using a mixture of 1×1 or 2×2 pivots chosen during the numerical factorization. The resulting $\mathbf{L}$ and $\mathbf{D}$ factors are then used to solve the augmented systems by forward and backward substitution in the usual way.

It must be emphasized that the method (2.2) is totally independent of the choice of the solvers for the underdetermined subproblems. For instance, in some problems it may be preferable to use a solver adapted to the structure of the block $\mathbf{A}^i$. In special cases, several different solvers might be used, one for each different kind of block $\mathbf{A}^i$.

7. Effect of the partitioning. We will now focus on some particular partitionings of the system (1.1), and show how the algorithm can exploit them.

7.1. Two-block partitioning. Assume that the matrix $\mathbf{A}$ is partitioned in two blocks

$$\begin{pmatrix} \mathbf{A}^1 \\ \mathbf{A}^2 \end{pmatrix},$$

where $\mathbf{A}^1$ and $\mathbf{A}^2$ have m_1 and m_2 rows, respectively. Assume, without loss of generality, that $m_1 \geqq m_2$. With such a partitioning, the iteration matrix $\mathbf{Q}_{RJ}$ can be considered as $\mathbf{I} - \omega(\mathbf{P}_1 + \mathbf{P}_2)$, where $\mathbf{P}_1 = \mathbf{P}_{\mathcal{R}(\mathbf{A}^{1^T})}$ and $\mathbf{P}_2 = \mathbf{P}_{\mathcal{R}(\mathbf{A}^{2^T})}$. So, as in § 4, the matrix $\mathbf{E}_{RJ} = \mathbf{P}_1 + \mathbf{P}_2$ can be reduced to $(\mathbf{Q}^1\mathbf{Q}^2)(\mathbf{Q}^1\mathbf{Q}^2)^T$, and from (4.2), the spectrum of the iteration matrix $\mathbf{Q}_{RJ}$ is the same as that of the matrix

$$\begin{pmatrix} (1-\omega)\mathbf{I}_{m_1 \times m_1} & -\omega\mathbf{Q}^{1^T}\mathbf{Q}^2 \\ -\omega\mathbf{Q}^{2^T}\mathbf{Q}^1 & (1-\omega)\mathbf{I}_{m_2 \times m_2} \end{pmatrix}. \tag{7.1}$$

Since we know from § 5 that the block Cimmino algorithm with conjugate gradient acceleration is independent of ω, we can take $\omega = 1$, and matrix (7.1) becomes

$$\begin{pmatrix} \mathbf{0}_{m_1 \times m_1} & -\mathbf{Q}^{1^T}\mathbf{Q}^2 \\ -\mathbf{Q}^{2^T}\mathbf{Q}^1 & \mathbf{0}_{m_2 \times m_2} \end{pmatrix}. \tag{7.2}$$

Then, looking at the shape of the rectangular submatrix $\mathbf{Q}^{1^T}\mathbf{Q}^2$, which has m_1 rows and m_2 columns ($m_1 \geqq m_2$) and therefore a rank not greater than m_2, it follows that there are at most $2m_2$ nonzero eigenvalues.

The finite termination property of the conjugate gradient algorithm suggests that the block Cimmino algorithm with CG acceleration generally works better for small m_2, and in exact arithmetic takes not more than $2m_2$ steps for convergence. This is another reason for the choice of the conjugate gradient acceleration.

7.2. Block tridiagonal structures. We now concentrate on block tridiagonal matrices, which are quite general patterns very common in partial differential equation discretization, and are obtainable from general systems as a byproduct of bandwidth reduction (see, for example, Duff, Erisman, and Reid (1986, pp. 153-157)). For such structures, a partitioning equivalent to the two-block one can be introduced and used efficiently for obtaining fast convergence.

Consider, for instance, a block tridiagonal matrix **A** with blocks of size $l\times l$, partitioned as follows:

$$\left[\begin{array}{ccccccccccccc}
* & * & & & & & & & & & & & \\
* & * & * & & & & & & & & & & \\
& * & * & * & & & & & & & & & \\
\hline
& & * & * & * & & & & & & & & \\
& & & * & * & * & & & & & & & \\
\hline
& & & & * & * & * & & & & & & \\
& & & & & * & * & * & & & & & \\
& & & & & & * & * & * & & & & \\
\hline
& & & & & & & * & * & * & & & \\
& & & & & & & & * & * & * & & \\
\hline
& & & & & & & & & * & * & * & \\
& & & & & & & & & & * & * & * \\
& & & & & & & & & & & * & *
\end{array}\right]=\left[\begin{array}{c}\mathbf{A}^1\\ \mathbf{A}^2\\ \mathbf{A}^3\\ \mathbf{A}^4\\ \mathbf{A}^5\end{array}\right],$$

where $\mathbf{A}^i$ has $k_i\times l$ rows, $k_i\geqq 2$, $i=1,2,\cdots,5$. Then $\mathcal{R}(\mathbf{A}^{i^T})$ and $\mathcal{R}(\mathbf{A}^{i+2^T})$, $i=1,2,3$, represent orthogonal subspaces, so that

$$\mathbf{P}_{\mathcal{R}(\mathbf{A}^{i^T})}+\mathbf{P}_{\mathcal{R}(\mathbf{A}^{i+2^T})}=\mathbf{P}_{\mathcal{R}(\mathbf{A}^{i^T})\oplus\mathcal{R}(\mathbf{A}^{i+2^T})}.$$

On the one hand, matrices of this form can therefore be easily partitioned in sufficient blocks to utilize all the processors of the target machine (provided the matrix **A** is large enough in comparison with the size of the tridiagonal substructure). On the other hand, the problem being solved is equivalent, after permutation of the rows of the matrix **A**, to the one defined by taking only two blocks $\mathbf{B}^1$ and $\mathbf{B}^2$, where

$$\mathbf{B}^1=\left\{\bigcup_i \mathbf{A}^i/i\ odd\right\},\qquad \mathbf{B}^2=\left\{\bigcup_i \mathbf{A}^i/i\ even\right\};$$

thus

$$\left[\begin{array}{c}\mathbf{B}^1\\ \mathbf{B}^2\end{array}\right]=\left[\begin{array}{c}\mathbf{A}^1\\ \mathbf{A}^3\\ \mathbf{A}^5\\ \mathbf{A}^2\\ \mathbf{A}^4\end{array}\right]=\left[\begin{array}{ccccccccccccc}
* & * & & & & & & & & & & & \\
* & * & * & & & & & & & & & & \\
& * & * & * & & & & & & & & & \\
& & & & * & * & * & & & & & & \\
& & & & & * & * & * & & & & & \\
& & & & & & * & * & * & & & & \\
& & & & & & & & & * & * & * & \\
& & & & & & & & & & * & * & * \\
& & & & & & & & & & & * & * \\
& & * & * & * & & & & & & & & \\
& & & * & * & * & & & & & & & \\
& & & & & & & * & * & * & & & \\
& & & & & & & & * & * & * & &
\end{array}\right].$$

Then the nice property of the two-block partitioning discussed in the previous section is maintained while a good degree of parallelism is obtained.

If m_1 and m_2 denote the number of rows of $\mathbf{B}^1$ and $\mathbf{B}^2$, respectively, we observe that it is easy to vary the number of zero eigenvalues in the iteration matrix, setting it close to zero by taking the same size for the $\mathbf{A}^i$ in $\mathbf{B}^1$ as for the $\mathbf{A}^i$ in $\mathbf{B}^2$ $(m_1 \sim m_2)$, or setting it as big as possible $(m_1 \gg m_2)$ by taking large blocks $\mathbf{A}^i$ in $\mathbf{B}^1$, and defining small interface blocks in $\mathbf{B}^2$ of size $2 \times l$ (the minimum required for making the $\mathbf{A}^i$ in $\mathbf{B}^1$ structurally orthogonal).

A good partitioning strategy must compromise between reducing the size of the interface block $\mathbf{B}^2$, with the aim of reducing the number of CG iterations, and maintaining the degree of parallelism of the method. The degree of parallelism involves two things: first, the number of blocks in the partitioning compared to the number of processors of the target machine, and second, the size of these blocks compared to the amount of work the different processors will have to perform, which affects granularity and load balancing.

We remember from § 3 that we can easily incorporate ellipsoidal norms, replacing orthogonal projectors by oblique ones. But in this case, when looking for partitionings leading to a two-block partitioning, we must ensure that the right-hand side preconditioning matrix $\mathbf{G}^{-1/2}$ (see § 3) preserves the orthogonality between the subspaces $\mathscr{R}(\mathbf{A}^{i^T})$ in each of the two blocks $\mathbf{B}^1$ and $\mathbf{B}^2$ of the given two-block partitioning.

8. Block SSOR method. Kamath and Sameh (1988) use a block SSOR iterative scheme with a two-block partitioning for block tridiagonal structures. They consider, however, only the partitioning defined by taking equal-sized blocks of the smallest possible size for a matrix partitioned into two blocks (twice the number of rows of one block in the block tridiagonal structure). This leads to a high degree of parallelism, but also to the largest number of nonzero eigenvalues in the iteration matrix.

The block SSOR algorithm with two-block partitioning (as described in § 7.2) can be written in the following way:

```
ALGORITHM 8.1 (block SSOR method).
    Choose $\mathbf{x}^{(0)}$, set $k=0$
    repeat until convergence
        begin
                            $\mathbf{z}^1 = \mathbf{x}^{(k)}$
            Forward sweep   $\left[\begin{array}{l} \mathbf{z}^2 = \mathbf{z}^1 + \omega \mathbf{B}^{1^+}(\mathbf{b}^1 - \mathbf{B}^1\mathbf{z}^1) \\ \mathbf{z}^3 = \mathbf{z}^2 + \omega \mathbf{B}^{2^+}(\mathbf{b}^2 - \mathbf{B}^2\mathbf{z}^2) \end{array}\right.$
            Reverse sweep   $\left[\begin{array}{l} \mathbf{z}^4 = \mathbf{z}^3 + \omega \mathbf{B}^{2^+}(\mathbf{b}^2 - \mathbf{B}^2\mathbf{z}^3) \\ \mathbf{z}^5 = \mathbf{z}^4 + \omega \mathbf{B}^{1^+}(\mathbf{b}^1 - \mathbf{B}^1\mathbf{z}^4) \end{array}\right.$
                            $\mathbf{x}^{(k+1)} = \mathbf{z}^5$, set $k = k+1$
        end
```

where the current iterate is projected in sequence onto the different subspaces using the updated value each time.

The iteration matrix for this algorithm can be expressed as a product of projectors instead of a sum as in the block Cimmino algorithm, viz.,

$$\mathbf{Q}_{RS} = (\mathbf{I} - \omega \mathbf{P}_{\mathscr{R}(\mathbf{B}^{1^T})})(\mathbf{I} - \omega \mathbf{P}_{\mathscr{R}(\mathbf{B}^{2^T})})^2(\mathbf{I} - \omega \mathbf{P}_{\mathscr{R}(\mathbf{B}^{1^T})})$$

where one projection in the central part of the algorithm can be avoided, because of the idempotency of projectors.

A detailed study of the block SSOR algorithm and of its iteration matrix is performed by Bramley (1989) and by Bramley and Sameh (1990). They show that, even when conjugate gradient acceleration is used, the first projection can also be avoided, provided that $\omega = 1$ and that $\mathbf{x}^{(0)}$ is such that $\mathbf{B}^1\mathbf{x}^{(0)} = \mathbf{b}^1$. Therefore, with the two-block partitioning, one iteration of block SSOR has the same cost in number of flops as one iteration of block Cimmino.

It was shown by Elfving (1980) that, with $\omega = 1$, and the same kind of partitioning, the spectrum of the iteration matrices for the two algorithms are:

$$\text{for block Cimmino:}\quad \begin{array}{ll} \lambda_k = +\cos \Psi_k, & k = 1, \cdots, m_2, \\ \lambda_k = -\cos \Psi_{k-m_2}, & k = m_2 + 1, \cdots, 2m_2, \\ \lambda_k = 0, & k = 2m_2 + 1, \cdots, n; \end{array}$$

$$\text{for block SSOR:}\quad \begin{array}{ll} \lambda_k = \cos^2 \Psi_k, & k = 1, \cdots, m_2, \\ \lambda_k = 0, & k = m_2 + 1, \cdots, n \end{array}$$

where $\{\Psi_k\}_1^{m_2}$ are the principal angles between $\mathcal{R}(\mathbf{B}^{1^T})$ and $\mathcal{R}(\mathbf{B}^{2^T})$.

Thus, with two-block partitioning, the block SSOR algorithm with CG acceleration should converge in about half the number of iterations as the block Cimmino algorithm with CG acceleration.

Similar comparisons between block Cimmino and block SSOR can also be found in Bramley (1989) and in Bramley and Sameh (1990). They show that, in general, the block SSOR iteration matrix has more eigenvalues collapsed to zero than that of block Cimmino, which will benefit the convergence of block SSOR. But we must not forget that the degree of parallelism in the block Cimmino method is higher because all the projections can be performed in parallel. This is not the case for the block SSOR method where information must be updated from one subproblem to the next so that the subspaces must be orthogonal to permit parallelism.

However, when using two-block partitioning on shared memory machines, we have found experimentally that block SSOR is still faster because memory bank conflicts prevent block Cimmino from attaining twice the speedup of block SSOR. At any rate, in the following, results and comments on various two-block partitionings will be given for the CG accelerated block Cimmino scheme only, because the aim here is not to compare the two methods in terms of efficiency, but to emphasize our approach to the implementation of row projection methods, using block Cimmino by way of illustration. Different techniques have been introduced in the previous sections, e.g., the augmented system, the ellipsoidal norms that can easily be incorporated in the augmented system approach, and the varying of the relative sizes of the blocks in the two-block partitionings. The experiments are designed to emphasize these novel aspects.

The main reason why we have chosen block Cimmino is that its degree of parallelism depends only on the partitioning and not on special properties of structural orthogonality between sets of equations. Thus block Cimmino provides more degrees of freedom for the choice of any ellipsoidal norm as a preconditioner because it is not necessary to preserve the structure of the original matrix to maintain the degree of parallelism.

9. Use of ellipsoidal norms. In this section we introduce a particular kind of ellipsoidal norm associated with two-block partitioning. This will be used to improve the convergence on some of the test problems.

Consider a two-block partitioning as shown in Fig. 9.1. The aim is to make the two subspaces, generated by the two sets of rows, nearly orthogonal so that the

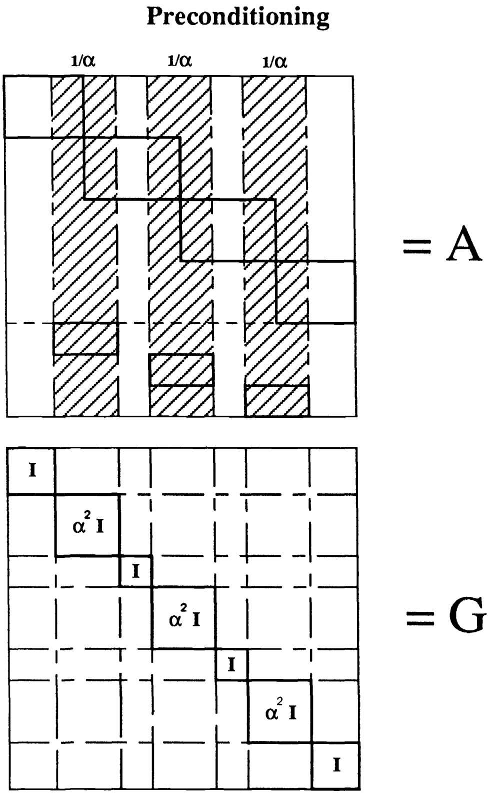


FIG. 9.1

eigenvalues of the iteration matrix (which are the cosines of the principal angles between the two subspaces, as shown in § 8) will all be small. The solution would then be obtained rapidly.

The simplest way to make two sets of rows in a matrix orthogonal is to multiply their overlapping part by zero. But obviously, if we do this, the resulting system can be rank deficient. In order to avoid this, we instead divide them by a large number α, as shown in Fig. 9.1. This leads to a diagonal matrix **G** with α^2 in positions corresponding to the overlapping columns and 1 elsewhere, remembering from § 3 that it is $\mathbf{G}^{-1/2}$ which preconditions the matrix **A**.

Of course, with such a preconditioner, some numerical instability in the factorization of the augmented systems can be expected if a very large value of α is used. We will see in § 10.4 that a value of $\alpha = 10$ gives the best compromise for our test problems and that much larger values for α introduce numerical instabilities without improving the rate of convergence. Also, the overlapping part must not involve all the columns of the matrix, because in that case the preconditioning would be equivalent to a scaling of the complete matrix, which would not modify the principal angles between the two spaces at all.

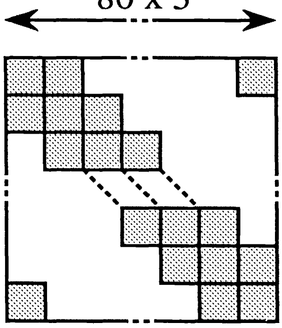
We note that this scaling preserves the block structure of the partitionings, as required in § 7.2, and thus can also be used with the block SSOR method of Kamath and Sameh (1988). However, it cannot be applied with their particular partitioning, because it involves a complete overlapping of the two blocks.

10. Numerical experiments. In this section we present some numerical experiments for testing the behaviour of our CG accelerated block Cimmino method. We first describe three different test problems and the particular points we wish to clarify by means of these tests. Then we analyse the results obtained on the eight-processor Alliant FX/80 at CERFACS, varying the two-block partitionings. Afterwards, we present results obtained using preconditioners of the kind discussed in § 9 and show how they decrease the number of iterations and the elapsed time for some of our test problems. Finally, we introduce a fourth test problem derived from Bramley (1989) and describe the results obtained using different kinds of partitioning from the two-block ones. These supplementary runs permit the comparison of our approach with the one of Bramley (1989) and Bramley and Sameh (1990) and an illustration of their differences.

10.1. The test problems. We will introduce the first three test problems. All are square problems, although we note that our algorithm does not require this. The first two problems come from simulation models of real flows developed at CERFACS in the fluid dynamics team.

The first model, developed by Perrel, concerns the study of a body entering the atmosphere at a high mach number. It is based on a finite-volume discretisation of the Navier–Stokes equations coupled with chemistry. For our test, we consider a two-dimensional problem which leads to three variables per mesh point (energy and two velocities), plus two species in the chemistry which lead to two density variables per mesh point, making a total of five variables. The discretization is performed using an implicit scheme on a curvilinear mesh of 69×60 points. This leads to block tridiagonal matrices of order $60 \times 69 \times 5 = 20{,}700$. The off-diagonal blocks are block diagonal with 69 blocks of 5×5 elements. The diagonal blocks have a block tridiagonal structure. The test problem solved is taken from the first time step in this time-dependent problem. The matrix is unsymmetric and nondiagonally dominant. The right-hand side is very large because the initial guess of the parameters is far from the steady-state solution. Therefore, a large correction to this guess is expected, and in this case, the solution need not be very accurate.

The second model, developed by Weinerfelt, is from the study of a two-dimensional wing profile at transonic flow (with no chemistry effects). It is still based on finite-volume discretisation, but this time, of the Euler equations. We are also looking for the steady-state solution, and the discretisation is performed using an upwind and implicit scheme on a curvilinear mesh of 80×32 points. This leads to unsymmetric but diagonally dominant block tridiagonal matrices of order $32 \times 80 \times 3 = 7{,}680$. The off-diagonal blocks are block diagonal with 80 blocks of 3×3 elements. The diagonal blocks are of the same order and have the following block triangular circulant structure:



This time the test problem solved is close to the steady-state solution, and a very accurate solution is required.

The other test problem comes from the paper by Kamath and Sameh (1988). It is obtained by application of the finite-difference method to a two-dimensional partial differential equation. The right-hand side is computed using a given solution

PROBLEM 3 (Kamath and Sameh (1988)).

$$-u_{xx}-u_{yy}+1000e^{xy}u_x-1000e^{xy}u_y=g, \qquad u=x+y,$$

on a 64×64 grid. We have selected this problem for the following reasons. First, as will be shown, it converges very slowly, but we will try to improve the situation using ellipsoidal norms. Second, this problem is the particular one for which Kamath and Sameh show that their block SSOR method converges while the iterative solver GMRES from the Yale package PCGPACK does not. By means of this test, we want to indicate that our method belongs to the same class of robust iterative solvers as the block SSOR method of Kamath and Sameh. This is also shown by Bramley and Sameh (1990), who compare different accelerated row projection methods, including the block Cimmino method with CG acceleration, with other iterative methods such as GMRES from PCGPACK, CG on the normal equations, and preconditioned versions of these two. They also use the previous equation and several other problems on three-dimensional grids.

10.2. Different partitionings. We now introduce the partitionings by which we hope to influence the number of iterations as indicated in the discussion in § 7. Since the tests are performed on an eight-processor Alliant, we consider the five following partitionings.

(1) A single block to provide a comparison with the direct solution of the augmented system on the overall matrix.

(2) Eight equal-sized blocks, in an attempt to balance the work on the eight processors of the Alliant FX/80. For this partitioning, however, the sizes of the two blocks in the equivalent two-block partitioning (after reordering the rows) are equal, so that the number of nonzero eigenvalues in the iteration matrix is expected to be maximum.

(3) Five large blocks in $\mathbf{B}^1$ and four minimum-sized blocks in $\mathbf{B}^2$. This partitioning should minimize the number of iterations, but gives a poor balancing of the work. It is then interesting to see if the gain in elapsed time due to the expected decrease in the number of iterations can overcome the loss of parallelism in the partitioning.

(4) A compromise between the size of the interface block and the degree of parallelism. For instance, if we consider a block tridiagonal structure with 32×32 blocks (which is the case for the problem given by Weinerfelt), we can partition by blocks of rows according to

$$\begin{array}{cccccccc} 4 & 4 & 4 & 4 & 4 & 1 & & (\mathbf{A}^i \text{ in } \mathbf{B}^1) \\ 1 & 2 & 2 & 2 & 2 & 2 & & (\mathbf{A}^i \text{ in } \mathbf{B}^2) \end{array}$$

where we give in sequence the number of block-rows for each partition. Here the size of the interface $\mathbf{B}^2$ is 11 block-rows, and the work is well balanced, with five processors handling blocks of size 4 (in number of block-rows) and the three remaining processors, each handling two blocks of size 2, or one block of size 2 plus two blocks of size 1. Of course, the number of rows in each block is only a rough heuristic for calculating the amount of work that will be performed by MA27 in the corresponding augmented systems. However, as we will see from the results, it gives a good estimate.

(5) The partitioning of Kamath and Sameh (1988), defined by taking equal-sized blocks of the smallest possible size for a matrix partitioned into two blocks (two times the number of rows in one block of the block tridiagonal structure). Although this partitioning also leads to a large interface block and to good load balancing, it has the added advantage that it minimizes the overall fill-in in the factorization of the different augmented systems because they are all of the smallest size. This leads to a smaller elapsed time for the factorization part, which can be of great benefit, especially in the case where the time for factorization is greater than the time for iterations.

10.3. Results. We will now present the results obtained on the Alliant FX/80. All the tests are performed in double precision (64 bit words) with machine precision $2.2\ 10^{-16}$.

For each test problem we give two figures, each with a histogram and a graph. In the first figure the histogram gives, for each partitioning, the number of floating-point operations for the factorization of all the augmented systems and the number of floating-point operations for the iterative part (computation of the minimum norm solutions and CG acceleration), and the graph in the same figure shows the number of iterations performed to reach the accuracy specified. With this histogram, we can see whether the factorization or the iteration is the more costly and also the effect of the partitionings on the number of iterations. The histogram in the second figure gives, for each partitioning, the total elapsed time for the solution (divided into the elapsed time for analysis and factorization of the augmented systems and the elapsed time for the iterative part), and the graph in the same figure gives the speedup of the method using the eight processors of the Alliant FX/80. The speedup measure used is the ratio between the elapsed time for execution on one processor and the elapsed time for execution on eight processors, using the same code. With this histogram, we can see which partitioning is the best, and also how well the work is balanced. We present, in tabular form in the Appendix, the results used in generating these figures.

For the stopping criterion of convergence, we use the scaled residual ω_k defined by

$$\omega_k = \frac{\|\mathbf{A}\mathbf{x}^{(k)} - \mathbf{b}\|_\infty}{\|\mathbf{A}\|_\infty \|\mathbf{x}^{(k)}\|_1 + \|\mathbf{b}\|_\infty}.$$

For an assigned value of *TOL*, if, at step k, $\omega_k \leqq TOL$, we stop the process. A small value for ω_k means that the algorithm is normwise backward stable (see Oettli and Prager (1964)) in the sense that the solution $\mathbf{x}^{(k)}$ is the exact solution of a perturbed problem where the max norm of the error matrix is less than or equal to ω_k. The bound

$$\frac{\|\mathbf{x}^{(k)} - \mathbf{x}\|_\infty}{\|\mathbf{x}\|_\infty} \leqq \omega_k (n+1) \|\mathbf{A}\|_\infty \|\mathbf{A}^{-1}\|_\infty + O(\omega_k^2)$$

also holds, which gives an estimate of the normwise relative error of the solution.

10.3.1. Problem given by Perrel. In the problem from Perrel, the iterations were stopped when $\omega_k \leqq TOL = 10^{-9}$ because, as we said in § 10.1, we do not need very high accuracy for this problem. Results for partitioning (1) are not given because the direct solution of the single augmented system produces too much fill-in for the memory of the Alliant. We present the results in Figs. 10.3.1 and 10.3.2 and, in tabular form, in Table A.1 in the Appendix.

The first remark is that the algorithm converges very quickly for this problem considering that the matrix is of order 20,700 and is not diagonally dominant. As expected from the discussion in § 10.2, the minimum number of iterations is attained

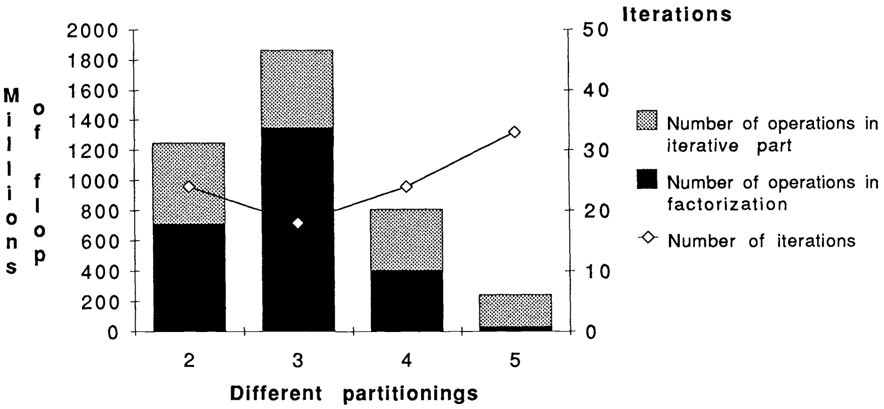


FIG. 10.3.1. *Perrel's test problem.* $\omega_k \leqq 10^{-9}$.

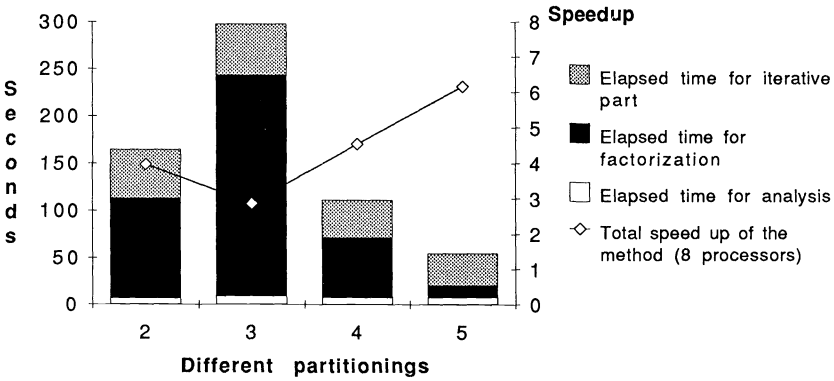


FIG. 10.3.2. *Perrel's test problem.* $\omega_k \leqq 10^{-9}$.

by partitioning (3), and partitioning (4) gives a good compromise between the number of iterations and the balancing of the work between factorization and iteration. In this problem, the fill-in in the direct solution of the subproblems is very significant and is more so when the subproblems are larger. Thus a partitioning with the smallest blocks possible should be the best even if this causes an increase in the number of iterations. This is illustrated by the results of partitioning (5), which has the least elapsed time although it has the greatest number of iterations. For such partitionings, we will also attain a higher speedup because, with the smaller blocks, there will be less cache conflict when solving the subsystems simultaneously on the different processors.

We also tried to obtain a more accurate solution with $\omega_k \leqq 10^{-14}$. The results are summarized in Table A.2 in the Appendix. All the partitionings achieved this accuracy with partitioning (5) again requiring the largest number of iterations but the least elapsed time because, even if the gap between the number of iterations for the different partitionings is increased for such an accuracy (it becomes roughly double that for the previous accuracy), the time and the amount of work performed during iterations still remains proportional to the fill-in produced during the factorization.

10.3.2. Problem given by Weinerfelt. In the problem from Weinerfelt, a very accurate solution is needed and so iterations are stopped when $\omega_k \leqq TOL = 10^{-14}$. We present

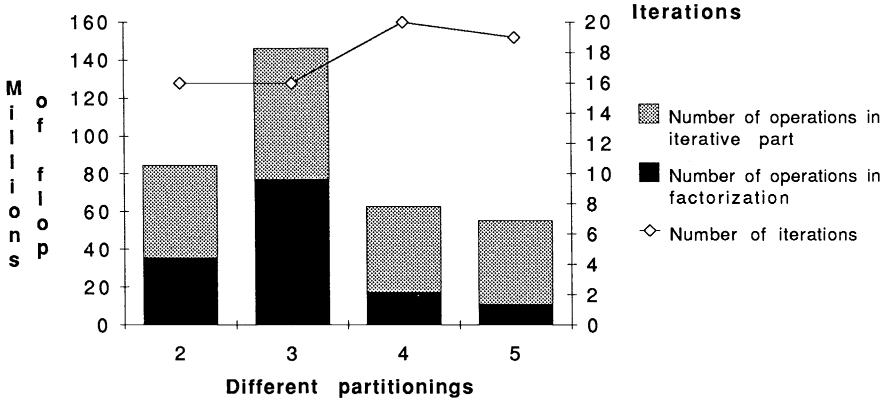


FIG. 10.3.3. *Weinerfelt's test problem.* $\omega_k \leqq 10^{-14}$.

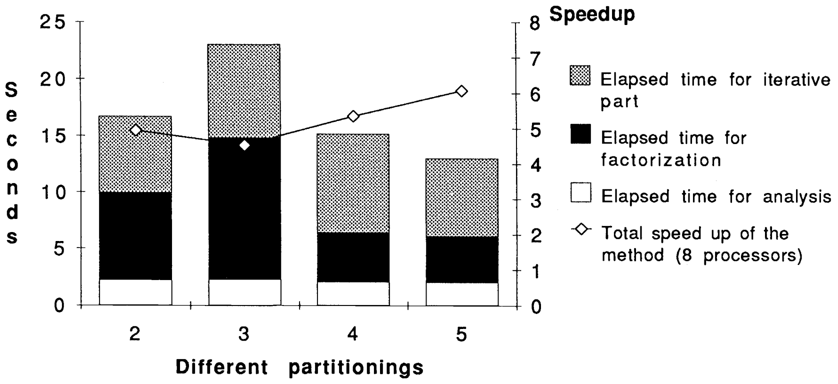


FIG. 10.3.4. *Weinerfelt's test problem.* $\omega_k \leqq 10^{-14}$.

the results in Figs. 10.3.3 and 10.3.4 and, in tabular form, in Table A.3 in the Appendix. Again, results for partitioning (1) are not shown in the figures, not because the direct solution of the augmented system with the original matrix does not fit in the memory of the Alliant, but because it takes so many operations and so much time that it would not make sense to display it on the same scales as for the other partitionings. However, the results from partitioning (1) can be found in Table A.3 in the Appendix.

Again, the algorithm converges very quickly. Additionally, the number of iterations is not very sensitive to the partitioning. We thus suspect a similar clustering of eigenvalues for the iteration matrices from the different partitionings, an effect which may be due to the diagonal dominance of the original matrix. Partitionings (4) and (5) give a good speedup. This supports what we have said in § 10.2, which is that the number of rows in the blocks is a reliable estimate for the amount of work that will be performed by MA27 in solving the augmented systems. In this case we cannot say that the factorization time dominates the time for iteration, but partitioning (5) still remains the best in terms of elapsed time, because now there is little difference in the number of iterations for the different partitionings and, of course, partitioning (5) still yields the least fill-in.

10.3.3. Problem 3 (two-dimensional PDE on a 64×64 grid). Problem 3 is the one on which Kamath and Sameh (1988) and Bramley and Sameh (1990) show that row projection methods are more robust than GMRES from the Yale package PCGPACK. The results, which are given in Figs. 10.3.5 and 10.3.6 and in Table A.4 in the Appendix, show that the block Cimmino method converges to high accuracy ($\omega_k \leqq TOL = 10^{-14}$), but the direct solution of the whole system is competitive, even though the solver MA27 does not exploit parallelism.

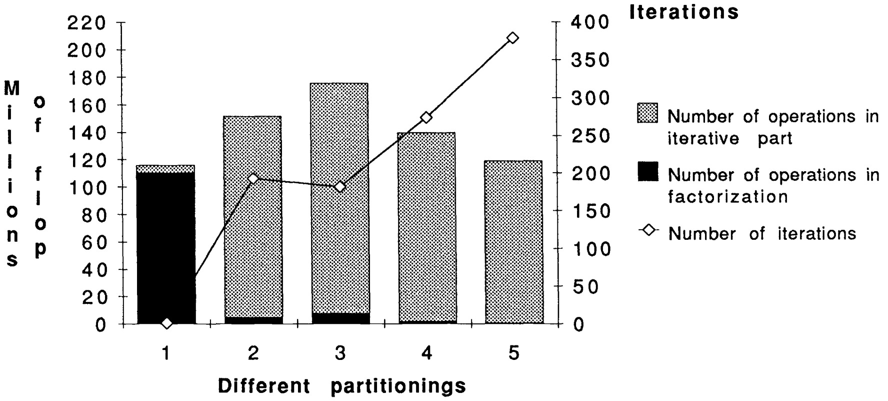


FIG. 10.3.5. *Problem* 3. $\omega_k \leqq 10^{-14}$.

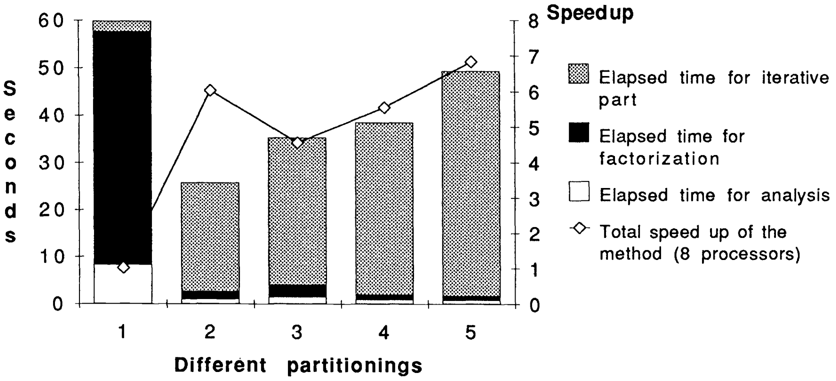


FIG. 10.3.6. *Problem* 3. $\omega_k \leqq 10^{-14}$.

The time for the iterations now completely dominates the time for factorization, and partitioning (3) requires the least number of iterations. Partitioning (2) with the eight equal-sized large blocks is the fastest. We remark that for partitionings (2), (4), and (5), the number of operations performed decreases and the speedup increases, respectively, but the elapsed time for reaching the solution increases gradually from partitioning (2) to (4) and (5). This effect, which is more deeply felt for smaller blocks, is due to a large number of delayed pivots and 2×2 pivots generated by MA27 during the factorization; the former increases the work involved while the latter disrupt the vectorization of the factorization and also of the forward and backward substitutions performed by MA27CD at each iteration.

10.4. Improvements with ellipsoidal norms. In this section we will show improvements to the previous results by using the ellipsoidal norms introduced in § 9.

A wide range of values for the parameter α in the matrix G (see § 9) were tried to improve the results on all of our test problems. This was not very helpful for the two problems coming from fluid dynamics simulations. For these, the convergence (as shown in § 10.3) was already very fast without the ellipsoidal norm. Thus, on the one hand, the number of iterations could be decreased by a small amount only and, on the other hand, more ill conditioned subsystems created more problems for the factorization. For this reason results are not given in this section for the two problems of Weinerfelt and Perrel. However, for the test coming from the set of Kamath and Sameh problems, and for which the convergence was quite slow, the use of ellipsoidal norms improved the results significantly.

For this test problem, better results, in terms of both accuracy and convergence, were obtained for all the partitionings on which we could apply this kind of ellipsoidal norm. These were partitionings (2), (3), and (4). Partitioning (1) is only one block, and partitioning (5), as we said in § 9, is one on which these ellipsoidal norms are equivalent to a complete scaling of the original matrix by a factor α.

Figure 10.4.1. shows, for Problem 3 with partitioning (4), the convergence for different values of the parameter α. The tests showed that it was not worth taking a very large α, but that the best improvements in convergence were obtained for α of order 10. Larger values for α introduce some oscillations in the convergence curve, as can be seen in Fig. 10.4.1. This phenomenon is due to the ill-conditioning of the subproblems when α increases. If iterative refinement (see Arioli, Duff, and de Rijk (1988)) is used in the computation on the subproblems, then the accuracy reached is the same, and the convergence behaviour for large α approaches that for α of order 10 but is never better than it.

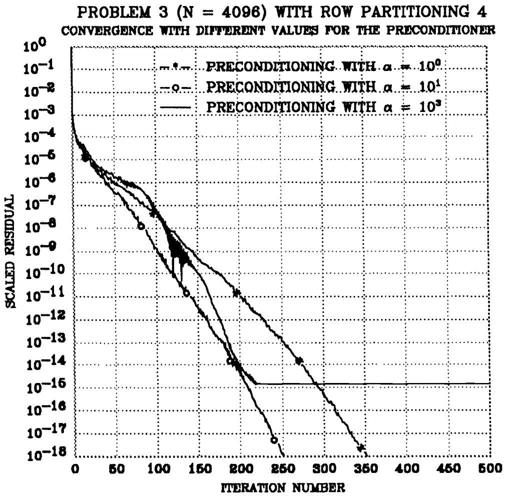


FIG. 10.4.1. *Effect of ellipsoidal norms. Two-dimensional* PDE *Problem* 3. *Partitioning* (4).

Table A.5 in the Appendix gives iteration counts, times, and flop counts obtained using ellipsoidal norms with $\alpha = 10$, on Problem 3 for partitionings (2), (3), and (4), and for a value of *TOL* equal to 10^{-14} in the stopping criterion.

The ratio between the number of flops for factorization before and after the preconditioning varies with the different partitionings. For example, in Problem 3, these ratios are 1.54 (partitioning (2)), 1.4 (partitioning (3)), and 1.47 (partitioning (4)).

10.5. Memory requirements. In this section we give the memory (integer and real) required by MA27 when solving the subproblems. These numbers do not take into account the arrays needed in the algorithm to perform all the other stages of the computations (conjugate gradient acceleration, etc.). These missing numbers can, however, be directly determined knowing the size of the original matrix and the partitioning in use.

It can be seen from Table 10.5.1 that the memory required decreases as the size of the blocks in the partitionings decreases.

TABLE 10.5.1
Memory requirements in thousands of words for handling the LU *factors of the different blocks.*

	Problem 1 $N = 20{,}700$ $NZ = 511{,}050$		Problem 2 $N = 7{,}680$ $NZ = 113{,}760$		Problem 3 $N = 4{,}096$ $NZ = 20{,}224$	
Part.	Int.	Real	Int.	Real	Int.	Real
1	—	—	123	4,328	65	697
2	368	4,881	171	611	50	167
3	400	6,328	142	907	55	207
4	430	3,539	187	429	48	103
5	676	1,067	172	375	60	56

We do not want, however, to focus too much on these numbers because the version of MA27 we are using does not perform well on augmented systems since the analysis phase assumes that the diagonal of the matrix is full. For instance, the ratio between the estimated amount of real storage (given by the analysis) and the actual amount of real storage used, can be as high as 2.5 for Problem 1 with partitioning (3) (the one with the largest blocks). A new version of MA27 that recognizes and exploits the zeros on the diagonal and behaves much better in terms of fill-in is being developed now at Rutherford Appleton Laboratory (Duff et al. (1991)).

Table 10.5.2 shows the memory required with and without the preconditioner, with $\alpha = 10$, for Problem 3. We see the bad effect on the fill-in due to the ill-conditioning that our scaling introduces in the augmented systems. In particular, partitioning (3) gives the least increase in memory requirement when going from the unpreconditioned

TABLE 10.5.2
Memory requirements in thousands of words for handling the LU *factors with and without preconditioning.*

	Problem 3 $N = 4{,}096$ $NZ = 20{,}224$				
	Unpreconditioned		Preconditioned		
Part.	Int.	Real	α	Int.	Real
2	65	167	10	53	212
3	55	207	10	56	247
4	48	103	10	48	125

case to the preconditioned one (19 percent more). In this partitioning, the size of the interface is minimized; therefore, the size of the overlapping part (see § 9) is minimized and the augmented systems are not strongly affected by the scaling.

10.6. More experiments. Bramley (1989) and Bramley and Sameh (1990) describe implementations of the block Cimmino and block SSOR algorithms that are different from the ones illustrated here. They test the robustness of these row projection methods on a set of linear systems coming from the discretization of three-dimensional partial differential equations by the seven-point formula.

We will now present more tests related to one of their test problems in order to illustrate the difference between our approach and theirs. Consider the following three-dimensional partial differential equation defined on the unit cube

$$u_{xx}+u_{yy}+u_{zz}+100xu_z-yu_y+zu_z+\frac{100(x+y+z)u}{xyz}=\mathbf{F}$$

where the right-hand side **F** is computed using the solution

$$u=\exp(xyz)\sin(\pi x)\sin(\pi y)\sin(\pi z).$$

The test problem is obtained by discretizing the previous equation by the seven-point finite-difference formula on a 24×24×24 mesh. This problem is Problem 3 in Bramley and Sameh (1990) and Problem 5 in Bramley (1989).

Bramley and Sameh implement block Cimmino and block SSOR using the normal equations approach for computing the projections instead of the augmented system approach described in § 6. According to this choice, they analyse the most suitable row partitioning strategies, considering other partitionings than the two-block one. In fact, taking advantage of the nested level of tridiagonal structures in the seven-point finite-difference operator matrices, we can define the following row partitionings:

(1) A two-block partitioning of the kind (5) described in § 10.2, with 12 blocks of 2×24×24 rows each.

(2) A three-block partitioning (see Kamath and Sameh (1988)) with 24 blocks of 24×24 rows each.

(3) A six-block partitioning, with 24×8 blocks of 3×24 rows each.

(4) A nine-block partitioning, with 24×24 blocks of 24 rows each.

Partitioning (4) is reported by Bramley and Sameh (1990) as the one giving the best average performance.

In the runs, the value of *TOL* in the stopping criterion has been set to $TOL=10^{-11}$ because it was the value giving results similar to those provided in Bramley (1989, p. 158) and in Bramley and Sameh (1990) in terms of the residual ($\|\mathbf{Ax}^{(k)}-\mathbf{b}\|_\infty$) and the error ($\|\mathbf{x}^{(k)}-\mathbf{x}\|_\infty$). For ease of comparison, we give in Table 10.6.1 the values for

TABLE 10.6.1
Behaviour of the block Cimmino method on an Alliant FX/80 (eight processors). Three-dimensional partial differential equation. $\omega_k \leqq 10^{-11}$.

		Elapsed time (seconds)				
Part.	No. iter.	Analysis	Factorization	Iterations	$\|\mathbf{Ax}^{(k)}-\mathbf{b}\|_\infty$	$\|\mathbf{x}^{(k)}-\mathbf{x}\|_\infty$
1	128	6.6	13.8	103.9	1.34E−5	9.57E−4
2	127	4.7	6.7	83.8	1.25E−5	8.37E−4
3	383	7.9	17.3	548.8	1.35E−5	1.85E−3
4	205	14.1	20.5	705.4	1.27E−5	7.90E−4

the errors and the residuals reached in the tests. However, it must be observed that the value of ω_k does not decrease much after it has been reduced to 10^{-11}.

The experiments show that the behaviour of the number of iterations as a function of the partitioning is quite irregular. Moreover, for this problem and with the partitioning (4), Bramley and Sameh (1990) report that the block Cimmino algorithm converges in 572 iterations when the block SSOR algorithm converges in 666, with a final value for the residual of 10^{-5}. The considerable difference from our number of iterations of 205 for the block Cimmino algorithm can be explained by the difference in the two stopping criteria.

The augmented system approach performs very poorly in term of efficiency with blocks having a small number of rows, as illustrated in the elapsed time of partitioning (4). However, with bigger blocks, as for partitionings (1) and (2), it takes care of ill-conditioning within the subproblems and yields the fastest convergence. In particular, for the augmented system approach, partitioning (2) is the most efficient, whereas Bramley (1989, p. 69) rejects the same partitioning because of instability problems in solving the normal equations.

Finally, the elapsed times shown in Table 10.6.1 might be not very attractive compared to those shown in Bramley (1989, p. 158) where convergence is obtained in 76 seconds. However, the elapsed time obtained using partitioning (2) is only 1.3 times greater, and significant improvement can be expected from the use of a new version of MA27 specially tuned for augmented systems.

11. Conclusions. In this paper, we have introduced a method for the solution of large sparse linear systems. There are many different components to this method, most of which we have explored here, but some of which warrant further study. The main components are: the partitioning, the choice of the method for solving the subproblems, the iterative scheme used, and the scaling (ellipsoidal norm) employed. We have shown that we can, for block tridiagonal matrices, partition the system cleverly in order to reduce the number of conjugate gradient iterations, while maintaining a good degree of parallelism.

Our numerical experiments reinforce what we expect from the theory, showing that we can vary the number of iterations and tune the speedup, using simple heuristics like the size of the subblocks in the two-block partitioning.

There are, however, some problems still requiring further work or investigation. First, the method is very dependent on the behaviour of MA27, which can generate much fill-in, as in Problem 1, or induce many 2×2 pivots and delayed pivots and perform poorly, as in Problem 3. This is because MA27 considers all diagonal entries as nonzero, which is not the case for the augmented systems. It then picks a poor pivot order at the beginning and corrects it during the factorization. This situation is being improved at the moment at Rutherford Appleton Laboratory by Duff et al. (1991). We also need to study the phenomenon of clustering of the eigenvalues, which can be quite pronounced for some matrices (see Problem 2) and seems to be linked to the partitioning. We effectively saw in all our tests that partitioning (2) with eight equal-sized large blocks generally requires the least number of iterations, although the size of the interface block for this partitioning is the largest.

The results show that the block Cimmino method with conjugate gradient acceleration can be considered a robust iterative solver similar to the conjugate gradient accelerated block SSOR method of Kamath and Sameh (1988). In fact, since we use conjugate gradient acceleration, we know that we must converge in exact arithmetic within at most n iterations, n being the size of the system. Moreover, we know that we can estimate the number of nonzero eigenvalues by considering the size of the interface block. Then, knowing the influence of the spectrum of the iteration matrix on the conjugate gradient method, we can use this as an upper bound for the number of iterations in order to stop the iterative process if it will not reach the required accuracy.

It is also possible to interpret our method as a preconditioned conjugate gradient method, considering the block Cimmino scheme as an implicit preconditioner and the use of ellipsoidal norms as an explicit preconditioner. From the experiments, the block Cimmino method appears to be a good preconditioner for the CG scheme (with respect to robustness) and, when the convergence would be otherwise slow, the ellipsoidal norms appear to be good complementary preconditioners. From the behaviour of the method, we can, as a first conclusion, say that if the time for factorization is much larger than the time for the iterations, it is better to look for partitionings having small blocks to minimize the fill-in and increase the speedup. But if the time for iterations is the larger, then the use of ellipsoidal norms as preconditioners can help greatly. We still need, however, to study and better understand the effect of these ellipsoidal norms on the convergence of the method and, if possible, to determine when they work well.

Our iterative scheme appears to be reliable, at least for general block tridiagonal matrices and is rather friendly, since it is easy to tune the speedup and the number of iterations. For all the examples, we have separated the time for analysis and the time for factorization because, in a time-dependent problem, for instance, we can avoid the analysis after the first time step, setting the block tridiagonal structure and the partitioning at the beginning and performing the analysis only once.

A particular strength of our approach is that it is not necessary to have disjoint blocks in the partitions in order to exploit parallelism, thus widening the set of applicable partitions. We plan to investigate this further, extending the technique to more general systems. With these other partitionings, we hope to decrease the ratio of the number of iterations between block Cimmino and block SSOR, enlarging the range of applicability of block Cimmino compared to block SSOR. We also need to consider other kinds of ellipsoidal norms, for instance, for partitionings that involve a complete overlap of the columns.

Our final goal is to make the algorithm as complete and as friendly as possible, by designing heuristics for automatically choosing a good partitioning, a good ellipsoidal norm, or a combination of the two, and also a heuristic which can choose automatically between block Cimmino and block SSOR.

Appendix. Tables of results of numerical experiments.

TABLE A.1

Behaviour of the block Cimmino method on an Alliant FX/80 (eight processors). Perrel's test problem. $\omega_k \leqq 10^{-9}$.

		No. operations (millions)		Elapsed time (seconds)			
Part.	No. iter.	Factorization	Iterations	Analysis	Factorization	Iterations	Speedup
2	24	709	539	6.7	105.9	51.9	4.0
3	18	1345	519	8.9	234.0	54.9	2.9
4	24	405	405	7.3	64.0	40.0	4.5
5	33	30	214	7.2	12.9	34.4	6.1

TABLE A.2

Behaviour of the block Cimmino method on an Alliant FX/80 (eight processors). Perrel's test problem. $\omega_k \leqq 10^{-14}$.

		No. operations (millions)		Elapsed time (seconds)			
Part.	No. iter.	Factorization	Iterations	Analysis	Factorization	Iterations	Speedup
2	48	709	1058	6.7	105.9	100.6	4.1
3	42	1345	1177	8.9	234.0	125.4	3.1
4	52	405	860	7.3	64.0	85.2	4.6
5	74	30	474	7.2	12.9	76.1	6.0

TABLE A.3

Behaviour of the block Cimmino method on an Alliant FX/80 (eight processors). Weinerfelt's test problem. $\omega_k \leqq 10^{-14}$.

		No. operations (millions)		Elapsed time (seconds)			
Part.	No. iter.	Factorization	Iterations	Analysis	Factorization	Iterations	Speedup
1	1	2257.2	35.3	14.7	773.6	10.7	1.0
2	16	35.3	49.3	2.2	7.7	6.7	4.9
3	16	76.9	69.4	2.3	12.5	8.2	4.5
4	17	17.1	45.6	2.1	4.3	8.8	5.4
5	19	10.8	44.2	2.0	4.1	7.0	6.1

TABLE A.4

Behaviour of the block Cimmino method on an Alliant FX/80 (eight processors). Problem 3. $\omega_k \leqq 10^{-14}$.

		No. operations (millions)		Elapsed time (seconds)			
Part.	No. iter.	Factorization	Iterations	Analysis	Factorization	Iterations	Speedup
1	1	110.1	5.7	8.3	49.2	2.3	1.0
2	193	4.4	147.3	1.0	1.6	23.1	6.0
3	182	7.4	168.1	1.5	2.5	31.3	4.6
4	274	1.8	138.0	0.9	1.1	36.5	5.6
5	379	0.4	118.3	0.8	0.9	47.6	6.8

TABLE A.5

Results for Problem 3 *with ellipsoidal norms,* $\alpha = 10$. $\omega_k \leqq 10^{-14}$.

		No. operations (millions)		Elapsed time (seconds)		
Part.	No. iter.	Factorization	Iterations	Analysis	Factorization	Iterations
2	95	6.8	90.8	1.0	2.0	15.8
3	105	10.3	115.0	1.5	3.3	23.0
4	195	1.3	117.0	0.9	2.6	35.2

REFERENCES

M. ARIOLI, I. S. DUFF, AND P. P. M. DE RIJK (1989), *On the augmented system approach to sparse least-squares problems*, Numer. Math., 55, pp. 667–684.

A. BJÖRCK AND G. H. GOLUB (1973), *Numerical methods for computing angles between linear subspaces*, Math. Comp., 27, pp. 579–594.

R. BRAMLEY (1989), *Row projection methods for linear systems*, Report No. 881, Center for Supercomputing Research and Development, University of Illinois, Urbana, IL.

R. BRAMLEY AND A. SAMEH (1990), *Row projection methods for large nonsymmetric linear systems*, Report No. 957, Center for Supercomputing Research and Development, University of Illinois, Urbana, IL.

S. L. CAMPBELL AND C. D. MEYER, JR. (1979), *Generalized Inverses of Linear Transformations*, Pitman, London.

I. S. DUFF AND J. K. REID (1983), *The multifrontal solution of indefinite sparse linear systems*, ACM Trans. Math. Software, 9, pp. 302–325.

I. S. DUFF, A. M. ERISMAN, AND J. K. REID (1986), *Direct Methods for Sparse Matrices*, Oxford University Press, London.

I. S. DUFF, N. I. M. GOULD, J. K. REID, J. A. SCOTT, AND K. TURNER (1991), *Factorization of sparse symmetric indefinite matrices*, IMA J. Numer. Anal., 11, pp. 181–204.

T. ELFVING (1980), *Block-iterative methods for consistent and inconsistent linear equations*, Numer. Math., 35, pp. 1–12.

G. H. GOLUB AND W. KAHAN (1965), *Calculating the singular values and pseudoinverse of a matrix*, SIAM J. Numer. Anal., 2, pp. 205–225.

G. H. GOLUB AND C. F. VAN LOAN (1989), *Matrix Computations*, Second Edition, The Johns Hopkins University Press, Baltimore, MD.

L. A. HAGEMAN AND D. M. YOUNG (1981), *Applied Iterative Methods*, Academic Press, New York, London.

C. KAMATH AND A. SAMEH (1988), *A projection method for solving nonsymmetric linear systems on multiprocessors*, Parallel Comput., 9, pp. 291–312.

W. OETTLI AND W. PRAGER (1964), *Compatibility of approximate solution of linear equations with given error bounds for coefficients and right-hand sides*, Numer. Math., 6, pp. 405–409.

C. R. RAO AND S. K. MITRA (1971), *Generalized Inverse of Matrices and its Applications*, John Wiley, Chichester, New York, Brisbane, Toronto.

F. SLOBODA (1988), *A projection method of the Cimmino type for linear algebraic systems*, Tech. Report n. 16, Dipartimento di Matematica, University of Bergamo, Bergamo, Italy.

SIAM J. Sci. Stat. Comput.
Vol. 13, No. 1, pp. 71-87, January 1992

SPARSE APPROXIMATION FOR SOLVING INTEGRAL EQUATIONS WITH OSCILLATORY KERNELS*

FRANCIS X. CANNING†

Abstract. An integral equation formulation of Helmholtz's equation is considered as an example of a problem with an oscillatory kernel. For such a problem, the field due to a localized source contains a phase that is a function of position. This allows directional radiation patterns to be produced by using a phase cancellation when constructing "extended" sources. This directional property may be used in solving the integral equation. The region containing the sources is decomposed into subregions, each containing many such extended sources. The directional properties of these extended sources result in an $N \times N$ full matrix having many very small elements (which may be approximated by zero), and approximately Order $[N]$ large elements. A matrix defining the transformation between localized and extended source formulations in two dimensions is introduced, and its condition number is calculated. This transformation is effective whenever a large enough number of the unknowns correspond to "smooth" regions containing the sources. Numerical examples for a formulation of scattering of waves obeying Helmholtz's equation are considered in two dimensions. A sample calculation illustrates that the sparse matrix that results allows a solution with Order $[N]$ operations per iteration. A permutation of matrix rows and columns is introduced and an example is given in which it moves large matrix elements towards the diagonal. This suggests that incomplete LU preconditioning might be quite effective. Even without preconditioning, the resulting method is the most efficient available for general surface problems using the Helmholtz equation.

Key words. integral equations, sparse matrices, iterative methods, preconditioners, Helmholtz equation

AMS(MOS) subject classification. 65F10

1. Introduction.

Integral equations with an oscillatory kernel arise from problems with a characteristic distance, i.e., a wavelength. Numerical solutions to these problems by either differential or integral equation approaches become computationally expensive for regions large compared to this wavelength. For example, consider two-dimensional problems involving "sources" on a surface with a characteristic size of L. For the exterior problem in an unbounded domain, a differential equation approach using a local radiation (i.e., absorbing) boundary condition typically requires the outer boundary to be at a distance away that scales as L [5]. Such a calculation requires a number of unknowns, N_d, and a storage, S_d, which scale with the wavenumber $k = 2\pi/\lambda$ as

$$N_d \sim (kL)^2, \qquad S_d \sim (kL)^2. \tag{1}$$

However, an integral equation only requires the surface to be discretized while the resulting matrix is full, so the resulting number of unknowns N and storage S scale as

$$N \sim kL, \qquad S \sim (kL)^2. \tag{2}$$

This paper gives a method for approximating the usual full matrix by a sparse one, thus reducing the storage required to be proportional to kL. When iterative techniques are used to solve the sparse equation, then each iteration for the integral equation method will only take a time proportional to kL, as opposed to proportional to $(kL)^2$ for the differential equation approach.

* Received by the editors April 5, 1990; accepted for publication February 5, 1991. This work was partially supported by Office of Naval Research contract N00014-91-C-0031.

† Science Center, Rockwell International Corporation, 1049 Camino Dos Rios, Thousand Oaks, California 91360.

The above motivation suggests that we study the full matrix that results from discretizing an integral equation containing an oscillatory kernel. Consider the generic form

$$\mathbf{Zj}=\mathbf{e}, \tag{3}$$

where $\mathbf{Z}$ is a square $N \times N$ matrix of complex numbers, while $\mathbf{j}$ and $\mathbf{e}$ are N-dimensional vectors. This form is general enough to apply to both first and second kind Fredholm equations, etc. It is desired to find a square matrix $\mathbf{A}$ such that the new matrix $\mathbf{T}$ defined by

$$\mathbf{T}=\mathbf{AZA}^{*} \tag{4}$$

has of Order$[N]$ large elements and very many elements that are quite small in magnitude. The resulting matrix problem to solve is

$$\mathbf{Ti}=\mathbf{v}, \quad \text{where } \mathbf{j}=\mathbf{A}^{*}\mathbf{i} \quad \text{and} \quad \mathbf{v}=\mathbf{Ae}. \tag{5}$$

If $\mathbf{A}$ and $\mathbf{Z}$ are both well-conditioned matrices, then $\mathbf{T}$ may be approximated by a sparse matrix.

At this point some motivation will be given for the type of matrices $\mathbf{A}$ chosen below. Consider a problem involving sources on a one-dimensional surface in a two-dimensional space. The element (μ, ν) of the matrix $\mathbf{Z}$ may be found from

$$(\mathbf{Z})_{\mu,\nu}=\int g(s,s')\exp[ik(\alpha s-\alpha' s')]t_{\mu}(s)b_{\nu}(s')\,ds\,ds'. \tag{6}$$

The product of $g(s, s')$ with the exponential function gives the kernel of the integral equation to be approximated. The integral operator has an oscillatory kernel, by which we mean that for the appropriate choices of α and α', $g(s, s')$ will be slowly varying over distances of several wavelengths. The function $g(s, s')$ and the parameters α and α' may be (slowly varying) functions of μ and ν, though we do not explicitly show these subscripts. The functions $t_{\mu}(s)$ and $b_{\nu}(s')$ are used to form the discrete version of the integral equation. These functions of arc length are each assumed to be nonzero only for s (or s') in some interval shorter than a wavelength. If desired, one may let $t_{\mu}(s)$ (or $b_{\nu}(s')$) be a delta function (or a sum of delta functions) to achieve a discretization such as that in Nystrom's method.

The elements of the matrix $\mathbf{T}$ are given by an integral similar to that in (6), but with $t_{\mu}(s)$ replaced by $t'_{\mu}(s)$ and with $b_{\nu}(s')$ replaced by $b'_{\nu}(s')$. In each case the primed functions are the linear combination of the unprimed functions determined by the matrix $\mathbf{A}$. In order to motivate a choice for these primed functions (and for the corresponding $\mathbf{A}$), consider the approximation

$$(\mathbf{T})_{\mu,\nu}\approx g(s,s')\int \exp[ik(\alpha s-\alpha' s)]t'_{\mu}(s)b'_{\nu}(s')\,ds\,ds'. \tag{7}$$

Clearly, a phase cancellation for each integral may be achieved by letting both $t'_{\mu}(s)$ and $b'_{\nu}(s')$ be nonzero over some "extended" region, meaning a region of length several wavelengths or more. An earlier paper [7] contrasted two ill-conditioned choices for t' and b' with a well-conditioned choice. For the ill-conditioned choices, $t'_{\mu}(s)$ and $b'_{\nu}(s')$ had an approximately constant phase. They resulted in a $\mathbf{T}$ with all of the matrix elements between two different "extended" regions of the "body" being very small. However, it is known that these regions do interact, so we might hope that the condition number would be reduced by allowing this interaction to occur through one or more large matrix elements between these regions. This suggested constructing an $\mathbf{A}$ for which a small number of extended sources on each region gives rise to (a small number of) large matrix elements. This is accomplished by allowing the functions $t'_\mu(s)$ and $b'_\nu(s')$ to have a variety of rates of phase evolution as a function of s or s'. A simple implementation of this idea gives the unitary **A** described in [7] and [4]. Since the important interactions are localized to a small (i.e., Order$[N]$) number of elements of **T**, this method might be called the "interaction matrix localization" method.

The physical principle involved is quite simple to state. The scattering body is (conceptually) broken into many regions, and on each region there are many sources $b'_\nu(s')$. The subscript ν determines both the region where that source is centered and the rate of its linear phase evolution. The rate of phase evolution determines the narrow range of directions in which it radiates strongly. The resulting matrix element connecting such a source with the radiated field as observed by some "testing" function $t'_\mu(s)$ in another region will be large or small depending on whether that region is located within this range of directions. This effect is increased by similarly choosing the "testing" functions to give them the same directional dependence.

In order to understand the largeness and the smallness of the matrix elements resulting from this approach, note that for a specific value of the pair (μ, ν), the parameters α, α', γ, γ' may be chosen so that integral is in terms of three exponential functions and the three slowly varying functions $g(s, s')$, $t''_\mu(s)$, and $b''_\nu(s')$. The calculation now is in the form

$$(8) \qquad (\mathbf{T})_{\mu,\nu} = \int g(s, s') \exp[ik(\alpha s - \alpha' s')] \exp[i\gamma s] t''_\mu(s) \exp[i\gamma' s'] b''_\nu(s')\, ds\, ds'.$$

These integrals are over the regions where $b'_\nu(s)$ and $t'_\mu(s')$ are nonzero. Clearly, when the net rate of phase evolution as a function of s or s' is approximately zero (e.g., $k\alpha + \gamma \approx 0$), then the corresponding integral will give a large answer that may be found by approximating the integrand by a constant. When this condition is not met, an approximation to the integral may be found by repeated integration by parts, where the product of the three exponential functions is integrated at each step. For the integral involving s, the first integrated part will be zero, provided $t''_\mu(s)$ is zero at both endpoints of the integration. Furthermore, if the first derivative of $t''_\mu(s)$ is also zero at these endpoints, then the next integrated part will also be zero, and so on. Thus, requiring $t''_\mu(s)$ to go to zero smoothly may increase the smallness of many matrix elements [7]. These conclusions and many further details are well known to antenna designers. Indeed, what is being done might be thought of as thinking of the scattering surface as a collection of antennas where many modes are considered on each antenna. For our purposes we note that a magnitude taper going smoothly to zero at the edges of an antenna produces a more distinct beam. Both the form of **A** and its condition number depend on this taper. A larger class of candidates for **A** is introduced in the next section, and their condition numbers are explicitly evaluated in terms of the taper used.

If one is dealing with a second kind Fredholm equation, then the matrix **Z** involves the sum of a multiple of the identity matrix and a matrix resulting from an operator with an oscillatory kernel. For various preconditioning strategies, such as incomplete LU (ILU) decomposition, it is desirable to be able to transform all of **Z** in the same way. Fortunately, it can be shown that as long as the functions $b'_\nu(s)$ and $t'_\mu(s')$ are very smooth (i.e., their Fourier series contain only low frequencies), then the matrix $\mathbf{AA}^*$ will have Order$[N]$ large elements. However, that issue will not be further discussed in this paper. The discussion given here is intended only to outline the motivation for the choice of matrix **A**. A more extensive discussion, in the context of a more restrictive class of matrices, was given in [7]. It was predicted there that, for the two-dimensional case, the number of large matrix elements is Order $[N]$ for an $N \times N$ matrix. The generalization to the matrices $\mathbf{A}$ described below is confirmed by the numerical study presented in § 3.

The present method may be contrasted with other fast solution methods for integral equations. The multilevel scheme of Beylkin, Coifman, and Rokhlin [1] using wavelet transforms does not apply to problems with an oscillatory kernel (i.e., their coarsest level must have at least two points per wavelength). The more standard multilevel methods (e.g., [2]) also have this two points per wavelength limitation. An earlier method by Rokhlin [12], [13] does apply to the oscillatory kernels of scattering theory and has an advantage over the present approach in that it does not require any smoothness property of the scatterer. Rokhlin's method allows the matrix $\mathbf{Z}$ to be applied to a trial solution $\mathbf{j}$ in Order $[N \log N]$ operations, by expressing $\mathbf{Z}$ in terms of blocks of convolutions, etc. The present method generates a truly sparse matrix that may allow efficient preconditioning, which is important since scattering by nonpenetrable bodies generally results in a $\mathbf{Z}$ with a condition number not very different from N (assuming that the number of points per wavelength is fixed at a small constant as N varies, which is the standard practice). Thus, although integral equations often result in better condition numbers than differential equations, the number of iterations required is still somewhat large. This observation is all the more significant when problems with multiple right-hand sides ($\mathbf{e}$) are considered (the number of right-hand sides is proportional to N for some cases of interest). Some iterative results are given in § 4, where a permutation of rows and columns is introduced that may allow effective preconditioning by ILU decomposition. As will be seen below, an additional advantage of the present approach is its simplicity, both in itself and as an addition to existing computer programs.

2. A class of matrices. In this section, transformation matrices $\mathbf{A}$ having a general form are considered. These matrices should be sparse, and the elements of the resulting $\mathbf{T}$ should be easily computed. This leads us to consider matrices where the sources implicit in the resulting $\mathbf{T}$ [7] have linear progressive phase (as a function of arc length along the surface), since multiplication by the corresponding $\mathbf{A}$ may then be accomplished using fast Fourier transforms, as described below. $\mathbf{A}$, $\mathbf{T}$, and $\mathbf{Z}$ are $N \times N$ matrices of complex numbers, and we choose N to be the product of integers m and n. The matrix $\mathbf{A}$ will be constructed from n blocks by n blocks, where each block is an $m \times m$ matrix.

To construct $\mathbf{A}$, we consider an $m \times m$ unitary matrix $\mathbf{U}$ and $m \times m$ real diagonal matrices $\mathbf{C}$, $\mathbf{D}$, and $\mathbf{E}$, which are defined in terms of their elements in row α and column β as

$$(\mathbf{U})_{\alpha,\beta} = m^{-1/2} \exp\{2\pi i[\alpha-(m+1)/2][\beta-(m+1)/2]/m\}, \tag{9}$$

$$(\mathbf{C})_{\alpha,\beta} = \delta_{\alpha,\beta} c_\alpha, \qquad (\mathbf{D})_{\alpha,\beta} = \delta_{\alpha,\beta} d_\alpha, \qquad (\mathbf{E})_{\alpha,\beta} = \delta_{\alpha,\beta} e_\alpha. \tag{10}$$

One possible form for $\mathbf{A}$ is the block circulant matrix whose (j, k) submatrix is given by

$$(\mathbf{A})_{(j,k)} = \delta_{j,k+1}\mathbf{UC} + \delta_{j,k}\mathbf{UD} + \delta_{j,k-1}\mathbf{UE}, \tag{11}$$

where the indices on the delta functions are to be taken mod n, e.g.,

$$\delta_{0,n} = \delta_{0,0} = 1, \qquad \delta_{1,n+1} = \delta_{1,1} = 1. \tag{12}$$

In this paper, only $\mathbf{A}$ in the form given above is considered. Clearly, additional diagonal subblocks could be added to (11), and many of the results in this paper immediately generalize to that case. If **C** and **E** are taken to be zero matrices and **D** is taken to be the identity matrix, then the resulting matrix, which we call **A**′, is a block diagonal matrix, where each block describes a discrete Fourier transform of order m. That is, we define **A**′ by

$$(\mathbf{A}')_{(j,k)} = \delta_{j,k}\mathbf{U}. \tag{13}$$

Thus **A**′ is unitary, and when using it for **A** the 2-norm condition number of the resulting matrix **T** is the same as that of the **Z** it was derived from. Some numerical results using this matrix were given in [7] and [4].

To find the condition number of **A** having the general form (11), note that it can be factored as

$$\mathbf{A} = \mathbf{A}'\mathbf{W}, \tag{14}$$

where **W** is defined to have a submatrix at (j, k) given by

$$(\mathbf{W})_{(j,k)} = \delta_{j,k+1}\mathbf{C} + \delta_{j,k}\mathbf{D} + \delta_{j,k-1}\mathbf{E}. \tag{15}$$

Again, here and throughout the rest of this paper, the indices of the delta functions are to be taken mod n. This makes **W** block circulant. Since **A**′ is unitary, the 2-norm condition number of **A** is equal to that of **W**. Forming the product of **W** with its transpose,

$$\begin{aligned}(\mathbf{W}\mathbf{W}^t)_{(j,k)} = \{&\delta_{j,k}[\mathbf{C}^2+\mathbf{D}^2+\mathbf{E}^2] + \delta_{j,k+1}[\mathbf{DC}+\mathbf{ED}] + \delta_{j,k-1}[\mathbf{CD}+\mathbf{DE}]\\ &+ \delta_{j,k+2}[\mathbf{EC}] + \delta_{j,k-2}[\mathbf{CE}]\}.\end{aligned} \tag{16}$$

Transposing **W** is equivalent to interchanging **C** and **E**, so $\mathbf{W}^t\mathbf{W}$ may be found from the above equation by interchanging **C** and **E**. Doing so, and using that **C**, **D**, and **E** are diagonal and thus commute,

$$\mathbf{W}\mathbf{W}^t = \mathbf{W}^t\mathbf{W}, \tag{17}$$

proving that **W** is normal. This allows the singular values of **W** to be calculated as the absolute values of its eigenvalues.

The factorization of **A** given in (14) allows an efficient method for multiplying **A** by an arbitrary $N \times N$ matrix. Rather than taking Order $[N^3]$ operations as for general matrices, multiplying separately by **W** and **A**′ takes Order $[N^2]$ operations for **W** (which has three or fewer nonzero elements per row) plus Order $[N^2 \log N]$ for **A**′, since fast Fourier transforms may be used. It is also important to notice that each $m \times m$ block of **T** may be calculated from a slightly larger block of **Z** by using (4) and (14). Thus if one wants to calculate the elements of **T** with magnitude larger than some value, this can be done one block at a time where only the larger elements are saved from block to block. This implies the total memory required is not that for N^2 matrix elements, but rather only for a small constant times m^2 elements (e.g., for a block of **Z** and of **T**) plus the number of large elements saved. We will see in the next section that typically one chooses m so that $m^2 \approx N$, giving each block about N elements. Since the number of large elements saved will also be linear in N, so will the total memory required in a computer program, even though all N^2 of the elements of **T** were calculated.

A trial form for the eigenvectors of **W** is the vector of length N whose jth segment is given by $(\mathbf{e})_j = \mathbf{v}z^{jl}$, where $\mathbf{v}$ is a vector of length m and the complex number z is defined by

$$z = \exp[2\pi i/n]. \tag{18}$$

Forming the product **We**, we find that its jth segment is given by

(19) $$(\mathbf{We})_j = \sum_k \{[\delta_{j,k+1}\mathbf{C} + \delta_{j,k}\mathbf{D} + \delta_{j,k-1}\mathbf{E}]\mathbf{v}z^{kl}\}.$$

For any (integer) value of l, z^{jl} is periodic in j with period n, so the eigenvalue equation will be satisfied whenever

(20) $$(\mathbf{C}z^{-l} + \mathbf{D} + \mathbf{E}z^{l})\mathbf{v} = \lambda\mathbf{v}.$$

This shows that the trial form does indeed generate all N of the eigenvectors of **W**. The matrix above operating on **v** is diagonal, allowing a trivial solution for the eigenvectors and eigenvalues. The N eigenvalues of **W** are

(21) $$\lambda_{l,s} = c_s z^{-l} + d_s + e_s z^{l} \qquad \text{for } l = 0, \cdots, n-1, \quad s = 1, \cdots, m.$$

The (2-norm) condition number of **A** is thus given by

(22) $$\kappa(\mathbf{A}) = \left\{\max_{l,s} \operatorname{Abs}(\lambda_{l,s})\right\} \Big/ \left\{\min_{l,s} \operatorname{Abs}(\lambda_{l,s})\right\}.$$

An alternate method for calculating $\kappa(\mathbf{A})$ would be to notice that the sth column of **W** is the pth column of the kth column of blocks, where

(23) $$s = p + (k-1)m.$$

If one symmetrically permutes the rows and columns of **W** so that the sth column of the permuted matrix is related to the original p and k by

(24) $$s' = k + (p-1)n,$$

then the resulting matrix is block diagonal and the blocks are circulant and are essentially tridiagonal (but with an upper right and lower left element included). This resulting matrix thus is normal and the elements of its sth block (i.e., c_s, d_s, e_s) immediately give (21). If (11) were generalized to allow additional blocks, then after the permutation just described the blocks would still be circulant but would then have more than three nonzero elements per row, so the generalization of (21) would have more than three terms.

The coefficients c_s for $s = 1, \cdots, m$, d_s for $s = 1, \cdots, m$, and e_s for $s = 1, \cdots, m$ taken collectively in this order correspond to the discretely sampled version of the taper function used on an antenna [7]. Use will now be made of our assumption that these coefficients are real numbers. One particular case of interest is for

(25) $$c_s, e_s \geqq 0 \quad \text{and} \quad d_s > c_s + e_s \qquad \text{for } s = 1 \cdots m, \quad \text{where } n \text{ is even,}$$

in which case

(26) $$\kappa(\mathbf{A}) = \left\{\max_s [d_s + c_s + e_s]\right\} \Big/ \left\{\min_s [d_s - (c_s + e_s)]\right\}.$$

The example that will be considered in our numerical calculations is motivated by a weighting function well known in antenna theory:

(27) $$f(r) = 1 + \cos[\pi r/m] = 2\cos^2[\pi r/2m], \qquad -m < r < m,$$

(28) $$f(r) = 0 \quad \text{otherwise.}$$

This function will be sampled at discrete, uniformly spaced points. When m is odd, a symmetric choice of sampling points gives

(29) $$c_s = f(s - 1.5m - 0.5) \quad \text{for } s = 1, \cdots, m,$$

(30) $$d_s = f(s - 0.5m - 0.5) \quad \text{for } s = 1, \cdots, m,$$

(31) $$e_s = f(s + 0.5m - 0.5) \quad \text{for } s = 1, \cdots, m.$$

Sample numerical calculations using this particular form of **A** were given in [8]. Comparing this with (21) and assuming that n is even we find that

$$\lambda_{\max} = d_{(m+1)/2} = 2, \tag{32}$$

$$\lambda_{\min} = d_1 - e_1 = \cos[\pi(-0.5+0.5/m)] - \cos[\pi(0.5+0.5/m)] \approx \pi/m. \tag{33}$$

Since **W** is normal, this gives a 2-norm condition number

$$\kappa(\mathbf{A}) \approx 2m/\pi. \tag{34}$$

If n is not even, this is an upper bound on the condition number, as follows from Gersgorin's theorem. The result of a numerical calculation of $\kappa(\mathbf{A})$ as a function of n for several block sizes m is given in Fig. 1. As n increases through odd integers, $\kappa(\mathbf{A})$ slowly approaches the upper bound of (34).

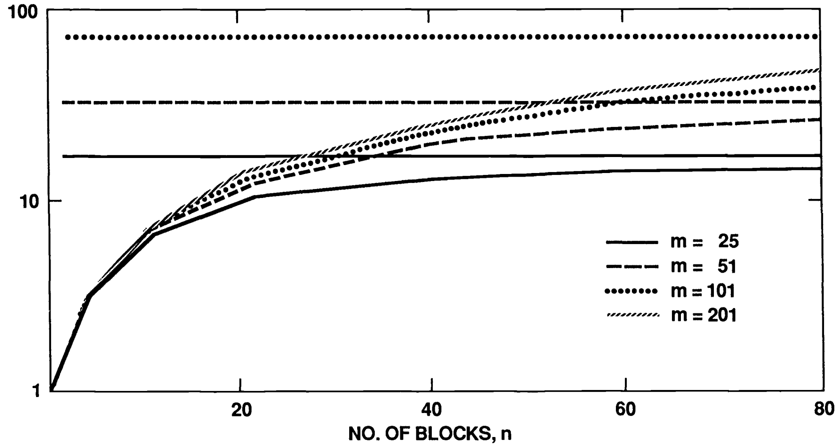


FIG. 1. *The 2-norm condition number, $\kappa(\mathbf{A})$, for $\mathbf{A}$ using (27)-(31) is plotted as a function of the number of blocks n, for several block sizes m. The results of calculations for even values of n are joined together, giving the horizontal lines. The results of calculations for odd values of n are similarly joined together, giving the curves that increase with n.*

The linear dependence of $\kappa(\mathbf{A})$ on m in (34) results from the linear behavior of $f(r)$ in (27) about $r = m/2$. Since it is theoretically interesting to find transformations with small κ, we consider modifying (27) so that $f(r)$ is still monotonically decreasing for Abs (r) increasing and Abs $(r) < m$, but so that for small Δr,

$$f(m/2+\Delta r) \approx \text{const.} + (\Delta r)^{1/\alpha}, \qquad \alpha > 1. \tag{35}$$

One can easily check that κ then satisfies

$$\kappa \sim m^{1/\alpha}. \tag{36}$$

While theoretically interesting, this result may not be especially useful in practice, since $\kappa(\mathbf{A})$ along with $\kappa(\mathbf{Z})$ gives only an upper bound on $\kappa(\mathbf{T})$, and that bound has been found to be quite pessimistic in several specific examples.

The conclusion that we have just reached is that in practice, the matrix **A** will be well conditioned enough so that **T** will be nearly as well conditioned as **Z**. It is important to stress the context in which this conclusion was reached. This paper describes a method for easing the severe limitations on problem size for computer solutions due to the storage and execution time restrictions of present and even near future computers.

For this reason, all methods used must be designed to work with the fewest possible unknowns (e.g., per wavelength) to reduce the order of the matrix to be solved. Thus, we should not be troubled that the method described here would not work in the limit that the number of points per wavelength used in the discretization approaches infinity (in that limit m approaches infinity and so does κ due to (34) or (36)).

3. Matrix element size. To give some numerical examples, we now specialize the discussion to consider the Helmholtz equation in an unbounded region in two dimensions with Dirichlet boundary conditions on a region Γ with boundary $\partial\Gamma$. When the field satisfying the Helmholtz equation in the exterior obeys an outgoing radiation condition and its (one-sided) normal derivative on the boundary exists, an integral equation for the "single layer source," $J(\rho)$, producing the field $E_i(\rho)$ on $\partial\Gamma$ is

$$E_i(\rho)=\int_{\partial\Gamma} H_0^{(2)}(k|\rho-\rho'|)J(\rho')\,ds' \tag{37}$$

where ρ is any vector on the boundary of the scatterer and ρ' is treated as a function of the arc length s' along the scatterer [9, § 3.5]. The scattered field external to Γ and on $\partial\Gamma$ due to the source J is found from

$$E_s(\rho)=-\int_{\partial\Gamma} H_0^{(2)}(k|\rho-\rho'|)J(\rho')\,ds'. \tag{38}$$

The sum E_i+E_s evaluated on $\partial\Gamma$ is clearly zero. If $E_i(\rho)$ is the value of an incident electric field at position ρ, then the total electric field, E_i+E_s, is zero on $\partial\Gamma$. This equation would then describe, for example, the scattering of an electric field aligned in the z-direction by a perfect electrical conductor which is in the shape of a cylinder, infinite (and uniform) in the z-direction [10]. The cross-section of this cylinder would be given by $\partial\Gamma$, which would lie in the x-y plane. The wavenumber k is defined as $2\pi/\lambda$, and $H_0^{(2)}$ is a Hankel function of zero order.

Although (37) is ill posed, that fact has not prevented the widespread practical use of matrix equations based on it in a variety of applications. This situation may seem strange, since it is well known that second kind integral equations may be discretized to matrix problems having condition numbers that are independent of N. At the same time, a first kind equation such as the one above is ill posed, and discretizations generally have condition numbers that grow with N. However, for the problem described above with a characteristic dimension L, the number of unknowns used in practice is nearly always a small constant times kL as kL varies. Thus, the relevant comparison in practice involves the condition numbers obtained with each method for the actual numbers of unknowns used.

To further illustrate this point, let the scatterer be a circle of radius a. For one choice of first and second kind equations the eigenvalues (and hence singular values since this geometry produces a circulant and therefore normal matrix) are

$$J_s(ka)H_s^{(2)}(ka)\to 1/s \quad \text{as } s\to\infty, \tag{39}$$

$$J_s(ka)H_s^{(2)\prime}(ka)\to 1/(\pi ka) \quad \text{as } s\to\infty. \tag{40}$$

In practice N and therefore the largest s used is proportional to ka; both approaches give matrices with similar condition numbers. A detailed discussion of the advantages of each approach including a justification for the discretization step for a first kind equation is beyond the scope of this paper. However, we do mention that there are some reasons for using a first kind equation, such as its applicability to both open and closed structures, its symmetry and the resulting savings in computer memory required and operation count [6], and the nonradiating character of the source associated with the resonant (interior) mode [3] at discrete values of k. While the numerical results presented here use (37), the method described for generating a sparse matrix is very general and also applies to second kind equations, etc.

For our calculations we will use a simple though quite crude discretization of (37), since the goal of this paper is to illustrate how our transformation and subsequent approximation *changes* an answer calculated by well-known techniques. This discretization may be described as Nystrom's method with a one-point rectangle rule, except for the diagonal elements where the asymptotic form of the Hankel function is used for the singular part. In the "moment method" context this is [10, § 3.2] delta function testing with pulse basis functions, with the same approximations to the integrals as above. Without examining under what conditions this (admittedly crude) method converges to the solution of the integral equation (37) as more sample points are taken, we observe that it has often been used with success at typical sampling densities. The form of **A** considered in § 2 is appropriate when the sampling density used to generate **Z** is kept constant, and the calculations that follow all have this property. Also, for this form of **A**, **Z** should be the discretization of the integral equation in terms of local unknowns. Using Nystrom's method, that means the quadrature formulas should involve only local points; using the moment method, it means the basis and testing functions (as in (6)) should be nonzero only in a local region, say, smaller than a wavelength or so. Using a variable sampling density for **Z** would require that **A** be modified so that the **T** created is the discretization of the integral equation in terms of expansion functions that have a phase evolution that is approximately linear with arc length along $\partial\Gamma$.

It was noted in [7] that the matrix **T** resulting from **A**′ may be thought of as giving the discretization of (37) in terms of an expansion of the sources $J(\rho')$ in the following functions. The scatterer is broken into blocks; each function has a magnitude of one throughout a given block, and is zero elsewhere. The βth function on a given block (for $\beta = 1, \cdots, m$) is nonzero only on that block and has a uniform phase progression at a rate determined by β. Since the above description allows one to think of each block as an antenna, each of the basis functions with support on a given block radiates most strongly (i.e., has a main beam) in a different direction. Let D be the size of this "antenna." For such beams nearly perpendicular to the surface, it is well known in antenna theory [15, Table 6.1] that the width of the main beam (i.e., the angle between the two relative minima enclosing it) is approximately $2\lambda/D$ and the first relative maximum past the main beam has a relative height (i.e., "sidelobe level") of about $10^{-1.32}$ (or -13.2 dB). The sidelobe level may be further reduced by tapering the current to decrease its discontinuity at the edges of the "antenna," although doing so increases the angular width of the beam. Note, however, that while nonoverlapping antennas with untapered currents describe **A**′, the **A** described by (27)-(31) corresponds to antennas twice as wide and overlapping. For this taper function, the width of the main beam is 4λ divided by the width of the antenna [15, Table 6.1], which is now $2D$, giving a width of $2\lambda/D$ as before, while the sidelobe level is now $10^{-3.2}$. Since **A** is applied to both sides of **Z**, the typical reduction in matrix element size is expected to be at least $10^{-6.4}$. Numerical results will now be presented that illustrate this reduction and confirm that it is quite an improvement over that for **A**′ as found in [7] and [4]. Although the taper function we have used as given by (27) and (28) gives impressive results, it was chosen in [7] because its analytical form allowed a quick estimation of matrix element size [7, Appendix]. As described in [14, § 9.1.1.1], the taper function of (27) was popular for antennas before computers were available, while more efficient functions are now in use. The general framework presented here should aid in the evaluation and use of such functions.

Before concluding this discussion of expected matrix element sizes, it is useful to note what would change if a double layer source had been used, as occurs in many second kind integral equations. A single layer (point) source radiates isotropically, while a double layer source in two dimensions radiates as sin (θ), where θ is the angle from tangential to the surface. The angular dependence resulting from an array of identical sources is the product of the angular dependences of the array and of the sources. Since the double layer source has a minimum along the tangent to the surface, the "resulting" part of **Z** and **T** would generally have smaller elements near the diagonal when using a double layer source. (There may be an additional part of **Z**, e.g., a multiple of the identity matrix.)

Numerical scattering calculations were performed for Γ consisting of a (two-dimensional) ogive with a blade at its end. The shape of $\partial\Gamma$ may be described by joining two circular arcs with a radius of curvature of 4.25 at their ends to give a chord length of 4 and a full width of 1. A blade of length 1 is then attached radially to one end. The total distance between the ends of Γ will be denoted L/λ, and several such sizes will be considered. The largest magnitude of an element of the resulting matrix **T** (using (29)-(31)) will be denoted γ, and τ will be used to denote a "relative drop tolerance." The matrix $\mathbf{T}'(\tau)$ may be defined by

$$(41)\qquad (\mathbf{T}'(\tau))_{\alpha,\beta} = (\mathbf{T})_{\alpha,\beta} \quad \text{if Abs}\,[(\mathbf{T}'(\tau))_{\alpha,\beta}] \geqq \tau\gamma,$$

$$(42)\qquad (\mathbf{T}'(\tau))_{\alpha,\beta} = 0 \quad \text{otherwise.}$$

The number of nonzero elements of **T** divided by that of $\mathbf{T}'(\tau)$ is plotted on Fig. 2 as the "sparseness factor." This sparseness factor is given for the ogive with blade of length $L/\lambda = 90$ using $N = 1701$ as a function of τ for various values of m. When $m = 1$, the corresponding matrix **A** is the identity, so in this case $\mathbf{T} = \mathbf{Z}$. This matrix does not have any elements smaller in magnitude than 0.02γ. Therefore, on Fig. 2, the $m = 1$ curve indicates a sparseness factor of 1 for $\tau < 0.02$. However, the situation changes dramatically as m increases. When $m = 63$ the sparseness factors for $\tau = 0.0005$ and

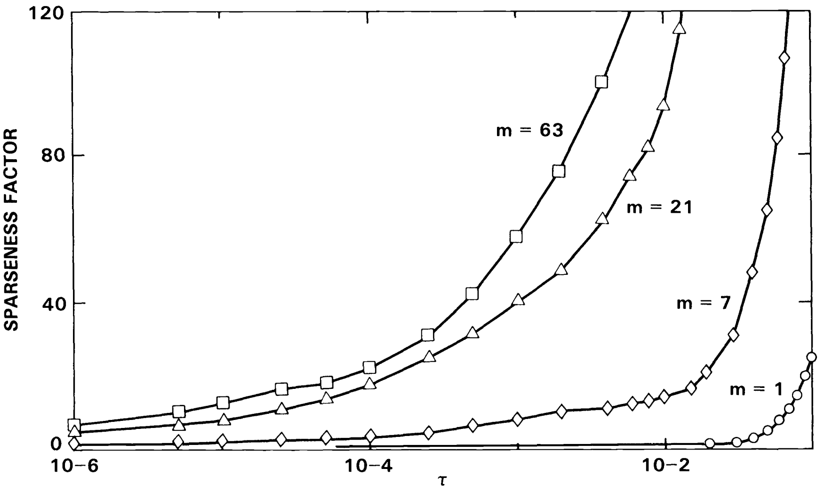


FIG. 2. *The sparseness factor for* $\mathbf{T}'(\tau)$ *as a function of relative drop tolerance* τ *for the ogive with blade described in the text and a chord length L of* 90 *wavelengths. Results are shown for* 1701 *unknowns and several block sizes m.*

0.0001 are 42 and 22, respectively. This transformation is much more effective than that involving $\mathbf{A}'$, as described in [7] and [4].

An earlier paper [7] argued that for a given (reasonably small) value of τ and for $N/(kL)$ fixed, the number of nonzero elements in $\mathbf{T}'(\tau)$ would be at most linear in N provided that N/m^2 is also constant. This scaling of the number of elements per block allows the main beam of each "antenna" to decrease in angular extent with increasing N proportionally to how the angle subtended by the "receiving" block (antenna) decreases. The discussions above about beamwidths, etc., assume that the receiving block is in the far field of the antenna (roughly speaking, the far field is where the approximation of (7) is valid). For an antenna of length D, the distance to the far field region is a constant times D^2/λ [15, § 6.9]. By varying N with both $N/(kL)$ and N/m^2 fixed, one can see that whether the far field condition is satisfied is independent of N but depends only on which parts of $\partial\Gamma$ are interacting (i.e., where the blocks are located). Rather than forming analytical estimates of the number of nonzero elements in $\mathbf{T}'(\tau)$, we now give some numerical results.

Figure 2 gave results for the ogive with blade using $m=63$ and $L/\lambda=90$ with $N=1701$. Keeping $N/(kL)$ constant at $1701/(2\pi 90)$, we now vary N from 21 to 15,309 by factors of three, and plot the resulting "sparseness factors" on Fig. 3. While in the previous figure we had $N/m^2=3/7$, in Fig. 3 the calculations for m satisfying this are plotted directly above the corresponding value of N; smaller and larger values of m (in factor of three increments) are plotted to the left and right, respectively. For example, the calculation for $N=1701$ and $m=63$ is given directly above $N=1701$, while the results for $m=21$ and $m=189$ are immediately to the left and right, respectively. For the values of τ of most interest (i.e., all $\tau<0.01$), this choice for m appears to be the best available (out of the discrete set plotted).

Having verified that $N/m^2=3/7$ is a good choice for m for this problem, the sparseness factor for only this value of m is now plotted as a function of N in Fig. 4.

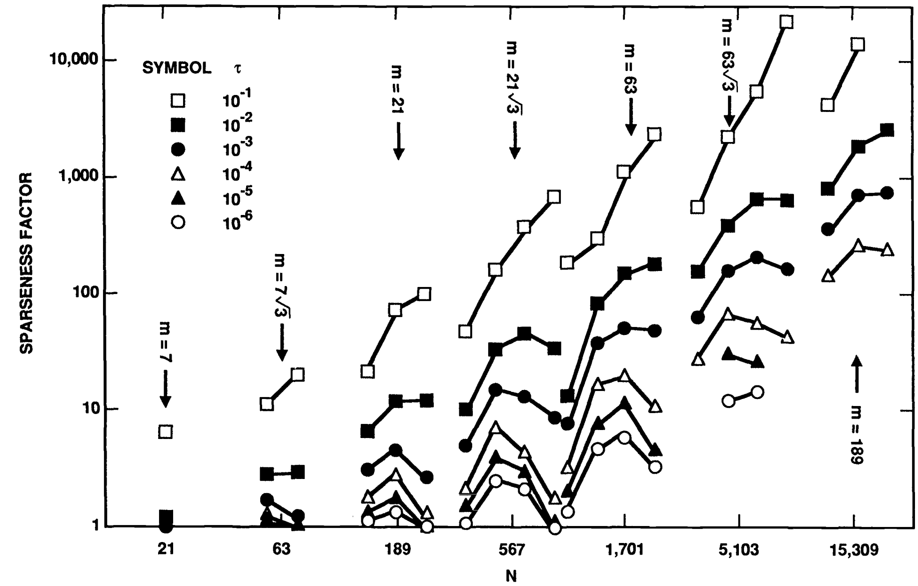


FIG. 3. *Sparseness factors for* $\mathbf{T}'(\tau)$ *as a function of m for m varying about* $m=[7N/3]^{0.5}$. *Calculations are shown for* $N=21$ *to* 15,309 *in factors of three and for* $\tau=10^{-6}$ *to* 10^{-1} *in factors of ten. The same ogive with blade is used as in Fig. 2, and its chord length L is given by* $L=N\lambda/18.9$.

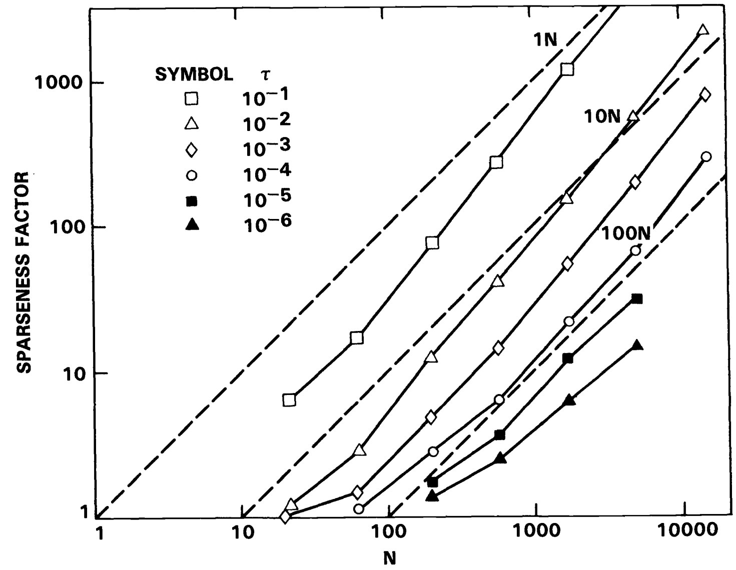


FIG. 4. *Sparseness factors for $m = [7N/3]^{0.5}$ from Fig. 3 are plotted as a function of N to illustrate how the storage required for a given value of τ scales with N. The dashed lines indicate a linear in N scaling of storage. The numerical results are steeper than these dashed curves, indicating the required storage is possibly even less than linear in N.*

When m from this equation would be some integer I times $\sqrt{3}$, the average of the sparseness factors for $m = I$ and $m = 3I$ is used. The dashed lines at 45 degrees on Fig. 4 indicate storage linear in N. The numerical results shown are asymptotically at least this steep, showing that for fixed τ, the storage required scales better than linear in N. Indeed, for $\tau = 10^{-4}$, the storage required is less than $100N$.

Since $\tau = 10^{-4}$ produces a sparse matrix, we now examine whether it produces accurate scattered fields. Fixing $N = 1701$, $m = 63$, and $L/\lambda = 90$, the cross-section for backward scattering of incident plane waves as a function of incidence angle is calculated and plotted in Fig. 5. Three curves are given, for $\tau = 0.0$, 0.0005, and 0.0001. The $\tau = 0.0001$ curve is nearly indistinguishable from the usual answer (i.e., for $\tau = 0.0$), while the $\tau = 0.0005$ curve is more than accurate enough for most practical applications. The condition numbers $\kappa[\mathbf{T}]$ for $m = 1$ and $m = 63$, respectively, were 2100 and 9400, while for $m = 63$, $\kappa[\mathbf{T}'(0.0001)]$ was 9700. Based on the condition number alone, we cannot say that $\tau = 0.0001$ will produce accurate scattering results. However, we have performed similar calculations for scatterers of several shapes and found accurate scattered fields in all cases examined. Note also that $\kappa[\mathbf{T}]/\kappa[\mathbf{Z}] < 5$, while the bound on this ratio is $\kappa^2[\mathbf{A}]$ which is over 100 (see Fig. 1).

4. Iterative results. In this section, first the results of standard iterative methods are given and then the possibility of preconditioning by incomplete LU (ILU) decomposition or one of its variants is discussed. The calculation of Fig. 5 will be used as an example, with a right-hand side given by a plane wave incident from $\theta = 0$. Using $\tau = 0.0001$ the iteration uses $\mathbf{T}'(0.0001)$. It was observed that $\mathbf{Z}$ is better conditioned than $\mathbf{T}$ for this problem, so it is likely that $\mathbf{A}^{-1}\mathbf{T}'(\tau)\mathbf{A}^{-*}$ would be a better matrix to use than $\mathbf{T}'(\tau)$. The factorization of $\mathbf{A}$ given by (14) into the product of matrices involving Fourier transforms and circulant matrices immediately shows how $\mathbf{A}^{-1}$ and $\mathbf{A}^{-*}$ may be applied in a sparse and efficient way. Although this approach is viable we will instead consider ILU preconditioning below.

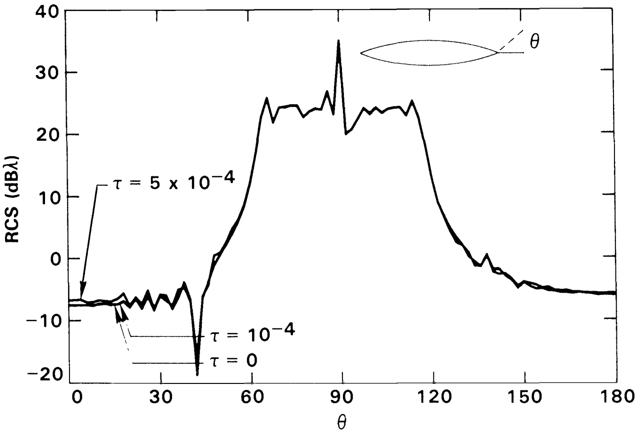


FIG. 5. *The backward scattering cross-section for the ogive with blade, as a function of the angle of incidence θ of an incident plane wave. The curves plotted for a standard calculation ($\tau = 0$) and for the approximation $\tau = 10^{-4}$ are indistinguishable, while the $\tau = 5 \times 10^{-4}$ curve differs only slightly. The two approximations that both used $m = 63$ gave sparseness factors of* 22 *and* 42, *respectively.*

The integral equation (37) is complex symmetric, and its eigenvalues all have a real part greater than or equal to zero [11]. When $\partial\Gamma$ encloses one or more regions, the interior Helmholtz problem(s) is known to have solutions at discrete values of k. Away from these values, the real part of the eigenvalues of the operator in (37) is positive. Since **Z** is fairly well conditioned, we might for the moment assume that it also has this property. The transformation to **T** given by (4) and the fact that **A** is nonsingular immediately imply that the eigenvalues of **T** would also lie in the right half plane. It is not completely clear if passing from **T** to **T**′(0.0001) would preserve this property, although the modest change in condition number between these two matrices is certainly encouraging. While this argument does not prove that the eigenvalues of **T**′(0.0001) lie in the right half plane, it certainly suggests trying an iterative method such as Orthomin.

The integral equation (37) and our symmetric discretization of it resulted in a complex symmetric **Z**. While that symmetry may be used to reduce the required storage for **Z**, the transformation (4) resulted in a nonsymmetric **T**. If the Hermitian conjugate indicated by * in (4) is replaced by simply a transpose, then the resulting matrix would still be complex symmetric. This complex symmetric form was used in [4], [7] and [8]. The two matrices (resulting from Hermitian conjugate and transpose) are simply related to each other by a permutation of columns, and they have the same condition numbers. However, as Fig. 6 shows, iteration using Orthomin [20] on these two matrices gives very different results. On the other hand, when using conjugate gradient on the normal equations the iteration matrix is **T*T** in one case and **PT*TP** in the other, where **P** gives the permutation of the columns of **T** mentioned above. These iteration matrices have identical eigenvalues, their eigenvectors are related by **P**, and each step of the iteration corresponds. The results of using (4) and conjugate gradient on the normal equations are shown on Fig. 6.

These iterative results show that an efficient solution has been demonstrated compared to other methods when one must solve for only one or a few right-hand sides. The sparseness of **T**′(0.0001) allows each iteration to be nearly 22 times quicker than it would be using **Z**. For example, on our Vax 6000 the matrix was generated in six minutes, and 300 iterations of conjugate gradient on the normal equations took an

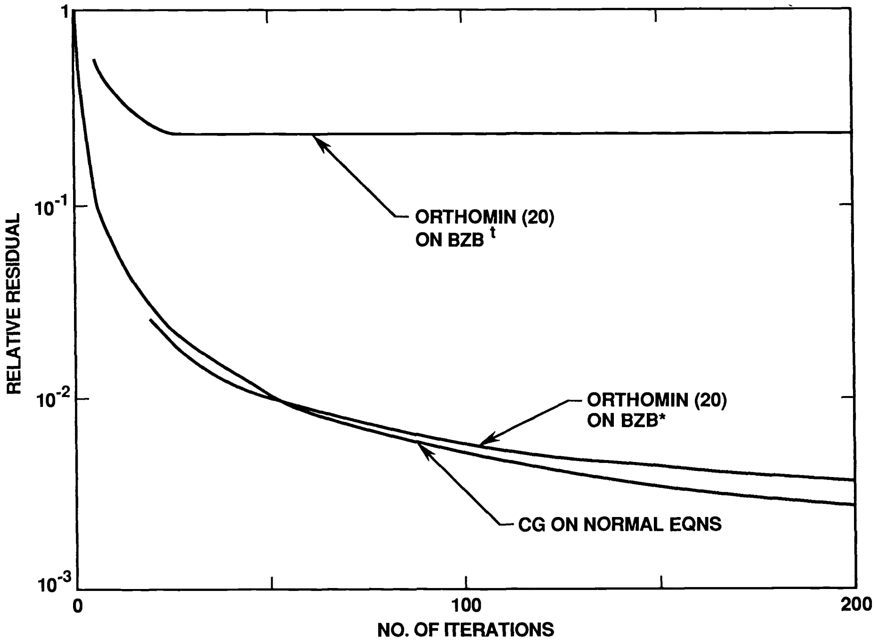


FIG. 6. *Reduction in residual as a function of the number of iterations for the ogive with blade of length* $L=90\lambda$ *using* $N=1701$. *Using the matrix* T′(0.0001) *the conjugate gradient applied to the normal equations and Orthomin* (20) *are used. Results also are shown using the corresponding matrix that results from replacing the Hermitian conjugate* (*indicated by* *) *in* (4) *and* (5) *by a transpose, and then using Orthomin* (20).

additional ten minutes. If enough memory had been available to store **Z**, then its full LU decomposition would have taken several hours. In contrast, the ILU decomposition of **T**′(0.0001) would take only two minutes.

ILU will be an effective preconditioner only if the large elements of **T** (or of a permutation of its rows and columns) tend to be near the diagonal. A discussion of the structure of the large elements of **T** is easier with an example to look at. A clear plot may be generated using $N=100$ and the scatterer given by a well-known airfoil shape, an NACA 0012 airfoil. A slight modification (to give a closed body) results in the shape defined by

$$(43)\qquad y=\pm 0.6\,[0.2969x^{1/2}-0.1281x-0.3516x^2+0.2843x^3-0.1015x^4].$$

T is calculated using $m=25$, $n=4$, and the **A** defined by (27)-(31), where this airfoil shape is scaled to have a chord length (tip to tail, in a straight line) of 5λ. Figure 7 gives a three-dimensional plot of the magnitudes of the elements of **T**. The block structure that appears is not very different from that found for this geometry using **A**′ to generate $\mathbf{A}'\mathbf{Z}\mathbf{A}'^{tr}$ as shown in [7, Fig. 5]. Unfortunately, in both cases the large elements in **T** occur successively further and further away from the diagonal. Fortunately, the spacing is somewhat regular, as we shall see below.

Column c of **T** may alternatively be identified as element a of block b, where $a=1,\cdots,m$, and $b=1,\cdots,n$. A permutation of the columns of **T** may be defined by relating the resulting column index c' to the old index c by

$$(44)\qquad c=a+(b-1)m,\qquad c'=b+(a-1)n.$$

In Fig. 7 large elements occur near the center of each block (i.e., for $a\approx(m+1)/2$ for $b=1,\cdots,n$) so we might expect that applying this permutation to both the rows and

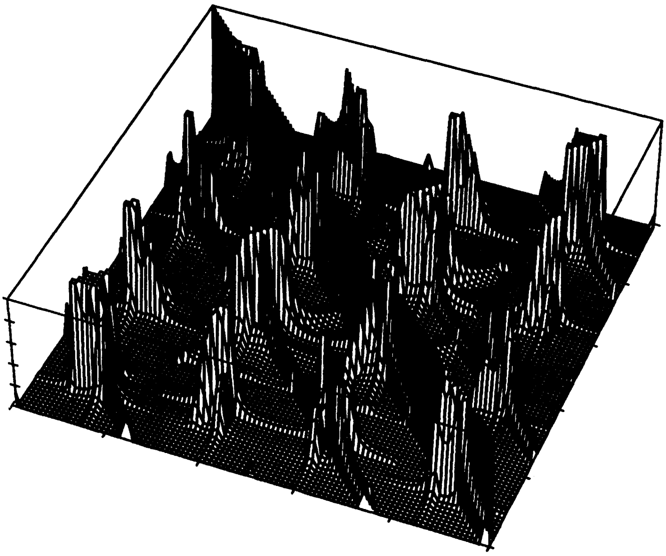


FIG. 7. *Three-dimensional plot of the magnitudes of the matrix elements for* **T** *using* (27)–(31) *for the modified* NACA 0012 *airfoil of* (43) *with a chord length of* 5λ. 100 *unknowns are used with* $m=25$ *and* $n=4$.

columns of **T** would group these large elements together. Figure 8 contains a three-dimensional plot of the permuted **T** and verifies that this is the case for all of these peaks. In addition, the large elements that had been on the diagonal are still on it since the permutation was done symmetrically. Although we do not yet have numerical results using ILU preconditioning, Fig. 8 strongly suggests that either it or a closely related method is likely to be highly effective.

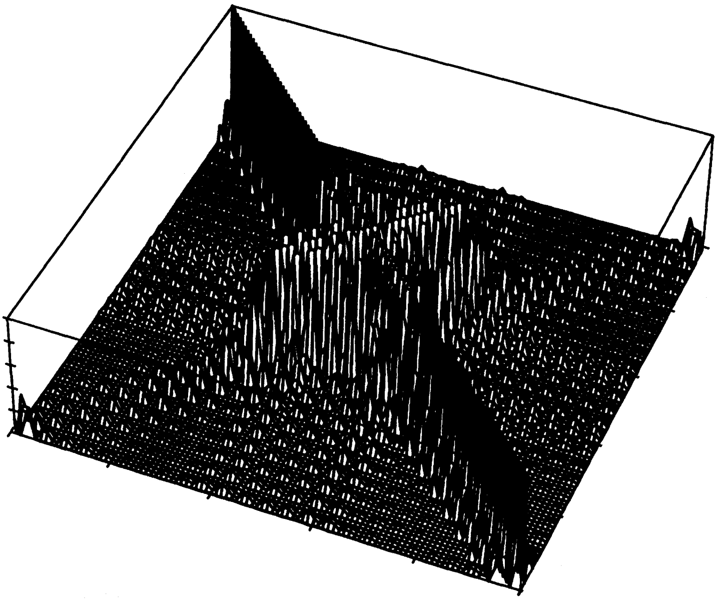


FIG. 8. *Three-dimensional plot of the magnitudes of the matrix elements of the permutation of the* **T** *of Fig.* 7 *as described by* (44) *and the associated text.*

The reason for the block structure in Fig. 7 and the size of the bandwidth in Fig. 8 are easy to find. The matrix $\mathbf{A}'$ contains $m \times m$ blocks of Fourier transforms and the mode with highest spatial frequency oscillates about $m/2$ times in m points (the spatial frequency in $\mathbf{A}$ is the same as that in $\mathbf{A}'$). If the discretization in $\mathbf{Z}$ used p points per wavelength of arc length, then m points correspond to a distance of m/p wavelengths. Thus, the rate of oscillation of this mode is $p/2$ times per wavelength, resulting in tangential propagation described by an effective wavenumber

$$k_t \approx (p/2)k. \tag{45}$$

The wavenumber describing the propagation of this mode perpendicular to the surface k_p satisfies

$$k_p = [k^2 - k_t^2]^{0.5} \approx k[1-(p/2)^2]^{0.5}. \tag{46}$$

Propagating modes require real k_p, so in these terms the sampling criterion of two points per wavelength (i.e., $p=2$) includes exactly the propagating modes. Since Figs. 7 and 8 used $p=10$, we would expect that only 2/10, or 20 percent, of the modes are propagating. That is, for $p=10$ and for "a" within a peak an approximation is

$$(m+1)/2 - 0.1m \leqq a \leqq (m+1)/2 + 0.1m. \tag{47}$$

The finite width of the "beams" causes some spillover from this region, which may be observed in Fig. 8.

Equation (47) has a remarkable consequence on the structure of the permutation of $\mathbf{T}$ and on our ability to solve it quickly. If N unknowns are used with p points per wavelength, then the permutation of $\mathbf{T}'(\tau)$ for some appropriate τ contains nonzero elements only in a central block of size $2N/p \times 2N/p$, or on or very near the diagonal. If this central block is treated as a full matrix, then the time for a direct solution would be nearly $(p/2)^3$ times faster than for the usual LU decomposition for large N. However, as N increases, the total number of elements in this block grows as N^2, while the number of nonzero elements is linear in N. Thus, for very large N, this block itself becomes sparse, and iterative methods preconditioned by ILU decomposition or related approaches are likely to be even more effective.

5. Conclusions. A class of transformations of the usual matrix formulations of integral equations involving oscillatory kernels was presented with the goal of generating a sparse $N \times N$ matrix from a full $N \times N$ matrix. In § 2 the basic properties of these transformations, such as their condition numbers, were calculated and the motivation for the transformations was developed. In § 3 the explanation of this transformation as one that generated a matrix whose elements each represented extended directional sources in a region interacting with similarly directional receiving elements was further developed. Numerical calculations based on an integral formulation of the Helmholtz equation in two dimensions showed that for a given smallness tolerance only a linear in N number of matrix elements need be kept after this transformation. This allows one to store profoundly larger physical problems with correspondingly larger numbers of unknowns in existing computers than has previously been possible. Sample calculations showed that the approximation of setting small elements of the transformed matrix $\mathbf{T}$ to zero allows good accuracy to be maintained. A permutation of the elements of $\mathbf{T}$ was introduced, which put it in a form allowing solution by banded matrix techniques orders of magnitude faster than by standard techniques. Some possible further increases in efficiency for very large N by the use of preconditioned iteration were also suggested.

The method described here should work for other integral equations where the fractional change in the magnitude of the kernel is small over one wavelength and the kernel's behavior is generally smooth. Such kernels may be described as oscillatory. The numerical results presented here were for a one-dimensional surface in two dimensions. Similarly, if the sources are spread over a two- or three-dimensional surface, the additionl integrals might serve to give further reductions in the magnitudes of the resulting matrix elements. Thus, there is reason to believe that this method will be useful for a variety of integral equations involving oscillatory kernels both in two and three dimensions.

The method described here reduces the storage requirements for a computer program based on an integral equation with oscillatory kernel with N unknowns from N^2 to Order $[N]$. The number of operations used to generate the matrix from the usual matrix is Order $[N^2 \log N]$. Directly generating this matrix can significantly reduce this operation count. The number of operations per iteration is similarly reduced from N^2 to Order $[N]$, allowing the most efficient method found to date storagewise, and also in terms of execution time, at least when a limited number of incident fields are considered.

REFERENCES

[1] G. Beylkin, R. Coifman, and V. Rokhlin, *Fast wavelet transforms and numerical algorithms* I, Yale Res. Report YALEU/DCS/RR-696, Department of Computer Science, Yale University, New Haven, CT, 1989.

[2] A. Brandt and A. A. Lubrecht, *Multilevel matrix multiplication and fast solution of integral equations*, J. Comp. Phys., 90 (1990), pp. 348-370.

[3] F. X. Canning, *Singular value decomposition of integral equations of* EM *and applications to the cavity resonance problem*, IEEE Trans. Antennas and Propagation, 37 (1989), pp. 1156-1163.

[4] ———, *Reducing moment method storage from Order N^2 to Order N*, Electron. Lett., 25 (1989), pp. 1274-1275. Reprinted in Moment Methods in Antennas and Scattering, R. C. Hansen, ed., Artech House, Norwood, MA, 1990.

[5] ———, *On the application of some radiation boundary conditions*, IEEE Trans. Antennas and Propagation, 38 (1990), pp. 740-745.

[6] ———, *Direct solution of the* EFIE *with half the computation*, IEEE Trans. Antennas and Propagation, 39 (1991), pp. 118-119.

[7] ———, *Transformations that produce a sparse moment method matrix*, J. Electromagnetic Waves Appl., 4 (1990), pp. 893-913.

[8] ———, *Sparse matrix approximation to an integral equation of scattering*, Comm. Appl. Numer. Methods, 6 (1990), pp. 543-548.

[9] D. Colton and R. Kress, *Integral Equation Methods in Scattering Theory*, John Wiley, New York, 1983.

[10] R. F. Harrington, *Field Computation by Moment Methods*, Reprint Edition, R. E. Krieger, Malabar, FL, 1983.

[11] R. F. Harrington and J. R. Mautz, *Theory of characteristic modes for conducting bodies*, IEEE Trans. Antennas and Propagation, 19 (1971), pp. 622-628.

[12] V. Rokhlin, *Rapid solution of integral equations of scattering theory in two dimensions*, J. Comp. Phys., 86 (1990), pp. 414-439.

[13] ———, *Rapid solution of integral equations of scattering theory in two dimensions*, Yale Res. Report YALEU/DCS/RR-440, Department of Computer Science, Yale University, New Haven, CT, 1985.

[14] A. W. Rudge, K. Milne, A. D. Olver, and P. Knight, *The Handbook of Antenna Design*, Peter Peregrinus, London, UK, 1983.

[15] S. Silver, *Microwave Antenna Theory and Design*, Radiation Laboratory Series, No. 12, McGraw-Hill, New York, 1949, p. 187.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 1, pp. 88-100, January 1992

005

A HIGHLY PARALLEL MULTIGRID-LIKE METHOD FOR THE SOLUTION OF THE EULER EQUATIONS*

RAY S. TUMINARO†

Abstract. A highly parallel multigrid-like method for the solution of the two-dimensional steady Euler equations is considered. The new method, introduced in [T. Chan and R. Tuminaro, in *Proc. Third Copper Mountain Conference on Multigrid Methods*, 1987, pp. 101-115] as a "filtering" multigrid, is similar to a standard multigrid scheme in that convergence on the finest grid is accelerated by iterations on coarser grids. In the filtering method, however, additional fine grid subproblems are processed concurrently with coarse grid computations to further accelerate convergence. These additional problems are obtained by splitting the residual into a smooth and an oscillatory component. The smooth component is then used to form a coarse grid problem (similar to standard multigrid) while the oscillatory component is used for a fine grid subproblem. The primary advantage in the filtering approach is that fewer iterations are required and that most of the additional work per iteration can be performed in parallel with the standard coarse grid computations.

In this paper, the filtering algorithm is generalized to a version suitable for nonlinear problems. It is emphasized that this generalization is conceptually straightforward and relatively easy to implement. In particular, no explicit linearization (e.g., formation of Jacobians) needs to be performed (similar to the FAS multigrid approach). The nonlinear version is illustrated by applying it to the Euler equations and presenting numerical results. Finally, a performance evaluation is made based on execution time models and convergence information obtained from numerical experiments.

Key words. multigrid, parallel, Euler equations, filtering

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

1. Introduction. Multigrid methods are among the fastest algorithms for a wide variety of problems and are now used in many scientific disciplines. Structurally, the algorithm iterates on a hierarchy of consecutively coarser and coarser grids until convergence is reached. While critical to its rapid convergence, the coarse grid computations are more difficult to parallelize efficiently due to the presence of fewer grid points (and hence less parallelizable work). We therefore consider a highly parallel multigrid-like method (see [6], [7], and [8] for other types of highly parallel multigrid-like methods). This new algorithm, "filtering" (proposed in [4]), uses additional fine grid subproblems to accelerate the convergence of the overall process. More specifically, these fine grid problems are created by splitting the residual into a smooth and an oscillatory component. The smooth component is used to form a coarse grid problem (similar to standard multigrid) while the oscillatory component is used for the fine grid subproblem. The primary benefit to this approach is that while more work per iteration is necessary, fewer iterations are required and more of the work within an iteration is parallelizable. In fact, if the additional work can be performed concurrently with coarse grid computations, the CPU time per iteration need not rise significantly.

In this paper we generalize the filtering algorithm into a version suitable for nonlinear problems. This new algorithm is conceptually straightforward and relatively easy to implement. In particular, no explicit linearization (e.g., formation of Jacobians)

* Received by the editors April 5, 1990; accepted for publication (in revised form) July 23, 1990. The majority of this work was performed while the author was at the Research Institute for Advanced Computer Science and was supported under Cooperative Agreement NCC 2-387 between NASA and the Universities Space Research Association. This work was also supported by the Applied Mathematical Science Program, U.S. Department of Energy, Office of Energy Research, and was performed in part at Sandia National Laboratories, operated for the U.S. Department of Energy, under contract DE-AC04-76DP00789.

† Applied and Numerical Mathematics, Sandia National Laboratories, P.O. Box 5800, Albuquerque, New Mexico 87185.

needs to be performed (similar to the FAS multigrid approach). We apply the nonlinear version to the solution of the Euler equations (see [4] and [11] for convergence analysis of the filtering algorithm for linear model problems). Specifically, we consider the filtering approach applied to the FLO52 algorithm. FLO52, written by Jameson [9], is a well-known multigrid code for the solution of the Euler equations describing transonic flow past an airfoil. The corresponding FLO52-filtering algorithm is similar to the ORIGINAL FLO52 with the exception of the additional subproblems. We begin our description of the algorithm with discussions of both the standard and filtering multigrid methods applied to linear problems in §§ 2 and 3. The generalization of both the standard multigrid and the filtering approach to nonlinear problems is then discussed in §§ 4 and 5. We conclude by comparing the convergence of the filtering and standard FLO52 algorithms on a fluid calculation. Based on these numerical experiments, as well as a mathematical execution time model, we make some predictions on the performance of the FLO52-filtering algorithm on massively parallel computers.

2. Standard multigrid algorithm. We begin our discussion with a brief sketch of the standard multigrid algorithm applied to linear elliptic partial differential equations (PDEs). More complete introductory material on multigrid methods can be found in [3] and [10].

Assume that a given elliptic partial differential equation is approximated by a discrete set of equations (finite differences or finite elements):

$$A_1 u = b, \tag{1}$$

where A_1 is a matrix, b is a vector, and u is a vector of unknowns for which we seek the solution. One iteration of a simple multigrid ("V" cycle) method consists of the following steps:

- relaxation iterations (e.g., Jacobi or SOR methods);
- formation of a correction equation for the error in the current approximation;
- projection of correction equation onto a coarser grid;
- "solution" of a coarse grid system;
- interpolation and addition of correction to previous approximation.

A key feature of this procedure is that the solution to the coarse grid equations can be approximated using the multigrid idea recursively. Thus the general algorithm consists of processing on a hierarchy of coarser grids (each processed in turn). We summarize this multigrid algorithm with a pseudocode fragment in Fig. 1.

For the most part, analysis of the two-level multigrid method reveals the general behavior of the multiple grid version. If we denote the projection operator by R_1, the

```
Proc Multigrid($A_i$, $b$, $u$, level)
{
    if (level = CoarsestLevel) then $u = A_i^{-1}b$
    else
        PreRelax($A_i$, $b$, $u$, level)
        ComputeResidual($b$, $u$, level, residual)
        ProjectResidual(level, residual, coarse_residual)
        Multigrid($A_{i+1}$, coarse_residual, v, level + 1)
        Interpolate(level, v, correction)
        $u = u +$ correction
    endif
}
```

FIG. 1. *Simple "V" cycle multigrid algorithm.*

interpolation operator by P_1, the relaxation iteration operator by G, and the coarse grid difference operator by A_2, we can express the two-grid iteration operator

$$e^{(k+1)} = Te^{(k)} \tag{2}$$

by

$$T = (A_1^{-1} - P_1 A_2^{-1} R_1) A_1 G, \tag{3}$$

where $e^{(k)}$ denotes the error after k iterations. In the next section, we will contrast this two-grid operator with that of the two-level filtering method.

3. Filtering algorithm. Conceptually, the filtering algorithm is similar to the standard multigrid method, the primary difference being that two correction equations are formed after the relaxation iterations. Specifically, let u_1 denote an approximate solution to the linear equation

$$A_1 u = b. \tag{4}$$

Within a standard multigrid method, a correction equation is usually formed by computing the residual $(r = b - A_1 u_1)$ and using this as the right-hand side to a new equation. Within the filtering algorithm, however, two subproblems are created by splitting the residual into the two components:

$$r_1 = Zr \quad \text{and} \quad r_2 = r - r_1, \tag{5}$$

which are then used as right-hand sides in

$$A_1 c^{(1)} = r_1 \quad \text{and} \quad A_1 c^{(2)} = r_2. \tag{6}$$

To approximate the solution of the first subproblem, the equation is projected onto a coarser grid, as within a standard multigrid method. The error associated with this approximation (solving the coarse problem exactly) is

$$e_1 = (A_1^{-1} - P_1 A_2^{-1} R_1) r_1, \tag{7}$$

where A_2 is the coarse grid operator. To approximate the solution of the second problem, q relaxation sweeps are performed on the fine grid. The error for this second approximation is given by

$$e_2 = S^q A_1^{-1} r_2, \tag{8}$$

where S is the iteration operator of the concurrent relaxation method, and we have assumed that the initial guess is zero. Thus the two-level algorithm generates one coarse grid correction (as in the standard multigrid method) and one additional fine grid problem to be processed by relaxation iterations. Once again, a multiple grid version of the method can be defined by recursively using the filtering procedure to "solve" the coarse grid subproblems. This multilevel version is summarized in Fig. 2, where the "FilterMultigrid ( )" and "ConcurrentRelax ( )" steps can occur in parallel.

```
Proc FilterMultigrid($A_i$, $b$, $u$, level)
  if (level = CoarsestLevel) then $u = A_i^{-1} b$;
  else
    PreRelax($b$, $u$, level);
    ComputeResidual($A_i$, $u$, $b$, level, res);
    SplitResidual(res, $r_1$, $r_2$);
    ProjRes($r_1$, $\hat{r}_1$);
    FilterMultigrid($A_{i+1}$, $\hat{r}_1$, $\hat{u}_1$, level + 1);
    ConcurrentRelax($A_i$, $r_2$, $u_2$, level + 1);
    Interpolate_and_add($\hat{u}_1$, $u_2$, level, $u$);
  endif
```

FIG. 2. *V cycle version of the filtering algorithm.*

Finally, an iteration operator for the two-level version of this algorithm is obtained by combining (7) and (8):

$$e^{(k+1)} = [(A_1^{-1} - P_1 A_2^{-1} R_1)Z + S^q A_1^{-1}(I - Z)]A_1 G e^{(k)}, \tag{9}$$

where we have included the possibility of performing one prerelaxation sweep with iteration operator G. That is,

$$r = A_1 G e^{(k)}. \tag{10}$$

Note that when the operator Z is the identity matrix, (9) is identical to (3).

Intuitively, the S^q term in (9) damps the high frequencies while the coarse grid correction damps the low frequencies. A critical element affecting the convergence behavior of the filtering algorithm is the properties of the splitting operator Z, used for computing r_1 and r_2. For the most part, this operator should decompose the residual so that high frequency errors remain on the fine grid while low frequency errors are projected onto the coarse grid subproblem. This can be accomplished by choosing an operator that filters out high frequencies in the residual. Many choices are possible. In this paper we consider only the operator

$$Z = P_1 R_1. \tag{11}$$

This corresponds to first projecting the residual onto the coarse grid and then interpolating back to the fine grid. See [4] for an alternative filter.

A detailed convergence analysis (via Fourier transform on the iteration operator) of a two-level filtering algorithm applied to the Poisson equation can be found in [4] and [11]. Using this analysis, it is possible to determine convergence rates for the filtering method. Table 1 lists the spectral radius of the two-grid iteration operator as a function of q when the algorithm is applied to the Poisson equation:

$$u_{xx} + u_{yy} = f. \tag{12}$$

The algorithm depicted uses full-weighted restriction, given by the stencil

$$\frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}, \tag{13}$$

bilinear interpolation; one Jacobi prerelaxation sweep; q concurrent relaxation iterations of damped Jacobi with damping parameter equal to $\frac{4}{5}$; and the exact solution of the coarse grid equations. Finally, discretization is obtained via central differences on both the fine grid (32 × 32) and the coarse grid (16 × 16). We note that by comparison, the corresponding standard multigrid method using one damped Jacobi relaxation sweep (with optimal damping parameter) has a spectral radius of 0.570. Thus even with only one concurrent relaxation sweep, the convergence rate is accelerated.

TABLE 1

Convergence rates of the filtering algorithm corresponding to a 32 × 32 *grid.*

q	$\rho(T_f)$
1	0.498
2	0.268
3	0.230
4	0.205

4. FAS-multigrid method. To apply the multigrid method to a nonlinear problem, the simple scheme described in § 2 must be modified to implement the FAS algorithm. For the most part these modifications ensure that the correction equations correspond to physically meaningful subproblems. To describe the algorithm, we consider the nonlinear systems

$$A_k(u) = f_k, \tag{14}$$

arising from discretization of a partial differential equation on grid $\mathcal{G}_k$ (where $k = 0$ corresponds to the finest grid). We write one iteration of the relaxation scheme as

$$u_k \leftarrow Relax(u_k, f_k). \tag{15}$$

Once again, let R_k denote projection from grid k to grid $k+1$. Similarly, let P_k denote interpolation from grid k to grid $k-1$. Then the coarse grid subproblem is defined as follows. Let the initial guess on the coarse grid be given by

$$u_{k+1} \leftarrow R_k u_k. \tag{16}$$

On the finest grid, f_0 corresponds to the discretization of the continuous right-hand side. The right-hand sides on the coarser meshes are recursively defined by

$$f_{k+1} \leftarrow A_{k+1}(u_{k+1}) - R_k r_k, \tag{17}$$

where

$$r_k \leftarrow f_k - A_k(u_k). \tag{18}$$

Finally, after the solution on the coarse grid is improved (either by relaxation or recursively applying the multigrid procedure), the solution on the fine grid is corrected by

$$u_k \leftarrow u_k + P_{k+1}(u_{k+1} - R_k u_k). \tag{19}$$

Below we summarize a two-grid version of the algorithm using one relaxation sweep on each grid level:

$$\begin{aligned}
&u_0 \leftarrow Relax(u_0, f_0)\\
&u_1 \leftarrow R_0 u_0\\
&r_0 \leftarrow f_1 - A_1(u_1)\\
&f_1 \leftarrow A_1(u_1) - R_0 r_0\\
&u_1 \leftarrow Relax(u_1, f_1)\\
&u_0 \leftarrow u_0 + P_1(u_1 - R_0 u_0).
\end{aligned}$$

For more than two grids, the algorithm is recursively defined by replacing

$$u_1 \leftarrow Relax(u_1, f_1)$$

with

$$u_1 = \text{result of FAS algorithm starting on } \mathcal{G}_1.$$

Note that when this method is applied to a linear problem, it is mathematically identical to the method described in § 2. See [2] for more on the FAS procedure.

5. FAS-filtering algorithm. Similar to the FAS algorithm, the filtering algorithm must be modified so that all the subproblems are physically meaningful. To describe the method, we consider the discrete nonlinear systems defined by (14):

$$A_k(u) = f_k. \tag{20}$$

As with the FAS scheme, a coarse grid subproblem is created after relaxation on $\mathcal{G}_k$ by first computing the residual on that level. However, in the filtering version this residual is further split into two components and then the smooth component is used in forming the coarse grid subproblem. That is,

$$r_k = f_k - A_k(u_k), \quad \hat{r}_k = Zr_k, \quad \text{and} \quad \tilde{r}_k = r_k - \hat{r}_k. \tag{21}$$

The formation of the coarse grid subproblem proceeds in a simlar fashion to that of a standard FAS algorithm with the exception that $\hat{r}_k$ is used instead of r_k. That is, the initial guess on $\mathcal{G}_{k+1}$ is

$$u_{k+1} = R_k u_k, \tag{22}$$

and the right-hand side of the coarse grid equations are recursively defined by

$$f_{k+1} = A_{k+1}(u_{k+1}) - R_k \hat{r}_k. \tag{23}$$

In addition to the coarse grid subproblem, a fine grid problem is created

$$A_k(\tilde{u}_k) = \tilde{f}_k, \tag{24}$$

which can be processed concurrently with the coarse grid problem. The right-hand side of this fine grid subproblem is defined by

$$\tilde{f}_k = A_k(u_k) + \tilde{r}_k, \tag{25}$$

and the initial guess to this system is taken to be u_k. Similar to the filtering algorithm described in § 3, a relaxation scheme is used to improve the approximation to (24). Finally, after the approximations to the coarse grid and fine grid subproblems have been improved, the solution of the original problem is corrected by

$$u_k \leftarrow u_k + P_{k+1}(u_{k+1} - R_k u_k) + (\tilde{u}_k - u_k). \tag{26}$$

Below we summarize the two-grid version of the algorithm using one prerelaxation sweep, one concurrent relaxation sweep on the fine grid, and one relaxation sweep on the coarse grid:

```
              $u_0 \leftarrow Relax(u_0, f_0)$
              $r_0 \leftarrow f_0 - A_0(u_0)$
$\hat{r}_0 \leftarrow Zr_0$                          $\tilde{u}_0 \leftarrow u_0$
$u_1 \leftarrow R_0 u_0$                          $\tilde{r}_0 \leftarrow r_0 - \hat{r}_0$
$f_1 \leftarrow A_1(u_1) - R_0 \hat{r}_0$          $\tilde{f}_0 \leftarrow A_0(\tilde{u}_0) + \tilde{r}_0$
$u_1 \leftarrow Relax(u_1, f_1)$              $\tilde{u}_0 \leftarrow Relax(\tilde{u}_0, \tilde{f}_0)$
```

In the above pseudocode fragment, independent parts of the two subproblems appear in separate columns. A multilevel version of the above algorithm can be obtained by replacing

$$u_1 \leftarrow Relax(u_1, f_1)$$

with

$$u_1 = \text{result of FAS-filtering algorithm starting on } \mathcal{G}_1.$$

We can easily verify that when the operators A_i are linear, the FAS-filtering method is mathematically identical to that described in § 3.

It is important to realize the relative simplicity of modifying an FAS method to implement the FAS-filtering algorithm. One advantage is that no linearization is needed. Only the splitting operator and the concurrent relaxation operator must now be developed. For the concurrent relaxation we can use the same (or a similar) routine as for the prerelaxation. Additionally, no additional work is required to implement the splitting operator if

$$Z = P_k R_k$$

is used as these operators are already defined for the interpolation and restriction. In fact the only aspects of the filtering algorithm that requires nontrivial modification are the routines necessary for the allocation of subproblems to different processors.

6. FLO52 and the Euler equations. We consider both the FAS and the filtering schemes applied to the Euler equations. We begin by describing the Euler equations and the FLO52 code.

The FLO52 algorithm written by Jameson solves the two-dimensional steady Euler equations describing flow around an airfoil. It is widely used in research and industrial applications throughout the world. It produces good results for problems in its domain of application (steady inviscid flow around a two-dimensional body), and converges rapidly.

We briefly describe the Euler equations and the FLO52 scheme (see [9] for more on FLO52). We begin with the unsteady time-dependent two-dimensional equations written in conservation integral form as

$$\frac{d}{dt}\iint w + \oint \mathbf{n}\cdot\mathbf{F} = 0, \tag{27}$$

where **n** is the outward-pointing normal on the boundary of the region. The variable w is the vector of unknowns

$$w = (\rho, \rho u, \rho v, \rho E)^T, \tag{28}$$

where ρ is density, u and v are velocity components directed along the x and y-axes, respectively, and E is total energy per unit mass. The function **F** is given by

$$\mathbf{F}(w) \leftarrow (E(w), F(w)), \tag{29}$$

where

$$E(w) = (\rho u, \rho u^2 + p, \rho uv, \rho uH)^T,$$
$$F(w) = (\rho v, \rho uv, \rho v^2 + p, \rho vH)^T.$$

Here p is pressure and H is enthalpy. These are defined by

$$p = (\gamma - 1)\rho[E - (u^2 + v^2)/2],$$
$$H = E + p/\rho,$$

where γ is the ratio of specific heats. The integral relation given by (27) expresses conservation of mass, momentum, and energy which is to hold for any region in the flow domain.

To produce a numerical method based on (27), the flow domain is divided into quadrilaterals. On each quadrilateral of the domain, the double integral in (27) is approximated by the centroid rule and the line integral is approximated by the midpoint rule. For numerical stability, a dissipation term that is a blend of second- and fourth-order differences is added.

A simple iterative method (such as the Jacobi algorithm) for the steady-state problem can be viewed as a time-marching method for the time-dependent equations (27). After spatial discretization, the equations form a system of ordinary differential equations

$$\frac{dw}{dt}+A_1(w)=0, \tag{30}$$

where $A_1(\,)$ denotes the nonlinear finite-difference operator corresponding to differencing of spatial derivatives. Thus for the steady-state solution, we are interested in solving

$$A_1(w)=0. \tag{31}$$

The application of both the FAS-multigrid and -filtering algorithms to this problem is relatively straightforward. We conclude this section with a description of the relaxation method used within the algorithm. Specifically, it is a general multistage Runge-Kutta-like method. Such a procedure can be written in the form

$$\begin{aligned} &w^{(0)}=w_n,\\ &w^{(k)}=w^{(0)}-\Delta t\sum_{j=0}^{k-1}\alpha_{kj}A_i(w^{(j)}), \quad \text{for } k=1 \quad \text{to} \quad m,\\ &w_{n+1}=w^{(m)}, \end{aligned} \tag{32}$$

where Δt, the time step, and α_{kj} are chosen so that the multigrid procedure converges rapidly (and not necessarily to maintain time accuracy). In our experiments we always use $m=4$ for both the concurrent relaxation and the prerelaxation. The only nonzero parameters for the α_{kj}'s are given by $\alpha_{10}=\frac{1}{4}$, $\alpha_{21}=\frac{1}{3}$, $\alpha_{32}=\frac{1}{2}$, and $\alpha_{43}=1$.[1] Finally, we mention that a few other acceleration techniques are employed. These include local time-stepping and a residual smoothing technique (see [1] and [9] for more details).

7. Algorithmic choices and operation counts. To compare the performance of the filtering approach to the standard FAS multigrid, we modified the FLO52 algorithm to implement the procedure described in the previous section (denoted FLO52-filtering) on a serial machine. Using these codes, we compare the number of iterations required for convergence. Since the code was not implemented on a parallel machine, operation counts are made for both algorithms to compare the time per iteration. Additionally, the operation counts are used to determine the number of concurrent iterations that can be performed in parallel with the coarse grid correction. In this section, we describe the algorithmic choices and briefly discuss the corresponding floating point operation counts for each of the FLO52 algorithms.

The operators used in our experiments are the same as those typically used within the FLO52 code. Specifically, the Runge-Kutta relaxation scheme is used for prerelaxation and concurrent relaxation. Bilinear interpolation and full-weighted restriction are used to transfer values between grids. The coarse grid contains one-fourth as many points as the fine mesh. Coarse grid operators are defined using the same discretization scheme as on the fine grid. Finally, the operator Z, which splits the residual in the filtering algorithm, is given by

$$Z=(P_kR_k)^q, \tag{33}$$

where q is an algorithm parameter.

[1] We note that this choice corresponds to the standard Runge-Kutta scheme used in the FLO52 code, which is probably not the best choice for the FLO52-filtering procedure.

In estimating the time per iteration of both algorithms, we make a number of assumptions on the implementation and the architecture of the parallel machine. In particular, we assume that a hypercube multiprocessor is used and that each processor is assigned to one of the fine grid points using a binary reflected Gray code (see [5]). This implies that many processors are inactive when processing coarser meshes in the FLO52 procedure. In particular, each successively coarser grid contains one-fourth as many points as the previous grid. Thus the number of idle processors when computing on $\mathcal{G}_k$ is given by

$$P(1-(.25)^k), \tag{34}$$

where P is the total number of processors. For the FLO52-filtering algorithm, we assume that these inactive processors are used for the concurrent iterations. Specifically, the following sequence is proposed for a two-level FLO52-filtering algorithm.

(1) Prerelaxation is performed and the residual is computed using the full hypercube.

(2) The residual is split into two components using the full hypercube.

(3) The coarse grid right-hand side and initial guess are computed.

(4) The right-hand side corresponding to the fine grid subproblem is computed.

(5) The coarse grid right-hand side and initial guess are sent to a processor subcube ($\frac{1}{4}$ the size of the original hypercube) according to an algorithm of Chan and Saad (see [5]). At the same time, the right-hand side and initial guess for the fine grid subproblem are sent to a processor subcube ($\frac{1}{2}$ the size of the original hypercube).

(6) Initiate processing on both the coarse grid and fine grid subproblems.

(7) Fine grid processing terminates and the correction from this subproblem is added to the previous fine grid approximation.

(8) The coarse grid processing terminates and the correction is interpolated and added to the fine grid approximation.

The corresponding FAS method is identical except that steps 2, 4, and 7 are omitted and that no fine grid computations are necessary in steps 5 and 6. It should be noted that the communication necessary for initiating the subproblems on the subcubes is local in nature. Furthermore, contention between the subproblems on the communication network is avoided (since each is in a different subcube).

By way of example, Fig. 3 illustrates the different subproblems in a FLO52-filtering algorithm using five grid levels. In the figure, the darker boxes correspond to

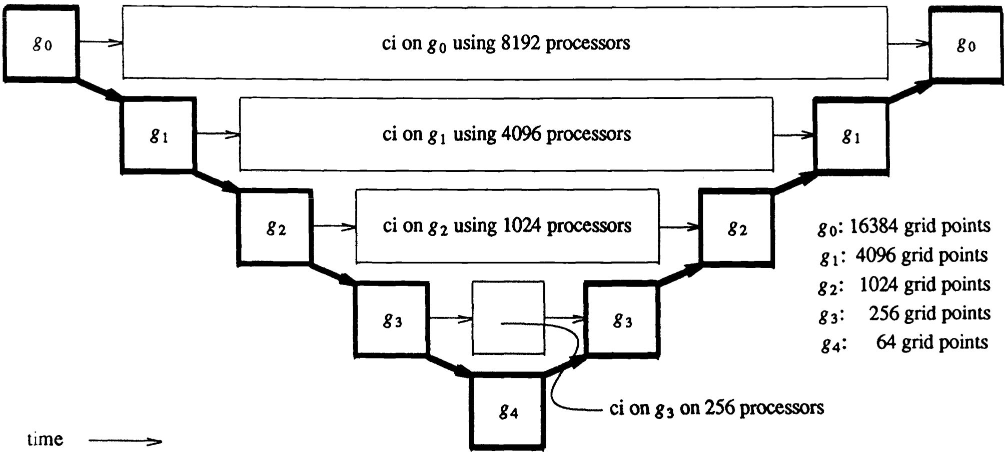


FIG. 3. *Illustration of the subproblems (and processor allocation) for a five-level filtering method.*

the standard multigrid "V" cycle. Within each box comprising the "V" cycle, the grid on which processing occurs is indicated. Each element of the first half of the "V" cycle (except for the coarsest grid) spawns a concurrent iteration problem. This is indicated by the lighter boxes, which are labeled "ci." Finally, we assume that for each element of the "V" cycle calculation, one grid point is assigned to each processor. The processors assigned to each concurrent relaxation subproblem are indicated on the figure.[2]

The operations per iteration of each algorithm are estimated using a timing model developed for an actual hypercube implementation of FLO52 (see [1]). In Table 2, we give the operation counts (not including communication operations) corresponding to a five-grid version of the two methods as a function of the number of prerelaxation sweeps μ and residual splitting cost q. Note that the filtering algorithm is only slightly more expensive. This additional cost is primarily due to the residual splitting computations. Of course, the numbers in Table 2 ignore communication. However, it can be argued that the additional cost of communication for both algorithms would scale in a similar fashion. That is, the amount of communication is proportional to the amount of computation.[3]

TABLE 2
A comparison of the maximum number of floating point operations performed by any processor as a function of prerelaxation sweeps μ and residual splitting costs q.

		FLO52-Filtering		
μ	FLO52	$q=1$	$q=2$	$q=3$
1	309	316	323	330
2	526	533	540	547
3	742	749	756	763

In addition to computing the time per iteration, we use the operation counts to determine the number of concurrent relaxation iterations that can be performed in parallel with the coarse grid correction. That is, the maximum number of concurrent relaxations that can be completed during the coarse grid correction is given by:

$$\begin{aligned}&\#\text{ concurrent iterations on } \mathcal{G}_k\\ &\quad= \text{floor}\left[\frac{\text{time for coarse grid correction for } \mathcal{G}_k}{\text{time for one fine grid relaxation on } \mathcal{G}_k}\right].\end{aligned} \tag{35}$$

In this way, we ensure that the concurrent iterations do not increase the overall time per iteration of the algorithm. In Table 3 we give the number of concurrent iterations that can be performed for each subproblem in Fig. 3 as a function of the number of prerelaxations used, μ.

8. Performance comparisons. A series of convergence experiments were run on a serial machine to evaluate the numerical properties of both the FLO52 and the FLO52-filtering code. For the most part, the relative performance of the two algorithms

[2] Note that for the concurrent iteration problem on grid $\mathcal{G}_0$, each processor contains two grid points. This is because there are not enough processors available to assign one processor for each grid point.

[3] This makes the assumption that the additional communication required for sending the fine grid information to a subcube can be overlapped with the communication necessary to set up the coarse grid correction.

TABLE 3
Number of concurrent iterations performed for each subproblem in Fig. 3 as a function of the number of Runge-Kutta prerelaxation sweeps, μ.

μ/grid	$\mathcal{G}_0$	$\mathcal{G}_1$	$\mathcal{G}_2$	$\mathcal{G}_3$
1	2	3	2	1
2	4	6	4	2
3	6	9	6	3

was fairly consistent for different grids and over a range of MACH numbers. In this paper we present results obtained from runs on a 256×64 grid. For these experiments an angle of attack of 1.25 degrees and a MACH number of 0.8 was chosen. As previously mentioned, a five-level multigrid scheme is used. The computation starts on the coarsest level, $\mathcal{G}_4$, using just relaxation until the norm of the residual is reduced below 10^{-2}. This solution is interpolated to $\mathcal{G}_3$ where a multigrid procedure is used to reduce the norm of the residual to 10^{-4}. The process is repeated for grids $\mathcal{G}_2$, $\mathcal{G}_1$, and $\mathcal{G}_0$, reducing the residual norm to 10^{-5}, 10^{-6}, and 10^{-9}, respectively.

Before proceeding with convergence results, we briefly comment on the serial run time of the filtering algorithm. In particular, we consider the FLO52 algorithm (using three prerelaxation iterations) and the FLO52-filtering algorithms (using two prerelaxation iterations, one concurrent iteration, and $q = 1$ for the residual splitting) on a serial machine. Since prerelaxation is more efficient than concurrent relaxation, it is not surprising to find that the FLO52 algorithm requires less time than the FLO52-filtering method (1568 seconds versus 1815 seconds). This inefficiency within the concurrent relaxation is a consequence of the fact that only the error associated with the fine grid suproblem is reduced (as opposed to prerelaxation, which reduces the entire error). Note that while the FLO52-filtering method is slower on serial machines, it is still competitive.

Of course the motivation for the filtering method is that more of the work is parallelizable. More specifically, we have argued that the cost per iteration of FLO52-filtering is not significantly greater than that of the corresponding FLO52 method when both are implemented on a massively parallel machine. Therefore, we can obtain a rough measure of the relative parallel performance of these two algorithms by comparing the number of iterations required for convergence. In Table 4, we present our results using different numbers of prerelaxation sweeps μ and splitting operators q.[4]

TABLE 4
Comparisons of multigrid iterations for the FLO52 *and* FLO52-*filtering algorithms where iterations correspond to the total (including one-, two-, and three-, four-, five-level multigrid iterations).*

		FLO52-Filtering	
μ	FLO52	$q = 1$	$q = 2$
2	90	78	64
3	53	44	39
4	48	35	33

[4] These results are representative of a number of different runs using different grid sizes, Mach number, and angle of attack.

As the table illustrates, the number of iterations required for the filtering algorithm is significantly less than the number of iterations for the standard method. Furthermore, the use of the more expensive filter, $q = 2$, yields somewhat better convergence results than the simpler filter. Additionally, it should be noted that the savings in using the filtering approach over the standard method is reduced when more prerelaxation is used: $\mu > 4$. This is to be expected, since the primary function of the new subproblems is to reduce high frequency errors in parallel with the coarse grid correction. However, when many prerelaxation sweeps are performed (before forming the new subproblem), the high frequency error is significantly reduced before the coarse grid correction begins.[5] Finally, we remark that it may be possible to boost the performance of the filtering method further by the use of more sophisticated operators. For example, the prerelaxation and concurrent relaxation operators use the identical Runge-Kutta coefficients that are used within the standard FLO52 scheme. These coefficients were specially chosen for the FLO52 method. Thus improvements may be possible if a new set of coefficients is chosen for the new method. Likewise, a more carefully chosen splitting operator Z may also yield significant improvements.

9. Conclusion. We have presented an FAS version of the filtering method. The principle idea is to use processors that could otherwise be idle in a standard multigrid method to perform relaxation iterations. These additional iterations are performed on subproblems that are created by splitting the residual into "smooth" and "oscillatory" components. The "smooth" component is used for the coarse grid correction, while the "oscillatory" component is used for the new subproblem. By using these otherwise idle processors, a filtering iteration need not cost significantly more than standard multigrid iterations. Additionally, we remark that the modifications necessary to implement a filtering algorithm from an FAS algorithm are relatively straightforward. This is primarily because no linearization of the operator is necessary.

We have applied the filtering approach to the FLO52 algorithm for solving the steady two-dimensional Euler equations. Based on numerical experimentation, it has been determined that this filtering method requires fewer iterations than the standard method, and consequently, it is an attractive alternative on parallel computers.

REFERENCES

[1] E. Barszcz, T. Chan, D. Jespersen, and R. Tuminaro, *Performance of a parallel Euler equation code on hypercube computers*, Tech. Memo 102260, NASA-Ames Research Center, Moffett Field, CA, April 1990.

[2] A. Brandt, *Multi-level adaptive solutions to boundary-value problems*, Math. Comp., 31 (1977), pp. 333-390.

[3] W. Briggs, *Multigrid Tutorial*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

[4] T. Chan and R. Tuminaro, *Design and implementation of parallel multigrid algorithms*, in Proc. Third Copper Mountain Conference on Multigrid Methods, S. McCormick, ed., Marcel Dekker, New York, 1987, pp. 101-115.

[5] T. Chan and Y. Saad, *Multigrid algorithms on the hypercube multiprocessor*, IEEE Trans. Comput., C-35 (1986), pp. 969-977.

[6] C. Douglas and W. Miranker, *Constructive interference in parallel algorithms*, SIAM J. Numer. Anal., 25 (1987), pp. 376-398.

[7] P. Frederickson and O. McBryan, *Parallel superconvergent multigrid*, in Proc. Third Copper Mountain Conference on Multigrid Methods, S. McCormick, ed., Marcel Dekker, New York, 1987, pp. 195-210.

[5] An alternative filter that distributes more of the middle frequency errors on the fine grid subproblem may yield better performance.

[8] W. Hackbusch, *A new approach to robust multi-grid methods*, in First Internat. Conference on Industrial and Applied Mathematics, Paris, 1987.

[9] A. Jameson, *Solution of the Euler equations for two-dimensional transonic flow by a multigrid method*, Appl. Math. Comp., 13 (1983), pp. 327-335.

[10] D. Jespersen, *Multigrid methods for partial differential equations*, in Studies in Numerical Analysis, G. Golub, ed., The Mathematical Association of America, 1984, pp. 270-318.

[11] R. Tuminaro, *Multigrid Algorithms on Parallel Processing Systems*, Ph.D. thesis, Department of Computer Science, Stanford University, Stanford, CA, 1989.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 1, pp. 101-118, January 1992

006

FAST PARALLEL ITERATIVE SOLUTION OF POISSON'S AND THE BIHARMONIC EQUATIONS ON IRREGULAR REGIONS*

A. MAYO† AND A. GREENBAUM‡

Abstract. In [*SIAM J. Numer. Anal.*, 21 (1984), pp. 285-299], a method was introduced for solving Poisson's or the biharmonic equation on an irregular region by making use of an integral equation formulation. Because fast solvers were used to extend the solution to an enclosing rectangle, this method avoided many of the standard problems associated with integral equations. The equations that arose were Fredholm integral equations of the second kind with bounded kernels. In this paper iterative methods are used to solve the dense nonsymmetric linear systems arising from the integral equations. Because the matrices are very well conditioned, conjugate gradient-like methods can be used and will converge very rapidly. The methods are very amenable to vectorization and parallelization, and parallel and vector implementations are described on shared memory multiprocessors. Numerical experiments are described and results presented for a three-dimensional interface problem for the Laplacian on a recording head geometry.

Key words. Laplace's equation, biharmonic equation, integral equations, iterative methods

AMS(MOS) subject classification. 65

1. Introduction.

In this paper we present efficient parallel numerical methods for solving Poisson's and the biharmonic equations on general regions. Aside from being parallel, these methods also vectorize very well. This is in contrast to other methods for solving these equations on general irregular regions, notably finite element methods. Finite element methods may require use of a fast scatter/gather operation in order to efficiently perform a matrix-vector multiply for the large sparse matrix that arises. In addition, many of the standard preconditioners used in the iterative solution of such equations require inherently sequential operations (e.g., backsolving with the incomplete Cholesky decomposition [12]). This is especially true in the case of the biharmonic equation, and we know of no other effective parallel and vectorizable methods for solving it.

The methods presented here combine integral equation formulations of the problems with rapid finite difference methods on a larger rectangular region in which the irregular region is embedded. Both the integral equation method and the finite difference method parallelize and vectorize well. By using well-conditioned integral equation formulations and iterative methods for solving them, the main part of the computation is reduced to the iterative solution of a dense nonsymmetric matrix equation, instead of a (much larger) sparse (although symmetric) system of equations with an irregular pattern of nonzero elements.

The method used is very similar to the method developed in [10]. The main idea is to use the integral equation formulation to define a discontinuous extension of the solution to the remainder of the embedding region. All the discontinuities between the original function and its extension can be expressed in terms of the solution of the integral equation. These discontinuities are used to compute an approximation to the discrete Laplacian of the combined function at mesh points near the

* Received by the editors April 5, 1990; accepted for publication (in revised form) May 22, 1991.

† IBM T. J. Watson Research Center, Yorktown Heights, New York 10598. Part of this work was performed while this author was visiting the Courant Institute of Mathematical Sciences, New York, New York.

‡ Courant Institute of Mathematical Sciences, 251 Mercer Street, New York, New York 10012. The work of this author was supported by the Applied Mathematical Sciences Program of the U.S. Department of Energy under contract DE-AC02-76ER03077.

original boundary. Fast Poisson solvers are then used to compute the extended function. This method can also be used to solve directly for the derivatives of the solution, which are usually the physically meaningful quantities.

This combination of integral equation formulations with finite difference methods on a larger rectangular region overcomes two of the standard problems usually associated with integral equation methods. First, because the solution is actually computed using a fast Poisson solver, it is not necessary to compute a costly integral in order to evaluate the solution at many points. Second, although the kernel of the integral is singular, and hence difficult to evaluate accurately by quadrature near the boundary, the solution can be computed very accurately using the fast Poisson solver. In addition, since the methods make use of integral equation formulations, they are very well suited to problems on exterior regions. A future aim is to use these methods for solving nonlinear magnetostatic problems in unbounded domains.

Work described in the current paper differs from that in [10] in several important respects. One difference is that we use a different integral equation formulation of the biharmonic equation, which has probably not been applied before in numerical computations. The other primary difference is the way in which the integral equations are solved. Conjugate gradient type iterative methods (i.e., the biconjugate gradient/conjugate gradient squared method [5], [18]; GMRES [16]; and the conjugate gradient method applied to the normal equations [6]), with simple diagonal preconditioners, are used to solve both Laplace's and the biharmonic equations. Since most of the operations in these methods consist of matrix-vector products, and since the matrices are dense, it is easy to see how the calculations can be performed in parallel and each of the parallel sections can be vectorized. The matrix can be divided into blocks, and each processor can handle multiplication by that block, while vectorization can be performed over the rows within the block. This can actually be accomplished using a level-2 Blas routine [4], and these routines have already been optimized on many parallel/vector supercomputers.

We also note that the integral equations used for solving both of these problems are well-conditioned Fredholm integral equations of the second kind, and so the matrices have eigenvalues and singular values that are bounded independent of the number of discretization points. Therefore we were able to achieve convergence in a small number of iterations. In particular, we were able to solve the biharmonic equation on general nonconvex regions, and, except on very eccentric ellipses, all of the iterative methods tested converged in fewer than 30 iterations. To solve the biharmonic equation on an ellipse with eccentricity 10 or 20, required no more than 32 GMRES iterations, provided enough direction vectors were saved.

In addition to using iterative methods to solve the integral equations, we also solved equations with different types of boundary conditions than before. Specifically, we solved a three-dimensional exterior interface problem in magnetostatics that arises from modeling a recording head, and a two-dimensional Dirichlet problem for Laplace's equation on a doubly connected region.

We note that fast Poisson solvers have also been used for solving Poisson's equation on irregular regions through the use of capacitance matrix methods [2], [14]. Our method is, however, better suited to problems on exterior regions. Although, by using a trick due to Hockney [7] and improved by James [8], it is possible to use capacitance matrix methods to solve exterior problems, both the operation count and the storage requirements for the fast solver are significantly increased. More important, capacitance matrix methods do not allow one to solve for derivatives directly. Moreover, we know of no implementation of a capacitance matrix method for solving the biharmonic equation on an irregular region or for solving interface problems.

The organization of the paper is as follows. Section 2 presents the integral equation formulations, and § 3 the method used for obtaining solutions to the differential equations, given the solutions of the integral equations. Section 4 describes results of numerical experiments carried out on the 8-processor New York University ultracomputer prototype, on the Cray X-MP and on the IBM 3090.

2. Integral equations.

2.1. Laplace's equation. Laplace's equation in two dimensions with Dirichlet boundary data $g(t)$ prescribed on the boundary curve $(x(s), y(s))$ is solved in terms of a double layer density function. In this formulation, the solution $u(z)$ is expressed as the integral over the boundary of the region of the product of an unknown density function μ with the normal derivative of the Green's function in the plane:

$$u(z)=\frac{1}{2\pi}\int_{\partial D}\frac{\partial \log r(s,z)}{\partial n_s}\mu(s)\,ds, \qquad z\in D, \tag{1}$$

where

$$r(s,z)=|s-z|.$$

If the region is simply connected, the density function μ satisfies

$$\mu(t)+\frac{1}{\pi}\int_{\partial D}\frac{\partial \log r(s,t)}{\partial n_s}\mu(s)\,ds=2g(t).$$

If the region is doubly connected, with boundary curves L_0 and L_1, then the boundary values can only be prescribed up to an additive constant. In this case $u(z)$ is the real part of a *single valued* analytic function, and the density function satisfies the following equation [13]:

$$\mu(t)+\frac{1}{\pi}\int_{\partial D}\left(\frac{\partial \log r(s,t)}{\partial n_s}+a(s,t)\right)\mu(s)\,ds=2g(t), \tag{2}$$

where

$$a(s,t)=\begin{cases}1 & \text{if } s \text{ and } t \text{ both lie on } L_1,\\ 0 & \text{otherwise.}\end{cases}$$

These are Fredholm integral equations of the second kind and, if the boundary ∂D is smooth enough, they can be solved very accurately numerically.

Outside D, (1) defines another harmonic function $\tilde{u}(z)$:

$$\tilde{u}(z)=\frac{1}{2\pi}\int_{\partial D}\frac{\partial \log r(s,z)}{\partial n_s}\mu(s)\,ds. \tag{3}$$

The function $\tilde{u}$ is, of course, a discontinuous extension of the function u. However, all the discontinuities between u and $\tilde{u}$ can be expressed in terms of the density μ and the boundary curve. In § 3 we show how to determine the discontinuities and how to use them to rapidly compute an approximation to the combined function $U(z)$, which is equal to $u(z)$ inside D, and $\tilde{u}(z)$ outside D.

We can also use (1) to express the conjugate function $v(z)$ to $u(z)$ in terms of the density μ. Since the conjugate harmonic function of $\partial \log r(s,z)/\partial n_s$ is $\partial \log r(s,z)/\partial s$, we have

$$v(z)=\frac{1}{2\pi}\int_{\partial D}\frac{\partial \log r(s,z)}{\partial s}\mu(s)\,ds. \tag{4}$$

2.2. Interface problems. A similar formulation can be used to solve certain interface problems. In linear magnetostatics, for example, the following problem arises. One seeks to find a function $u(z)$ defined on a magnetizable region D and on the infinite exterior region outside D such that

$$\nabla \cdot a\nabla u(z) = \phi(z),$$

where $\phi(z)$ is the given potential function for the applied field, a is the magnetic permeability of the region, and $u(z)$ is the unknown potential function for the H field (i.e., the function satisfying $\nabla u = -H$). In typical linear problems, a is equal to a large constant a_1 inside the magnetizable region D, and equal to a small constant a_0 in the exterior region. Continuity of the H field and the normal component of the B field imply that $u(z)$ and $au_n(z)$ are continuous across the boundary of D.

It turns out that both inside and outside D, the potential u can be expressed as the sum of a term involving the applied potential and the integral of a double layer density function, where the density is the value of the potential on the boundary:

$$(5) \qquad u(z) = \frac{\tilde{a}+1}{\tilde{a}} \int_{\partial D} \frac{\partial G(s,z)}{\partial n} u(s)\,ds + \frac{\phi(z)}{\tilde{a}}, \qquad z \in D$$

$$(6) \qquad u(z) = (\tilde{a}+1) \int_{\partial D} \frac{\partial G(s,z)}{\partial n} u(s)\,ds + \phi(z), \qquad z \in D^c,$$

where $\tilde{a} = a_1/a_0$ is the relative permeability. See [9]. On the boundary of the region, the potential $u(t)$ satisfies

$$(7) \qquad u(t) - \frac{\tilde{a}-1}{\tilde{a}+1} \int_{\partial D} \frac{\partial G(s,t)}{\partial n} u(s)\,ds = 2\phi(t).$$

In two dimensions, the kernel (Green's function) is $(1/2\pi)\log r(s,z)$, while in three dimensions it is $1/4\pi r(s,z)$. We have solved this integral equation in three dimensions, in which case the kernel, while in $L^2(\partial D)$, is unbounded.

2.3. The biharmonic equation. The integral equation formulation used for the biharmonic equation relies on the representation of a biharmonic function in terms of Goursat functions. Any two-dimensional biharmonic function $W(z)$ can be expressed as

$$(8) \qquad W(z) = \operatorname{Re}(\bar{z}\phi(z) + \chi(z)),$$

where $\phi(z)$ and $\chi(z)$ are analytic functions. The functions $\phi(z)$ and $\psi(z) = \chi'(z)$ are known as the Goursat functions. For a given biharmonic function $W(z)$, the Goursat functions are not uniquely determined. More precisely, $\phi'(z)$ is determined to within a purely imaginary additive constant, and so $\phi(z)$ is determined to within an additive term of the form

$$i\alpha z + \beta,$$

where α is real and β is complex. Similarly, $\psi(z)$ is not uniquely determined. However, all the *physically meaningful quantities*, such as velocities in fluid mechanics or stresses and displacements in elasticity can be expressed in terms of certain derivatives of the Goursat functions, which are uniquely determined. Therefore, it suffices to compute these two functions, and not the biharmonic function itself. Both of these functions can be expressed as Cauchy integrals:

$$(9) \qquad \phi(z) = \frac{1}{2\pi i} \int_{\partial D} \frac{\omega(s)}{s-z}\,ds, \qquad \psi(z) = \frac{1}{2\pi i} \int_{\partial D} \frac{\bar{\omega}(s) - \bar{s}\phi'(s)}{s-z}\,ds.$$

The integral equation one uses to determine the density function $\omega(t)$ depends on the boundary conditions. We have considered the case in which W_x and W_y (or, equivalently, W and W_n) are prescribed. In previous work [10], the following integral equation, originally developed by Sherman [17], was used:

$$\begin{aligned}\omega(t)+\frac{1}{\pi}\int_{\partial D}\omega(s)\,d\theta-\frac{1}{\pi}\int_{\partial D}\bar{\omega}(s)\,e^{2i\theta}\,d\theta\\ +\frac{b}{\pi i}\operatorname{Re}\left(\int_{\partial D}\frac{\omega(s)}{(s-c)^2}\,ds\right)=f(t),\end{aligned}\tag{10}$$

where

$$f(t)=W_x(t)+iW_y(t),$$

$$\theta(s,t)=\arctan\left(\frac{y(s)-y(t)}{x(s)-x(t)}\right),\qquad b=\frac{1}{t-c}-\frac{1}{(\bar{t}-\bar{c})^2}+\frac{t-c}{(\bar{t}-\bar{c})^2},$$

and c is any point inside the region D. It is important to note that the kernel of this equation is always bounded.

It is known [13] that if a function $\omega(t)$ satisfies this equation, then the third integral in the equation is zero, provided $f(t)$ satisfies the natural compatibility condition $\operatorname{Re}\left(\int_{\partial D}\bar{f}(t)\,dt\right)=0$. Thus (10) can be replaced by

$$\omega(t)+\frac{1}{\pi}\int_{\partial D}\omega(s)\,d\theta-\frac{1}{\pi}\int_{\partial D}\bar{\omega}(s)\,e^{2i\theta}\,d\theta=f(t).\tag{11}$$

We chose to solve this integral equation instead of (10), despite the fact that this equation need not have a unique solution. This is acceptable for the following reason. Any solution $\omega_0(t)$ of the homogeneous equation

$$\omega_0(t)+\frac{1}{\pi}\int_{\partial D}\omega_0(s)\,d\theta-\frac{1}{\pi}\int_{\partial D}\bar{\omega}_0(s)\,e^{2i\theta}\,d\theta=0$$

corresponds to the Goursat functions $\phi_0(z)=i\alpha z$ and $\psi_0(z)=0$, where α is real [13]. Since $\phi(z)$ is only determined up to terms of this form, and since the physically meaningful quantities are not changed by adding terms of this type to $\phi(z)$, any solution of the modified equation will provide a physically correct solution. Existence of multiple solutions is not a problem for the iterative methods we used, as will be explained in § 4. Furthermore, it is easy to show that whenever the kernel $\partial\theta/\partial s$ is symmetric, the kernel of (11) will be symmetric. This is true, for example, when the region is an ellipse. A symmetric kernel is an advantage, since in this case the minimum residual variant of the conjugate gradient algorithm can be used to solve the problem (and, in fact, the biconjugate gradient method and the (unrestarted) GMRES method reduce to the minimum residual conjugate gradient algorithm). Conjugacy of direction vectors is maintained without having to save vectors and explicitly orthogonalize. In addition, even when the kernel is not symmetric, we found that the kernel of (11) is closer to being normal than that of (10). Experimental evidence indicated that convergence was always more rapid with (11) than with (10), so it was decided to use this modified equation.

3. Solving the differential equations. After solving the integral equations of § 2 for the density function μ, we are then able to compute the solution $u(z)$, at points z within the region D, in the following manner. Discontinuous extensions of the solutions to the differential equations are defined throughout a larger embedding rectangular region. Specifically, (1) defines the extension of the solution of Laplace's equation, and (9) defines the extensions of the real and imaginary parts of the Goursat functions for the biharmonic equation. These functions are all harmonic, except at the boundary of the original region. Therefore, their discrete Laplacians are all zero, up to truncation error, except at mesh points near the boundary. We compute approximations to the discrete Laplacians at these points near the boundary and approximations to the extended function at the edge of the embedding region. Then we apply fast Poisson solvers to obtain approximate solutions to the differential equations, throughout the domain D.

3.1. Solution of Laplace's equation. The integral equation (1) can be used with the second-order accurate five-point discrete operator Δ_h^5 to compute a second-order accurate solution to $\Delta u = 0$ on an irregular region D with smooth boundary $\partial D = (x(s), y(s))$ on which smooth Dirichlet boundary data $u(s) = g(s)$ is prescribed. The procedure is as follows.

First embed D in some regular region R, such as a square with a uniform mesh of width h in the x and y directions. Define a discontinuous extension $\tilde{u}$ of u throughout the region R, using (3). As noted previously, all the jump discontinuities in the derivatives of u and $\tilde{u}$ can be expressed in terms of the density function and the derivatives of the boundary curve. Once the discontinuities in the derivatives of the combined function are known and the distances from the boundary curve to the neighboring mesh points are computed, it is easy to compute the discrete Laplacian of the combined function. Knowing an approximation to the discrete Laplacian throughout the regular region R, a fast Poisson solver can then be used to obtain an approximation to the solution u.

Define $U(z)$ to be combined function,

$$U(z) = \begin{cases} u(z) & z + D, \\ \tilde{u}(z) & z \in D^c. \end{cases}$$

The discontinuity between u and $\tilde{u}$ at a point on the boundary of D is equal to the value of the density at that point [13]. Therefore, the discontinuity between their tangential derivatives is equal to the derivative of the density,

$$u_s - \tilde{u}_s = \frac{d\mu}{ds}.$$

There is no discontinuity between their normal derivatives:

$$u_n = \tilde{u}_n.$$

Using these two facts and the direction of the curve, we can compute the discontinuities between u_x and $\tilde{u}_x$ and between u_y and $\tilde{u}_y$:

$$(12) \qquad u_x(s) - \tilde{u}_x(s) = \frac{\mu'(s)x'(s)}{x'(s)^2 + y'(s)^2}, \qquad u_y(s) - \tilde{u}_y(s) = \frac{\mu'(s)y'(s)}{x'(s)^2 + y'(s)^2}.$$

Discontinuities in higher-order derivatives of U can be obtained by differentiating these expressions.

In two dimensions it is also possible to compute these discontinuities by using the fact that $U(z)$ is the real part of a Cauchy integral with density function μ. Let

$$f(z) = \frac{1}{2\pi i} \int_{\partial D} \frac{\mu(\zeta)}{\zeta - z} \, d\zeta.$$

The kernel in (1) is the real part of the Cauchy kernel:

$$\operatorname{Re}\left(\frac{1}{2\pi i}\,\frac{d\zeta/ds}{\zeta-z}\,ds\right)=\frac{1}{2\pi}\,\frac{y'(s)[x(s)-x(t)]-x'(s)[y(s)-y(t)]}{(x(s)-x(t))^2+(y(s)-y(t))^2}\,ds$$
$$(13)\qquad =\frac{1}{2\pi}\,\frac{\partial \log r(s,z)}{\partial n_s}\,ds,$$

where

$$\zeta(s)=x(s)+iy(s), \qquad z(t)=x(t)+iy(t).$$

Therefore $u(z)=\operatorname{Re}(f(z))$ for z in D, and $\tilde{u}(z)=\operatorname{Re}(f(z))$ for z outside D. To find the discontinuities between u and $\tilde{u}$, we use the known discontinuities of Cauchy integrals across the boundary curve and the fact that Cauchy integrals are analytic functions. For details see [10].

These discontinuities are then used to compute approximations to the discrete Laplacian at mesh points near the boundary of D. Define the mesh function $U_{i,j}$ by

$$U_{i,j}=\begin{cases} u(x_i,y_j) & (x_i,y_j)\in D,\\ \tilde{u}(x_i,y_j) & (x_i,y_j)\in D^c.\end{cases}$$

An approximation to $\Delta_h^5 U_{i,j}$ at all mesh points of R can be computed as follows. At mesh points (i,j) of R, which have all four of their neighboring mesh points $(i+1,j)$, $(i-1,j)$, $(i,j+1)$, and $(i,j-1)$ on the same side of ∂D, set $\Delta_h^5 U_{i,j}=0$, since u and $\tilde{u}$ are harmonic. Consider the set B^+ of mesh points that have neighboring mesh points on both sides of ∂D. This set consists of two rings of mesh points, one inside and one outside D. Let p, for example, be a mesh point that is in D but whose neighbor to the right, p_E, is not. Let p^* be the point on ∂D on the line between p and p_E, let h_1 be the distance between p and p^*, and let $h_2=h-h_1$.

By manipulating the Taylor series at p and p_E, both evaluated at p^*, one can derive the following expression for $\tilde{u}(p_E)-u(p)$. (For details see [10].)

$$\begin{aligned}(14)\qquad \tilde{u}(p_E)-u(p)&=[\tilde{u}(p^*)-u(p^*)]+h_2[\tilde{u}_x(p^*)-u_x(p^*)]\\ &\quad+\tfrac{1}{2}h_2^2[\tilde{u}_{xx}(p^*)-u_{xx}(p^*)]+\tfrac{1}{6}h_2^3[\tilde{u}_{xxx}(p^*)-u_{xxx}(p^*)]\\ &\quad+\tfrac{1}{24}h_2^4[\tilde{u}_{xxxx}(p^*)-u_{xxxx}(p^*)]\\ &\quad+\tfrac{1}{120}h_2^5[\tilde{u}_{xxxxx}(p^*)-u_{xxxxx}(p^*)]\\ &\quad+hu_x(p)+\tfrac{1}{2}h^2u_{xx}(p)+\tfrac{1}{6}h^3u_{xxx}(p)+\tfrac{1}{24}h^4u_{xxxx}(p)\\ &\quad+\tfrac{1}{120}h^5u_{xxxxx}(p)+O(h^6).\end{aligned}$$

Note that the first six terms depend on the discontinuities between u and $\tilde{u}$ and their derivatives at the boundary. The other terms are the usual Taylor series terms. Therefore, once these discontinuities are known, the right-hand side of (14) is just the sum of known quantities and Taylor series terms.

Now let p_W be the point to the left of p. Then we have

$$\begin{aligned}U(p_W)-U(p)&=\{\text{known quantities}\}-hu_x(p)+\tfrac{1}{2}h^2u_{xx}(p)-\tfrac{1}{6}h^3u_{xxx}(p)\\ &\quad+\tfrac{1}{24}h^4u_{xxxx}(p)-\tfrac{1}{120}h^5u_{xxxxx}(p)+O(h^6),\end{aligned}$$

where the quantities in braces are zero if p_W is in D. In any case, we have

$$\begin{aligned}(15)\qquad U(p_W)+U(p_E)-2U(p)&=\{\text{known quantities}\}+h^2u_{xx}(p)\\ &\quad+\tfrac{1}{12}h^4u_{xxxx}(p)+O(h^6).\end{aligned}$$

If p_N is the point above p and p_S is the point below p, then by the same arguments we have

$$\begin{aligned}U(p_N)+U(p_S)-2U(p) &= \{\text{known quantities}\}+h^2u_{yy}(p)\\ &\quad +\tfrac{1}{12}h^4u_{yyyy}(p)+O(h^6).\end{aligned} \tag{16}$$

Let $g^+_{i,j}$ denote the sum of the quantities in braces in (15) and (16). Adding (15) and (16) and using the fact that $u_{xx}(p)+u_{yy}(p)=0$, we obtain

$$h^2\Delta^5_h U(p)=g^+_{i,j}+\frac{h^4}{12}[u_{xxxx}(p)+u_{yyyy}(p)]+O(h^6). \tag{17}$$

The same equation holds with u replaced by $\tilde{u}$ if p is a point outside D since $\tilde{u}$ is then harmonic at p. (It is important to notice that it is *not* assumed that either of the harmonic functions u or $\tilde{u}$ can be extended beyond ∂D. If, however, they could both be extended one mesh width, then the formulas we would obtain for the discrete Laplacian clearly agree with those we have derived.)

Therefore, if the integral equation can be solved accurately enough, one can obtain a fourth-order accurate approximation to $h^2\Delta^5_h U(p)$ at points of B^+. This guarantees the accuracy of the solution obtained after applying a fast solver. Define U^h_1 to be the solution of the following set of equations

$$\begin{aligned}(\Delta^5_h U^h_1)_{i,j} &= 0 & (x_i,y_j)&\in R-B^+\\ (\Delta^5_h U^h_1)_{i,j} &= g^+_{i,j} & (x_i,y_j)&\in B^+\\ (U^h_1)_{i,j} &= U_{i,j} & (x_i,y_j)&\in \partial R\end{aligned} \tag{18}$$

where the boundary values $U_{i,j}$ on ∂R are computed by evaluating the integral (3) using the computed density μ. In practice, we have used the trapezoid rule as the quadrature formula for two-dimensional problems.

If the values of $g^+_{i,j}$ and $U_{i,j}$ are sufficiently accurate, then U^h_1 will be a second-order accurate approximation to U. For a proof see [10]. By using a higher-order accurate approximation to the Laplacian and retaining more terms in the Taylor series, a higher-order accurate solution can be obtained.

3.1.1. Computation of the conjugate function. The conjugate of a harmonic function can also be computed at small additional cost. Using the Cauchy-Riemann equations, the discontinuities in the conjugate function v can be expressed in terms of the discontinuities in u. For example,

$$v_y-\tilde{v}_y=-(u_y-\tilde{u}_y)=\frac{\mu'(s)y'(s)}{x'(s)^2+y'(s)^2}.$$

These discontinuities could also be computed using the fact that v is the imaginary part of the same Cauchy integral that u determines:

$$v(z)=\text{Im}\left(\frac{1}{2\pi}\int_{\partial D}\frac{\mu(\zeta)}{\zeta-z}\,d\zeta\right)=\frac{1}{2\pi}\int_{\partial D}\frac{\partial \log r(s,z)}{\partial s}\mu(s)\,ds.$$

Knowing these discontinuities, we can, in the same way as before, compute an approximation to the discrete Laplacian of v, and then apply a fast Poisson solver to obtain an approximation to v.

3.1.2. Computation of derivatives. An important property of these methods is that one can easily compute the derivative of a harmonic function directly, without having to compute the function itself. This follows because all the derivatives of a harmonic function are themselves harmonic and because one can compute the discontinuities in all these derivatives.

3.2. Solution of interface problems. Since the integrals in (6) and (7) are also integrals of double layer density functions, the same method can be used to solve interface problems.

3.3 Solution of the biharmonic equation. As noted in § 2, the solution of the biharmonic equation can be reduced to the evaluation of two analytic functions that satisfy (9) and the prescribed boundary conditions. More precisely, the evaluation of the physically meaningful quantities reduces to the evaluation of the Goursat functions (8).

Both of the Goursat functions can be expressed as Cauchy integrals whose densities are given in terms of the solution of the integral equation (11). Therefore, it suffices to be able to evaluate a Cauchy integral $g(z)=\int_{\partial D}(f(t)/z-t)\,dt$ with prescribed density function $f(t)$. Suppose $f(t)=p(x,y)+iq(x,y)$. The integral $g(z)$ can be expressed in terms of certain integrals of double layer density functions and their conjugates:

$$g(z)=\frac{1}{2\pi i}\int_{\partial D}p(s)\left(\frac{\partial \log r}{\partial n_s}+i\frac{\partial \log r}{\partial s}\right)ds+\frac{1}{2\pi}\int_{\partial D}q(s)\left(\frac{\partial \log r}{\partial n_s}-i\frac{\partial \log r}{\partial s}\right)ds.$$

Therefore, the same methods can be used to evaluate the Goursat functions.

4. Numerical experiments. In this section we report on calculations we have performed. Parallel implementation of the method is discussed, and details are given on the performance of the iterative methods used for solving the integral equations. Accuracy of the method was discussed in detail in [10] and will not be repeated here.

The complete solution method consists of the following steps:

1. The irregular region D is embedded in a rectangle R with a uniform mesh, and distances from neighboring mesh points to the boundary are computed.

2. The boundary(ies) of the region D are discretized, and the integral operators are replaced by quadrature formulae. Since the kernels are bounded for the two-dimensional problems, we used the trapezoid rule, since it is extremely accurate on periodic regions. For the three-dimensional interface problem, a different procedure was used since the kernel is unbounded. We triangulated the surface of the region, which was flat. We assumed a continuous, piecewise linear density function and integrated the product of the kernel function with the piecewise linear basis functions exactly. Explicit quadrature formulae can be found in [9]. (These formulae require the surface to be flat.)

3. Having formed the dense, nonsymmetric matrix arising from the integral equation, the appropriate linear system is then solved for the density μ using an iterative method. Methods that we experimented with include the biconjugate gradient/conjugate gradient squared method [5], [18], GMRES [16], and the conjugate gradient method applied to the normal equations [6]. In each case, the diagonal of the matrix was used as a preconditioner.

4. Approximations to the discrete Laplacian are computed near the boundary ∂D, and approximations to the extended function are computed along the boundaries of R.

5. A fast Poisson solver is applied to obtain an approximate solution to the differential equation.

4.1. Test problems and performance of iterative methods. Although the matrices that arise from the integral equations of § 2 are dense and nonsymmetric, they are also well conditioned and, in most cases, have tightly clustered eigenvalues and singular values. For this reason, all of the nonsymmetric conjugate gradient-type iterative methods that we have tried have performed well. In each case, the diagonal of the matrix was used as the preconditioner. Results are presented primarily for the biconjugate gradient (BCG) method, but some of the problems were also solved using GMRES [16], conjugate gradients on the normal equations (CGNR) [6], and the conjugate gradient squared (CGS) method [18].

The effectiveness of these iterative methods for solving the double layer integral equation for Laplace's equation on simply connected regions is well documented. (See, for example, [11].) We have performed calculations on doubly connected regions and have found the results to be similar. In particular, we have tested the biconjugate gradient method on a region bounded by two nonconcentric ellipses. The outer ellipse had semiaxes .35 and .40, the inner ellipse had semiaxes .08 and .10, and the center of the inner ellipse was offset .14 from the center of the outer ellipse along the major axis. Starting with zero as an initial guess, we iterated until the relative size of the pseudo-residual was reduced below 10^{-6}:

$$\frac{\|M^{-1}r^k\|}{\|M^{-1}f\|}<10^{-6}.$$

Here M is the diagonal of the matrix, r^k is the residual at step k, and f is the right-hand side of the matrix equation. In all cases, the true residual, $f-Ax^k$ was computed, since, for some problems this may differ from the vector r^k generated by the biconjugate gradient algorithm.

Table 1 shows the number of iterations required to satisfy this convergence criterion, for different values of n, the number of boundary discretization points. Table 2 shows results for a slightly different region. Now the outer ellipse has semiaxes .35 and .15.

The biconjugate gradient method was also used to solve the three-dimensional integral equation (7) for an interface problem in magnetics. The region D, shown in Fig. 1, is shaped like a U-shaped recording head, with equal pole tips of width 5

TABLE 1
Doubly connected Region 1.

n	Number of BCG Iterations
50	6
100	6
200	6
360	9

TABLE 2
Doubly connected Region 2.

n	Number of BCG Iterations
50	9
100	9
200	9
360	12

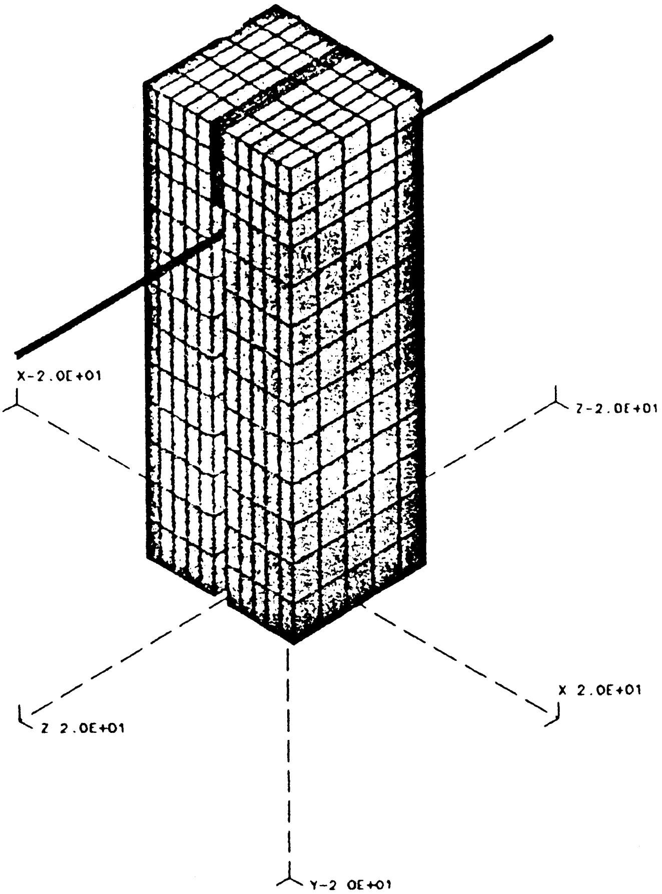


FIG. 1. *U-shaped recording head.*

meters, gap 1 meter, height 30 meters, and uniform depth 10 meters. The function $\phi(z)$ was the applied field due to an infinitely long straight wire with 100 amps of current, and the relative permeability was set to 1000. In one case we used 361 boundary elements and in another case we used 1086 elements. In both cases our initial guess for the potential at a point on the boundry was the value of the applied field at that point. The biconjugate gradient method required 28 iterations to achieve convergence for the smaller problem, 31 iterations for the larger problem.

What is perhaps more interesting is that good results were also obtained using iterative methods to solve the integral equation for the biharmonic equation. Table 3 shows the number of iterations required by various iterative methods to satisfy the convergence criterion on several different regions. We always placed 256 points on the boundary of the region, so the matrix was of order 512. The crescents mentioned in the table are nonconvex and are pictured in Fig. 2 and Fig. 3. They are given parametrically by

$$x(t) = \cos t + .15 \sin^2 t, \qquad y(t) = .5 \sin t + d \cos^2 t.$$

In crescent 1, $d = .4$, and in crescent 2, $d = .7$.

Most of the numbers in the table were obtained by taking the right-hand side of the equation to correspond to the biharmonic function $W(x, y) = x^2 + y^2 + x$, and using

TABLE 3
Biharmonic equation (11).

Region	BCG	CGS	GMRES (5)	GMRES (40)	CGNR
1. Circle of radius 1	4	3	4	4	4
2. Ellipse with semiaxes 1 and 2	9	6	10	9	12
3. Ellipse with semiaxes 1 and 10	32 (20)	27 (18)	73 (58)	32 (20)	74 (41)
4. Ellipse with semiaxes 1 and 20	58 (22)	19 (10)	44 (23)	32 (14)	145 (21)
5. Crescent 1	11	8	12	11	16
6. Crescent 2	16	11	23	15	26

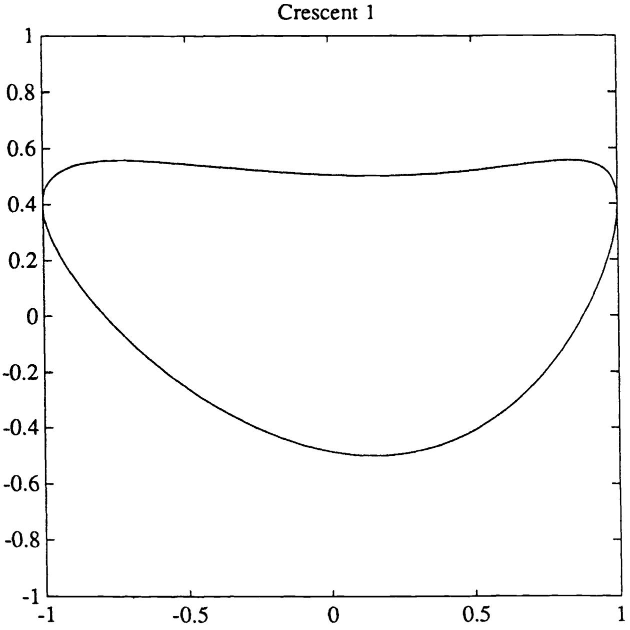


FIG. 2

a *random* initial guess. In practice, however, it is usually advantageous to use a smooth initial guess, such as zero or the right-hand side, since the true solution will be similarly smooth. To determine whether a reasonably chosen initial guess could significantly reduce the number of iterations, we solved the biharmonic problem for several more complicated right-hand sides, using a zero initial guess. The results did not differ significantly from those obtained previously, except on the highly eccentric ellipses. The numbers in parentheses in Table 3 are iteration counts using an initial guess of zero, when the biharmonic function $W(z)$ is given by

$$W(z) = \text{Re}\,(\bar{z}\phi(z) + \chi(z)), \quad \text{where}$$

$$\phi(z) = \sum_{k=0}^{5} (-1)^k \frac{z^k}{k+1} + e^z, \qquad \chi(z) = \sum_{k=0}^{5} \frac{z^k}{k+1} - 2e^z.$$

We believe that these convergence rates may be more realistic than those obtained with a random initial guess, for most applications.

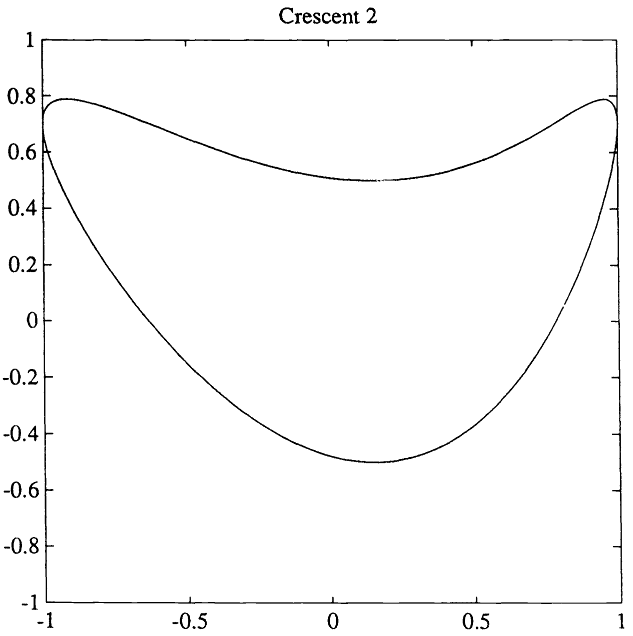


FIG. 3

Note that the most difficult problems for the iterative methods (with either initial guess) were those defined on highly eccentric ellipses. For problems 3 and 4, the GMRES method converged in 32 iterations (for the random initial guess), provided all direction vectors were saved (GMRES (40)), but required significantly more iterations when only five direction vectors were saved (GMRES (5)). Similarly, the other iterative methods were slower to converge on these regions.

It should be noted that some of these iterative methods require more work per iteration than others. Since the bulk of the work at each iteration is in applying the dense matrix to a vector, the BCG, CGS, and CGNR methods require almost twice as much work per iteration as the GMRES method, even when 40 direction vectors are saved and orthogonalized against at each step. (This is because the matrix is dense and so a standard matrix-vector multiply requires n^2 operations, compared to about $40n$ for orthogonalizing. In future work, it is planned to replace this standard matrix-vector multiply routine by an $O(n)$ method [3]. Still, the matrix-vector multiply will require significantly more than $40n$ operations, so the comparison of operation counts per iteration for the various iterative methods probably will not change.) Thus, to compare the methods according to total work performed, the iteration counts for BCG, CGS, and CGNR should be multiplied by two and compared with that of GMRES (5) and GMRES (40). It can then be seen that the GMRES (40) method is the most efficient, with CGS or GMRES (5) second. In addition, it may be necessary to compute the residual directly (as we have done) at each step of the CGS algorithm, since the vector r^k computed by updating may not resemble the true residual. If this is done, then the number of matrix-vector multiplies per CGS iteration is three instead of two, and the GMRES methods look even better in comparison. Of course, GMRES (40) requires almost 40 vectors more in storage than these other methods, and it is difficult to predict just how many vectors will be needed in order to obtain rapid convergence with GMRES. The least efficient iterative method for these problems is CGNR.

To understand why the iteration counts are as described, we computed the eigenvalues and singular values for these problems. The 512 singular values for each region are plotted in decreasing order of magnitude in Fig. 4. The eigenvalues were very close to the singular values (almost indistinguishable on graphs such as Fig. 4) and had tiny imaginary parts and nonnegative real parts. This tends to be an advantage for the GMRES method over, say, CGNR, because the convergence rate of GMRES is governed by the eigenvalues while that of CGNR is determined by the *squared* singular values. We note that in all cases, the eigenvalues and singular values cluster around 1. For the eccentric ellipses, more eigenvalues and singular values appear near the ends of the spectrum, causing some difficulty for the iterative methods. The ratio of the largest to the second smallest singular value for each of the six regions is 2.0, 5.2, 265, 2031, 8.5, and 22.3, respectively.

The presence of a zero singular value does not adversely affect the convergence rate of the iterative methods. The reason for this is as follows. For each of these

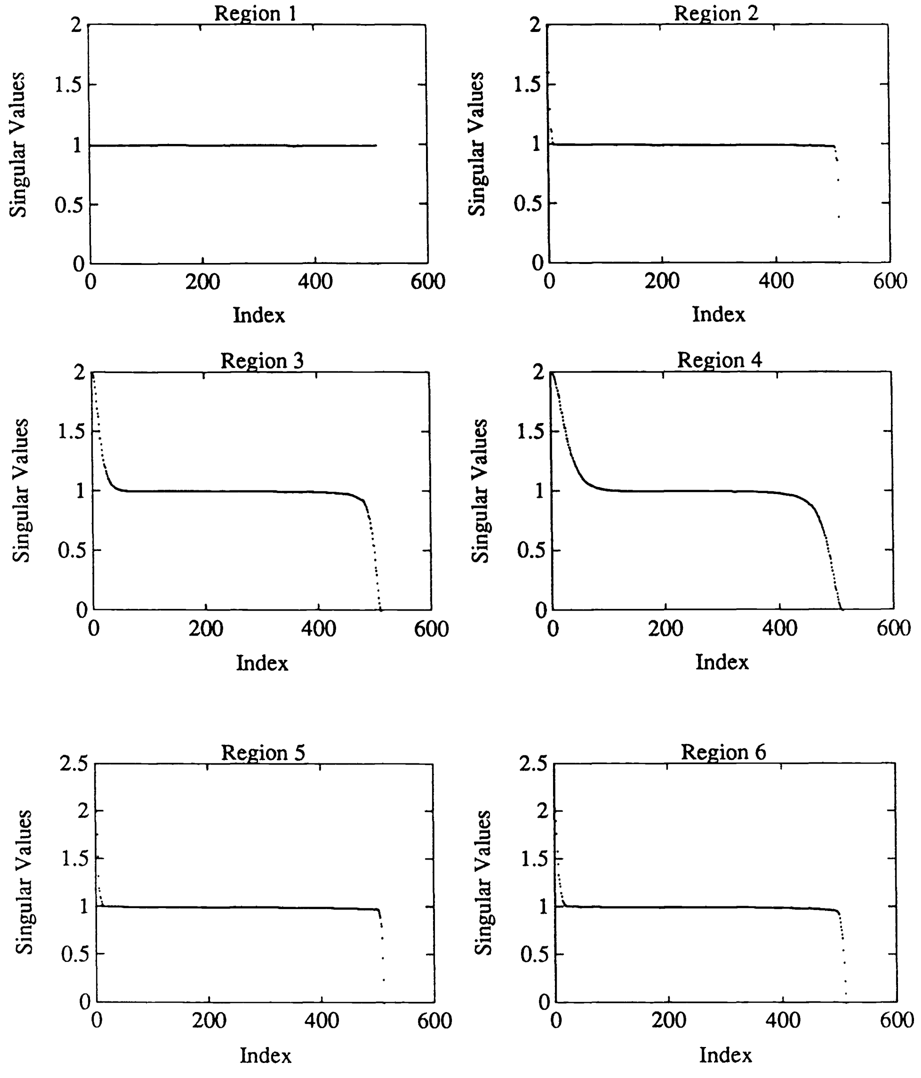


FIG. 4

iterative methods, the residual r^k satisfies

$$r^k = P_k(B)r^0,$$

where P_k is a kth or $2k$th for (CGS) degree polynomial with value one at the origin and $B = A$ for the GMRES, BCG, and CGS methods (actually, $M^{-1}A$, where M is the diagonal of A), $B = AA^T$ for CG on the normal equations. Assuming B has a complete set of eigenvectors V (which it does for each of these problems), we can write $B = V\Lambda V^{-1}$ and

$$V^{-1}r^k = P_k(\Lambda)V^{-1}r^0. \tag{19}$$

Now suppose some eigenvalue λ_i is zero. Since the equations are consistent, the ith component of V^{-1} times the right-hand side f of the equation $Bx = f$ must also be zero; for we have

$$\lambda_i(V^{-1}x)_i = (V^{-1}f)_i.$$

It also follows that the ith component of V^{-1} times the initial residual $r^0 = f - Bx^0$ is zero:

$$(V^{-1}r^0)_i = (V^{-1}f)_i - \lambda_i(V^{-1}x^0)_i.$$

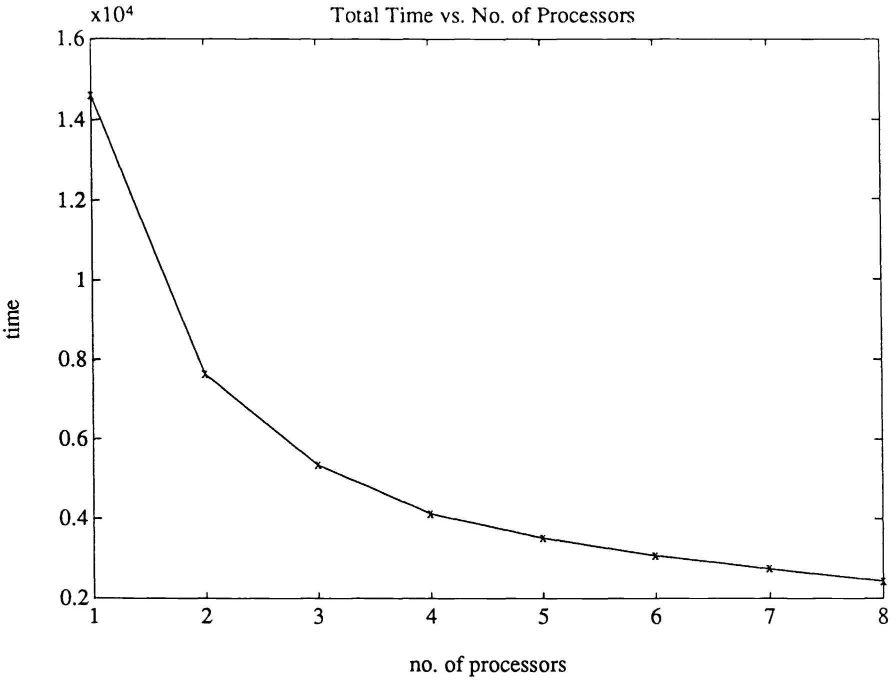
It then follows from (19) that the ith component of $V^{-1}r^k$ is zero for all steps k, and it is easy to check that the coefficients computed by each of these iterative methods are the same as those that would be computed if this zero eigenvalue were not present. Thus, these methods converge to *a* solution of the singular linear system at the same rate as they would converge to *the* solution of a nonsingular linear system having only the nonzero eigenvalues.

Table 4 shows the number of iterations needed to satisfy the same convergence criterion, when we discretized (10) instead of (11). Note that the number of iterations required was significantly greater for each of the iterative methods. The GMRES (5) method failed on the crescent problems, stagnating (ceasing to reduce the error) well before it reached the desired level of accuracy.

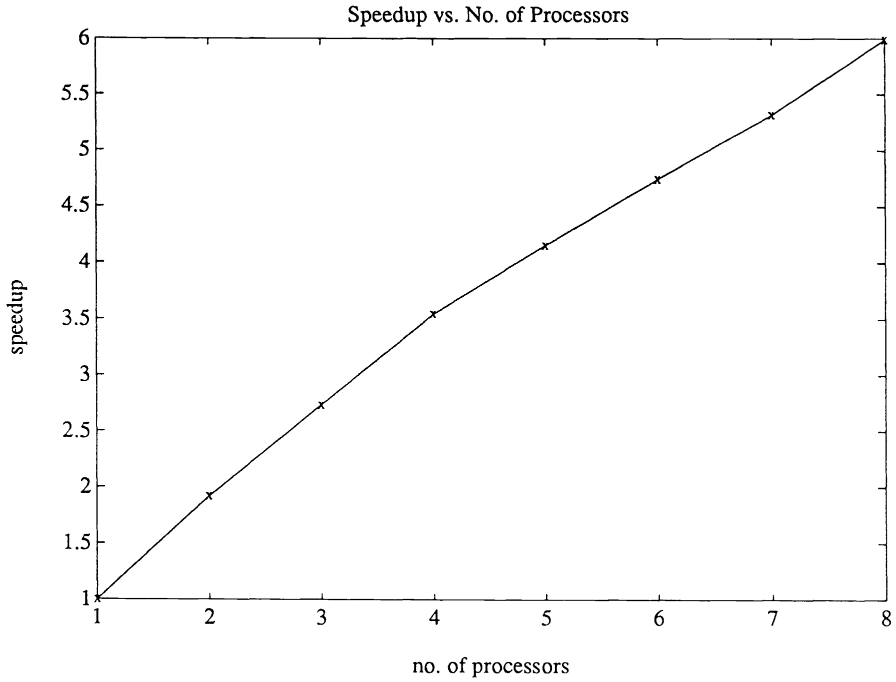
4.2. Parallel and vector implementation of the method. The major part of the work at each iteration of the biconjugate gradient algorithm is in multiplying the matrix and its transpose by an arbitrary vector. This operation is easily vectorized and/or parallelized, as are the other operations (dot products, saxpy's, etc.) required for an iteration. On a machine with both vector and parallel capabilities, different blocks of the matrix can be assigned to different processors and vectorization can be performed over the rows within each block. This can actually be accomplished using a level-2 Blas routine

TABLE 4
Biharmonic equation (10) (*random initial guess*).

Region	BCG	CGS	GMRES (5)	GMRES (40)	CGNR
1. Circle of radius 1	8	7	10	8	11
2. Ellipse with semiaxes 1 and 2	12	9	15	12	17
3. Ellipse with semiaxes 1 and 10	45	35	76	34	92
4. Ellipse with semiaxes 1 and 20	86	41	119	50	188
5. Crescent 1	30	30	>200	24	45
6. Crescent 2	50	64	>200	30	82


x10⁴
Total Time vs. No. of Processors
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
time
1
2
3
4
5
6
7
8
no. of processors

(a)


Speedup vs. No. of Processors
6
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
speedup
1
2
3
4
5
6
7
8
no. of processors

(b)

FIG. 5

[4], and these routines have already been optimized on many parallel/vector supercomputers. Another significant part of the work is in applying the fast Poisson solver after the integral equations have been solved. This is also easily parallelized, with different processors computing FFTs simultaneously, while the individual FFTs may be vectorized.

We implemented the entire algorithm for computing a harmonic function, its conjugate, and its first derivatives on an 8-processor shared memory MIMD machine—the NYU Ultracomputer prototype—and obtained good speedup over the serial code. The region was the first doubly connected region bounded by two ellipses described above. We again placed 200 points on the boundary and used a 33 by 33 rectangular mesh on the rectangle in which the region was embedded.

In this case the biconjugate gradient routine accounted for about 40 per cent of the time required by the serial code. To obtain good speedup of the entire code, it was also necessary to parallelize the other sections of the code—generating the matrix, computing the Laplacian at irregular mesh points, and applying the fast Poisson solver. These sections were also easily parallelized, and the use of a parallel DO loop facility, called DOALL, on the ultracomputer allowed efficient parallelization of the entire code with only minor modifications to the original serial code. DOALL prespawns tasks and assigns loop indices to processors dynamically [1]. Different elements of the matrix were generated in parallel and the Laplacian was computed at irregular mesh points in parallel. The two-dimensional fast Poisson solver consists of one-dimensional FFTs, which were also executed in parallel.

Figure 5 shows the time and speedup on one to eight processors for the above problem. Using eight processors, an overall speedup of about a factor of 6 was obtained, with the biconjugate gradient routine obtaining about a factor of 7 speedup over its time on one processor. The time for the parallel code run on one processor was about 5 percent slower than the time for the serial code. A factor of seven speedup on the ultracomputer appears to be near the best attainable, due to bus traffic.

On the Cray X-MP, we measured the difference between the time required for the matrix generation only, when vectorized and unvectorized. For this section of the code, vectorization resulted in a factor of 9 speedup, and its effect on the other sections appears to be similar.

Acknowledgment. The authors thank the referees for many helpful comments.

REFERENCES

[1] W. BERKE, *ParFOR—A structured environment for parallel Fortran*, Ultracomputer Note 137, Courant Institute, New York, NY, 1988.

[2] B. BUZBEE, F. W. DORR, J. A. GEORGE, AND G. H. GOLUB, *The direct solution of the discrete Poisson equation on irregular regions*, SIAM J. Numer. Anal., 8 (1971), pp. 722-736.

[3] J. CARRIER, L. GREENGARD, AND V. ROKHLIN, *A fast adaptive multipole algorithm for particle simulations*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 669-686.

[4] J. DONGARRA, J. DUCROZ, S. HAMMARLING, AND R. HANSON, *An extended set of Fortran basic linear algebra subprograms*, ACM Trans. Math. Software, 14 (1988), pp. 1-17.

[5] R. FLETCHER, *Conjugate gradient methods for indefinite systems*, in Proc. of the Dundee Biennial Conference on Numerical Analysis, G. A. Watson, ed., Springer-Verlag, New York, 1975.

[6] M. R. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 409-436.

[7] R. W. HOCKNEY, *The potential calculation and some applications*, in Methods in Computational Physics, Vol. 9, Academic Press, New York, 1970, pp. 135-211.

[8] R. JAMES, *The solution of the Poisson equation for isolated source distributions*, J. Comput. Phys., 25 (1977), pp. 71-93.

[9] D. Lindholm, *Notes on boundary integral equations for three-dimensional magnetostatics*, IEEE Trans. Magnetics, 16 (1980), pp. 1409-1417.

[10] A. Mayo, *The fast solution of Poisson's and the biharmonic equations on general regions*, SIAM J. Numer. Anal., 21 (1984) pp. 285-299.

[11] ——— *Rapid, high order accurate evaluation of volume integrals of potential theory*, J. Comput. Phys., submitted.

[12] 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.

[13] S. G. Mikhlin, *Integral Equations and their Applications*, Pergamon Press, New York, 1957.

[14] W. Proskurowski and O. Widlund, *On the numerical solution of Helmholtz's equation by the capacitance matrix method*, Math. Comp., 30 (1976), pp. 433-468.

[15] V. Rokhlin, *Rapid solution of integral equations of classical potential theory*, J. Comput. Phys., 60 (1985), pp. 187-207.

[16] Y. Saad and M. H. Schultz, GMRES: *A generalized minimum residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856-869.

[17] I. Sherman, *The solution of the plane static problem of the theory of elasticity with given external forces*, Dokl. Akad. Nauk SSSR, 28 (1940).

[18] P. Sonneveld, *CGS, a fast Lanczos-type solver for nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 36-52.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 1, pp. 119-127, January 1992

007

COMPACT MULTIGRID*

VICTOR PAN† AND JOHN REIF‡

Abstract. The bit-complexity is a realistic complexity measure for computations on parallel computers such as the CONNECTION MACHINE (CM1) and the MASPAR. For a large class of linear PDEs satisfying some routine assumptions of the multigrid methods, the N point discretization of their solution is compressed to a constant number of bits per discretization point with no loss of information and without introducing errors beyond the order of the discretization error. Namely, it is shown that the bit-complexity of the compressed solution is $O(N)$ for the storage space and, if the PDE has (piecewise) constant coefficients, then also for the total number of bit-parallel operations. The compressed solution is also computed by using time $O(\log N)$ and $N/\log N$ bit-serial processors. The known bounds on the bit-complexity (for both sequential time and storage space) were at least $N \log N$; moreover, the order of $N \log N$ bit-serial processors was required to support the $O(\log N)$ parallel time in the known algorithms. It is believed that this is the first time when the solution to a linear system has been provably compressed (i.e., the bit-complexity of storage of the compressed solution is less than the solution size) and also the first case where the use of data compression provably speeds up the time to solve the system (in the compressed form).

Key words. multigrid, data compression, partial differential equations, low precision computing, algorithms, complexity

AMS(MOS) subject classifications. 65P05, 65F10

1. Introduction.

1.1. Motivation and limitations.

Approximate solution of partial differential equations (PDEs) is usually obtained by means of their discrete approximation and by solving a sparse linear algebraic system of equations. Of course, the amount of the discretization affects the accuracy of the approximation. If N discrete approximation points have been chosen over a regular (two- or three-dimensional) grid, then for a very large and important class of linear PDEs, which we will call *weakly smooth*, the solution to the linear algebraic systems approximates to the solution of the PDE with an error of the order of N^{-g}, for some constant $g > 0$. Such discretization errors have been well studied by numerical analysts from the 1920s [CFL], [BR], [BS], [FW], [SF].

Thus the approximation gives us, at each discretization point, the first $O(\log N)$ bits of the actual value of the solution to the PDE, and the high accuracy solution of two- and three-dimensional PDEs requires in particular that the number of discretization points N grows very large. The computation is often more limited by the storage constraints than by the time constraints (particularly if it is desired to store the solution in the primary storage memory without the use of the much slower secondary storage of the conventional machines). It is therefore important to investigate methods that substantially reduce this storage by compressing the data in the solution. As we will see, this is indeed possible in some important cases and is a surprisingly fundamental

* Received by the editors April 5, 1990; accepted for publication (in revised form) December 19, 1990.

† Computer Science Department, State University of New York, Albany, New York 12222 and Mathematics and Computer Science Department, Lehman College, City University of New York, Bronx, New York 10468. The work of this author was supported by National Science Foundation grant CCR-8805782 and by PSC-CUNY awards 661340, 668541, and 669290.

‡ Department of Computer Science, Duke University, Durham, North Carolina 27706. The work of this author was supported by Air Force contract AFOSR-87-0386, Office of Naval Research contract N00014-87-K-0310, Defense Advanced Research Projects Agency contract N00014-88-K-0458, Office of Naval Research contract DAAL03-88-K-0195, and National Aeronautics and Space Administration/CESDIS subcontract 550-63NAS5-30428URSA.

property of PDEs, which will also lead us to a substantial economization of bit-operations and (in parallel implementation) of bit-serial or nibble-serial processors involved. In this paper we will demonstrate the power of our new techniques in the simplest cases, that is, under the limitations of the linearity of a PDE and of the choice of simple lattice grids for its discretization; furthermore, to save bit-operations and bit-processors, we will assume that the PDE has piecewise constant coefficients and reduces to linear systems that we may solve by using linearly convergent iterations. Moreover, our present paper addresses special purpose hardware, rather than most of the existing computers. We believe that all these limitations will be at least partially relaxed in our further work, and substantial progress will be reported in our next publication.

1.2. The bit-complexity model. Certain sequential machines (such as the CRAY) were specifically designed to solve these large linear systems: their processor has the ability to do sequential arithmetic operations very quickly and also has a relatively large amount of primary storage. For such a machine, the arithmetic complexity model has generally been considered appropriate. However, a machine such as the CRAY is capable of performing a bit-vector operation in one parallel step (for example, it may perform AND, OR, or NOT operations on hundreds of bits), so that the parallel bit-complexity of such machines can also be of interest.

On the other hand, fine grained massively parallel machines (such as the MASPAR and the CONNECTION MACHINE, CM1) have been designed with large numbers of bit-serial or nibble-serial (4-bit-serial) processors (requiring a relatively long time to execute an arithmetic operation) and with a very limited memory, which is generally accessed bit- (or 4-bit-) serially. A complexity model for parallel algorithms must take into account both the bit-serial nature of the processors and the limited memory constraints; in particular, we feel that the parallel complexity of computing on such machines is most reasonably measured by the bit-complexity. In this model, we assume that each memory cell holds only one bit, and each processor can do a single bit-operation per step.

1.3. Previous solutions of PDEs. The solution of the linear algebraic systems approximating PDEs can be computed by means of a number of well-known and now classical algorithms. For example, for a large class of PDEs, we may apply the standard linearly convergent iterative solution algorithms, such as Gauss and Siedel's, and find the solutions to the auxiliary linear systems up to the maximum accuracy of $O(\log N)$ bits at N points in $O(\log N)$ iterations, using N processors. However, each such iteration generally involves arithmetic operations over the $O(\log N)$-bit numbers and hence requires at least $O(\log N)$ bit-operations per point. Thus the total work is $O(N \log N)$ arithmetic operations or $O(N \log N)$ bit-operations.

Various multigrid methods were first proposed by Fedorenko and Bakhvalov in the 1960s, and then by Astrakhantzev and Brandt in the early 1970s for the solution of these linear systems approximating PDEs (see [Ast], [Bak], [Fed], and [Bran2, 3, 4]). In [Bran2], it was claimed that these multigrid methods required only $O(N)$ arithmetic operations; this was rigorously proved for a large class of PDEs in [BD] (also see [Hack1, 2, 3], [HT], [McCor2], and [McT]). (Actually, the multigrid methods are effective even in many cases where the classical iterative algorithms converge too slowly.) The works of [Bran1], [FMc], [CSS], and [McV1, 2] all describe parallel algorithms that take $O(\log N)$ arithmetic steps using N processors, and thus use the order of $N \log N$ arithmetic operations, which is off by the factor of $\log N$ from the optimum. Even if the order of s bit-operations sufficed to add and to multiply two integers modulo 2^s, which is actually a lower estimate, whereas the current record upper bound is only $O(s \log s \log \log s)$ [AHU], it would follow that the known multigrid methods require a total of at least the order of $N \log N$ bit-operations, and at least the order of $N \log N$ bit-serial processors to support the order of $\log N$ time. The bit-operation bound appears to be unbeatable since the binary representation of the solution occupies a total of the order of $N \log N$ bits.

1.4. Our results. Our main goal is a rigorous study of the bit-complexity of these linear algebraic systems approximating linear PDEs. In spite of the lack of theoretical investigation into this area, we feel that the problems are fundamental in nature. In this paper we will show some surprising properties of the linear algebraic systems approximating to linear PDEs under some mild assumptions specified below and in § 2:

(1) The *weakly smooth* solutions to PDEs can be significantly compressed to $O(1)$ bits per solution point (which, by the factor of $\log N$, improves the previous storage requirements), by using a data structure that we call the *Compact Multigrid Data Structure.* (We do not know of any previous provable results for data compression of the solutions to any class of linear systems.)

(2) For a large class of linear PDEs with constant coefficients, their compressed solutions can be very efficiently computed (both sequentially and in parallel) by an algorithm that we also call *Compact Multigrid* and that only uses $O(\log N)$ time, $N/\log N$ bit-serial processors, and a total of $O(N)$ bit-operations, which is optimum since the size of the compressed solution is of the order of N. This improves by the factor of $\log N$ the bounds on both sequential time and storage space of the known algorithms and decreases by the factor of $\log^2 N$ the number of bit-serial processors supporting $O(\log N)$ time.

Note that already the $\log N$ factor is significant for even relatively small problems; for example, this factor is 10 or more for problems of size $N > 1000$, such as the three-dimensional grid of size $10 \times 10 \times 10$.

A bit-serial data communication required by our compact multigrid algorithm happens to be what is known as a pyramid network, consisting of a sequence of grids, $G_0, G_1, \cdots, G_k$, where the grid G_i has about 2^{di} points and where each nonboundary node of the ith grid is connected to $2d$, its neighbors in this grid, and also to the corresponding nodes of the $(i+1)$th and $(i-1)$th grids.

The weak smoothness assumption, sufficient for property (1) to hold, is just the very mild and customary bound $O(1/N^c)$, for a constant c, on the discretization errors (see (2.3) below), but even the assumptions required for the property (2) to hold are still satisfied for a large class of (piecewise) constant coefficient linear PDEs. Specifically, besides the weak smoothness, for a given linear (piecewise) constant coefficient PDE, we essentially need only an iterative algorithm (such as multigrid or SSOR) for solving the auxiliary linear systems over all the grids that use not more than a fixed constant number of steps on each grid in order to decrease the approximation error norms by a fixed constant factor—the same for all the grids. This is in fact a *routine assumption* of the multigrid methods (cf. [McCor1]).

We may compress a given N point uncompressed solution by using $O(\log N)$ time and N bit-serial processors, for a total of $O(N \log N)$ amount of work, which is optimal since the input solution is of size $O(N \log N)$. A very simple decompression algorithm requires only $O(\log N)$ sequential bit-operations to access the full precision (of $O(\log N)$ bits) solution value at any discretization point.

The compressed solution can be stored, and it can be decompressed only when its values need to be output. In many practical applications the solutions need not be decompressed. For example, in the solution of the time-dependent PDEs, the most customary solution methods perform at a discrete sequence of, say, T time steps. In each time step, a PDE is approximately solved, by using an N point discretization of the PDE fixed at that time value and by using the approximate solution obtained (in the compressed form) at the previous time step as an initial approximation to the current solution. Thus the solutions at these time steps need not be decompressed, except for the solution at the final time step. The total bit-complexity estimate for this computation, including decompression of the final solution and T calls for compact multigrid, would be $O(N(T+\log N))$ bit-operations (requiring $O(T \log N+\log^2 N)$ time and using $N/\log N$ bit-serial processors). Here we need, in particular, the linear convergence assumption; if it holds initially, we will preserve it by using sufficiently small time steps.

1.5. Organization of the paper. We will specify our compression techniques for PDEs in §§ 2–5. We will simplify our presentation, assuming, in particular, the simple lattice grids, although our results hold for more general discretization sets. In § 6 we will indicate some further extensions of our results, in particular, to more general discretization sets. We include Appendix A, where we bound interpolation errors in terms of discretization errors.

2. Definitions and assumptions. Let a linear PDE on the unit d-dimensional cube be discretized over a family of d-dimensional grids $G_0, G_1, \cdots, G_k$, having the distance $h_j=2^{-j}$ between each point of G_j and $2d$, its nearest neighbors, so that there are $|G_j|=N_j(1+o(1))$, $N_j=2^{dj}$, points in G_j, for $j=0, 1, \cdots, k$, and $N_k=N=2^{dk}$, where $k=(\log_2 N)/d$. Hereafter, for simplicity, we will ignore the smaller order terms $o(1)$, corresponding to the "extra" boundary points of the grids.

Let $u(x)$, a function in the d-dimensional vector, denote the solution to a PDE, let

$$D_j u_j = b_j, \tag{2.1}$$

denote the linear system of the difference equations generated by the discretization of the PDE over the grid G_j, and let $u_j(x)$ for a fixed $x \in G_j$ denote the respective component of the $|G_j|$-dimensional vector u_j representing the solution to this linear system, so that $\Delta_j(x)=u(x)-u_j(x)$ denote the discretization error functions on G_j, for $x \in G_j$ and $j=1, \cdots, k$.

Surely, discretization of the linear PDE gives matrices D_j with $O(1)$ nonzero coefficients per row as $j \to \infty$. Let $u_0(x)=0$ for $x \in G_0$, and let $\hat{u}_{j-1}(x)$, $j=1, 2, \cdots, k$, denote the prolongation of $u_{j-1}(x)$ from G_{j-1} to G_j, obtained by the interpolation (which usually means just the averaging) of the values of $u_{j-1}(x)$ at the appropriate subarray of points of G_{j-1} lying near x. Then

$$u_j(x)=\hat{u}_{j-1}(x)+e_j(x), \qquad x \in G_j, \qquad j=1, \cdots, k, \tag{2.2}$$

where $e_j(x)$ denotes the interpolation error on G_j.

We will assume that the discretization and interpolation errors satisfy the two following bounds, which we will call *the weak smoothness assumption* for the PDE:

$$|\Delta_j(x)| \leqq 2^{c-\alpha j}, \tag{2.3}$$

$$|e_j(x)| \leqq 2^{c-\alpha j} \tag{2.4}$$

for all $x \in G_j$, $j=1, \cdots, k$, and for fixed $c \geqq 0$ and $\alpha \geqq 1$. In the Appendix, we will deduce (2.4) from (2.3), whereas the assumption (2.3) holds for a wide class of PDEs (see, for instance, [Ame, p. 29], [LP]), including the well-posed linear PDEs, as well as many nonlinear PDEs. Such an assumption is routinely made in the analysis of the multigrid methods (e.g., [Bran2, 3, 4], [BD]); in particular, the auxiliary grid problems are said to be "solved to the level of truncation" defined by (2.3) (see [McCor1, p. 26]).

As a part of the weak smoothness assumption, let us further assume that $u(x)$ has been scaled so that $|u_j(x)| \leqq 1$ for $x \in G_j$ and for all j and that every $e_j(x)$ is represented with (that is, rounded off to) α binary bits (digits).

Remark. We may replace the bounds (2.3), (2.4), and $|u_j(x)| \leqq 1$ by the bounds

$$\|\Delta_j\| \leqq 2^{c-\alpha j} \|u_j\|,$$
$$\|e_j\| \leqq 2^{c-\alpha j} \|u_j\|,$$

for a fixed vector norm, provided that Δ_j, e_j, and u_j are considered as $|G_j|$-dimensional vectors with the components $\Delta_j(x)$, $e_j(x)$, and $u_j(x)$ for $x \in G_j$. This modification would not change our resulting estimates for the complexity of the Compact Multigrid.

In § 4, we will assume (in addition to weak smoothness) *interpolation regularity*, which means that the prolongation from G_{j-1} to G_j only requires $O(1)$ time using N_j bit-serial processors (which is surely the case for the interpolation by averaging).

In § 5, in addition to weak smoothness and interpolation regularity, we will assume the following:

(1) A fixed iterative algorithm (such as SSOR or a multigrid algorithm) applied to linear systems with matrices D_j uses $O(1)$ multiplications of submatrices of D_j by vectors for every j, in order to decrease, by the factor independent of j and N, the norm of the error of the approximation to the solution $u_j(x)$ of the system (2.1) (*linear convergence assumption*).

(2) The entries of the matrices D_j for all j, as well as the components of b_j, are integers having magnitudes $O(1)$ or turn into such integers after the scaling and truncation of the entries of the system (2.1) (*bounded coefficient assumption*, which holds for the constant coefficient PDEs).

Finally, for convenience, we will assume the fixed point binary representation, although shifting to the floating point representation would not actually lead to any substantial changes in our estimates.

3. Compression of the output. In this section, we will assume the weak smoothness relations and will compress approximations to all the values of $u_k(x)$ on G_k within absolute errors of at most $2^{c-\alpha k}$, so as to decrease the storage space required.

For the straightforward fixed point binary representation of these values of $u_k(x)$, we generally need $N\lceil \alpha k - c\rceil$ binary bits.

Alternatively, let us store $u_k(x)$ on G_k in the compressed form by recursively approximating within $2^{c-\alpha j-\alpha}$ to the fixed point binary values $e_j(x)$ for $x \in G_j$, $j = 1, \cdots, k$. The storage space of $2^d\lceil \alpha - c\rceil + \alpha(N_2 + N_3 + \cdots + N_k) < 2^d\lceil \alpha - c\rceil + 2\alpha N = O(N)$ binary bits suffices for this compressed information, which means saving roughly the factor of $0.5k = 0.5 \log_2 N$ binary bits against the straightforward representation.

4. Recovery of the solution values from the compressed data. In this section, we will assume the weak smoothness and the interpolation regularity. To recover $u_k(x)$ on G_k from the compressed information given by $e_j(x)$ on G_j for $j = 1, \cdots, k$, we start with $u_0(x) = 0$ for $x \in G_0$ and recursively, for $j = 1, \cdots, k$, compute the values

(a) $\hat{u}_{j-1}(x)$ on G_j, by prolongation of $u_{j-1}(x)$ from G_{j-1} to G_j, and then

(b) $u_j(x)$ on G_j, by applying (2.2).

Both stages (a) and (b) are performed with precision $2^{c-\alpha j-\alpha}$, so that stage (b) amounts to appending α binary bits of $e_j(x)$ to the available string of binary bits in the fixed point binary representation of $\hat{u}_{j-1}(x)$ for each $x \in G_j$, and stage (a) amounts to scanning the values of $u_{j-1}(x)$ on G_{j-1} and to the summation of few β-bit binary numbers (where, say, $\beta = O(\alpha)$) defined by the β least significant binary bits in the representation of $u_{j-1}(x)$ for appropriate x from G_{j-1}. Since $\sum_j N_j = O(N)$, the computational complexity estimates for stages (a) and (b) stay within the bounds stated in the Introduction.

5. Computing the compressed solution by compact multigrid. In this section, we will use all the assumptions of § 2, that is, the weak smoothness, interpolation regularity, linear convergence, and bounded coefficient assumptions. The time complexity of computing the compressed data structure is dominated by the time required to obtain the solution vectors e_j for the linear systems of equations over G_j for $j = 1, \cdots, k$:

$$D_j e_j = r_j, \tag{5.1}$$

where

$$r_j = b_j - D_j \hat{u}_{j-1}, \tag{5.2}$$

the matrices D_j and the vectors b_j are from the linear systems (2.1), and the vectors e_j and $\hat{u}_{j-1}$ have components $e_j(x)$ and $\hat{u}_{j-1}(x)$ for $x \in G_j$, defined by (2.1) and (2.2), respectively.

We will follow the routine of the V-cycle multigrid methods (cf. [McCor1] and [FMc]), and will recursively evaluate the vectors e_j for $j = 1, \cdots, k$. Initially, we will let $u_0(x) = 0$ for $x \in G_0$, and at stage j, we will successively compute for all $x \in G_j$:

(a) $\hat{u}_{j-1}(x)$ (by prolongation of $u_{j-1}(x)$ from G_{j-1} to G_j);

(b) $r_j(x)$ (by using (5.2));

(c) $e_j(x)$ (by solving the linear system (5.1) "to the level of truncation");

(d) $u_j(x)$ in the compressed form (by using (2.2), as in § 4).

We may then restrict $u_{j+1}(x)$ to $u_j(x)$ for $j = k-1, \ldots, 0$ (as in some customary multigrid algorithms) and then recursively repeat such a V-cycle.

The linear convergence assumption means that the errors of the approximations to $u_j(x)$ decrease by a constant factor independent of j and N when stages (a)-(d) are repeated once for all j, even if only $O(1)$ iteration steps are used at stage (c) for solving linear systems (5.1) for every j. Such convergence results have been proven for the customary multigrid algorithms applied to a wide class of PDEs (see [BD], [Hack2, 3], [HT], [McCor2], [McCT], and [FMc2]).

Let us estimate the time complexity of these computations, dominated by the time needed for solving the linear systems (5.1).

The size $N_j = 2^{dj}$ (within $O(N_j^{1/d})$) of the linear system (5.1) increases by 2^d times as j grows by 1. Even if we assume that the solution time for the system (5.1) is linear in $|G_j|$, the overall solution time for all the k such systems in terms of the number of arithmetic operations involved is less than $1/(1-2^{-d})$ times the solution time for the single system (2.1) for $j = k$, which gives us the uncompressed output values $u_k(x)$ for $x \in G_k$. The bit-operation count is even more favorable to the solution of the systems (5.1) for all j, as opposed to the single system (2.1) for $j = k$, because the output values $e_j(x)$, satisfying the systems (5.1), are sought with the lower precision of α binary bits.

Furthermore, we solve the linear systems (5.1) by iterative methods where each step is essentially reduced to a constant number (say, one or two) multiplications of a matrix D_j or its submatrices by vectors. Due to the linear convergence assumption we made, a constant number of iterations suffices at each step j in order to compute the α desired binary bits of $e_j(x)$.

The computational cost of multiplication of D_j by a vector is $O(N_j)$ arithmetic operations for a sparse and structured discretization matrix D_j (having $O(1)$ nonzero entries in each row). Furthermore, a parallel acceleration to the parallel time bound

$O(1)$ is possible by using N_j processors (for we multiply a vector by a matrix having $O(1)$ nonzero entries per row). Thus we arrive at Proposition 1, whose processor bound follows similarly to the proof of Proposition 2 below.

PROPOSITION 1. *$O(\log N)$ parallel arithmetic steps and $N/\log N$ processors suffice to compute the vectors e_j for all j, that is, to compute the smooth compressed solution to a* PDE *discretized over the grid G_k, under all the assumptions of* § 2.

Furthermore, we only need $O(1)$ binary bits in order to represent $e_j(x)$ for every $x \in G_j$ and every j. Since D_j has only $O(1)$ nonzero entries per row and since these entries are integers having magnitudes $O(1)$ (due to the bounded coefficients assumption), it suffices to use $O(1)$ bits to represent $r_j(x)$ (see (5.1)). (These bits may occupy not all the positions where the corresponding components of b_j have nonzero bits, for the most significant binary digits of these components may be canceled in subtracting $D_j\hat{u}_{j-1}$.) Thus, we will perform all the arithmetic operations with $O(1)$-bit operands and will arrive at Proposition 2.

PROPOSITION 2. *Under all the assumptions of* § 2, *$O(\log N)$ steps, $N/\log N$ bit-serial processors, and $O(N)$ storage space under the Boolean model of computation suffice in order to compute the compressed solution to a* PDE *by using the Compact Multigrid algorithm.*

Proof. As described above, our algorithm has stages $j=1,\cdots,k=(\log N)/d$, where at stage j we require $O(1)$ time for each of the $N_j=2^{dj}$ bit-serial processors. Thus our parallel algorithm (if naively implemented) appears to take $O(\log N)$ time using N bit-serial processors. However, the first $(\log N)/d$-$\log\log N$ stages only require $N/\log N$ bit-serial processors. Thus, at each of the last $\log\log N$ stages $j, j=(\log N)/d$-$\log\log N+1,\cdots,(\log N)/d$, we will slow down the computation to the time $O(2^{dj}\log N/N)$, and then we only need $N/\log N$ bit-serial processors. The overall time of our resulting parallel algorithm is then still $O(\log N)$.

6. Extensions of the results. Our results can be immediately extended to the case of more general sequences of the sets $S_0, S_1, \cdots, S_k$ of the discretization of the PDEs, provided that each set S_j consists of $c_j\sigma^j$ points where $0<c<c_j<c^*$, $\sigma>1$, c, c^*, and σ are constants (this includes the grids with step sizes that vary depending on the direction of the steps), and that the required assumptions of § 2 are, respectively, extended to the case of the sets S_j. We also need to assume that each discretization point of S_j has at most $O(d)$ neighbors: this will imply that each equation of the associated linear algebraic system has at most $O(d)$ nonzero coefficients.

Finally, the presented approach can be further extended to some nonlinear PDEs, as long as our assumptions (such as (2.3) and (2.4)) hold and as long as dealing with nonlinear systems of difference equations replacing the linear systems (2.1) remains relatively inexpensive.

Appendix A. Let us show that the bound (2.3) implies the bound (2.4) assuming that $\hat{u}_{j-1}(x_{j-1})=u_{j-1}(x_{j-1})$ for $x_{j-1}\in G_{j-1}$ and that $\hat{u}_{j-1}(x_j)$ has been defined on G_j-G_{j-1}, say, as the average value $u_{j-1}(x)$ at all the points x_{j-1} of G_{j-1} such that $|x_{j-1}-x_j|=gh_j$, for the minimum g. Then $|e_j(x_j)|\leqq|u_j(x_j)-u_{j-1}(x)|$ for at least one of these points. First rewrite the bound (2.3) as follows:

$$|\Delta_j(x_j)|\leqq ah_j^\gamma,\qquad x_j\in G_j, \tag{A.1}$$

where a and γ are positive constants, and h_j is the length of a side of the mesh G_j, so that $h_{j-1}=2h_j$ for the lattices G_j that we have chosen. Let $\gamma<1$, $x_{j-1}\in G_{j-1}$, $x_j\in G_j$, $|x_j-x_{j-1}|=h_j$, and $|e_j(x_j)|\leqq|u_j(x_j)-u_{j-1}(x_{j-1})|$. Then deduce from (A.1) that for some Θ, $0\leqq\Theta\leqq1$, $|e_j(x_j)|\leqq|u_j(x_j)-u_{j-1}(x_{j-1})|\leqq|u(x_j)-u_j(x_j)|+|u(x_{j-1})-u_{j-1}(x_{j-1})|+$

$|u(x_j)-u(x_{j-1})|\leqq a^*h_j^\gamma$, that is,

$$|e_j(x_j)|\leqq a^*h_j^\gamma, \tag{A.2}$$

$a^*=a(1+(h_{j-1}/h_j)^\gamma)+g|u'(x_j+\Theta h_j)|h_j^{1-\gamma}\leqq(1+2^\gamma)a+g\max|u'(x)|h_j^{1-\gamma}$. The bounds (A.1) and (A.2) turn into the bounds (2.3) and (2.4) for $c=\log_2 a^*$, $\alpha=-\gamma(\log_2 h_j)/j$, and $x=x_j$. If $x_j\in G_{j-1}$, then we just replace x_{j-1} by x_j above and cancel the term $|u(x_j)-u(x_{j-1})|$.

Acknowledgments. The authors thank the referee and the editor for helpful comments.

REFERENCES

[AHU] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1976.

[Ame] W. F. Ames, *Numerical Methods for Partial Differential Equations*, Academic Press, New York, 1977.

[Ast] G. P. Astrakhantzev, *An iterative method of solving elliptic net problem*, Z. Vychisl. Mat. i Mat. Fiz., 11 (1971), pp. 439-448. (In Russian.)

[Bak] N. S. Bakhvalov, *On the convergence of a relaxation method under natural constraints on an elliptic operator*, Zh. Vychisl. Mat. i Mat. Fiz., 6 (1966), pp. 861-883. (In Russian.)

[BR] R. Bank and D. J. Rose, *Analysis of a multilevel iterative method for nonlinear finite-element equations*, Math. Comp., 39 (1982), pp. 453-465.

[BD] R. Bank and T. Dupont, *An optimal order process for solving finite element equations*, Math. Comp., 36 (1981), pp. 35-51.

[BS] R. Bank and A. Sherman, *Algorithmic aspects of the multi-level solution of finite element equations*, in Sparse Matrix Proceedings, 1978, I. S. Duff and G. W. Stewart, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979.

[Bran1] A. Brandt, *Multi-grid solvers on parallel computers*, ICASE Technical Report 80-23, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, VA, 1980.

[Bran2] ———, *Multi-level adaptive technique* (MLAT) *for fast numerical solutions to boundary value problems*, in Proc. Third Internat. Conference on Numerical Methods in Fluid Mechanics, Paris, France, 1972; Lecture Notes in Physics 18, Springer-Verlag, Berlin, Germany, 1984, pp. 82-89.

[Bran3] ———, *Multi-level adaptive solutions to boundary value problems*, Math. Comp., 31 (1977), pp. 333-390.

[Bran4] ———, *Multigrid Techniques*: 1984 *Guide, with Applications to Fluid Dynamics*, available as Gesellschaft für Mathematik und Datenverardeitung (GMD) Studien Nr. 85, GMD-AIW, St. Augustin 1, Germany, 1984.

[CSS] T. F. Chan, Y. Saad, and M. H. Schultz, *Solving elliptic partial differential equations on hypercubes*, Hypercube Multiprocessors 1986, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1986.

[CFL] R. Courant, K. O. Friedrichs, and H. Lewy, *Uber die partiellen Differenzengleich-ungen der mathematischen Physik*, Math. Ann., 100 (1928), pp. 32-74.

[Fed] R. P. Fedorenko, *The speed of convergence of one iteration process*, Z. Vychisl. Mat. i Mat. Fiz., 4 (1964), pp. 559-663. (In Russian).

[FW] G. E. Forsythe and W. R. Wasow, *Finite Difference Methods for Partial Differential Equations*, John Wiley, New York, 1960.

[FMc1] P. O. Frederickson and O. A. McBryan, *Parallel superconvergent multigrid*, in Multigrid Methods: Theory, Applications and Supercomputing, S. McCormick, ed., Lecture Notes in Pure and Applied Math., Vol. 100, Marcel Dekker, New York, 1988, pp. 195-210.

[FMc2] ———, *Superconvergent multigrid methods*, preprint, Cornell Theory Center, Cornell University, Ithaca, NY, May 1987.

[Hack1] W. Hackbusch, *Convergence of multi-grid iterations applied to difference equations*, Math. Comp., 34 (1980), pp. 425-440.

[Hack2] ———, *Multigrid Methods and Applications*, Springer-Verlag, Berlin, New York, 1985.

[Hack3] ———, *On the convergence of multi-grid iteration applied to finite element equations*, Report 77-8, Mathematics Department, Universität zu Köln, Köln, Germany, July 1977.

[HT] W. Hackbusch and U. Trottenberg, eds., *Multigrid Methods*, Lecture Notes in Mathematics 960, Springer-Verlag, Berlin, New York, 1982.

[LP] L. Lapidus and G. F. Pinder, *Numerical Solution of Partial Differential Equations in Science and Engineering*, John Wiley, New York, 1982.

[McV1] O. A. McBryan and E. Van de Velde, *Parallel algorithms for elliptic equations*, Comm. Pure Appl. Math., 38 (1985), pp. 769–795.

[McV2] ———, *The multigrid method on parallel processors*, in Multigrid Methods II, W. Hackbusch and U. Trottenberg, eds., Lecture Notes in Mathematics 1228, Springer-Verlag, Berlin, New York, 1986.

[McCor1] S. McCormick, ed., *Multigrid Methods*, Frontiers in Mathematics 3, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

[McCor2] ———, *Proceedings of the Second Copper Mountain Multigrid Conference*, Appl. Math. Comp., 19 (1986), pp. 1–372 (special issue).

[McCT] S. McCormick and U. Trottenberg, eds., *Multigrid methods*, Appl. Math. Comp., 13 (1983), pp. 213–474 (special issue).

[SF] G. Strang and G. Fix, *An Analysis of the Finite Element Method*, Prentice-Hall, Englewood Cliffs, NJ, 1973.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 1, pp. 128-145, January 1992 008

PARALLEL PERFORMANCE OF DOMAIN-DECOMPOSED PRECONDITIONED KRYLOV METHODS FOR PDEs WITH LOCALLY UNIFORM REFINEMENT*

WILLIAM D. GROPP† AND DAVID E. KEYES‡

Abstract. Preconditioners based on domain decomposition appear natural for the Krylov solution of implicitly discretized partial differential equations (PDEs) on parallel computers. Two-scale preconditioners (involving a global coarse-grid solve, independent solves over interfaces connecting the coarse-grid points, and independent subdomain solves) have been known since the early 1980s to be "near optimal" in the sense of ensuring a bounded, or at most logarithmically growing, iteration count as the mesh is refined. As a result, the refinement of the mesh can be chosen locally on the basis of truncation error, and the granularity of the domain decomposition can be chosen globally on the basis of parallel computing considerations with only mild effects on the convergence rate of the algorithm. However, overall computational complexity depends not only on the algebraic convergence rate, but also on the operation counts of the components of the preconditioner that must be applied at each iteration. The costs of solving the subdomain systems and the crosspoint system show superlinear growth in their respective (and inversely related) sizes. On the subdomains, the superlinear terms arise from arithmetic only; in the crosspoint system the cost of nonlocal data exchange is also superlinear. These factors make the preconditioner granularity and the choice of its components problem- and machine-dependent compromises.

The tradeoffs involved are illustrated through numerical experiments on both shared- and distributed-memory computers for convection-diffusion problems. Because of the development of boundary layers, these problems benefit from local mesh refinement, which is straightforward to accommodate within the domain decomposition framework in a locally uniform sense, but which introduces load balancing as a further consideration in selecting the granularity of the preconditioner. In spite of the tradeoffs, cumulative speedups are obtainable out to at least medium-scale granularity (up to 64 processors in our tests). The largest problems involve $\mathcal{O}(10^5)$ unknowns partitioned into $\mathcal{O}(10^3)$ subdomains and converge in $\mathcal{O}(10)$ iterations requiring $\mathcal{O}(1)$ seconds on the Intel iPSC/860.

Key words. domain decomposition, elliptic problems, parallel algorithms, mesh refinement

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

1. Introduction.

Several tributaries of applied mathematics and computer science meet at large-scale PDE-based scientific computing applications on parallel computers, where feasibility and efficiency rank alongside existence, uniqueness, and conditioning as essential considerations.

Iterative methods based on choosing the best solution in incrementally expandable subspaces have proved effective in many contexts and have been generalized to nonsymmetric operators in such archetypal forms as conjugate gradient squared (CGS) and generalized minimal residual (GMRES). For computational economy, these methods allow different representations of the underlying operator to be used, ultimately converging to a "high quality" representation, through a series of applications of the inverse of a "lower quality" representation, called a preconditioner. Though

* Received by the editors October 4, 1990; accepted for publication (in revised form) April 20, 1991. This work is a revision and update of a presentation at the Conference on Iterative Methods, Copper Mountain, Colorado, April 1-5, 1990, previously co-issued as Argonne National Laboratory preprint MCS-P147-0490 and Yale University Department of Computer Science Research Report 773, April 1990.

† Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois 60439. The work of this author was supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy contract W-31-109-Eng-38, by Office of Naval Research contract N00014-86-K-0310, and by National Science Foundation contract DCR 8521451.

‡ Department of Mechanical Engineering, Yale University, New Haven, Connecticut 06520. The work of this author was supported by National Science Foundation contracts EET-8707109 and ECS-8957475, by the IBM Corporation, and by the 3M Company.

already advantageous in linear problems and on serial computers, the ability to operate with multiple representations of the operator proves even more significant in nonlinear problems and in parallel.

Adaptive discretizations determine the feasibility boundary for many scientific computing applications. Unfortunately, the irregularity of adaptivity can complicate the management of parallel resources, a situation that has led to various compromises of adaptivity with quasi uniformity. Generally, the use of highly structured discretizations in a modular way at different scales has been favored.

Distributed computation puts a premium on the exchange of information between cooperating processors, since such operations are generally slower than local operations. The fact that nearby points usually interact more strongly than distant points motivates decomposition by domain and allows for a "nearly" local algorithm. Nevertheless, the physics intrinsic to many PDE-based problems, expressible in the form of Green's functions with global support, requires global sharing of information. Function-space decompositions based on coarsening processes, which propagate information on multiple scales simultaneously and even asynchronously, have evolved as another active research tributary.

This contribution does not push the frontier of any particular one of the methodologies mentioned above, but lies in their intersection. An implementation in software of a particular domain-decomposed locally uniform mesh refinement (LUMR) GMRES method with a two-scale preconditioner is offered as an effective prototype, which brings various tradeoffs to light. These include adaptivity and convergence rate, adaptivity and truncation error, and adaptivity and parallel load-balance. Except for details at the highest and lowest levels, the same code has been successfully ported from its serial incarnation to both shared- and distributed-memory computers with little sacrifice on any one machine, relative to a custom implementation for that machine alone. Our performance results on these machines allow comparisons with other parallel PDE algorithms, not based on a priori domain decomposition.

The mathematical framework of our domain-decomposed linear solver is described in § 2, followed by a discussion in § 3 of the shared and distributed parallel implementations. Section 4 contains selected performance measurements in the form of tabular cross-sections of a rather large parameter space. Finally, in § 5 we draw some conclusions and comment on bottlenecks and desired extensions.

2. GMRES, LUMR, and domain decomposition.

The generalized minimal residual method [21] is the most economical of the Krylov algorithms to which it is equivalent in the absence of round-off (including GCR [8] and ORTHODIR [25]) and is proposed for retention in [20] (along with CGNR [12] and CGS [23]) in any systematic comparative study of the performance of nonsymmetric Krylov methods. For the purpose of this study it is sufficient to recall the following salient features of GMRES, when used to solve $Ax=b$ in the right-preconditioned form $(AB^{-1})y=b$, $Bx=y$: (1) its implementation requires the ability to evaluate the action of the preconditioned operator (AB^{-1}), but not A^{-1}; (2) it converges monotonically in a number of iterations related to the number of separated clusters of eigenvalues in the spectrum (or the pseudo-spectrum; see, e.g., [24]) of the preconditioned operator; (3) its work and storage estimates are quadratic and linear in iteration count, respectively, because of the construction and retention of an orthonormalized basis for the Krylov subspace based on AB^{-1}; and (4) storage estimates and (sometimes) work estimates can be decreased for many problems by a restarted version of GMRES, which periodically accumulates the solution updates, discards the Krylov subspace, and starts anew.

The application of A is usually an easily parallelized operation, requiring only local communication for local discretization schemes (finite differences, finite elements, finite volumes, etc.). Good choices for B^{-1}, in the sense of producing the clustering mentioned in (2) above, will generally involve some nonlocal communication, but less than in parallel direct algorithms for forming the action of A^{-1}. Domain decomposition is one means of deriving efficient parallel preconditioners for elliptic operators, as well as other operator types.

Locally uniform mesh refinement is a means of resolving multiple spatial scales in PDE problems based on local discretizations. LUMR lies in between the extremes of excess mesh points/low overhead per point, on the one hand, and minimum mesh points/high overhead per point, on the other. It is motivated by the observation that threshold levels of certain continuous function(al)s are often good indicators for enhanced resolution requirements. Hence, adaptively inserted mesh points can be allocated in *quanta* of a convenient size and structure without a serious loss of efficiency through overresolution. In this paper, we assume that the refinement requirements are specifiable a priori, but this is not an inherent limitation of the technique. Serial demonstrations of the LUMR concept have been provided by many investigators; see, e.g., [1], [11] and references therein. In our implementation for a parallel port we are somewhat more restrictive than is serially natural in basing our quanta on initial coarse-grid subdivisions called "tiles." Tiles have standard data interfaces with each other but may support different discretizations and preconditioner modules internally. They are software analogs of the geometry defining processors (GDPs) of [5] and lead to a code that is object-oriented in concept, but not competely so in implementation. By requiring that corners of tiles meet other tiles at corners only (and thus that tiles are elements of a logically tensor product structure) we greatly simplify the portion of the code that performs interdomain information exchange, with dividends for parallel efficiency. For standardized "docking," tiles are tensor products of half-open intervals. On average, in two dimensions, each tile "owns" one of its corners, two of its sides, and all of its interior. At physical boundaries, a tile may "own" additional pieces of its border.

The term "domain decomposition" spans many approaches based on a geometric partitioning of the domain of a PDE, which thus also impose a partitioning on the matrix representations of A and B^{-1}. In this contribution, we discuss nonoverlapping partitionings of a two-dimensional region into subdomain interiors, subdomain interfaces, and crosspoints at interface intersections. A sample decomposition of an L-shaped region into many square subdomains is pictured in Fig. 1.

We write A as

$$A=\begin{pmatrix} A_I & A_{IB} & 0 \\ A_{BI} & A_B & A_{BC} \\ 0 & A_{CB} & A_C \end{pmatrix},$$

where the submatrices arise from a simple permutation of a naturally ordered local discrete operator as follows. A_I is a block diagonal matrix representing the discretization of all of the independent subdomain interior problems, also including physical boundary points other than crosspoints. (The boundary points are retained as degrees of freedom in order to accommodate general boundary conditions, including spatial coupling and also intercomponent coupling in multicomponent problems.) In a standard five-point finite difference (FD) discretization of a single two-dimensional PDE, for instance, the blocks of A_I (one for each domain) are each smaller five-point operators with the discrete subdomain diameter as the maximum bandwidth. A_B is a block diagonal matrix representing the discrete coupling *along* the artificial internal boundaries induced by the decomposition. In our five-point FD examples, each block of A_B is tridiagonal. The crosspoint matrix A_C is purely diagonal in the case of a local discretization of a single PDE. (This is based on the design assumption that the subdomains are too large to be completely spanned by a single discretization stencil, whatever its order.) The doubly subscripted blocks of A contain the coupling between the respective different categories of points. When neighboring subdomains are discretized at the same resolution, the appropriate coefficients are unambiguous. When resolution changes, the finite element method remains unambiguous, and a version of the finite volume method has been likewise rendered in [18], but special provisions are required for FD discretizations. In this paper, we use a biquadratic interpolation scheme (consistent with the best accuracy to be expected from the use of a five-point FD discretization) in the subdomain interiors, when constructing a fine stencil with data requirements from a coarse region. When constructing a coarse stencil with data requirements from a fine region, we employ unweighted injection. This is a consistent discretization, but not a conservative one.

As a preconditioner, we consider

$$B=\begin{pmatrix} \tilde{A}_I & \tilde{A}_{IB} & 0 \\ 0 & B_B & \tilde{A}_{BC} \\ 0 & 0 & B_C \end{pmatrix},$$

where the tilde quantities are related to the correspondingly subscripted blocks of A according to a variety of prescriptions. A key property of B is its block triangularity, which represents a compromise between a perfectly parallel block diagonality on one hand and the structurally symmetric form of A itself on the other. The formal inverse of B is, of course, also block triangular. The block B_C, solved first, is based on a coarse-grid discretization of the original PDE. This step requires the exchange of data between tiles. The resulting solution provides Dirichlet endpoint conditions, through $\tilde{A}_{BC}$ $(=A_{BC}$ in our examples), for the solution of the independent interface blocks which together constitute B_B. We use for B_B a discretization of the tangential part of the operator, defined as those terms that remain when the derivatives in directions normal to the interface in the local coordinate system are set to zero. Dense but FFT-implementable interfacial blocks have also been incorporated into B_B [3]. Finally, the blocks of $\tilde{A}_I$ are solved independently by using Dirichlet data at all artificial boundary points, through $\tilde{A}_{IB}$ $(=A_{IB})$, and actual boundary conditions at physical boundary points. In this paper, $\tilde{A}_I=A_I$, although other less expensive approximate replacements (such as fast solvers) can (and often should) be employed. Our choices for the components of B lead to good convergence results but are not the most convenient ones for convergence proofs of Krylov iteration based on AB^{-1}. A proof of near-optimality has been given [3] for the case where the blocks of B_B consist of the square root of the corresponding one-dimensional discrete Laplacian operator, but there is no known proof for the case employed herein, where the blocks of B_B derive directly from the tangential terms of the overall operator.

A discussion of the utility of the block triangular form of B (as opposed to a structurally symmetric B) in the case where A is already nonsymmetric is given in [4], and a detailed discussion of the preconditioner of this paper is given in [11]. In the opposite direction, the alternative of a purely block diagonal B is discussed in [10] and [15] and references therein. It may often, but not always, be dismissed as requiring too many extra iterations to justify the savings per iteration. Multicomponent problems in which source terms dominate provide examples in which a block diagonal preconditioning can outperform the coupled alternatives in overall wall-clock time.

3. Shared- and distributed-memory implementations. The tile-based decomposition described above is motivated partly by its yield of a suitably convergent preconditioned iterative method and partly by its control of overhead in a LUMR context, but our main interest in this contribution is its straightforward adaptation to parallel processing. There are a variety of ways to find independent threads of computation of commensurate size in the tile-based decomposition described above. Indivisible tiles can be parceled out over an MIMD processor array, or a collection of identical tiles can be broken up over a vector or SIMD processor array, to be operated on in lockstep. In between these extremes, an MIMD array can be subpartitioned into SIMD- or vector-like clusters and different sets of identical tiles allocated to each cluster for lockstep processing within each. Thus, a workload consisting of a large number of tiles in at most a moderate number of distinct sizes maps well onto a variety of parallel machines and hybridizes well on a hierarchical MIMD/SIMD machine. Our implementations to date are confined to the first type: tiles are not divided over processors, but processors may be responsible for more than one tile each. The number of processors in simultaneous use is therefore bounded below by the number of tiles, but is otherwise independent of it. In our current implementation, it is determined towards the outset of execution and held fixed throughout.

In the serial implementation, work arrays for the data structures associated with each tile are allocated separately, and the tiles are placed on a list that is traversed in an arbitrary order each time a grid-oriented operation (such as forming the action of A or B^{-1}, performing a DAXPY, or taking a residual norm) is required. A highly modular approach is adopted in which each tile has its own local coordinate system and its own subroutines for problem definition, preprocessing, application of an iteration step, exchange of data with neighbors, and (as a consequence of all this flexibility) its own workload estimates for each major iterative phase. The most complex of these phases is usually B^{-1}. In the examples of this paper, the only aspect of this modularity that we exploit nontrivially is the data exchange, in which different interface handling routines mediate the changes in mesh refinement across subdomain boundaries. However, we anticipate applying the same code to nonlinear problems such as reacting or high Reynolds number flows, in which every aspect of this modularity can purchase a significant work advantage. For instance, fast or frozen kinetics, or Euler or Navier–Stokes fluid mechanics, could be adaptively selected on each subdomain.

Code development on a serial workstation offers the obvious advantages of uncontested resources, high graphical display bandwidth, and ease of tracebacks in debugging. Furthermore, it allows the performance testing of algorithmic options apart from communication considerations. We mention that iteration-by-iteration contour plots of solution, error (when knowable), and residual were instrumental in shortening development time.

3.1. Shared-memory implementation. The port to a shared-memory machine (the Encore Multimax 320, equipped with 18 processors) required attention to three issues beyond the serial version: data access, load balance, and work-to-processor mapping. The Multimax offers its global address space to each processor; thus, when the traversal of the single sequential tile list is broken up into several simultaneous traversals of smaller tile lists, access to shared data must be controlled by locks and barriers. In anticipation of a further distributed-memory port, and in keeping with a structured, minimal side-effect programming paradigm, we restricted the use of data sharing to synchronization variables, accumulation registers (for inner products or max/min reductions), work arrays for the global crosspoint system, and data buffers at the artificial interfaces of the decomposition. A list of all tiles allowing neighbor inference is also shared.

In an earlier shared-memory code series for stripwise domain decompositions of tensor-product grids (see, e.g., [14], [15]), we shared substantially more data, including entire unknown fields. In earlier codes, data belonging to neighboring tiles was accessed by simply supplying an absolute index into the global array. In the current code, a buffer is maintained around the perimeter of each tile of a width corresponding to the semibandwidth of the difference stencil in use of that tile. These buffers are refreshed (by injection or interpolation) at appropriate synchronization points. The miserliness of the current code in allowing shared access only to interfacial buffers obviously incurs some unproductive time and space overhead, relative to the earlier versions. It is therefore natural to question whether this is an optimal shared-memory code or just a distributed-memory code implemented on a shared-memory processor, and to assess the impact of the extra overhead on parallel performance.

To quantify the cost of this overhead, we repeatedly executed on the Multimax a loop consisting of a MATVEC (an application of A to a vector), a vector DAXPY, and a vector inner product under both globally shared and buffered versions of the stripped-down code. A five-point FD stencil with hard-coded coefficients was used for A. This is a very conservative comparison, from the point of view of establishing the validity of the buffered version as a·reasonable shared-memory implementation for domain-decomposed elliptic PDE problems, because it includes all but one of the required forms of nonlocal data access per iterative cycle, together with the minimum amount of local arithmetic per access. Almost any mix of operations in a typical preconditioned iteration, except for the solution of the crosspoint system (considered separately below), will enjoy a better computation to "communication" ratio than these loops. Nevertheless, the penalty in wall-clock time for the buffered version was less than 10 percent of the globally shared version. In our judgment, this is a small penalty for the convenience of a code that is readily ported to distributed-memory configurations. Of course, this is a machine-dependent conclusion. However, we note further that many current and anticipated "shared-memory" multiprocessor designs have memory hierarchies that put an execution efficiency premium on global sharing. Fast global shared memory is destined to be a scarce resource on loaded multiprocessors. The Multimax 320 enjoys the evanescent luxury of a bus that is very fast relative to the data-sharing requirements of 16 processors cooperating on an elliptic PDE.

Banded Gaussian elimination, used in the inversion of B_C, does not map as gracefully onto multiprocessors as do the rest of the algorithmic modules. Its parallel performance (both factorization and backsolve) has been evaluated separately from the basic loops above for a five-point stencil Dirichlet problem on the Multimax. The problem sizes varied from 16 (which corresponds to the crosspoint system of a 3×3 grid of tiles) to 4,096 (a 63×63 grid of tiles), and the processing force from 1 to 16. For problem sizes up to 1,024 (bandwidth 32) there is speed*down* at 16 processors, and the case of dimension 4,096 is only 45 percent efficient at 16 processors. For our PDE examples (which employ a maximum of 1,024 tiles), we therefore factor and solve the crosspoint system with single-threaded code, letting other processors remain idle. For a sufficiently high tile-to-processor ratio, it will become more important to parallelize the crosspoint solve itself.

Our load balancing strategy is a primitive and static one that relies on asymptotic order estimates for the largest complexity terms in both the preprocessing and iteration phases of the computation. To be specific, to an $n \times n$ tile we assign the work estimate

$$W = n^2(I^2 + I(1+n) + n^2),$$

where I is an estimate for the number of (nonrestarted) Krylov iterations that will be necessary for convergence. The term in I^2 represents the cumulative orthogonalization work of GMRES; the two terms in I the applications of A and B^{-1}, respectively; and the remaining n^4 term the factorization preprocessing of the subdomain interior matrices of $\tilde{A}_I$ when a bandsolver is employed. Though this primitive estimate contains asymptotic leading terms only, for reasons of granularity we rarely work with asymptotically large tiles and would switch to sparse subdomain solvers if we did. In [11] it is shown, however, that a bandsolver outperforms a sparse direct solver based on nested dissection for sufficiently small n, such as 8 or 16.

Though I can be estimated more carefully on the basis of experience (§ 4), we simply set it to 20 in all examples herein. There is obviously considerable room for refinement of the per-tile work estimate on the basis of a priori or evolving information about the tile and the overall problem. For sufficiently large tile-to-processor ratios, very balanced load distributions become possible in spite of the tile-based quantization of work, and greater effort in work estimation pays dividends.

The question of work-to-processor mapping goes beyond load balancing in the sense that the objective of load balancing is finding simultaneous executable threads of execution of comparable durations, while mapping is further concerned with the data requirements of these threads in relation to the network and memory hierarchies of the machine. From the viewpoint of overall performance, these considerations are rarely independent, of course. We chose a fixed allocation of subdomains to processors based on either of two strategies. The first is tied to the work estimates of the preceding paragraphs. Tiles are ranked in order of decreasing W and parceled out to processors without regard to spatial proximity. While tiles remain, the tile with the largest W is assigned to the processor with the least amount of work, with ties broken by a fixed but arbitrary enumeration ordering.

The second strategy, called "wrap mapping," is based on the heuristic argument of § 2 that led initially to LUMR, namely, that regions requiring refinement tend to occur in contiguous clumps in elliptic PDE discretizations. Therefore, the assignment of tiles based on some regular lexicographic labeling is likely to be as good as a work estimate-based assignment in the limit of large tile-to-processor ratios. The two strategies are compared in § 4.2. Obviously, the ability to dynamically remap will be important to either mapping strategy once dynamic levels of refinement are implemented and should be pursued at the next level of coding sophistication.

3.2. Distributed-memory implementation. The tile-based domain decomposition code has also been tested on the Intel iPSC/860 in a 64-processor hypercube configuration with 8 Mb of memory per node. The port from shared to distributed memory was relatively straightforward because of the miserly use of shared memory in the Multimax version. The sharing of data is accomplished by explicit calls to message-passing subroutines. Since we make no assumptions about the locality in the processor domain of tiles sharing data, all intertile messages are routed via calls to the send/receive ethernet subroutines. For small numbers of processors, and hence strong locality of cooperating tiles, this approach is clearly suboptimal. However, our design is motivated primarily for extension to large numbers of processors, where going off-board is a reasonable default.

The distinctions, rooted in hardware, between "shared" and "distributed" memory are blurred by software. These should not be regarded as distinct opposites, but as different extremes on a continuum of memory hierarchies, with various cached memory designs as intermediates. At one end is the idealized model of a flat time cost for accessing different pieces of data, and at the other, a (usually nonlinear) time cost variation based on distance and packet size. The synchronized tile-interface buffering discussed above is well suited for the distributed-memory model. In order to minimize redundant storage, data areas used exclusively in a small number of processors should not be replicated on all processors. This procedure leads to a custom paring on each processor of the master tile data structure to the minimum size required for neighbor identification and cooperation in the global crosspoint solves and inner products. Without such storage optimizations, distributed-memory processors waste significant memory.

On the iPSC it is even less inviting than on the Multimax to perform the crosspoint solve in parallel. Experiments (see, e.g., [9], since updated in unpublished form for new hardware) show that a problem of size 1,024 (bandwidth 32) can be speeded up slightly by parallelization on up to 32 processors, but at very small efficiency. On the other hand, the single-threaded execution used on the Multimax is also unattractive, since it would have to be preceded and followed by significant global communication to gather the right-hand side and scatter the solution. Our compromise, which dates back to our earliest hypercube domain decomposition code series [13], is to broadcast the data required for the right-hand side to each processor, then to solve the crosspoint system redundantly on each processor. A future optimization for large numbers of tiles on either shared- or distributed-memory computers is cooperative solution of the crosspoint system within subclusters of processors. The crosspoint problem is sufficiently different from the balance of the code in its computation-to-communication ratio that either extreme of employing one or all processors in it is too restrictive. Like the coarse-grid solution in multigrid, it can be regarded as a black box which, though extremely important, is somewhat detachable. Many parallel algorithm designers concern themselves with this black box, and good solutions in the point-per-processor regime are readily incorporated.

4. Performance on model problems. We select for parallel study two-dimensional model problems from the suite of ten studied in serial in [11]. The examples are obtained by taking three different values of the convection, namely, $c=0$, $c=-1$, and $c=10$, in the cylindrically separable reentrant corner convection-diffusion problem:

$$-\nabla^2 u+\frac{c}{r}\frac{\partial u}{\partial r}=0,$$

$$u(x,y)=r^{\alpha}\sin\left(\frac{2}{3}\left(\theta-\frac{\pi}{2}\right)\right),$$

$$\text{where}\quad r=\sqrt{(x-1)^2+(y-1)^2}\quad\text{and}$$

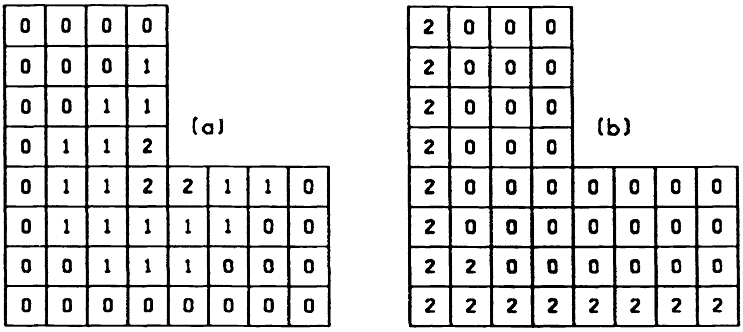
$$\theta=\arg\left((x-1)+i(y-1)\right),\qquad 0\leqq\theta<2\pi,$$

$$\text{Dirichlet data on } \partial\Omega,$$

$$\Omega=L\text{-shaped region.}$$

The domain Ω is shown in Fig. 1.

The first of these corresponds to pure diffusion, and the second and third correspond to convection towards the reentrant corner and away from it, respectively, at a



(a)

0	0	0	0				
0	0	0	1				
0	0	1	1				
0	1	1	2				
0	1	1	2	2	1	1	0
0	1	1	1	1	1	0	0
0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	0

(b)

2	0	0	0				
2	0	0	0				
2	0	0	0				
2	0	0	0				
2	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
2	2	0	0	0	0	0	0
2	2	2	2	2	2	2	2

FIG. 1. *The domain* Ω *and sample tessellations for* (a) *diffusion and inflow problems and* (b) *outflow problem. The integer labels in each tile give the logarithm of the maximum refinement ratio employed in that tile, over all the tests. For the (indicated) second level of refinement, the finest tiles are* 16×16 *subintervals, while the coarsest ones are* 4×4. *Some of the test runs are performed at smaller refinement ratios. In first-level tests, all tiles showing* "2" *are set to* "1" (8×8). *In zeroth-level refinement,* all *tiles are set to* "0".

rate inversely proportional to the radius (thus satisfying mass conservation). We refer to them as the *diffusion*, the *inflow*, and the *outflow* problems, respectively. The respective values of the radial eigenfunction exponent α are $\frac{2}{3}, \frac{1}{3}$, and approximately 10.0442. Figure 2 displays $u(x, y)$ for the three problems. The first two solutions lack derivatives at the reentrant corner. The last is everywhere twice differentiable, but the solution is characterized by steep variation in three of the *non-reentrant* corner regions, where $r>1$. Some form of local mesh refinement is critical to improving the accuracy of a finite difference solution. *Uniform* refinement is somewhat of a brute-force approach to a problem in which the strength of the singularity is known analytically. Instead of uniform refinement, or in conjunction with it, a simple change to the finite difference scheme in the vicinity of the reentrant corner can substantially improve the accuracy of the solution, as noted below.

4.1. The effect of adaptivity on convergence rate and truncation error. Tables 1 through 3 provide a purely serial demonstration of the value of local uniform mesh refinement: comparable accuracy in considerably fewer operations, compared with global uniform refinement. We solve these problems at effective refinement levels of $h^{-1}=32$, 64, and 128, where h^{-1} refers to the number of equal subintervals along one of the long edges of Ω. The computations are carried out to a reduction in the residual of 10^{-8}, so our report of the truncation error is not contaminated by incomplete

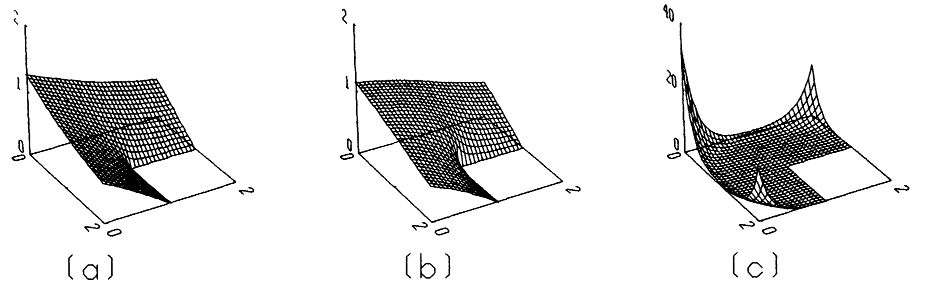


FIG 2. *Surface plots of* $u(x, y)$ *for* (a) *the diffusion problem, nondifferentiable at* $r=0$, (b) *the inflow problem, the singularity strengthened, and* (c) *the outflow problem, the singularity eliminated (note the change of vertical scale).*

TABLE 1
Number of unknowns N, sup-norm of the error e, iteration count I, and execution time T (sec) for the diffusion problem, globally and locally refined, along with execution time ratios, for a reduction in the initial residual of 10^{-8}.

	Global				Local				Ratio
h^{-1}	N_G	e_G	I_G	T_G	N_L	e_L	I_L	T_L	T_G/T_L
32	833	1.30 (−2)	18	1.1	833	1.30 (−2)	18	1.1	1.0
64	3201	8.30 (−3)	22	4.0	1817	8.39 (−3)	22	2.6	1.5
128	12545	5.25 (−3)	26	24.4	2409	5.26 (−3)	23	3.9	6.3

TABLE 2
Same as Table 1, *except for the inflow problem.*

	Global				Local				Ratio
h^{-1}	N_G	e_G	I_G	T_G	N_L	e_L	I_L	T_L	T_G/T_L
32	833	6.97 (−2)	18	1.1	833	6.97 (−2)	18	1.1	1.0
64	3201	5.65 (−2)	23	4.2	1817	5.66 (−2)	23	2.7	1.6
128	12545	4.53 (−2)	28	26.2	2409	4.58 (−2)	25	4.1	6.4

TABLE 3
Same as Table 1, *except for the outflow problem.* (*The error values are large in absolute terms, but still small relative to the sup-norm of the solution. See Fig.* 2(*c*) *for the scale of the solution.*)

	Global				Local				Ratio
h^{-1}	N_G	e_G	I_G	T_G	N_L	e_L	I_L	T_L	T_G/T_L
32	833	7.35 (−1)	19	1.2	833	7.35 (−1)	19	1.2	1.0
64	3201	4.15 (−1)	23	4.2	1609	4.30 (−1)	22	2.2	1.9
128	12545	2.19 (−1)	29	27.0	4697	2.40 (−1)	27	9.6	2.8

algebraic convergence. Global refinement results are on the left, and local on the right. Each of these columns lists the number of unknowns, the sup-norm of the error, the number of iterations, and the total execution time on a SPARCstation-1. The rightmost column gives the global-to-local execution time ratios for each refinement level.

Though the particular refinement strategy illustrated in Fig. 1 is by no means considered optimal, appreciable sixfold improvements in computational work are available in the first two problems. In the third problem, the number of tiles refined to a high degree is a substantial fraction of the total, resulting in only a nearly threefold gain.

The roughly linear behavior of iteration count with each doubling a global refinement in Tables 1 through 3 is consistent with the logarithmic growth in conditioning with h^{-1} derived in [3].

Case-by-case comparisons between the globally and locally refined results show that correctly placed local refinement can replace global refinement in representative problems with essentially no loss in accuracy and with gains in both storage and iteration count. The gain in iteration count is modest; it may be better to simply summarize with the observation that iteration count is not significantly affected in the context of the two-level preconditioner. This result is expected on the basis of theory: the condition number of the overall system is bounded by the condition number of the worst-conditioned subdomain, not by the details of the distribution of subdomain problems. The gain in storage, on the other hand, is potentially very significant and shows up both in a lower cost per iteration in already feasible problems and in an enlargement of the class of feasible problems, given a fixed-size memory. These conclusions are not new; they are stated here to provide self-contained motivation for the parallel results below, where the tradeoffs between adaptivity and parallel load balance are addressed.

Before leaving this section, we comment briefly on two means other than ordinary h-type refinement for improving the truncation error of either the global or the local discretizations compared above: singularity fitted discretizations and higher-order upwind discretizations.

As expected in cases where the exact solution is nondifferentiable at the reentrant corner (Tables 1 and 2), the sup-norm of the truncation error (which is determined by the data at the corner) shows slow improvement in h when ordinary finite differences based on Taylor series expansions are employed. As was demonstrated nearly half a century ago [19], the error can be improved by making a local rediscretization in the neighborhood of the corner based on an expansion of the solution in most singular modes of the known local series representation. For this purpose, three special-purpose tiles (simple rotations of each other) are employed at the reentrant corner, and the remainder of the discretization is unchanged. A standard Cartesian grid is employed, but the operator coefficients near the corner are derived from a cylindrical coordinate analysis. The results are shown in Table 4, the Taylor part of which is repeated from Table 1 for convenient comparison.

TABLE 4

Number of unknowns N, sup-norm of the error e, iteration count I, and execution time T (sec) for the diffusion problem, for ordinary Taylor series-based finite differences and for a discretization fitted to the singularity at the reentrant corner.

		Ordinary Taylor			Singularity fitted		
h^{-1}	N	e	I	T	e	I	T
32	833	1.30 (−2)	18	1.1	1.63 (−3)	17	1.1
64	3201	8.30 (−3)	22	4.0	1.04 (−3)	22	4.0
128	12545	5.25 (−3)	26	24.4	6.54 (−4)	26	24.3

We observe that the alternative discretization leaves the convergence rate of the preconditioned Krylov method and the execution time virtually unchanged, while the error at each resolution of the grid is a constant eightfold factor better than the corresponding error of the ordinary discretization. (The error improves with h at the same rate.)

The first-order accurate upwind treatment of convection leaves its signature in the truncation error of the outflow problem (Table 3). As was demonstrated in [16], second-order convergence can be recovered in the overall method by improving A to second order while leaving the preconditioner B unchanged. This is a very convenient compromise from the point of view of practical computing, since the preconditioner is more complex to code and the application of its inverse consumes a higher proportion of the execution time than does the forward application of the original operator. The results are shown in Table 5, the first-order part of which is repeated from Table 3 for convenient comparison.

We observe that the higher-order discretization of A is not "free," in that both the iteration count and the cost per iteration are increased. However, the truncation accuracy increases more rapidly, so that the computational cost to achieve a given accuracy is less with the higher-order method. Fortuitously, a direct example of this advantage is obtained by comparing the second row of the first-order method with the first row of the second-order method. In both cases, $\|e\|_\infty$ is 4.15×10^{-1} (approximately 1.2 percent of $\|u\|_\infty$), and the number of iterations is nearly identical as well. The second-order method achieves this result with one-quarter of the gridpoints and one-half of the execution time, however. Second-order convergence of the sup-norm in h is clearly visible in the right-hand set of columns.

TABLE 5

Number of unknowns N, sup-norm of the error e, iteration count I, and execution time T (sec) for the outflow problem, for first-order and second-order upwind finite differences.

		First-order upwind			Second-order upwind		
h^{-1}	N	e	I	T	e	I	T
32	833	7.35 (−1)	19	1.2	4.15 (−1)	24	2.1
64	3201	4.15 (−1)	23	4.2	9.59 (−2)	29	9.0
128	12545	2.19 (−1)	29	27.0	2.10 (−2)	34	53.0

These examples show that global h-type refinement can be synergistically combined with singularity fitted and mixed-order discretizations of A and B. Local h-type refinement can also be so combined; however, full exploitation of discretization improvements places a greater burden on automatic refinement algorithms.

4.2. Adaptivity and parallel load balance. To prevent the parameter space from growing unwieldy as multiple processors are added and parallel performance is studied, we focus on the diffusion problem ($c = 0$) alone. Overall conclusions based on studies of the inflow and outflow problems are similar, however. Since our attention is not on truncation error in these tests, we relax our convergence tolerance to a level more typical of an intermediate step in a sequence of nonlinear iterations: we enforce a relative reduction of 10^{-5} in the residual.

In the first set of tests we fix the number of tiles and vary the maximum level of refinement, h^{-1}, and the number of processors, p ($= 2^k$). In Tables 6 and 7, we present separately the time spent on preprocessing ("factor," predominantly factorizing the interior $\tilde{A}_I$ and crosspoint B_C blocks) and the time spent on the GMRES iteration ("solve," which includes the application of the preconditioner in factored form). Actual times are shown for $p = 1$; for $p > 1$ we show T_1/T_p, where T_p is the maximum time reported by any of the p processors. For the Encore Multimax, efficiencies (namely, the tabulated speedups divided by the number of processors) at $p = 16$ range from 21 percent to 48 percent in the preprocessing, and from 33 percent to 51 percent in the solution. For the Intel iPSC/860, efficiencies at $p = 16$ are worse: from 24 percent to 34 percent in the preprocessing, and from 15 percent to 21 percent in the solution. With its large ratio of communication to computation rates, the Intel iPSC/860 is not a very well balanced machine for problems with relatively small tiles. On the earlier Intel iPSC/2-SX, efficiencies of up to 50 percent, comparable to the shared-memory Multimax, were achieved.

TABLE 6

Number of unknowns N, iteration count I, execution times (for $p = 1$*), and parallel speedups (for* $p > 1$*) for the diffusion problem, through two levels of refinement on the Encore Multimax 320, for reduction in the initial residual of* 10^{-5}.

	Level 0		Level 1		Level 2	
N I	833 10		1817 13		2409 13	
p	Factor	Solve	Factor	Solve	Factor	Solve
1	*0.19s*	*3.55s*	*0.75s*	*9.19s*	*2.24s*	*13.3s*
2	1.72	1.79	1.88	1.86	1.54	1.74
4	2.88	3.15	3.44	3.50	2.94	3.32
8	4.23	5.09	5.24	5.69	3.33	4.52
16	6.09	6.41	7.60	8.08	3.51	5.30

TABLE 7

Same as Table 6, *except on the* Intel iPSC/860.

	Level 0		Level 1		Level 2	
N	833		1817		2409	
I	10		13		13	
p	Factor	Solve	Factor	Solve	Factor	Solve
1	*0.021s*	*0.217s*	*0.049s*	*0.465s*	*0.094s*	*0.613s*
2	2.10	1.40	2.23	1.56	1.49	1.50
4	3.00	1.90	3.77	1.98	3.13	2.15
8	4.20	2.12	4.90	2.67	3.36	2.69
16	4.20	2.38	5.44	3.34	3.76	3.16

The tests used eight tiles along one of the long edges of Ω (as in Fig. 1) and a "level 0" global resolution of $h^{-1} = 32$ (as in the first rows of Tables 1 through 3). The unrefined cases therefore employ 48 tiles, each 4×4. The uniformity at level 0 makes load balancing trivial (modulo boundary effects) for $p = 1, 2, 4, 8$, and 16, all of which evenly divide 48. However, the small size of each tile means short threads between synchronization points and intrinsic parallel inefficiency resulting from data exchanges and sequential data dependencies in the crosspoint equation solution are revealed. At level 1, there are 20 tiles of size 8×8 and 28 of size 4×4. The relatively better utilization of multiple processors at this level is due to the longer purely arithmetic threads of computation on the 8×8 tiles, which overcome a modest degree of load imbalance. At level 2, 3 of the 20 refined tiles at the previous stage are further refined to 16×16. Since there are few of these largest problems, load balance is much more distorted, but speedups still occur up to 16 processors. Load imbalance shrinks the "paydirt" region of parameter space for local uniform refinement in the parallel setting. For a problem in the small tile-to-processor ratio regime, which would otherwise lead to a lumpy work distribution, a certain amount of extra (numerically unnecessary) refinement, padding out the work distribution, is "free." The extra storage it requires is presumed available on a homogeneous distributed-memory array, the extra data movement is likely to be masked by comparably sized messages, the solution of the crosspoint system is independent of the degree of refinement, and the algebraic convergence rate is likewise unaffected.

The parallel efficiencies quoted above are not impressive, but the usual motivation for large-scale algorithm design is rapid solution, not linear speedup. Tables 8 and 9 pertain to the same diffusion problem at a finer resolution ($h^{-1}=128$) for a variety of tile sizes and processor numbers. In these tables, we show in three-column groups the speedups for the factorization and solution phases of the algorithm and the absolute execution times for the total run. In Table 8 all tiles are uniform, while in Table 9, the tiles near the corner of the L-shaped domain are refined to level 1. Note that all five sets of columns in both tables solve the original PDE to nearly identical levels of truncation error; therefore, the total execution times can be compared directly to determine the best "bottom line" discretization/decomposition/parallelization combination. To facilitate such comparisons, we present the unreduced "Total" timings alongside the speedups for each phase.

The best speedups in both tables are for the coarsest tessellations (48 tiles). These have the sparsest crosspoint systems and the largest tiles, and hence the smallest

TABLE 8

Number of unknowns N, iteration count I, execution times (italic), and parallel speedups for the diffusion problem, globally refined to $h^{-1}=128$ on the Intel iPSC/860, *for reduction in the initial residual of* 10^{-5}.

	48 tiles each 16×16			192 tiles each 8×8			768 tiles each 4×4		
N	12,545			12,545			12,545		
I	14			11			8		
p	Factor	Solve	Total	Factor	Solve	Total	Factor	Solve	Total
1	*0.91s*	*3.21s*	*4.11s*	*0.33s*	*2.23s*	*2.56s*	*0.55s*	*2.69s*	*3.24s*
2	1.98	1.87	*2.17s*	1.84	1.72	*1.48s*	1.23	1.15	*2.80s*
4	3.85	3.41	*1.17s*	3.28	3.02	*0.84s*	1.41	1.84	*1.85s*
8	7.48	5.63	*0.69s*	4.94	4.80	*0.53s*	1.50	2.98	*1.27s*
16	13.7	9.27	*0.41s*	6.90	6.59	*0.39s*	1.58	4.09	*1.01s*

TABLE 9

Same as Table 8 *except locally refined* (*level* 1) *to* $h^{-1}=128$.

	48 tiles 8×8 or 16×16			192 tiles 4×4 or 8×8		
N	7,089			6,697		
I	14			11		
p	Factor	Solve	Total	Factor	Solve	Total
1	*0.41s*	*1.77s*	*2.18s*	*0.17s*	*1.44s*	*1.62s*
2	1.98	1.82	*1.18s*	1.64	1.55	*1.04s*
4	3.76	2.99	*0.71s*	2.85	2.38	*0.67s*
8	6.31	4.46	*0.46s*	3.78	3.22	*0.49s*
16	9.76	6.00	*0.34s*	4.70	4.42	*0.36s*

communication-to-computation ratios. However, among the cases in Table 8, the most parallel-efficient is not the fastest, at any number of processors. The intermediate tessellation in Table 8 gives the best wall-clock times, in spite of mediocre efficiency. The last set uses tiles that are too small, and it is choked by the large crosspoint system. In terms of total execution time, this is the worst case. Similar findings were made in [11] for a variety of convection-diffusion problems solved on a uniprocessor: the optimum granularity of tessellation (for a two-scale preconditioner with exact subdomain solves) occurs between the extremes of many small tiles (dominated by the crosspoint system) and few large tiles (dominated by the interior systems).

Among the locally refined cases in Table 9, the preferred tessellation is different at different numbers of processors, as a result of the interaction of load imbalance with communication-to-computation ratio. It is interesting to observe that the best time for this problem, 0.34*s*, is within 15 percent of the best time on the globally refined problem of the same effective resolution, though the number of unknowns involved is over 40 percent less. Load imbalance resulting from a small tile-to-processor ratio takes away from the advantage of local refinement in parallel.

Finally, we investigate differences between the work estimate and the wrap mappings of tiles to processors. This study is presented for a level-1 refinement at the upper end of the problem sizes we have tested in two dimensions. In Tables 10 and 11 we consider the L-shaped diffusion problem on the iPSC/860 for effective resolutions of $h^{-1} = 256$ and $h^{-1} = 512$ on 768 tiles with up to 64 processors. Since the largest problem does not fit onto a single processor, absolute speedup data are not available, and we report relative speedups with each doubling of processors ($T_p/T_{p/2}$) under the "Factor" and "Solve" headings.

TABLE 10

Number of unknowns N, iteration counts I, execution times (italic), and relative parallel speedups for the tile algorithm based on the work-estimated tile-to-processor mapping.

	768 tiles 4×4 or 8×8			768 tiles 8×8 or 16×16		
N	25,969			103,201		
I	10			12		
p	Factor	Solve	Total	Factor	Solve	Total
1	*0.90s*	*5.76s*	*6.66s*	—	—	—
2	1.44	1.16	*5.61s*	—	—	—
4	1.30	1.75	*3.32s*	*1.74s*	*7.87s*	*9.60s*
8	1.15	1.51	*2.29s*	1.67	1.74	*5.57s*
16	1.11	1.44	*1.67s*	1.47	1.65	*3.44s*
32	1.05	1.29	*1.36s*	1.36	1.49	*2.36s*
64	1.03	1.17	*1.21s*	1.18	1.31	*1.84s*

We observe that neither the work estimate nor the wrap mapping is systematically better than the other. It is noteworthy that as simple a scheme as wrap mapping competes as well as it does with a first attempt at a work estimation scheme. Both schemes suffer from an interesting nonmonotonicity in relative speedup versus *p* in going from one to two processors. We attribute this to poor tile-processor locality; that is, tiles have neighbors in different processors in numbers substantially greater

TABLE 11
Same as Table 10 *except for the wrap mapping.*

	768 tiles 4×4 or 8×8			768 tiles 8×8 or 16×16		
N	25,969			103,201		
I	10			12		
p	Factor	Solve	Total	Factor	Solve	Total
1	*0.90s*	*5.76s*	*6.66s*	—	—	—
2	1.42	1.07	*6.03s*	—	—	—
4	1.29	2.06	*3.11s*	*1.79s*	*7.66s*	*9.45s*
8	1.15	1.73	*1.94s*	1.54	1.77	*5.49s*
16	1.12	1.41	*1.45s*	1.51	1.67	*3.36s*
32	1.05	1.18	*1.27s*	1.40	1.44	*2.35s*
64	1.04	1.12	*1.16s*	1.21	1.33	*1.81s*

than an optimal decomposition would require. As the number of processors increases, the amount of off-processor intertile data traffic does not decrease, but more of it can be overlapped in the larger number of communication channels available.

Finally, we report breakdowns of the factorization and solution times. For the factorization phase of the largest problem on 64 processors under the work estimate-based mapping, 79 percent was devoted to factoring the crosspoint system, 20 percent to the subdomain interior systems, and 1 percent to the interface systems. Of the time required to iterate it to convergence, 49 percent was spent solving the crosspoint system, 15 percent solving the interior systems, and 1 percent solving the interface systems. The remaining solution time went into the work of the GMRES algorithm apart from the inner products (23 percent), the inner products themselves (8 percent), and the application of A to a vector (4 percent). Execution rates are of interest as measures of compatibility of the software and the machine hardware and architecture, though they reveal little about the appropriateness of the algorithm itself. On the 64 processor run, 362 Mflops (5.7 Mflops per node) were achieved in the interior block factorization step, 271 Mflops in the interior solution steps, and 210 Mflops in the matrix-vector multiply with the A matrix. At the other end of the performance spectrum, the crosspoint factorization garnered only an aggregate 5.3 Mflops and the crosspoint system solution only 2.0 Mflops. The crosspoint system contributes to a substantial per-tile communication overhead, the penalty of which we look forward to reducing through faster communication and perhaps better algorithms. So-called asynchronous algorithms, allowing simultaneous solution of crosspoint and interior systems *within a single iteration step* represent one possibility. The high quality preconditioning resulting in only 12 iterations to solve an elliptic problem with 103,201 unknowns depends crucially on the crosspoint system.

5. Summary and shopping list. We conclude with some brief assessments of the state of parallel domain decomposition algorithms.

The potential advantages of domain-decomposed preconditioned iterative methods for PDE problems with complex geometry, adaptive refinement requirements, and vast numbers of unknowns are clear. A two-level preconditioner that includes a global crosspoint solve is capable of maintaining good conditioning, thus providing good algebraic convergence and preserving the usefulness of double-precision computer arithmetic in the context of very large problems. The key features to be addressed in a parallel implementation are the inefficiency of the crosspoint solve itself and load imbalance and data locality in tile-to-processor mapping.

The communication-intensive crosspoint system is the single largest consumer of time in both the preprocessing and iteration phases, when the number of tiles is sufficiently large; and a large number of tiles is often required to realize the advantages of the method in complex geometry and adaptive refinement contexts. Thus, the distributed solution of sparse matrix problems should remain the focal point of research that it is today. It is the importance of having a conveniently structured crosspoint system that keeps us from a much looser framework for our tessellations. This issue is described in greater detail in [11]. A preconditioner based only on nearest neighbor interactions would impose almost no rules on the topology of the decomposition, but the deterioration in convergence rate in the many tile limit would eventually become unacceptable, and no further useful parallelism could be found without subdividing the tile.

It is important to gain experience with domain decomposition algorithms in three-dimensional problems, since this is the true province of promise for parallel computing. Motivated by the fact that a two-level hierarchical preconditioner with crosspoint basis functions that are merely coarsened versions of the fine-grid basis functions cannot lead to (near) optimal convergence in three dimensions as it does in two, pioneering works [2], [6], [17] (see also [7]) considered preconditioners with greater coarse-grid implicitness requiring sequential computations on data associated with the full "wire basket" of the decomposition. A recent theoretical work written with an eye toward parallelism in three-dimensional applications is the doctoral thesis of Smith [22]. In it Smith presents an algorithm valid in two or three dimensions in which the coarse and multiple fine-grid problems can be attacked simultaneously.

Load imbalance can seriously detract from the parallel performance of an adaptive algorithm. It is desirable to have mapping algorithms that are superior to the simple ones employed herein, but they obviously must not be so complex as to begin to dominate the total execution time themselves. Attention to the processor locality of frequently cooperating (i.e., neighboring) tiles could also yield important dividends, depending on the architecture.

Tile subdivision and the clustering of homogeneously refined tiles into larger rectangular patches (meta-tiles) break the connection between the parallel data structure granularity and the crosspoint system dimension. These are two extensions that we have only begun to investigate, although their promise is considerable. Neither would necessarily replace the indivisible heterogeneous tile algorithm evaluated herein; rather, they would generalize it to move into parallel realms beyond moderate granularity MIMD. Tile subdivision accommodates numbers of processors larger than the maximum useful size of the crosspoint system. Homogeneous tile clustering for all algorithmic phases other than the multilevel preconditioning accommodates vector nodes in an MIMD array or in vector-like SIMD clusters within an MIMD array, the latter hybrid architectures being likely for massively parallel supercomputers because of their cost-effective use of the silicon.

REFERENCES

[1] M. J. Berger and J. Oliger, *Adaptive mesh refinement for hyperbolic partial differential equations*, J. Comput. Phys., 53 (1984), pp. 484–512.

[2] J. H. BRAMBLE, J. E. PASCIAK, AND A. H. SCHATZ, *The construction of preconditioners for elliptic problems by substructuring*, IV, Math. Comp., 53 (1989), pp. 1-24.

[3] X.-C. CAI, W. D. GROPP, AND D. E. KEYES, *Convergence rate estimate for a domain decomposition method*, Numer. Math., to appear.

[4] T. F. CHAN AND D. E. KEYES, *Interface preconditionings for domain-decomposed convection-diffusion operators*, in Proc. Third Internat. Symposium on Domain Decomposition Methods, T. F. Chan, R. Glowinski, J. Périaux, and O. B. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990, pp. 245-262.

[5] D. DEWEY AND A. T. PATERA, *Geometry-defining processors for partial differential equations*, in Special Purpose Computers, Academic Press,. New York, 1988, pp. 67-96.

[6] M. DRYJA AND O. B. WIDLUND, *Some domain decomposition algorithms for elliptic problems*, Tech. Report TR 438, Courant Institute, New York University, New York, 1989.

[7] ———, *Towards a unified theory of domain decomposition algorithms for elliptic problems*, in Proc. Third Internat. Symposium on Domain Decomposition Methods, T. F. Chan, R. Glowinski, J. Périaux, and O. B. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990, pp. 3-21.

[8] H. C. ELMAN, *Iterative methods for large, sparse, nonsymmetric systems of linear equations*, Tech. Report RR-229, Department of Computer Science, Yale University, New Haven, CT, 1982.

[9] W. D. GROPP, *Solving* PDE*s on loosely-coupled parallel processors*, Parallel Comput., 5 (1987), pp. 165-173.

[10] W. D. GROPP AND D. E. KEYES, *Domain decomposition on parallel computers*, Impact Comput. Sci. Eng., 1 (1989), pp. 421-439.

[11] ———, *Domain decomposition methods in computational fluid dynamics*, Tech. Report 91-19, Institute for Computer Applications in Science and Engineering, Hampton, VA, 1991.

[12] M. R. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 33-53.

[13] D. E. KEYES AND W. D. GROPP, *A comparison of domain decomposition techniques for elliptic partial differential equations and their parallel implementation*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s166-s202.

[14] ———, *Domain decomposition techniques for nonsymmetric systems of elliptic boundary value problems: Examples from* CFD, in Proc. Second Internat. Symposium on Domain Decomposition Methods, T. F. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989, pp. 321-339.

[15] ———, *Domain decomposition techniques for the parallel solution of nonsymmetric systems of elliptic boundary value problems*, Appl. Numer. Math., 6 (1990), pp. 281-301.

[16] ———, *Domain-decomposable preconditioners for second-order upwind discretizations of multicomponent systems*, in Proc. Fourth Internat. Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, Y. A. Kuznetsov, G. Meurant, J. Périaux, and D. B. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1991.

[17] J. MANDEL, *Efficient domain decomposition preconditioning for the p-version finite element method in three dimensions*, Tech. Report, Computational Mathematics Group, University of Colorado, Denver, CO, 1989.

[18] S. F. MCCORMICK, ED., *Multilevel Adaptive Methods for Partial Differential Equations*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989.

[19] H. MOTZ, *The treatment of singularities of partial differential equations by relaxation methods*, Quart. Appl. Math., 4 (1946), pp. 371-377.

[20] N. M. NACHTIGAL, S. C. REDDY, AND L. N. TREFETHEN, *How fast are nonsymmetric matrix iterations?* Tech. Report, Numerical Analysis Report 90-2, Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA, 1990.

[21] Y. SAAD AND M. H. SHULTZ, GMRES*: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856-869.

[22] B. F. SMITH, *Domain decomposition algorithms for the partial differential equations of linear elasticity*, Tech. Report 517, Courant Institute, New York University, New York, 1990.

[23] P. SONNEVELD, CGS, *a fast Lanczos-type solver for nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 36-52.

[24] L. N. TREFETHEN, *Approximation Theory and Numerical Linear Algebra*, Algorithms for Approximation, J. C. Mason and M. G. Cox, eds., Chapman and Hall, London, 1990, pp. 336-360.

[25] D. M. YOUNG AND K. C. JEA, *Generalized conjugate gradient acceleration of nonsymmetrizable iterative methods*, Linear Algebra Appl., 34 (1980), pp. 159-194.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 1, pp. 146-167, January 1992

OPTIMAL MULTILEVEL ITERATIVE METHODS FOR ADAPTIVE GRIDS*

WILLIAM F. MITCHELL†

Abstract. Many elliptic partial differential equations can be solved numerically with near optimal efficiency through the uses of adaptive refinement and multigrid solution techniques. This paper presents a more unified approach to the combined process of adaptive refinement and multigrid solution which can be used with high order finite elements. Refinement is achieved by the bisection of pairs of triangles, corresponding to the addition of one or more basis functions to the approximation space. An approximation of the resulting change in the solution is used as an error indicator. The multigrid iteration uses red-black Gauss-Seidel relaxation with local black relaxations. The grid transfers use the change between the nodal and hierarchical bases. This multigrid iteration requires only $O(N)$ operations, even for highly nonuniform grids, and is defined for any finite element space. The full multigrid method is an optimal blending of the processes of adaptive refinement and multigrid iteration. To minimize the number of operations required, the duration of the refinement phase is based on increasing the dimension of the approximation space by the largest possible factor, given the error reduction of the multigrid iteration. The algorithm (i) uses only $O(N)$ operations, (ii) solves the discrete system to the accuracy of the discretization error, and (iii) achieves optimal convergence of the discretization error in the presence of singularities. Numerical experiments confirm this for linear, quadratic, and cubic elements.

Key words. finite elements, elliptic partial differential equation (PDE), adaptive refinement, multigrid, hierarchical basis

AMS (MOS) subject classifications. 65F10, 65N30, 65N50

1. Introduction. The efficient numerical solution of elliptic partial differential equations (PDEs) has been an important area of research in numerical analysis for several decades. Over the years, many new methods have been discovered in the areas both of discretizing the differential equation and of solving the discrete problem. While every method has restrictions on the subclass of problems to which it is applicable, the efficiency of the methods has increased manyfold since even the best solvers of 25 years ago. A thorough presentation of most practical methods is provided by Birkhoff and Lynch [7]. Additionally, the ELLPACK project [22] has provided us with robust software for many of these methods and a sound environment in which to perform numerical experiments to determine the relative merits of each method.

Today, we are at a point where many problems can be solved with near optimal efficiency. Many of the recent improvements have occurred through the uses of adaptive refinement, multigrid solution techniques, and parallelism. We will not consider parallelism here but concentrate on adaptive refinement and full multigrid solution. At first glance, the two concepts seem almost meant for each other. Each is a process that alternately performs phases of refinement and solution, with one concentrating on refinement and the other on solution. Yet, when examined more deeply, subtle difficulties arise in combining them. Most researchers who have attempted to join the two have maintained the individual structures of the two phases and then developed strategies to overcome the problems that emerge. It is our goal to develop a more unified approach to the combined processes of adaptive refinement and multigrid solution that is so natural that it becomes obvious how to extend these important

* Received by the editors April 5, 1990; accepted for publication July 22, 1990. This work was performed while the author was attending the Department of Computer Sciences, University of Illinois at Urbana-Champaign and supported in part by the U.S. Department of Energy under grant DOE DEFG02-87ER25026.

† General Electric Advanced Technology Laboratories, Bldg. 145-2 Moorestown Corporate Center, Moorestown, New Jersey 08057.

techniques to more complicated situations, such as with high order methods and for three-dimensional problems. This unification and extendability is achieved by interpreting the parts of the method from the viewpoint of the hierarchical basis, in which successive coefficients represent a change in the solution rather than the value of the solution. In particular,

(i) adaptive refinement is considered to be the selective enrichment of the approximation space by adding new basis functions, rather than the division of triangles or rectangles. Local relaxations that are identical to those of the multigrid iteration are performed with the addition of each new basis function. The choice of which basis functions to add is based on how much each potential new basis function will reduce the error and is determined by approximating the hierarchical coefficients. This computation uses the equations that will be added to the linear system when the space is enriched by this function, and is the same as the local relaxation.

(ii) the multigrid iteration is defined strictly in terms of the hierarchical basis and is not restricted to the approximation spaces we consider. Grid transfers are achieved through the change between nodal and hierarchical bases. Relaxations are performed in both the nodal and hierarchical bases, essentially supersaturating the approximation space with an excessive number of directions in which to minimize the error.

(iii) the full multigrid method is a very natural, optimal blending of adaptive refinement with multigrid iterations. The approximation space is enriched with as many new basis functions as is possible with respect to the error-reducing power of the multigrid iteration. This minimizes the number of operations used to obtain a solution whose accuracy is comparable with the discretization error.

The method is presented in the context of the second order selfadjoint elliptic problem

$$\begin{aligned} Lu &= (pu_x)_x + (qu_y)_y + ru = f \quad \text{in } \Omega \\ u &= g \quad \text{on } \partial\Omega \end{aligned} \tag{1.1}$$

where Ω is a polygonal domain in $\mathbf{R}^2$ and p, q, r, f, and g are functions of x and y. The Galerkin finite element method is used to discretize the problem over a triangular mesh which covers Ω exactly. The method will be developed for the higher order spaces of C^0 pth degree piecewise polynomials over triangles, where p is any given positive integer, but the usual space of continuous piecewise linear functions over the triangles will frequently be used for illustration purposes.

In § 2, the adaptive refinement aspect of the method is presented. Triangles are divided by bisection, similar to the bisection method of Rivara [24], [25] and identical to that of Sewell [27], [28], the difference between the two being in the method for determining which side of the triangle is to be bisected, the longest edge or the side opposite the newest vertex, respectively. Compatibility of the triangulation is maintained during the refinement process, rather than before, as in Sewell [27], [28], or after as in Rivara [24], [25] and Bank and Sherman [3], [4]. The error indicator uses the hierarchical basis to estimate the norm of the change in the solution invoked by dividing the triangle, similar to that proposed by Zienkiewicz et al. [32] for bilinear elements. The overall structure of the adaptive refinement algorithm is based on the division of a small number of triangles, rather than local refinement of the entire grid, to provide more control over how often multigrid iterations occur.

Section 3 presents and analyzes the multigrid iteration used to solve the linear system of equations. This multigrid uses a V-cycle, a restricted form of red-black Gauss-Seidel relaxation, and restriction and prolongation operators which arise naturally from the hierarchical basis. In the case of uniform grids the method is equivalent to that studied by Braess [8], [9], [10] and is related to the MGR methods [16], [23]. It is also very closely related to the hierarchical basis multigrid method recently developed by Bank, Dupont, and Yserentant [6], and to the composite grid methods of McCormick.

In § 4 a full multigrid method is presented to combine the adaptive refinement procedure with the multigrid iteration into a very efficient unified solution method that is easily extended to high order finite element spaces. The full multigrid method consists of alternately performing refinement and solution phases. Properties of the convergence of the discretization error are used to justify basing the frequency of solution phases, not on the levels of refinement, but on the increase in the number of nodes. Thus, the refinement phase continues until the number of nodes has been increased by some given factor. A formula is derived to determine how large this factor can be in terms of the error reduction factor of the multigrid iteration and the convergence rate of the discretization error for the finite element space being used.

In § 5, the results of numerical experiments are presented to demonstrate the performance of the method.

2. Adaptive refinement. The use of adaptive refinement to obtain a grid for the discretization of a partial differential equation has been the subject of much research in the past decade [2], [3], [14], [17], [18], [20], [25], [27], [32]. The idea is to automatically construct a grid that is coarse where the solution is well behaved, fine near singularities, boundary layers, etc., and has a smooth transition between the coarse and fine parts. Such a grid can dramatically reduce the number of nodes needed to obtain an accurate solution for marginally smooth problems, and can recover the optimal order of convergence for nonsmooth problems.

Triangle division. Central to any adaptive refinement algorithm for a finite element grid is a method for dividing (refining) the elements, triangles in our case. There are two major methods for dividing triangles in adaptive refinement algorithms. Regular division divides a triangle into four similar triangles by connecting the midpoints of the sides. Bank and Sherman [3], [4] show how to use a regular division in an adaptive refinement algorithm. Bisection division connects one of the vertices of the triangle to the midpoint of the opposite side. Two approaches are in use regarding the selection of the vertex to be divided. Rivara [24], [25] chooses the vertex opposite the longest edge. Sewell [27], [28] chooses the "newest" vertex. The method presented here, *newest vertex bisection*, is nearly identical to the method presented by Sewell. Much of the terminology of this section is due to Sewell.

In bisection division a triangle is divided to form two new triangles by connecting one of the vertices, called the *peak*, to the midpoint of the opposite side, called the *base*, as in Fig. 2.1. The original triangle is called the *parent*, and the two new triangles are called the *children*. The children are said to have *generation* $i+1$ where i is the generation of the parent. The initial triangle is assigned generation 1. The new vertex

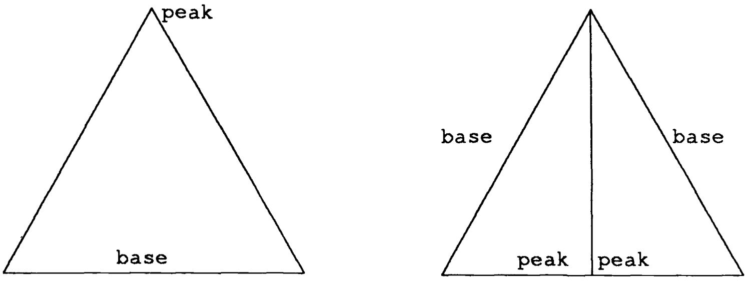


FIG. 2.1. *Propagation of the peak with newest node bisection.*

created at the midpoint of the base is assigned to be the peak of the children, hence the name *newest vertex bisection.*

It is easily shown that only four similarity classes of triangles [27] and only eight distinct angles [19] are created by this method, so the angles satisfy the important condition of being bounded away from 0 and π [1], [15].

Compatibility. One of the difficulties in adaptive refinement is that of maintaining compatibility of the triangulation. A triangulation is said to be *compatible* if for any two triangles t_i and t_j, $t_i \cap t_j$ is either empty, a common vertex, or a common side. In other approaches, some set of triangles with large error indicators is divided, producing an incompatible triangulation, and then a second process is performed to regain compatibility by dividing more triangles. In the approach that follows, compatibility is maintained *during* the refinement process, by dividing *pairs* of triangles rather than individual triangles, eliminating the need for a separate follow-up process to recover compatibility.

A triangle is said to be *compatibly divisible* if its base is either the base of the triangle that shares that side or part of the boundary of the domain. If a triangle is compatibly divisible, then divide the triangle and the neighbor opposite the peak (if such a neighbor exists) simultaneously as a pair. If a triangle is not compatibly divisible, then after a single bisection of the neighbor opposite the peak, it will be. So in this case, first divide the neighbor by the same process, and then divide the triangle and neighbor opposite the peak simultaneously. This is illustrated in Fig. 2.2. This leads to the recursive Algorithm 2.1, which is easily implemented in FORTRAN by constructing a stack of the triangles that need to be divided. It is easily seen that the length of the recursion is bounded by the generation of the triangle, since the generation of the triangles decrease with each recursive call. It can be shown [19] that it is always possible to assign peaks to the initial triangulation so that every triangle is compatibly divisible.

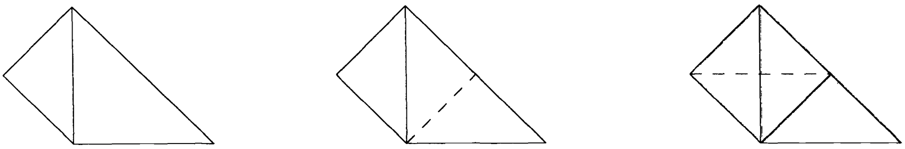


FIG. 2.2. *Maintaining compatibility during refinement.*

```
ALGORITHM 2.1. divide_triangle($t$)
  if $t$ is not compatibly divisible then
    divide_triangle(neighbor of $t$ opposite peak)
  endif
  divide the triangle pair $t$ and the neighbor opposite the peak of $t$
  return
```

Error indicator. Another critical element of any adaptive refinement algorithm is the error indicator, which is used to determine which triangles should be divided. Several good error indicators have been proposed [2], [3], [5], [23], [32]. Most of these are based on estimating the discretization error over each triangle. A slightly different approach is obtained by attempting to determine which admissible basis functions would reduce the discretization error the most. This becomes a type of error indicator for *pairs* of triangles. Mitchell [18] surveyed several error indicators and triangle division methods and compared them in a numerical experiment. He found that, among the methods considered, there is no universally "best" adaptive refinement method, and that most of the methods performed approximately the same. The method described here performed well in that experiment. This method is similar to that proposed by Zienkiewicz et al. [32] for bilinear rectangular elements. Interpretation in terms of hierarchical bases makes it possible to define this error indicator for any finite element space. It will be presented for the spaces of C^0 pth degree polynomials over bisected triangles, considering first linear elements, and then extending to arbitrary degree.

Instead of attempting to divide the triangles over which the error is the largest, attempt to divide the triangles whose division makes the greatest change in the solution. Since this change in the solution reduces the error, it is an attempt to divide the triangles that make the greatest reduction in the error, i.e., reduce the error the fastest for the number of divisions performed.

To approximate how much of a change in the solution will occur, use the hierarchical basis, defined in § 3. For what follows it suffices to know that when a function is expanded using the hierarchical basis, the coefficients represent displacements rather than nodal values as with the usual nodal basis, i.e., the coefficient of a hierarchical basis represents how much change occurs in the approximating function when the grid is refined by adding the new node.

The division of a pair of triangles, as in Fig. 2.3, by newest vertex bisection corresponds to the addition of one new basis function. Let v_i and ϕ_i, $i=1$, 2, 3, and 4 be the vertices of the triangles and corresponding hierarchical basis functions, and v_5 and ϕ_5 be the new vertex and basis function. To approximate how much change would occur if this division were to be performed, approximate the coefficient of ϕ_5, α_5, by assuming that α_i, the coefficients of ϕ_i, remain unchanged for $i=1$, 2, 3 and 4. Then

$$\alpha_5=\left((f,\phi_5)-\sum_{i=1}^{4}\alpha_i\langle\phi_i,\phi_5\rangle\right)\Big/\langle\phi_5,\phi_5\rangle \tag{2.1}$$

where $\langle\cdot,\cdot\rangle$ and $(\cdot,\cdot)$ are the usual inner products used to obtain the stiffness matrix and load vector, respectively, and f is the right-hand side of the differential equation. If $\|\cdot\|$ is the energy norm defined by $\|u\|^2=\langle u,u\rangle$, $\|\alpha_5\phi_5\|^2$ is the amount by which the square of the energy norm of the error is reduced by adding $\alpha_5\phi_5$ to the approximate solution. $\|\alpha_5\phi_5\|$ is the error indicator for this pair of triangles.

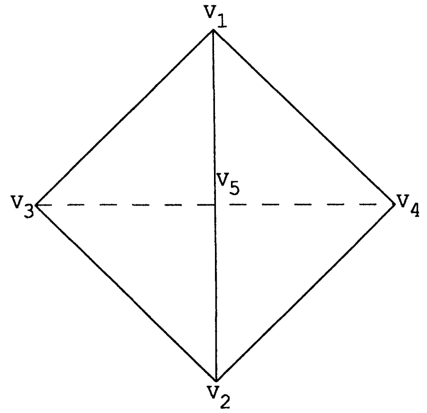


FIG. 2.3. *Triangle pair for error indicator.*

The extension of this error indicator to C^0 pth degree polynomials is straightforward. The only difference is that there are p^2 new basis functions rather than just one, so the amount of change depends on more than one basis function. Equation (2.1) is replaced by a linear system of p^2 equations in p^2 unknowns. This system can be solved by Cholesky decomposition in an acceptable number of operations, and one can compute $\|\sum \alpha_i\phi_i\|$, where the sum is over the p^2 new nodes associated with the prospective new vertex. Clearly this error indicator can be further extended to any finite element space with a hierarchical basis.

Refinement process. Given the error indicators for every triangle (or pairs of compatibly divisible triangles), one would ideally want to select the next triangle to divide by choosing the triangle with the largest error indicator. However, for an algorithm to use only $O(N)$ operations it is imperative that the selection of the next triangle to divide requires only $O(1)$ operations. This means that there is not enough time to search every triangle to find one with the largest error indicator. Instead, a triangle whose error indicator is close to the largest will be selected.

Let e be the largest error indicator at the beginning of the refinement phase. Partition the triangles (only including one triangle from each pair of compatibly divisible triangles) into Q sets such that each set contains all the triangles whose error indicators fall in a certain range. Specifically, for a given $0<c<1$, a triangle is in the qth set if and only if its error indicator is between $c^{q-1}e$ and $c^q e$ for $1 \leqq q \leqq Q-1$ and is in the Qth set if its error indicator is less than $c^{Q-1} e$. The first set contains all the triangles whose error indicator is larger than ce, and any one of these triangles can be selected as the next triangle to be divided. The selection of c and Q will determine the precision with which the correct triangles are selected. Mitchell [19] recommends $Q=4$ and $c=f^{-\alpha-1/2}$, where f is the factor by which the number of nodes is increased during the refinement phase, and α is the rate of convergence of the energy norm of the discretization error.

Combining the material presented in this section into an adaptive refinement algorithm results in the rather simple looking Algorithm 2.2. What is meant by "enough refinement" will be discussed in § 4. The actual process of dividing a triangle involves several steps which redefine the grid, change the linear system, provide a first approximation to the solution and update error indicators. The details of these steps are presented in [19]. Algorithm 2.3 summarizes the process.

```
ALGORITHM 2.2. Adaptive refinement
  repeat
    determine which triangle to divide
    divide the triangle
  until enough refinement has occurred
```

```
ALGORITHM 2.3. Divide triangle pair
  change grid specification
  add new equation(s) to the linear system
  change other affected equations
  block relaxation for new nodes
  Gauss-Seidel point relaxation for neighboring old nodes
  compute error indicators for new triangles and neighboring triangles
```

3. Multigrid solution. The multigrid method [11], [12], [29] has recently been established as perhaps the most efficient method for solving the linear systems that arise from the discretization of differential equations. The popularity of this method can be attributed to the fact that it is optimal in the sense that one multigrid iteration can reduce the norm of the error of the approximate solution of the linear system by a factor that is bounded away from 1 independent of N, the size of the linear system, while using only $O(N)$ operations. This section presents and analyzes a multigrid iteration suitable for the linear systems that arise from using the finite element method with low or high order bases on the triangulations generated by the adaptive refinement algorithm of § 2. In the special case of linear elements, uniform grids and certain domains, the method is equivalent to that studied by Braess [8], [9], [10] and the MGR methods [16], [23]. The method will be developed in terms of hierarchical bases from which it will be easy to extend the method to nonuniform grids, more general domains and high order bases.

Hierarchical bases. The use of hierarchical bases for finite elements has been considered by Zienkiewicz et al. [32]; Yserentant [30], [31]; Craig and Zienkiewicz [13]; and Bank, Dupont, and Yserentant [6]. The usual nodal basis, $\{\phi_i\}_{i=1}^N$, for a space of piecewise polynomials can be defined on a given grid by

$$\phi_i=\begin{cases}1 & \text{at node } i\\ 0 & \text{at all other nodes.}\end{cases}$$

In contrast, the hierarchical basis is defined using the family of nested grids, $\{T_i\}_{i=1}^L$, from the refinement process. The hierarchical basis begins with the nodal basis on the initial grid, T_1. As refinement proceeds, with each division one or more new nodes are added, and for each node a new basis function is defined so that it has the value 1 at the new node and 0 at all other nodes, *but the existing basis functions remain unchanged.* The *level* of a basis function is the same as the generation of the elements created when the basis function is added, i.e., higher level basis functions refer to those associated with smaller elements.

A *2-level hierarchical basis* can also be defined by starting with the nodal basis for the grid T_{L-1} and defining the higher level basis functions as above. The superscripts (N), (H), and (2) are used to indicate which basis is in use. Thus, $\phi_i^{(\mathrm{N})}$, $\phi_i^{(\mathrm{H})}$, and $\phi_i^{(2)}$ represent basis functions from the nodal, hierarchical, and 2-level bases, respectively.

Any function f that lies in the space of piecewise polynomial functions on T_L has an expansion in terms of any of the bases. Forms of α are used to denote the coefficients in this expansion. Thus

$$f=\sum_{i=1}^{N}\alpha_i^{(\mathrm{N})}\phi_i^{(\mathrm{N})}=\sum_{i=1}^{N}\alpha_i^{(\mathrm{H})}\phi_i^{(\mathrm{H})}=\sum_{i=1}^{N}\alpha_i^{(2)}\phi_i^{(2)}.$$

Conversion between bases is a linear process. S denotes the matrix that converts the coefficient vector of the hierarchical basis to the nodal basis, thus $\alpha^{(\mathrm{N})}=S\alpha^{(\mathrm{H})}$ and $\alpha^{(\mathrm{H})}=S^{-1}\alpha^{(\mathrm{N})}$. S_l denotes the conversion from the 2-level basis to the nodal basis on an l level grid. If the rows and columns of S_l are ordered such that rows corresponding to nodes of the same level are grouped together with lower level rows first, and S_l is partitioned into two parts corresponding to levels 1 through $l-1$ and level l, then S_l has the form

$$S_l=\begin{bmatrix}\mathrm{I} & 0\\ s & \mathrm{I}\end{bmatrix}$$

and

$$S_l^{-1}=\begin{bmatrix}\mathrm{I} & 0\\ -s & \mathrm{I}\end{bmatrix}$$

where $s_{ij}=\phi_j^{(2)}(x_i, y_i)$ and x_i and y_i are the coordinates of the ith node. Conversion between bases amounts to multiplying a vector by S_l or S_l^{-1}. If N_{l-1} is the number of nodes in the first $l-1$ levels

$$\alpha_i^{(\mathrm{N})}=\begin{cases}\alpha_i^{(2)} & \text{if node } i \text{ does not have level } l\\ \alpha_i^{(2)}+\sum\limits_{j=1}^{N_{l-1}}\phi_j^{(2)}(x_i, y_i)\alpha_j^{(2)} & \text{if node } i \text{ has level } l.\end{cases}$$

With newest vertex bisection refinement and pth degree piecewise polynomials, there are at most $(p+1)^2$ nonzeros in the sum. Moreover, the value of $\phi_j^{(2)}(x_i, y_i)$ depends only on p and the relative placements of nodes i and j in the same triangle, and is independent of the triangle shapes and sizes, problem being solved, etc. Thus a small $((p+1)^2\times p^2)$ table of these values can be constructed to be used whenever a basis change is desired.

To multiply a vector by S_l^{T} and $S_l^{-\mathrm{T}}$ the value of $\alpha_i^{(2)}$ is distributed over neighboring nodes when node i has level l, rather than collecting values from the neighboring nodes. The process of multiplying by S_l and S_l^{T} is summarized in Algorithm 3.1 and Algorithm 3.2. Here s_{ij} represents $\phi_j^{(2)}(x_i, y_i)$. Multiplication by S_l^{-1} and $S_l^{-\mathrm{T}}$ are the same but with $+s_{ij}$ changed to $-s_{ij}$.

```
ALGORITHM 3.1. Multiply $\alpha$ by $S_l$
  for each node $i$ with level $l$
    for each neighbor $j$ of $i$ with level $<l$
      $\alpha_i \leftarrow \alpha_i + s_{ij}\alpha_j$
    next $j$
  next $i$
```

```
ALGORITHM 3.2. Multiply $\alpha$ by $S_l^{\mathrm{T}}$
  for each node $i$ with level $l$
    for each neighbor $j$ of $i$ with level $<l$
      $\alpha_j \leftarrow \alpha_j + s_{ij}\alpha_i$
    next $j$
  next $i$
```

As with S, the rows and columns of the stiffness matrix, A, are ordered so that rows corresponding to nodes of the same level are grouped together, and smaller levels come first. The stiffness matrix will be called the *nodal matrix* or *hierarchical matrix* when we need to indicate which basis is in use. Yserentant [31] showed that the hierarchical matrix can be obtained from the nodal matrix by $A^{(\mathrm{H})}=S^{\mathrm{T}}A^{(\mathrm{N})}S$. Algorithm 3.3 shows how to change the basis of the matrix from nodal to 2-level.

```
ALGORITHM 3.3. Replace $A$ by $S_l^{\mathrm{T}}AS_l$
  for each node $i$ with level $l$
    for each neighbor $j$ of $i$ with level $<l$
      row $j \leftarrow$ row $j + s_{ij}$ * row $i$
      column $j \leftarrow$ column $j + s_{ij}$ * column $i$
    next $j$
  next $i$
```

Relaxation operator. The basis of the relaxation operator is the red-black Gauss-Seidel iteration. The red phase is done first, where the red nodes are those that are in the current grid, but not in the next coarser grid. ν_1 iterations of the relaxation operator are performed before coarse grid correction and ν_2 iterations after, allowing ν_1 and ν_2 to be multiples of $\frac{1}{2}$, where half an iteration means only the red phase. Bank, Dupont, and Yserentant [6] use $\nu_1=\nu_2=\frac{1}{2}$, i.e., they perform relaxations at the red nodes only. This V-cycle is equivalent to a symmetric Gauss-Seidel iteration using the hierarchical matrix. The condition number of the hierarchical matrix is $O(L^2)$, where L is the number of refinement levels [30], [31]. Since $L \geqq \log N$ and the number of Gauss-Seidel iterations depends on the condition number, their method requires at least $O(\log N)$ iterations to reduce the error by a given factor. To overcome this difficulty, use $\nu_1=\frac{1}{2}$ and $\nu_2=1$ by adding in relaxation at the black nodes after the coarse grid correction. This is a special case of the values of ν_1 and ν_2 considered by Braess [10]. He uses $\nu_1=\nu_2-\frac{1}{2}$, but performs the black phase first if ν_1 is an integer. Braess shows that by using $\nu_1=\frac{1}{2}$ and $\nu_2=1$, the V-cycle reduces the error by at least a factor of .5 independent of N for certain convex polygonal domains. Strong evidence is provided later in this section that for a square domain and uniform grid, the error is reduced by a factor of at least .125.

While the use of red-black Gauss-Seidel with $\nu_1=\frac{1}{2}$ and $\nu_2=1$ is an effective relaxation operator for uniform grids, it presents a problem with nonuniform grids, because the number of nodes might not grow exponentially with the number of levels. Suppose that the number of new nodes in each level, n_l, grows polynomially, i.e., $n_l=O(l^{p-1})$ for some power $p \geqq 1$. Then the total number of nodes in each level, N_l, satisfies $N_l=O(l^p)$. The number of operations used for relaxation with $\nu_1=\frac{1}{2}$ and $\nu_2=1$ is

$$\sum_{l=1}^{L} O(l^{p-1})+\sum_{l=1}^{L} O(l^p)=O(L^{p+1})=O(N_L^{1+1/p}).$$

This can be as bad as $O(N_L^2)$, which is unacceptable. To overcome this problem, the amount of relaxation performed must be restricted so that the number of operations used for the relaxation on one grid is proportional to the number of red nodes in that grid, not the total number of nodes. To achieve this, Bank, Dupont, and Yserentant perform the relaxation only at the red nodes. However, as noted earlier, this restriction is too strong and destroys the N-independence of the convergence. We propose the weaker restriction of performing the black phase only at black nodes that are immediate neighbors of red nodes. We call this *local black relaxation.* Since each red node has at most four black neighbors, the number of operations in the relaxation is proportional to the number of red nodes.

The relaxation operator is easily extended to higher order finite elements with only one minor change. To be specific, consider the spaces of C^0 pth degree polynomials over triangles. The difference for the higher order spaces is that the basis functions of the same level are not mutually orthogonal, as with the linear basis, so a simple red phase of red-black Gauss-Seidel does not solve the subsystem exactly. However, for the spaces considered, the submatrix is block diagonal with blocks of size p^2, so the subsystem can still be solved exactly by solving many small systems. For high order finite elements the bisection of a pair of triangles adds one new vertex, p^2 new nodes and p^2 new basis functions. These new nodes are the red nodes in the relaxation. Note that there are $(p+1)^2$ black nodes associated with each group of p^2 red nodes as illustrated in Fig. 3.1. The p^2 basis functions associated with the new vertex are not orthogonal to each other, but are orthogonal to all other basis functions of the same level, hence the block diagonal structure of the submatrix. Since the size of the symmetric positive definite diagonal blocks depends only on p, these small subsystems can be solved using Cholesky decomposition in a constant (with respect to N) number of operations and the $O(N)$ operation count for relaxation is maintained. Algorithms 3.4 and 3.5 give the red and black relaxation algorithms, respectively.

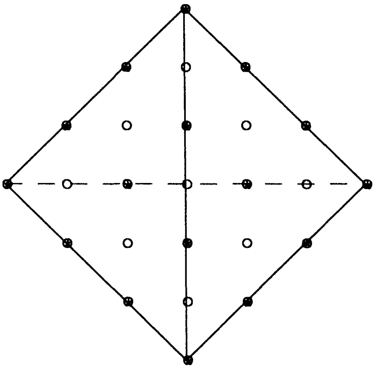


FIG. 3.1. *Red nodes* (○) *and black nodes* (●) *associated with a new vertex* (*cubic elements*).

ALGORITHM 3.4. Red relaxation

```
black list ← empty
for each vertex of level $l$
  set up and solve system for $p^2$ associated red nodes
  if black relaxation will follow then
    for each associated black node
      if black node is not on the black list then
        add to black list
      endif
    next black node
  endif
next vertex
```

ALGORITHM 3.5. Black relaxation

```
for each black node on the black list of Algorithm 3.4
  point Gauss-Seidel relaxation at this node
next node
```

Transfer operators. The other main parts of a multigrid algorithm are the two transfer operators, which are used to move between fine grids and coarse grids. The restriction operator $\mathrm{I}_{\mathrm{f}}^{\mathrm{c}}$ transfers the problem from the fine grid to the coarse grid and the prolongation operator $\mathrm{I}_{\mathrm{c}}^{\mathrm{f}}$ transfers the problem from the coarse grid to the fine grid. The transfer operators proposed turn out to be those of the Galerkin approach [29]. In this approach the restriction and prolongation operators are adjoint and the coarse grid operator, A_{c}, is related to the fine grid operator, A_{f}, by $A_{\mathrm{c}}=\mathrm{I}_{\mathrm{f}}^{\mathrm{c}}A_{\mathrm{f}}\mathrm{I}_{\mathrm{c}}^{\mathrm{f}}$. In this presentation the change between the nodal and 2-level hierarchical bases is used to describe the transfer processes. Since the definition of the transfers depends only upon the basis change, this method applies to any finite element space with a hierarchical basis. The resulting operators are very natural and of the correct order of accuracy for the approximation space being used.

Let the nodal matrix for the linear system be

$$A=\begin{bmatrix} A_{11} & A_{12}^{\mathrm{T}} \\ A_{12} & A_{22} \end{bmatrix}$$

and the nodal solution vector and right side be $x=[x_1\ \ x_2]^{\mathrm{T}}$ and $b=[b_1\ \ b_2]^{\mathrm{T}}$, where the partition is such that the second part contains values corresponding to basis functions of the highest level. Let $\tilde{A}$, etc., be the corresponding entities using the 2-level basis, S be the matrix that converts from the 2-level basis to the nodal basis, and s be the lower left submatrix of S. From $\tilde{A}=S^{\mathrm{T}}AS$ and the equivalence of $Ax=b$ and $S^{\mathrm{T}}ASS^{-1}x=S^{\mathrm{T}}b$, it follows that $\tilde{x}=S^{-1}x$ and $\tilde{b}=S^{\mathrm{T}}b$. Since the lower level of the 2-level basis is the nodal basis of the coarse grid, $\tilde{A}_{11}$ is the nodal matrix for the coarse grid. To obtain the coarse grid problem, use the lower level part of the 2-level basis fine grid problem, i.e., extract the equations corresponding to the coarse grid nodes. This is

$$[\tilde{A}_{11}\ \ \tilde{A}_{12}^{\mathrm{T}}]\begin{bmatrix}\tilde{x}_1\\ \tilde{x}_2\end{bmatrix}=[\tilde{b}_1].$$

So the problem to be solved on the coarse grid is

$$\tilde{A}_{11}\tilde{x}_1=\tilde{b}_1-\tilde{A}_{12}^{\mathrm{T}}\tilde{x}_2.$$

This process is summarized in Algorithm 3.6. Recall that the algorithms for the basis changes were given earlier.

Algorithm 3.6. Restriction

```
$A \leftarrow S^{\mathrm{T}}AS$
$x \leftarrow S^{-1}x$
$b \leftarrow S^{\mathrm{T}}b$
$b_1 \leftarrow b_1 - A_{12}^{\mathrm{T}}x_2$
```

coarse grid problem is $A_{11}x_1=b_1$

From this derivation, it is not clear that this coarse grid problem is equivalent to the standard multigrid coarse grid problem, nor is it clear what the transfer operators are. In [19], an alternate derivation of this method is provided by using the Galerkin approach

$$(\mathrm{I}_{\mathrm{f}}^{\mathrm{c}}A_{\mathrm{f}}\mathrm{I}_{\mathrm{c}}^{\mathrm{f}})(\tilde{x}_1^{\mathrm{new}}-\tilde{x}_1^{\mathrm{old}})=\mathrm{I}_{\mathrm{f}}^{\mathrm{c}}(b-A_{\mathrm{f}}x^{\mathrm{old}})$$

with the transfer operators

$$\mathrm{I}_{\mathrm{f}}^{\mathrm{c}}=[\mathrm{I}\quad s^{\mathrm{T}}]\quad\text{and}\quad \mathrm{I}_{\mathrm{c}}^{\mathrm{f}}=\begin{bmatrix}\mathrm{I}\\ s\end{bmatrix}.$$

The prolongation process is simply to restore the part of the solution due to the high level basis functions and return the system back to the nodal basis. The changes made in x_1 by solving the coarse grid problem are carried into x_2 as corrections during the basis change $x\leftarrow S\tilde{x}$. The prolongation process is given in Algorithm 3.7.

Algorithm 3.7. Prolongation

```
$b_1 \leftarrow b_1 + A_{12}^{\mathrm{T}}x_2$
$b \leftarrow S^{-\mathrm{T}}b$
$x \leftarrow Sx$
$A \leftarrow S^{-\mathrm{T}}AS^{-1}$
```

This provides all the necessary components for a multigrid iteration. Algorithm 3.8 performs one V-cycle using $\nu_1=\frac{1}{2}$ and $\nu_2=1$. L is the number of levels in the grid. The exact solve on level 1 can be performed by Cholesky decomposition or a sufficient number of Gauss-Seidel iterations.

```
ALGORITHM 3.8. V-cycle
  for level = $L$ downto 2
    red relaxation
    restriction
  next level
  exact solve on level 1
  for level = 2 to $L$
    prolongation
    red relaxation
    black relaxation
  next level
```

It is not sufficient to just know that a method requires only $O(N)$ operations; if the constant of proportionality is extremely large, the method is useless. Table 3.1 contains the asymptotic constants of proportionality for one multigrid iteration. It contains the constants for each part of the method, and for the complete V-cycle. In addition to the counts of the number of multiplications (equal to the number of additions) for each of linear, quadratic, and cubic elements, the count of the number of multiplications and additions for a specialized implementation of linear elements for Poisson's equation are provided, where one can take advantage of properties of the discrete operator.

Convergence of the multigrid iteration. In order to determine how many V-cycles are required to keep the *solution error* (the difference between the current solution vector and the exact solution of the discrete problem) of the same order as the *discretization error* (the difference between the exact solution of the discrete problem and the true solution of the continuous problem), it is necessary to know the factor by which one iteration reduces the error. In particular, the worst case reduction of the error in the energy norm is of interest, because the energy norm relates the error-reducing power of the multigrid iteration to the convergence of the discretization error through the orthogonality of the discretization error to the approximation space. This will be used in § 4.

Let V be the iteration operator for an l-level V-cycle, i.e., $e_{\text{new}} = Ve_{\text{old}}$, where e_{old} and e_{new} are the solution errors before and after the V-cycle, respectively. Let $\|\cdot\|$ be the energy norm defined by $\|x\|^2 = x^{\mathrm{T}}Ax$, where A is the stiffness matrix. Also let $\|\cdot\|$ denote the subordinate matrix norm. Define σ_l to be $\|V\|$, and σ, the *rate of convergence of the multigrid iteration*, to be $\sigma = \sup_{l \geqq 1} \sigma_l$. σ is a bound on the amount by which the energy norm of the error will be reduced by one V-cycle. The following proposition is proven in [19].

TABLE 3.1
Number of multiplications per node for the multigrid iteration.

	Poisson		Linear	Quadratic	Cubic
	mults	adds	elements	elements	elements
red relaxation	1	4	5	18	37 5/9
local black relaxation	1-4	4-16	9-36	14-37 1/4	19 2/3-43 1/9
right side basis change	1	2	2	4	6 2/9
solution basis change	1	2	2	4	6 2/9
matrix basis change	0	0	12	56	167 5/9
V-cycle total	6-9	18-30	49-76	174-197 1/4	448 5/9-472

PROPOSITION 3.1. *Let x_0 not be orthogonal to the dominant eigenspace of $V^{\mathrm{T}}AVA^{-1}$, and $x_{i+1} = V^{\mathrm{T}}AVA^{-1}x_i$ for $i \geqq 0$. Then*

$$\lim_{i\to\infty} \frac{x_{i+1}^{\mathrm{T}}x_i}{x_i^{\mathrm{T}}x_i} = \|V\|^2.$$

As a consequence, σ_l can be computed numerically using the power method. The matrix V in Proposition 3.1 can be any linear operator, so this procedure can be used to compute the rate of convergence of both the V-cycle and the 2-grid iteration, which can be compared to the known theoretical 2-grid error reduction. Table 3.2 presents the theoretical and computed 2-grid iteration error reductions and the computed V-cycle error reduction. The computed 2-grid values agree with the theoretical values, and the V-cycle also appears to be bounded by $\frac{1}{8}$. Table 3.3 shows the reduction of the energy norm of the error by one V-cycle for linear, quadratic, and cubic elements. The rate of convergence slows as the order of the elements is increased, and σ is approximately .31 for quadratics and .38 for cubics.

4. Full multigrid with adaptive refinement. The multigrid iteration provides a method for reducing the error between the approximate solution and the exact solution of the discrete problem by a factor which is bounded away from 1 independent of N using $O(N)$ operations. From an arbitrary initial guess, however, it would take

TABLE 3.2
Reduction of energy norm of error by one cycle for the model problem using linear elements.

Level of grid	2-grid theoretical	2-grid computed	V-cycle
2	.12500	.12500	.12500
3	.04419	.04419	.07329
4	.12500	.12500	.12500
5	.09857	.09857	.10297
6	.12500	.12500	.12500
7	.11793	.11793	.11823
8	.12500	.12500	.12500
9	.12320	.12320	.12330
10	.12500	.12500	.12500
11	.12455	.12455	.12463

TABLE 3.3
Reduction of energy norm of error by one V-cycle for the model problem using higher order elements.

Level of grid	Linear elements	Quadratic elements	Cubic elements
2	.125	.289	.333
3	.073	.291	.349
4	.125	.297	.363
5	.103	.301	.371
6	.125	.302	.375
7	.118	.303	.377

$O(\log N)$ iterations to reduce this error to the order of the discretization error. The *full multigrid method* provides a means to keep the solution error on the same order as the discretization error with $O(N)$ total operations. The basic idea of the full multigrid method [29] is to begin with a very coarse grid and alternately perform refinement and solution phases. At the end of each solution phase the solution error should be less than the discretization error.

For uniform grids, full multigrid is now a well-established method. The refinement phase consists of one uniform refinement of the grid (divide each triangle once) to obtain a grid of one level higher. This reduces the grid spacing h by a factor of $\sqrt{2}$ or 2, depending on the type of refinement used, and increases the number of nodes by about a factor of 2 or 4, respectively. Using linear elements and a refinement that cuts h in half as an example, the discretization error is cut in half. The solution phase must then perform enough cycles to cut the solution error at least in half, which can be done by a fixed number of cycles of the multigrid iterations. Since the number of nodes grows like 4^l where l is the number of levels, the total amount of work for this is about $\frac{4}{3}$ the amount of work done on the final grid, which is $O(N)$.

For adaptive grids, the number of nodes need not grow geometrically with the number of levels. If one used the full multigrid method the way it is used for uniform grids, the number of operations can be larger than $O(N)$. In the worst case where the number of nodes is proportional to the number of levels, the operation count is $O(N^2)$. In the usual approach used to overcome this [4], [26], the grid is refined to get one level higher, but if the number of nodes has not been increased by a factor of 2 or 4 (depending on the type of refinement used) the refinement is repeated rather than moving on to the solution phase. While this approach does result in an $O(N)$ algorithm, it does not flow smoothly, has unnecessary overhead in starting and ending refinements, and may overshoot the target increase factor of 2 or 4.

Since the basic step of the refinement of § 2 is the division of one pair of triangles and compatibility is always present, a more elegant approach to the full multigrid method can be taken. The refinement phase proceeds until the number of nodes has been increased by exactly some given factor f. Moreover, there is no reason for f to be the factor 2 or 4 from uniform refinement, so f can be any real number larger than 1. The solution phase consists of performing v V-cycles (multigrid iterations) where v is large enough to keep the solution error smaller than the discretization error.

The crucial missing element of the algorithm is a method for determining when to switch from one phase to the other, i.e., a way of determining reasonable values for f and v. The most efficient values of f and v in terms of α, the rate of convergence of the discretization error, and σ, the rate of convergence of the multigrid iteration, are determined in this section.

For uniform grids, the rate of convergence of the discretization error is usually given in terms of h, a measure of the size of the triangles. A method is said to have order 2α if the energy norm of the error decreases like $O(h^{2\alpha})$ as h gets small. For adaptively refined grids, h is not such a meaningful entity, but N can be used to measure the rate of convergence of the discretization error. For a uniform grid, $N = O(h^{-2})$ so the error decreases like $O(N^{-\alpha})$ as N gets large. Thus, α is said to be the *rate of convergence of the discretization error* if α is the largest value such that the discretization error is $O(N^{-\alpha})$, i.e., $\|u - u_N\| \sim cN^{-\alpha}$ for some constant c, where u is the true solution of the differential equation, u_N is the exact solution of the discrete problem with N nodes, and $\|\cdot\|$ is the energy norm. Normally, $\alpha = \frac{1}{2}$, 1, and $\frac{3}{2}$ for linear, quadratic, and cubic elements, respectively. Recall that σ is the rate of convergence of the multigrid iteration defined to be a bound on the factor by which the energy norm of the solution error is reduced in one V-cycle of the multigrid iteration, where the solution error is $\tilde{u}_N - u_N$ and $\tilde{u}_N$ is the approximate solution.

THEOREM 4.1. *Let α be the rate of convergence of the discretization error and f be the factor by which the refinement phase increases the number of nodes. Then, asymptotically, the solution error will remain less than the discretization error if the solution phase reduces the solution error by a factor of at least* $(2f^{2\alpha}-1)^{-1/2}$.

Proof. The true solution of the partial differential equation, u, lies in a Hilbert space $\mathbf{H}$ endowed with the energy inner product $\langle\cdot,\cdot\rangle$ and the subordinate energy norm $\|\cdot\|$. Let $\mathbf{S}_N \subseteq \mathbf{H}$ be the space of C^0 piecewise pth degree polynomials over the triangulation with N nodes, and $\mathbf{S}_{fN}$ be the space associated with the refined triangulation with fN nodes. $\mathbf{S}_N \subseteq \mathbf{S}_{fN}$ since the triangulation with fN nodes is a refinement of the triangulation with N nodes. Let u_N be the exact solution of the discrete problem in $\mathbf{S}_N$, i.e., u_N is the unique function in $\mathbf{S}_N$ such that $\langle u-u_N, w\rangle = 0\ \forall w \in \mathbf{S}_N$, and let u_{fN} be the exact solution of the discrete problem in $\mathbf{S}_{fN}$. Let $\tilde{u}_N$ be the approximate solution in $\mathbf{S}_N$. We assume that $\|\tilde{u}_N - u_N\| \leqq \|u-u_N\|$. We then wish to find a $\tilde{u}_{fN} \in \mathbf{S}_{fN}$ such that $\|\tilde{u}_{fN} - u_{fN}\| \leqq \|u - u_{fN}\|$ so that the solution phase will keep the solution error smaller than the discretization error.

From the definition of α, we obtain

$$\frac{\|u-u_{fN}\|}{\|u-u_N\|} \sim \frac{c(fN)^{-\alpha}}{cN^{-\alpha}} = f^{-\alpha}.$$

Thus $\|u-u_N\| \sim f^{\alpha}\|u-u_{fN}\|$.

Since $\tilde{u}_N \in \mathbf{S}_N$ and $\tilde{u}_N \in \mathbf{S}_{fN}$, we have the Pythagorean identities

$$\|\tilde{u}_N - u_N\|^2 + \|u-u_N\|^2 = \|\tilde{u}_N - u\|^2$$

and

$$\|\tilde{u}_N - u_{fN}\|^2 + \|u-u_{fN}\|^2 = \|\tilde{u}_N - u\|^2$$

and thus, using $\|\tilde{u}_N - u_N\| \leqq \|u-u_N\|$

$$\begin{aligned}\|\tilde{u}_N - u_{fN}\|^2 &= \|\tilde{u}_N - u_N\|^2 + \|u-u_N\|^2 - \|u-u_{fN}\|^2\\ &\leqq 2\|u-u_N\|^2 - \|u-u_{fN}\|^2\\ &\sim 2f^{2\alpha}\|u-u_{fN}\|^2 - \|u-u_{fN}\|^2\\ &= (2f^{2\alpha}-1)\|u-u_{fN}\|^2.\end{aligned}$$

Therefore, to insure that $\|\tilde{u}_{fN} - u_{fN}\| \leqq \|u-u_{fN}\|$ we must reduce the error $\|\tilde{u}_N - u_{fN}\|$ by a factor of $(2f^{2\alpha}-1)^{-1/2}$. □

COROLLARY. *For given α, σ and number of* V*-cycles v, f must be bounded by*

$$f \leqq \left(\frac{\sigma^{-2v}+1}{2}\right)^{1/2\alpha}.$$

Let $N_r = c_1 f^r$ be the number of nodes in the triangulation after r refinement phases, $c_2 N_r$ be the (asymptotic) number of operations used by one V-cycle on a triangulation with N_r nodes and the final triangulation be the result of R refinement phases. Then, the number of operations used by the full multigrid solution is

$$\sum_{r=1}^{R} vc_2N_r = \sum_{r=1}^{R} vc_2c_1f^r = vc_2c_1\frac{f(f^R-1)}{f-1} \sim v\frac{f}{f-1}c_2c_1f^R = v\frac{f}{f-1}c_2N_R.$$

For given v, the operation count is minimized by using the largest possible f. Thus one should choose $f = ((\sigma^{-2v}+1)/2)^{1/2\alpha}$. The most efficient choice of v is then given by the v that minimizes $v(f/(f-1))$ with this choice of f. Clearly, this is an increasing function of v when v is sufficiently large. Thus, for given α and σ, one can compute $v(f/(f-1))$ for a few small positive integer values of v to determine the most efficient choice for f and v. Usually, $v=1$ is best. Table 4.1 gives these values for linear, quadratic, and cubic elements. The last column contains the value of $v(f/(f-1))$. This represents the amount of work required with respect to one V-cycle on the finest grid, e.g., with quadratic elements the full multigrid is slightly faster than two V-cycles.

5. Numerical results. The method described in this paper has been implemented in a FORTRAN program to solve (1.1). Computations were performed on a Pyramid 90x with a floating point accelerator operating under the Pyramid Technology OSx 3.1 Operating System, which is a dual port of AT&T Bell Laboratories' System V Release 2.0 and the University of California, Berkeley's 4.2BSD. The Pyramid Technology Optimizing FORTRAN 77 compiler was used with single precision, which has about seven decimal digits.

Three sample problems were solved to demonstrate the method. The first two problems contain a singularity in the solution; the third problem illustrates a complicated differential operator.

Problem 1. Laplace's equation on the L-shaped domain of Figure 5.1(a) with the Dirichlet boundary conditions chosen so that the true solution is $r^{2/3}\sin(2\theta/3)$. Both x and y range from -1 to 1 and the reentrant corner is located at the origin. Figure 5.1(a) shows a sample adaptively refined grid with the initial six triangles in bold. The solution exhibits the leading term of the singularity due to the 270° reentrant corner.

Problem 2. Laplace's equation on the hexagonal domain of Fig. 5.1(b). The domain has a slit along the positive x axis. x ranges from -1 to 1 and y from $-\sqrt{3}/2$ to $\sqrt{3}/2$. The Dirichlet boundary conditions are chosen so that the true solution is $r^{1/2}\sin(\theta/2)$. The reentrant corner is located at the origin. Figure 5.1(b) shows a sample adaptively refined grid with the initial six triangles in bold. The solution exhibits the leading term of the singularity due to the 360° reentrant corner.

Problem 3. This is Problem 54 in the elliptic PDE population of Rice et al. [21], [22]. The differential equation is

$$((1+x^2)u_x)_x+((1+A^2)u_y)_y-(1+(8y-x-4)^2)u=f$$

on the unit square, where $A=4y^2+.9$. The right-hand side and Dirichlet boundary conditions are chosen so that the exact solution is

$$2.25x(x-A)^2(1-D)/A^3+1/(1+(8y-x-4)^2)$$

where

$$B=\max\{0,(3-x/A)^3\}$$
$$C=\max\{0,x-A\}$$
$$D=\begin{cases}0 & \text{if } C<.02\\ e^{-B/C} & \text{if } C\geqq .02.\end{cases}$$

TABLE 4.1
Optimal choices for v and f for the model problem.

Type of elements	α	σ	v	f	$vf/(f-1)$
linear	.5	.125	1	32.5	1.03
quadratic	1.0	.31	1	2.39	1.72
cubic	1.5	.38	1	1.58	2.72

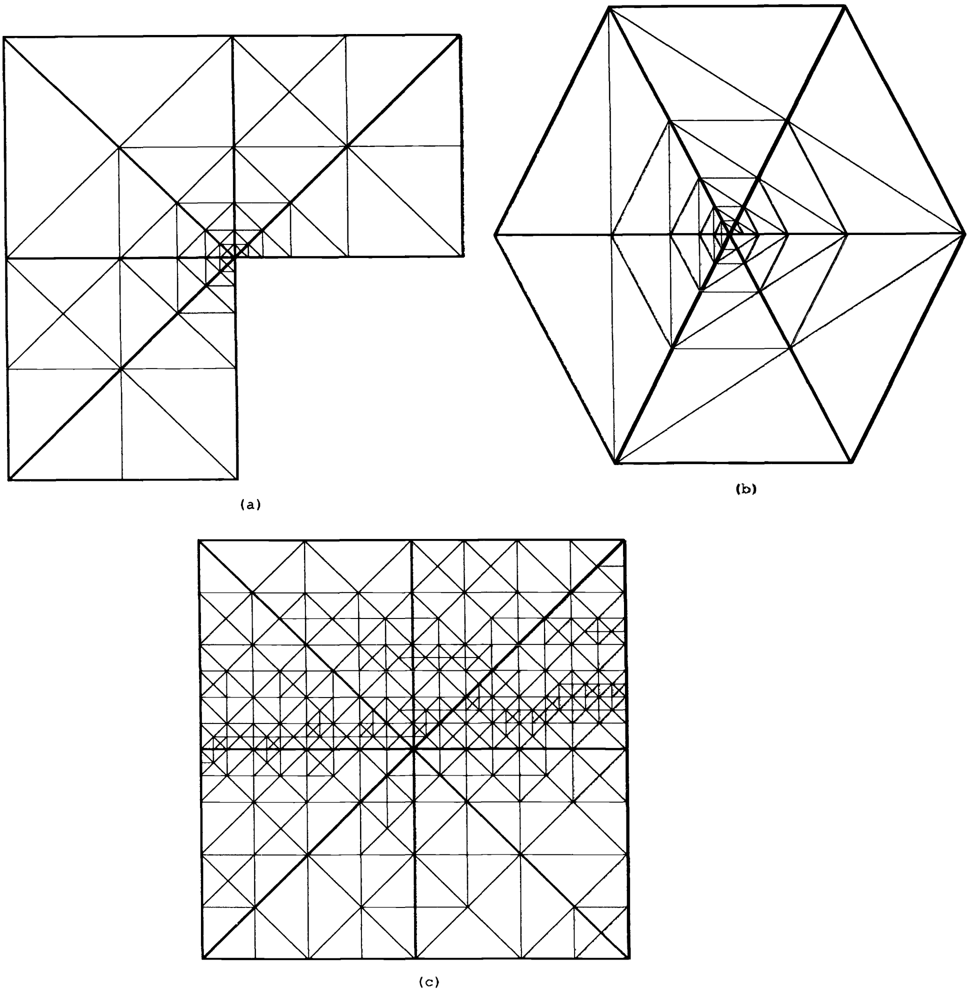


FIG. 5.1. *Domains and sample grids for* (a) *Problem* 1, (b) *Problem* 2, (c) *Problem* 3.

Figure 5.1(c) shows a sample adaptively refined grid with the initial eight triangles in bold. A contour plot of the solution can be found in [21] or [22]. The solution has a ridge in the vicinity of $y \approx .6-.7$.

Convergence of the discretization error. With uniform refinement, the rate of convergence of the discretization error depends on the smoothness of the solution. For Problems 1 and 2, the best one can hope for is $\alpha = \frac{1}{2}$ and $\frac{1}{4}$, respectively, no matter what degree polynomials are used. It is possible for adaptive refinement to recover the optimal rate of convergence. Problems 1 and 2 were solved using linear, quadratic, and cubic elements with both uniform and adaptively refined grids. For these solutions, one V-cycle was used for the solution phase, and the number of vertices was increased by the factor $f = 4$, 2.39, and 1.58 for linear, quadratic, and cubic elements, respectively, except for Problem 2, where $f = 2$ was used for quadratics. $f = 4$ was used instead of 32.5 for linear elements to give a sufficient number of data points for the graphs. The results are presented in Figs. 5.2, 5.3, and 5.4 for Problems 1, 2, and 3, respectively.

The data points on the graphs are labeled with A, B, and C for linear, quadratic, and cubic elements, respectively, for the uniform grids, and 1, 2, and 3 for linear, quadratic, and cubic elements, respectively, for the adaptive grids. The observed rate of convergence is given by the slope of a linear least squares fit of the data. When appropriate, some of the first data points were discarded in determining the slope. These slopes are given in Tables 5.1 and 5.2.

For uniform grids, the rate of convergence is about $\frac{1}{3}$ and $\frac{1}{4}$ for Problems 1 and 2, respectively, for all three polynomial degrees. For adaptive grids, the rate of convergence is about $\frac{1}{2}$, 1, and $\frac{3}{2}$ for linear, quadratic, and cubic elements, respectively, for both problems. For Problem 3 the rate of convergence is slightly larger than $\frac{1}{2}$, 1, and $\frac{3}{2}$ for linear, quadratic, and cubic elements, respectively, for both uniform and adaptive grids. Although the uniform grids achieve the optimal order of convergence for Problem 3, the adaptive grids have a smaller constant of proportionality. The convergence of

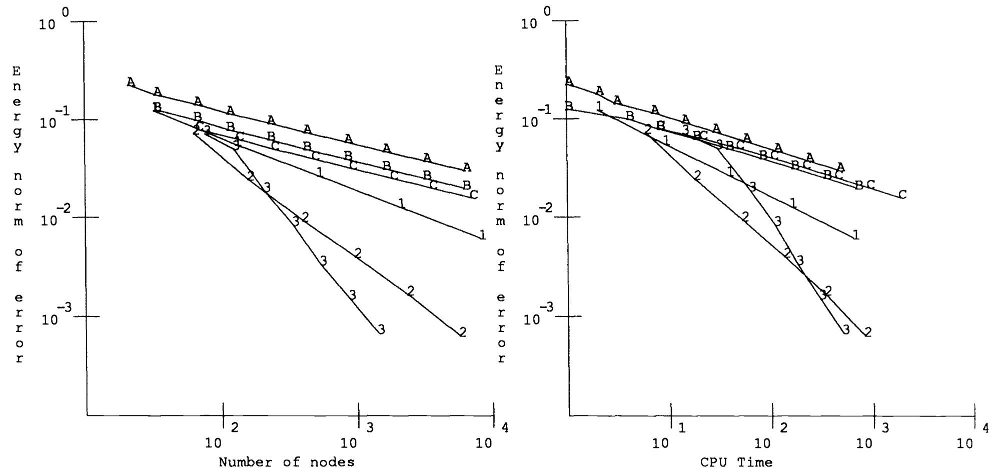


FIG. 5.2. *Results for Problem* 1.

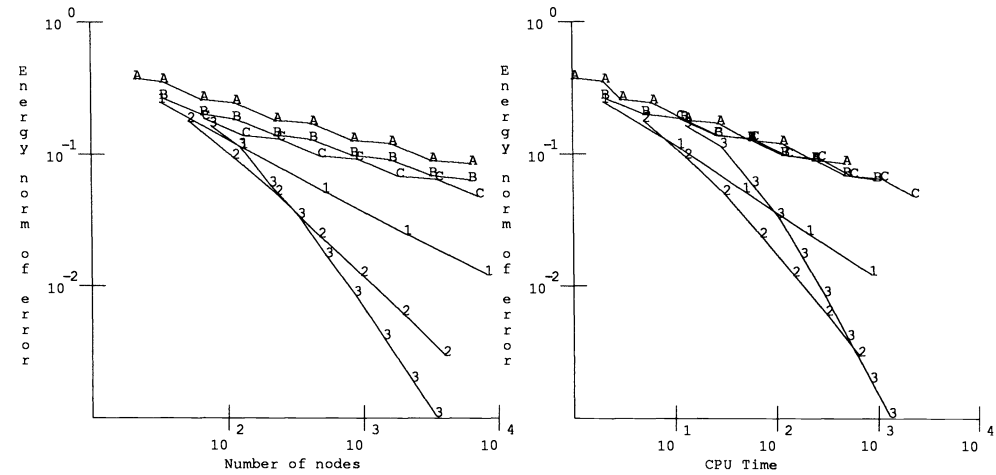


FIG. 5.3. *Results for Problem* 2.

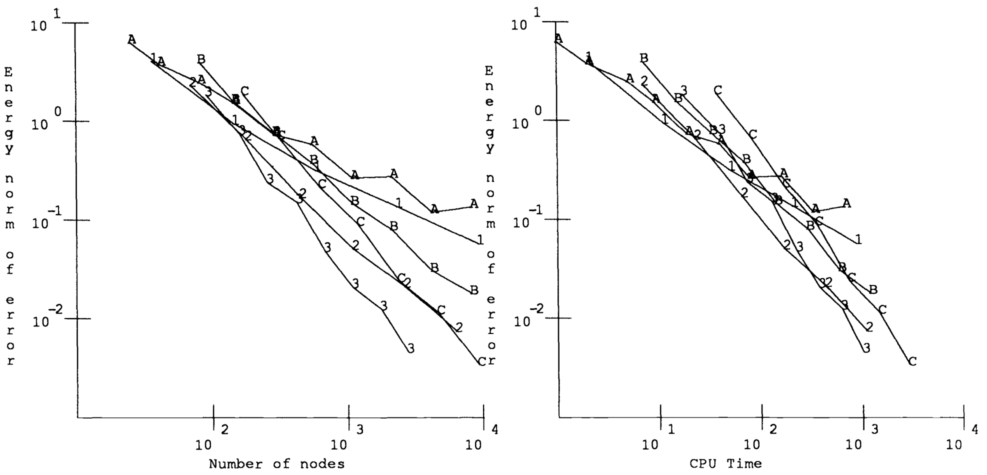


FIG. 5.4. *Results for Problem* 3.

TABLE 5.1
Observed order of convergence with uniform grids.

Problem	Linear elements	Quadratic elements	Cubic elements
1	.355	.355	.349
2	.284	.273	.264
3	.697	1.131	1.555

TABLE 5.2
Observed order of convergence with adaptive grids.

Problem	Linear elements	Quadratic elements	Cubic elements
1	.540	1.011	1.633
2	.542	.967	1.496
3	.616	1.159	1.680

the error with CPU time is also of optimal order for the adaptive grids, illustrating the optimality of the full multigrid method.

In numerical experiments that compare low and high order methods with uniform grids for problems with well-behaved solutions (e.g., [22]) it is usually observed that for very low accuracy it is more efficient to use linear elements, but for moderate and high accuracy the high order elements are more efficient. The same result is observed when using adaptively refined grids for Problems 1 and 2.

Convergence of the multigrid iteration. The effect of several factors on the rate of convergence of the multigrid iteration was also considered. Proposition 3.1 was used to determine the rate of convergence for the L-shaped domain of Problem 1 with linear elements. This was computed using both a uniform grid and an adaptively refined grid. For the adaptive grid three forms of relaxation are compared:

(i) full black, in which relaxtion is at all the black nodes after solving the coarse grid problem ($\nu_1 = \frac{1}{2}, \nu_2 = 1$);

(ii) local black, in which relaxation is at the black nodes that are neighbors of red nodes as discussed in § 3;

(iii) no black relaxation ($\nu_1 = \frac{1}{2}, \nu_2 = \frac{1}{2}$).

The results of these computations are presented in Table 5.3.

Note first that the reentrant corner has a pronounced effect on the rate of convergence. The only difference between the uniform grid here and the V-cycle in Table 3.2 is the shape of the domain, yet the rate of convergence has slowed from 0.125 to about 0.2. The use of a nonuniform grid has almost no effect on the asymptotic rate of convergence. Using local black relaxation slows the rate of convergence very slightly. The difference is small enough to ignore, especially when one considers that the full black relaxation requires more than $O(N)$ operations for a highly nonuniform grid. When no black relaxation is performed, the rate of convergence deteriorates rapidly. The experiments show that the rate of convergence behaves like $1 - O(1/\log N)$, so it is not bounded away from 1, and, in fact, $O(\log N)$ iterations are required to reduce the error by a given constant.

6. Conclusion. In this paper, we presented a unified multilevel adaptive refinement method for elliptic PDEs. The adaptive refinement is done by bisecting pairs of triangles, corresponding to the addition of a hierarchical basis function. An estimate of the coefficient of the new basis function is used as an error indicator to select triangle pairs for division. The multigrid iteration uses the change between the nodal basis and 2-level hierarchical basis for the grid transfers. Red-black Gauss-Seidel relaxations are used, with local black relaxations performed only at black nodes that neighbor red nodes, to keep the operation count at $O(N)$. The full multigrid alternately performs refinement and multigrid iterations, with the amount of refinement and number of iterations determined to be optimal while keeping the solution error smaller than the

TABLE 5.3
Rate of convergence of the multigrid iteration with the L*-shaped domain and linear elements.*

	Uniform grid		Adaptive grid			
Levels	Nodes	σ	Nodes	Full black	Local black	No black
3	21	.081	13	.081	.081	.188
4	33	.127	18	.070	.075	.217
5	65	.137	20	.081	.093	.256
6	113	.154	25	.084	.097	.273
7	225	.165	27	.090	.103	.289
8	417	.177	40	.086	.109	.347
9	833	.187	53	.105	.129	.419
10	1601	.196	78	.118	.143	.498
11	3201	.205	82	.120	.146	.498
12			114	.141	.162	.544
13			116	.135	.163	.544
14			155	.139	.168	.580
15			159	.136	.169	.580
16			312	.178	.200	.645
17			316	.173	.201	.645
18			443	.192	.212	.679
19			459	.186	.212	.679
20			606	.194	.217	.705

discretization error. The method was developed for the high order finite element spaces of C^0pth degree piecewise polynomials. Numerical computations were performed with linear, quadratic, and cubic elements to demonstrate the optimality of the adaptive grids in the presence of singularities, and the optimality of the full multigrid solution method.

REFERENCES

[1] I. Babuška and A. K. Aziz, *On the angle condition in the finite element method*, SIAM J. Numer. Anal., 13 (1976), pp. 214-226.

[2] I. Babuška and W. 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 Advances in Computer Methods for Partial Differential Equations III, R. Vichnevetsky and R. S. Stepleman, eds., IMACS, 1979, pp. 33-39.

[4] ——— *An adaptive multilevel method for elliptic boundary value problems*, Computing, 26 (1981), pp. 91-105.

[5] R. E. Bank and A. Weiser, *Some a posteriori error estimators for elliptic partial differential equations*, Math. Comp., 44 (1985), pp. 283-301.

[6] R. E. Bank, T. F. Dupont, and H. Yserentant, *The hierarchical basis multigrid method*, Preprint SC-87-1, Konrad-Zuse-Zentrum für Informationstechnik, 1987.

[7] G. Birkhoff and R. E. Lynch, *Numerical Solution of Elliptic Problems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1984.

[8] D. Braess, *The contraction number of a multigrid method for solving the Poisson equation*, Numer. Math., 37 (1981), pp. 387-404.

[9] D. Braess and W. Hackbusch, *A new convergence proof for the multigrid method including the V-cycle*, SIAM J. Numer. Anal., 20 (1983), pp. 967-975.

[10] D. Braess, *The convergence rate of a multigrid method with Gauss-Seidel relaxation for the Poisson equation*, Math. Comp., 42 (1984), pp. 505-519.

[11] A. Brandt, *Multi-level adaptive solutions to boundary-value problems*, Math. Comp., 31 (1977), pp. 333-390.

[12] ——— *Multi-level adaptive techniques (MLAT) for partial differential equations: Ideas and software*, in Mathematical Software III, J. R. Rice, ed., Academic Press, New York, 1977, pp. 277-318.

[13] A. W. Craig and O. C. Zienkiewicz, *A multigrid algorithm using a hierarchical finite element basis*, in Multigrid Methods for Integral and Differential Equations, D. J. Paddon and H. Holstein, eds., Clarendon Press, Oxford, 1985, pp. 301-312.

[14] J. P. De, S. R. Gago, D. W. Kelly, O. C. Zienkiewicz, and I. Babuška, *A posteriori error analysis and adaptive processes in the finite element method: Part* II—*Adaptive mesh refinement*, Internat. J. Numer. Methods Engrg., 19 (1983), pp. 1621-1656.

[15] I. Fried, *Condition of finite element matrices generated from nonuniform meshes*, AIAA J., 10 (1972), pp. 219-221.

[16] D. Kamowitz and S. Parter, *On MGR[ν] multigrid methods*, SIAM J. Numer. Anal., 24(1987), pp. 366-381.

[17] D. W. Kelly, J. P. De, S. R. Gago, O. C. Zienkiewicz, and I. Babuška, *A posteriori error analysis and adaptive processes in the finite element method: Part I—Error analysis*, Internat. J. Numer. Methods Engrg., 19 (1983), pp. 1593-1619.

[18] W. F. Mitchell, *A comparison of adaptive refinement techniques for elliptic problems*, ACM Trans. Math. Software, 15 (1989), pp. 326-347.

[19] ——— *Unified multilevel adaptive finite element methods for elliptic problems*, Ph.D. thesis, Report No. UIUCDCS-R-88-1436, Department of Computer Science, University of Illinois, Urbana, IL, 1988.

[20] W. C. Rheinboldt, *On a theory of mesh-refinement processes*, SIAM J. Numer. Anal., 17 (1980), pp. 766-778.

[21] J. R. Rice, E. N. Houstis, and W. R. Dyksen, *A population of linear, second order elliptic partial differential equations on rectangular domains*, Math. Comp., 36 (1981), pp. 475-484.

[22] J. R. Rice and R. F. Boisvert, *Solving Elliptic Problems Using* ELLPACK, Springer-Verlag, New York, 1985.

[23] M. Ries, U. Trottenberg, and G. Winter, *A note on MGR methods*, Linear Algebra Appl., 49 (1983), pp. 1-26.

[24] M. C. RIVARA, *Mesh refinement processes based on the generalized bisection of simplices*, SIAM J. Numer. Anal., 21 (1984), pp. 604-613.

[25] ——— *Algorithms for refining triangular grids suitable for adaptive and multigrid techniques*, Internat. J. Numer. Methods Engrg., 20 (1984), pp. 745-756.

[26] ——— *Design and data structure of fully adaptive, multigrid, finite-element software*, ACM Trans. Math. Software, 10 (1984), pp. 242-264.

[27] E. G. SEWELL, *Automatic generation of triangulations for piecewise polynomial approximation*, Ph.D. thesis, Purdue University, West Lafayette, IN, 1972.

[28] ——— *A finite element program with automatic user-controlled mesh grading*, in Advances in Computer Methods for Partial Differential Equations III, R. Vichnevetsky and R. S. Stepleman, eds., IMACS, 1979, pp. 8-10.

[29] 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.

[30] H. YSERENTANT, *Hierarchical bases give conjugate gradient type methods a multigrid speed of convergence*, Appl. Math. and Comput., 19 (1986), pp. 347-358.

[31] ——— *On the multi-level splitting of finite element spaces*, Numer. Math., 49 (1986), pp. 379-412.

[32] O. C. ZIENKIEWICZ, D. W. KELLY, J. GAGO, AND I. BABUŠKA, *Hierarchical finite element approaches, error estimates and adaptive refinement*, in The Mathematics of Finite Elements and Applications IV, J. R. Whiteman, ed., Academic Press, New York, 1982, pp. 313-346.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 1, pp. 168–193, January 1992

010

ROW PROJECTION METHODS FOR LARGE NONSYMMETRIC LINEAR SYSTEMS*

R. BRAMLEY† AND A. SAMEH†

Abstract. Three conjugate gradient accelerated row projection (RP) methods for nonsymmetric linear systems are presented and their properties described. One method is based on Kaczmarz's method and has an iteration matrix that is the product of orthogonal projectors; another is based on Cimmino's method and has an iteration matrix that is the sum of orthogonal projectors. A new RP method, which requires fewer matrix-vector operations, explicitly reduces the problem size, is error reducing in the two-norm, and consistently produces better solutions than other RP algorithms, is also introduced. Using comparisons with the method of conjugate gradient applied to the normal equations, the properties of RP methods are explained.

A row partitioning approach is described that yields parallel implementations suitable for a wide range of computer architectures, requires only a few vectors of extra storage, and allows computing the necessary projections with small errors. Numerical testing verifies the robustness of this approach and shows that the resulting algorithms are competitive with other nonsymmetric solvers in speed and efficiency.

Key words. iterative methods, nonsymmetric linear systems, Kaczmarz, Cimmino

AMS(MOS) subject classifications. 65, 15

1. Introduction.

Over the last few years much research effort has been invested in developing iterative solvers for nonsymmetric linear systems $Ax = b$, where A is large, sparse, and nonsingular. These solvers can be grouped roughly into the four categories of *matrix splitting*, *CG-like*, *residual polynomial*, and *symmetrization* methods.

Matrix splitting methods and their acceleration via CG include the earliest iterative techniques for solving linear systems, and are based on splitting the coefficient matrix as $A = M - N$. This category includes the Jacobi, Gauss–Seidel, and successive overrelaxation methods, and convergence is assured if the spectral radius $\rho(M^{-1}N)$ is less than 1. This condition can often be shown to hold if, e.g., A is irreducible and diagonally dominant. Hageman and Young [18] describe many of these splitting techniques and provide other conditions on A that imply convergence.

The second category, CG-like methods, was motivated by the success of the conjugate gradient (CG) method for symmetric positive definite systems. Generalized conjugate residuals (GCR), Orthomin, generalized minimum residual (GMRES), Axelsson's method, and Orthodir are included in this category (see Saad and Schultz [31] for a summary of these methods). Like CG, these solvers generate a Krylov subspace K using only matrix-vector products with A and then enforce some minimization or orthogonality property on K; they differ primarily in how the basis of K is formed and which inner product is used to define orthogonality or minimality. Since computation and storage grow with the iteration number for these methods, they are most often used in a truncated or restarted form. In 1982, Elman [14] proved that restarted versions of many of these methods converge provided that the symmetric part $(A + A^T)/2$ of A is positive definite. In spite of this restriction, the restarted

* Received by the editors October 2, 1989; accepted for publication (in revised form) October 24, 1990. This work was supported by Department of Energy grant DE-FG02-85ER25001; AT&T grant AFFL-67-SAMEH; Air Force Office of Scientific Research grant AFOSR-85-0211; National Science Foundation grants NSF-MIP-8410110, NSF-CCR-8717942, and CCR-900000N(NCSA); and the Digital Equipment Corporation.

† Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801.

GMRES(k) algorithm in particular is currently one of the more popular nonsymmetric solvers.

The second category generates residuals $r_k = b - Ax_k$ that satisfy

$$r_k = p_k(A)r_0, \tag{1}$$

where r_0 is the initial (restarted) residual and $p_k(\lambda)$ is a residual polynomial, i.e., $p_k(0) = 1$. A third category, residual polynomial methods, directly uses this idea by finding a convex region in the complex plane containing the eigenvalues of A. Iterates are then formed that satisfy (1) for some class of polynomials. Chebyshev polynomials [3], [27], [28] use ellipses for the convex region and Chebyshev polynomials to define $p_k(\lambda)$. Least squares polynomial methods [32] use an approximation of the convex hull cvx $(\sigma(A))$ of the extremal eigenvalues of A for the enclosing region and polynomials $p_k(\lambda)$ that minimize a weighted sum of squares on the boundary of cvx $(\sigma(A))$. Because of the restriction $p_k(0) = 1$, residual polynomial methods fail when the origin is in cvx $(\sigma(A))$.

All three categories above are thus restricted in applicability, requiring, e.g., $A + A^T$ to be positive definite or the spectrum of A to lie on one side of the imaginary axis. Symmetrization methods, the fourth category, implicitly or explicitly create a related symmetric positive definite system and then use one of the powerful iterative methods applicable to such systems. The most popular such approach is to use CG on the normal equations $A^TAx = A^Tb$. Actually forming A^TA can cause a loss of information and entail a large preprocessing cost as well as increased storage. Even if this is not done, a more serious problem is that the condition number $\kappa(A^TA)$ is the square of $\kappa(A)$. This can lead to outright failure of the solver.

In general, most nonsymmetric solvers either require storage and computation that grow excessively with the iteration number, special spectral properties of A to assure convergence, or a symmetrization process with potentially disastrous effects on the coefficient matrix. One group of methods that avoids these difficulties is that of accelerated row projection (RP) algorithms. Partition $A \in \Re^{N\times N}$ into m block rows:

$$A^T = [A_1, A_2, \cdots, A_m], \tag{2}$$

and partition the vector b conformally. A *row projection* (RP) method is any algorithm that requires the computation of the orthogonal projections $P_ix = A_i(A_i^TA_i)^{-1}A_i^Tx$ of a vector x onto range(A_i), $i = 1, 2, \cdots, m$. Note that the nonsingularity of A implies that A_i has full rank and so $(A_i^TA_i)^{-1}$ exists.

This paper presents three such methods and describes their properties. The first (KACZ) has an iteration matrix formed from the product of orthogonal projectors. A new method (V-RP) is derived from KACZ to reduce the number of matrix-vector operations needed and to reduce the problem size explicitly. The third RP method (CIMM) uses the sum of orthogonal projectors. Conjugate gradient (CG) acceleration is used on all three, and for KACZ this is shown to allow a reduction in the amount of work needed per iteration, while for V-RP an error-reducing algorithm results. Most importantly, the underlying relationship between RP methods and CG applied to the normal equations is shown. This provides an explanation for the behavior of RP methods, a basis for comparing them, and a guide for their effective use.

Possibly the most important implementation issue for RP methods is that of choosing the row partitioning that defines the projectors. An approach is shown for three-dimensional elliptic partial differential equations that yields algorithms with large scale parallelism, requires only a few extra vectors of storage, and allows the computation of the necessary projections with small errors. Numerical testing shows that the algorithms have superior robustness and can be competitive with other nonsymmetric solvers in speed and efficiency.

2. Three row projection methods.

2.1. Row projection method KACZ. The simplest RP method can be derived geometrically. Let $H_i = \{x : A_i^T x = b_i\}$ be the affine set of solutions to the ith block row of equations. The solution x^* to $Ax = b$ is the unique intersection point of those affine sets, and the method of successive projections gives the iteration

$$x_{k+1} = Q_u x_k + b_u = (I - P_m)(I - P_{m-1}) \cdots (I - P_1)x_k + b_u, \tag{3}$$

where

$$\begin{aligned} b_u = \hat{b}_m + (I - P_m)\hat{b}_{m-1} + (I - P_m)(I - P_{m-1})\hat{b}_{m-2} + \cdots \\ +(I - P_m) \cdots (I - P_2)\hat{b}_1, \end{aligned}$$

and $\hat{b}_i = A_i(A_i^T A_i)^{-1} b_i$.

In 1939 Kaczmarz [20] proposed iteration (3) and proved convergence for the $m = N$ case where each block row A_i^T consists of a single row of A. Since then many authors [10], [11], [19], [29], [30], [37] have examined the convergence of related iterative methods. The theoretical robustness of (3) is remarkable and the iteration converges even when A is singular or rectangular. However, as with any linear stationary process, the rate of convergence is determined by the spectral radius of Q_u and can be arbitrarily slow. For this reason Elfving [12] and Björck and Elfving [4] proposed symmetrizing Q_u by following a forward sweep through the rows with a backward sweep, and introduced an iteration parameter to get

$$x_{k+1} = Q(\omega)x_k + T(\omega)b, \tag{4}$$

$$Q(\omega) = (I - \omega P_1)(I - \omega P_2) \cdots (I - \omega P_m)^2 \cdots (I - \omega P_2)(I - \omega P_1), \tag{5}$$

with $T(\omega)$ defined by (8). When A is nonsingular and $0 < \omega < 2$, the spectrum $\sigma(I - Q(\omega))$ lies in the interval $(0, 1]$ and so the CG method can be applied to solve

$$(I - Q(\omega))x = T(\omega)b. \tag{6}$$

Also, (4) is equivalent to applying block SSOR to

$$\left\{ \begin{array}{rcl} AA^T y & = & b \\ x & = & A^T y \end{array} \right. \tag{7}$$

where the blocking is that induced by the row partitioning of A [12]. This gives a simple expression for $T(\omega)$:

$$T(\omega) = A^T(D + \omega L)^{-T} D (D + \omega L)^{-1}, \tag{8}$$

where $AA^T = L + D + L^T$ is the usual splitting into block lower triangular, diagonal, and upper triangular parts. This also shows that solving (6) using the CG algorithm can be placed in the category of symmetrization methods.

Although Björck and Elfving tested an accelerated RP algorithm, their work concentrated on the single row ($m = N$) case applied to weighted least squares problems.

Previous work dealing with the use of block forms is given by Kamath and Sameh [21], [22]. Using sample problems drawn from two-dimensional nonselfadjoint elliptic partial differential equations, they numerically examined the issues of suitable row partitionings and methods for the numerical solution of the induced projections, primarily for the $m = 2$ or 3 case. By comparisons with CG-like methods and preconditioned CG applied to the normal equations, they showed that the RP algorithm solved selected problems for which most of the other methods failed.

The first implementation issue is the choice of ω in (6). Normally the "optimal" ω is defined as the $\omega_{\min}$ that minimizes the spectral radius of $Q(\omega)$. In [22] $\omega_{\min} = 1$ is proven for the case where A is partitioned into two block rows, i.e., $m = 2$. This is no longer true for $m > 3$, as can be seen by considering

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} A_1^T \\ A_2^T \\ A_3^T \end{bmatrix}.$$

For $Q(\omega)$ defined in (4) it can be proven that

$$\begin{aligned} \rho(Q(1)) &= (7+\sqrt{17})/16 = 0.69519 \pm 10^{-5}, \\ \rho(Q(0.9)) &\leq 0.68611 \pm 10^{-5} < \rho(Q(1)), \end{aligned}$$

and so $\omega_{\min} \neq 1$.

Throughout this paper, however, $\omega = 1$ will be used for three reasons. Because CG acceleration is to be applied, the entire distribution of the spectrum must be considered, not simply the spectral radius. When $\omega = 1$, many of the eigenvalues of the system matrix in (6) are exactly 1:

FACT 2.1. *At least* $\operatorname{rank}(A_1)$ *of the eigenvalues of* $Q(1)$ *are zero.*

Proof. $(I - P_1)x = 0$ for $x \in \operatorname{range}(P_1) = \operatorname{range}(A_1)$. From the definition of $Q(\omega)$, $\operatorname{null}(I - P_1) \subseteq \operatorname{null}(Q(1))$. □

Since CG acceleration requires in exact arithmetic a number of iterations equal to the number of distinct eigenvalues, this suggests that numerically fewer iterations are needed as compared to when $\omega \neq 1$. A second reason is that numerical experience shows $\rho(Q(\omega))$ is not sensitive to changes in ω. This matches classical results for SSOR iterations, which are not as sensitive to ω as the corresponding SOR methods. The small improvement that does result from choosing $\omega_{\min}$ is more than offset by the introduction of extra nonzero eigenvalues. The third reason for using $\omega = 1$ is given by Fact 2.2.

FACT 2.2. *When* $\omega = 1$ *and* $x_0 = 0$, $A_1^T x_k = b_1$ *holds in exact arithmetic on every iteration of* (4).

Proof. Using the definition $P_i = A_i(A_i^T A_i)^{-1} A_i^T$, (8) can be expanded to show that the ith block column of $T(1)$ is given by

$$\prod_{j=1}^{i-1}(I - P_j) \left[I + \prod_{j=i}^{m}(I - P_j) \prod_{j=m-1}^{i+1}(I - P_j) \right] (A_i^T)^{\dagger}. \tag{9}$$

The first product above should be interpreted as I when $i = 1$ and the third product should be interpreted as I when $i = m - 1$ and 0 when $i = m$, so that the first summand in forming $T(1)b$ is $[I + (I - P_1) \cdots (I - P_m) \cdots (I - P_2)](A_1^T)^{\dagger} b_1$. Since $A_1^T(I - P_1) = 0$, $A_1^T x_1 = A_1^T (A_1^T)^{\dagger} b_1 = b_1$. The succeeding iterates x_k are obtained by adding elements in $\operatorname{range}(Q(1)) \subseteq \operatorname{range}(I - P_1) = \operatorname{null}(A_1^T)$ to x_1. □

Later we show that this continues to hold when CG acceleration is used. This feature means that when $\omega = 1$ equations deemed more important than the others can be put into the first block and kept satisfied to machine precision.

The most important reason for choosing $\omega = 1$ is an a posteriori one; the resulting algorithm works well. For the remainder of the paper $Q = Q(1)$ and $T = T(1)$ will be used. The resulting system,

$$(I - Q)x = \tilde{b}, \tag{10}$$

$$Q = (I - P_1)(I - P_2)\cdots(I - P_m)\cdots(I - P_2)(I - P_1), \tag{11}$$

$$\tilde{b} = Tb, \tag{12}$$

will be called the KACZ system.

2.2. Row projection method V-RP. A new RP iteration with computational and theoretical advantages over (11) is now introduced. Let $A_i = Q_iU_i$ be the QR decomposition of A_i for $i = 1, 2, \cdots, m$. Note that $P_i = Q_iQ_i^T$ in this case. Let $AA^T = L + D + L^T$ be the decomposition of AA^T into the strictly lower block triangular, block diagonal, and strictly upper block triangular parts induced by the block row partitioning. As shown in [18, p. 31], the block SSOR iteration for $AA^Ty = b$ can be written as

$$\begin{aligned} y_{k+1} &= [I - \omega(2-\omega)(D+\omega L)^{-T}D(D+\omega L)^{-1}AA^T]y_k \\ &+ \omega(2-\omega)(D+\omega L)^{-T}D(D+\omega L)^{-1}b. \end{aligned}$$

For $\omega = 1$ this can be viewed as an iteration to solve the system

$$[(D+L)^{-T}D(D+L)^{-1}AA^T]y = (D+L)^{-T}D(D+L)^{-1}b. \tag{13}$$

The KACZ system is obtained by replacing A^Ty with x and premultiplying by A^T, or equivalently applying a similarity transformation with $W = A^T$ to the coefficient matrix in (13) to yield

$$[A^T(D+L)^{-T}D(D+L)^{-1}A]x = A^T(D+L)^{-T}D(D+L)^{-1}b. \tag{14}$$

In [4], applying a similarity transformation using $W = D^{-1/2}(D+L)^T$ was proposed; when CG acceleration is used this has the desirable property of being an error reducing process in the two-norm, instead of reducing the error in an elliptic norm. Unfortunately, computing $D^{-1/2}$ is only practical in the $m = N$ case where each A_i^T is a single row of A.

However, the triangular factors U_i can be found in the block case either from an orthogonal decomposition of A_i or a Cholesky factorization of $A_i^TA_i$. Let

$$Z = \operatorname{diag}(U_1, U_2, \cdots, U_m) \in \Re^{N\times N}$$

and consider applying a similarity transformation using $W = Z^{-T}(D+L)^T$ to (13). The resulting system is

$$Z(D+L)^{-1}AA^T(D+L)^{-T}Z^Tz = Z(D+L)^{-1}b \tag{15}$$

with $z = Z^{-T}(D+L)^Ty$ and

$$x = A^Ty = A^T(D+\omega L)^{-T}Z^Tz. \tag{16}$$

The motivation for proposing this new system follows. Suppose for convenience that each A_i^T contains the same number N/m of rows of A; in this case the KACZ system has at least $1/m$ of its eigenvalues equal to 1. When CG is applied to $I - Q$, implicitly a system only $(m-1)/m$ the size of the original system is being solved. Can this reduction in problem size be made explicit? Using (15) and some algebraic manipulation, the coefficient matrix can be written as $V = S_V^T S_V$, where

$$\begin{aligned} S_V &= A^T(D+L)^{-T}Z^T \\ &= [Q_1, (I-P_1)Q_2, \cdots, (I-P_1)\cdots(I-P_{m-1})Q_m]. \end{aligned} \tag{17}$$

Since Q_1 has orthonormal columns, the (1,1) block entry of $S_V^T S_V$ is $Q_1^T Q_1 = I$, and the (1,i) block is

$$Q_1^T(I-P_1)\left(\prod_{j=2}^{i-1}(I-P_j)\right)Q_i.$$

Since $Q_1^T(I-P_1) = Q_1^T - Q_1^T Q_1 Q_1^T = 0$, and $S_V^T S_V$ is symmetric, the first row and column are zero except for the (1,1) block entry. For the case $m = 3$, e.g., this implies that $I - V$ has the form

$$\begin{bmatrix} I & 0 & 0 \\ 0 & {Q_2}^T(I-P_1)Q_2 & {Q_2}^T(I-P_1)(I-P_2)Q_3 \\ 0 & {Q_3}^T(I-P_2)(I-P_1)Q_2 & {Q_3}^T(I-P_2)(I-P_1)(I-P_2)Q_3 \end{bmatrix}. \tag{18}$$

Clearly, the first block of unknowns requires no iterations to find, and so the CG algorithm only need be applied to a reduced system. An important point is that this reduction has no detrimental effect on the spectrum of the iteration matrix. This is because (11) and (15) have matrices with identical eigenvalues since they are similar to the system matrix in (13). The resulting system,

$$\begin{aligned} (I-V)z &= b_V, \\ V &= I - Z(D+L)^{-1}AA^T(D+L)^{-T}Z^T, \\ x &= A^T(D+L)^{-T}Z^T z, \end{aligned} \tag{19}$$

$$b_V = Z(D+L)^{-1}b, \tag{20}$$

will be called the the V-RP system.

Similar to KACZ, V-RP keeps the first block of equations exactly satisfied when used as a linear stationary iteration, but without requiring $x_0 = 0$.

THEOREM 2.1. *For any starting vector z_0, the iteration $z_{k+1} = Vz_k + Z(D+L)^{-1}b$ yields iterates x_k that satisfy $A_1^T x_k = b_1$, where x_k satisfies* (16).

Proof. Let the vector $z_k = (z_k^{(1)}, \cdots, z_k^{(m)})$ be partitioned conformally with the row partitioning of A. Referring to the system matrix shown in (18), note that $z_k^{(1)}$ is simply the first block of the modified right-hand side $Z(D+L)^{-1}b$. $(D+L)$ is block lower triangular with its (1,1) block equal to $A_1^T A_1$, so the first block of $Z(D+L)^{-1}b$ is $z_k^{(1)} = U_1(A_1^T A_1)^{-1}b_1 = U_1(U_1^T U_1)^{-1}b_1 = U_1^{-T}b_1$. From (17), (16), $A_1 = Q_1 U_1$, and $Q_1^T(I-P_1) = 0$,

$$\begin{aligned} A_1^T x_k &= U_1^T Q_1^T [Q_1, (I-P_1)Q_2, \cdots, (I-P_1)\cdots(I-P_{m-1})Q_m]\, z_k \\ &= U_1^T z_k^{(1)} = U_1^T U_1^{-T} b_1 = b_1. \end{aligned} \qquad \square$$

TABLE 1
Comparison of system matrices for four methods.

Method	W^T
CGNE	$[A_1,\ A_2,\ \cdots\ ,A_m]$
CIMM	$[Q_1,\ Q_2,\ \cdots\ ,Q_m]$
KACZ	$\left[P_1,\ (I-P_1)P_2,\ (I-P_1)(I-P_2)P_3,\ \cdots\ ,\prod_{i=1}^{m-1}(I-P_i)P_m\right]$
V-RP	$\left[Q_1,\ (I-P_1)Q_2,\ (I-P_1)(I-P_2)Q_3,\ \cdots\ ,\prod_{i=1}^{m-1}(I-P_i)Q_m\right]$

2.3. Row projection method CIMM. The third RP method can be derived as a preconditioner for the CG algorithm. Premultiply the system $Ax = b$ by $\tilde{A} = [A_1(A_1^TA_1)^{-1}, A_2(A_2^TA_2)^{-1}, \cdots, A_m(A_m^TA_m)^{-1}]$ to obtain

$$(P_1 + P_2 + \cdots + P_m)x = \tilde{A}b. \tag{21}$$

This system can also be derived as a block Jacobi method applied to the system (7); see [11]. For nonsingular A this system is symmetric positive definite and the CG algorithm can be applied. The advantage of this approach is that the projections can be computed in parallel and then added. In 1939 Cimmino [9] first proposed an iteration related to (21), and since then it has been examined by several others [1], [11], [12], [15], [25], [26], [38]. Later we will show how the individual projections can, for a wide class of problems, be computed with parallelism. In this case, the Cimmino method allows computations to proceed with two levels of parallelism, making it especially suitable for hierarchical memory machines such as Cedar [24].

2.4. Connection between RP systems and the normal equations. Although KACZ and V-RP can be derived as a block SSOR, and CIMM as a block Jacobi, method for (7), a more instructive comparison can be made with CGNE, conjugate gradients applied to the normal equations $A^TAx = A^Tb$. All four methods consist of CG applied to a system with coefficient matrix W^TW, where W^T is shown for the four methods in Table 1. Intuitively, an ill-conditioned matrix A has some linear combination of rows approximately equal to the zero vector. For a row partitioned matrix, near linear dependence may occur *within* the blocks, that is, some linear combination of the rows within a particular block is approximately zero, or *across* the blocks, that is, the linear combination must draw from more than one block row A_i^T. Now let $A_i = Q_iU_i$ be the orthogonal decomposition of A_i, and note that both Q_i and $P_i = Q_iQ_i^T$ have the perfect condition number of 1. Examining the matrices W shows that CGNE works on A^TA, and the ability to form a near linear dependence from both within and across blocks enters into the system matrix. CIMM replaces each A_i with Q_i, which has orthonormal columns. Hence CIMM removes the ability to form linear dependence within the blocks, but has no effect on that between the blocks. V-RP also replaces each A_i with the perfectly conditioned Q_i and then goes a step further by recursively orthogonalizing between blocks in the following way: The first block column of W^T is orthogonal to the others since $Q_1(I - P_1) = 0$. If the orthogonalizing $I - P_1$ factor is removed from the other block columns of W^T, then the second block column is orthogonal to the remaining ones since $Q_2(I - P_2) = 0$. This process continues until the last block is reached. The recursive nature of this partial orthogonalization can be seen by rewriting W^T for V-RP as

$$W^T = [Q_1,\ \ (I-P_1)[Q_2,\ \ (I-P_2)[Q_3,\ \ (I-P_3)\cdots[Q_{m-1},\ \ (I-P_{m-1})Q_m]\cdots]]].$$

KACZ has the same effect as V-RP, since $P_i^T(I - P_i) = 0$ in the same way that $Q_i(I - P_i) = 0$.

Several implications follow from this heuristic argument. First, the system matrix spectrum for KACZ and V-RP should be better than that of CIMM, which in turn should be better than that of CGNE, where "better" means having fewer small eigenvalues and many more eigenvalues near the maximal one. Section 4 will show that for $m = 2$ the first comparison is true, and the second comparison holds if condition numbers are used to measure "better." Furthermore, by computing the spectra for small grid sizes we have found these comparisons are valid for the problems in §6, where $m = 9$.

A second implication of this argument is that RP methods will require fewer iterations for matrices A where the near linear dependence comes primarily from within a block row rather than between block rows. A third implication of the heuristic argument for the relative performance of RP methods is that the number of block rows should be kept as small as possible. The reason is that the partial orthogonalization between blocks in the W of Table 1 becomes less effective as more block rows appear. In terms of the heuristic, progressively more orthogonalizing factors $I - P_i$ must be stripped away before the orthogonalization effect between block row i and the succeeding block rows is felt. Keeping the number of block rows small can also be seen to be important because, e.g., for the $m = N$ case all of the ability to form a near linear dependency between rows of the system matrix occurs between the block rows, where the outer CG acceleration method must deal with it.

3. CG acceleration. Although the CG algorithm can be applied directly to the row projection systems, special properties allow a reduction in the amount of work required by KACZ and provide another advantage of V-RP. CG acceleration for RP methods was proposed in [4] and tested in [7], [21], [22]. The reason that a reduction in work is possible and $A_1^T x_k = b_1$ on every iteration of CG applied to KACZ follows from Theorem 3.1.

THEOREM 3.1. *Suppose that the* CG *algorithm is applied to the* KACZ *system and let $r_k = \tilde{b} - (I - Q)x_k$ be the residual and d_k be the search direction. If $x_0 = \tilde{b}$ is chosen as the starting vector, then r_k, $d_k \in$* range$(I - P_1)$ *for all k.*

Proof.

$$\begin{aligned} r_0 &= \tilde{b} - (I - Q)\tilde{b} = Q\tilde{b} \\ &= (I - P_1)(I - P_2)\cdots(I - P_m)\cdots(I - P_2)(I - P_1)\tilde{b} \in \text{range}(I - P_1). \end{aligned}$$

Since $d_0 = r_0$, the same is true for d_0. Suppose the theorem holds for $k - 1$. Then $d_{k-1} = (I - P_1)d_{k-1}$ and so

$$\begin{aligned} w_k &\equiv (I - Q)d_{k-1} \\ &= (I - P_1)d_{k-1} - (I - P_1)(I - P_2)\cdots(I - P_m)\cdots(I - P_2)(I - P_1)d_{k-1} \\ &\in \ \text{range}(I - P_1). \end{aligned}$$

Since r_k is a linear combination of r_{k-1} and w_k, $r_k \in$ range$(I - P_1)$. Since d_k is a linear combination of d_{k-1} and r_k, $d_k \in$ range$(I - P_1)$. The result follows by induction. □

This reduces the requisite number of projections from $2m - 1$ to $2m - 2$ because the first multiplication by $(I - P_1)$ when forming $(I - Q)d_k$ can be omitted. For V-RP, the next section will show that the algorithm can be implemented so that effectively only $2m - 2$ projections are required on each iteration, for any starting vector. Using $x_0 = \tilde{b}$ for KACZ also keeps the first block of equations satisfied in exact arithmetic.

COROLLARY 3.2. *If* $x_0 = \tilde{b}$ *is chosen in the* CG *algorithm applied to the* KACZ *system, then* $A_1^T x_k = b_1$ *for all* $k \geq 0$.

Proof. The proof of Fact 2.2 shows that $\tilde{b} \in (A_1^T)^\dagger b_1 + \text{range}(I - P_1)$. Since $A_1^T(I - P_1) = A_1^T(I - A_1 A_1^\dagger) = 0$, $A_1^T x_0 = A_1^T(A_1^T)^\dagger b_1 = b_1$ because A_1 has full column rank. For $k > 0$, $d_{k-1} \in \text{range}(I - P_1)$, so

$$A_1^T x_k = A_1^T(x_{k-1} + \alpha_k d_{k-1}) = A_1^T x_{k-1} = \cdots = A_1^T x_0 = b_1. \qquad \square$$

V-RP has already been shown to allow an explicit reduction of the problem size. Its second major advantage is that when accelerated by CG, the resulting algorithm is error-minimizing in the two-norm.

THEOREM 3.3. *Suppose* CG *acceleration is applied to the* V-RP *system and let* x^* *be the solution to* $Ax = b$. *Then the kth iteration minimizes* $\| x - x^* \|$ *over all* $x \in S_V^T * \text{span}[x_0 - x^*, \cdots, x_{k-1} - x^*]$, *where* S_V *is defined by* (17).

Proof. Recall that $x_k = A^T(D + L)^{-T} Z^T z_k = S_V z_k$, where $V = I - S_V^T S_V$. The CG algorithm minimizes the quadratic form [16]

$$\begin{aligned} z^T(I - V)z - 2z^T b_V &= z^T S_V^T S_V z - 2z^T S_V^T A^{-1} b \\ &= x^T x - 2x^T x^* = \| x - x^* \|^2 - \| x^* \|^2 . \end{aligned}$$

Since $\| x^* \|^2$ is a constant, this is equivalent to minimizing $\| x - x^* \|^2$. The subspace over which this minimization occurs is

$$\begin{aligned} \text{span}[r_0, \cdots, r_{k-1}] &= \text{span}[S_V^T A^{-1} b - S_V^T S_V z_0, \cdots, S_V^T A^{-1} b - S_V^T S_V z_{k-1}] \\ &= S_V^T * \text{span}[x^* - x_0, \cdots, x^* - x_{k-1}]. \qquad \square \end{aligned}$$

In contrast, CG acceleration in the KACZ system minimizes $\| S_V^T(x_k - x^*) \|$ over the subspace $S_V S_V^T * \text{span}[x_0 - x^*, \cdots, x_{k-1} - x^*]$, a result established in [4].

In summary, CG acceleration for KACZ allows one projection per iteration to be omitted and one block of equations to be kept exactly satisfied, provided that $x_0 = \tilde{b}$ is used. CG acceleration for V-RP minimizes the two-norm rather than the A-norm of the error.

4. The $m = 2$ case. When $m = 2$ is chosen, the matrix A is partitioned into two block rows, and a complete eigenanalysis of RP methods is possible using the concept of the angles θ_k between the two subspaces $L_i = \text{range}(A_i)$, $i = 1, 2$. The definition presented here follows that of Björck and Golub in [5], but for convenience, L_1 and L_2 are assumed to have the same dimension. The smallest angle $\theta_1 \in [0, \pi/2]$ between L_1 and L_2 is defined by

$$\begin{aligned} \cos\theta_1 &= \max_{u \in L_1} \max_{v \in L_2} u^T v \\ &\text{subject to } \| u \| = \| v \| = 1. \end{aligned}$$

Let u_1 and v_1 be the attainment vectors; then for $k = 2, 3, \cdots, N/2$ the remaining angles between the two subspaces are defined as

$$\begin{aligned} \cos\theta_k &= \max_{u \in L_1} \max_{v \in L_2} u^T v \\ &\text{subject to} \begin{cases} \| u \| = \| v \| = 1 \\ u_i^T u = v_i^T v = 0, \quad i = 1, 2, \cdots, k - 1. \end{cases} \end{aligned}$$

Furthermore, when u_i and v_j are defined as above, $u_i^T v_j = 0$, $i \neq j$ also holds. From this one can obtain the CS decomposition [16], [36], which is stated below in terms of the projectors P_i.

THEOREM 4.1 (CS Decomposition). *Let $P_i \in \Re^{N \times N}$ be the orthogonal projectors onto the subspaces L_i, for $i = 1, 2$. Then there are orthogonal matrices U_1 and U_2 such that*

$$P_1 = U_1 \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} U_1^T, \quad P_2 = U_2 \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} U_2^T, \quad U_1^T U_2 = \begin{bmatrix} C & -S \\ S & C \end{bmatrix},$$

$$\begin{aligned} C &= \operatorname{diag}(c_1, c_2, \cdots, c_{N/2}), \\ S &= \operatorname{diag}(s_1, s_2, \cdots, s_{N/2}), \\ I &= C^2 + S^2, \\ 1 &\geq c_1 \geq c_2 \geq \cdots \geq c_{N/2} \geq 0. \end{aligned} \tag{22}$$

In the theorem $c_k = \cos\theta_k$ and $s_k = \sin\theta_k$, where the angles θ_k are defined above. Now consider the nonsymmetric RP iteration matrix $Q_u = (I - \omega P_1)(I - \omega P_2)$. Letting $\alpha = 1 - \omega$, and substituting in the expressions for P_1 and P_2 from the CS decomposition gives

$$Q_u = U_1 \begin{bmatrix} \alpha^2 C & -\alpha S \\ \alpha S & C \end{bmatrix} U_2^T.$$

Applying a similarity transformation using the matrix U_2^T gives

$$U_2^T Q_u U_2 = \begin{bmatrix} \alpha^2 C^2 + \alpha S^2 & (1-\alpha) CS \\ \alpha(1-\alpha) CS & C^2 + \alpha S^2 \end{bmatrix}. \tag{23}$$

Since each block in the above 2×2 matrix is diagonal, $U_2^T Q_u U_2$ has the same eigenvalues as its scalar 2×2 principal submatrices. These eigenvalues are given by

$$(1-\alpha)^2 c_i^2 + 2\alpha \pm |1-\alpha| c_i \sqrt{(1-\alpha)^2 c_i^2 + 4\alpha} \tag{24}$$

for $i = 1, 2, \cdots, N/2$. For a given α the modulus of this expression is a maximum when c_i is largest, i.e., when $c_i = c_1$. The spectral radius of Q_u can then be found by taking $c_i = c_1$ and the positive sign in (24). Doing so and minimizing with respect to α gives $\omega_{\min} = 1 - \alpha_{\min} = 2/(1 + s_1)$ and a spectral radius of $(1 - s_1)/(1 + s_1)$. The same result was derived in [12] using classical SOR theory. The benefit of the CS decomposition is that the full spectrum of Q_u is given and not simply the eigenvalue of largest modulus. In particular, when $\omega = 1$ the eigenvalues of the nonsymmetric RP iteration matrix Q_u become $\left\{c_1^2, c_2^2, \cdots, c_{N/2}^2, 0\right\}$ with 0 repeated $N/2$ times. Now applying the CS decomposition to the symmetrized RP iteration matrix $Q(\omega)$ gives

$$\begin{aligned} Q(\omega) &= (I - \omega P_1)(I - \omega P_2)(I - \omega P_1) \\ &= U_1 \begin{bmatrix} \alpha^2(\alpha C^2 + S^2) & \alpha(\alpha - 1) CS \\ \alpha(\alpha-1) CS & \alpha S^2 + C^2 \end{bmatrix} U_1^T, \end{aligned} \tag{25}$$

where again $\alpha = 1 - \omega$. When $\omega = 1$, the eigenvalues of the symmetrized matrix are identical to those of the unsymmetrized matrix Q_u. One objection to the symmetrization process is that $Q(\omega)$ requires three projections while Q_u only needs two. The previous section, however, showed that when $\omega = 1$, KACZ can be implemented with only two projections per iteration.

When $m = 2$ the value $\omega = 1$ minimizes the spectral radius of $Q(\omega)$, as was shown in [22]. This result can be obtained from the representation (25) in the same way as (23) was obtained.

The CS decomposition also allows the construction of an example showing that no direct relationship need exist between the singular value distribution of A and its related RP matrices. Define

$$A = \begin{bmatrix} A_1^T \\ A_2^T \end{bmatrix} = \begin{bmatrix} 0 & D \\ S & C \end{bmatrix}, \tag{26}$$

where each block is $N/2 \times N/2$, C and S satisfy (22), and $D = \mathrm{diag}(d_1, d_2, \cdots, d_{N/2})$. The projectors become

$$P_1 = \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} \quad \text{and} \quad P_2 = \begin{bmatrix} C \\ S \end{bmatrix} \begin{bmatrix} C & S \end{bmatrix},$$

and the eigenvalues of $I - Q(1)$ are $\{s_1^2, s_2^2, \cdots, s_{N/2}^2, 1\}$, while those of A are $\left[c_i \pm (c_i^2 + 4s_id_i)^{1/2}\right]/2$. A is nonsingular if and only if each s_i and d_i is nonzero. If the s_i's are chosen to be close to 1 while the d_i's are chosen to be close to 0, $I - Q(1)$ has eigenvalues clustered near 1 while A has singular values close to both 0 and 1. Hence A is badly conditioned while $I - Q(1)$ is very well conditioned. Conversely, if the d_i's are chosen to be large while the s_i's are close to 0 in such a way that d_is_i is near 1, then A is well conditioned while $I - Q(1)$ is badly conditioned. Hence the conditioning of A and its induced RP system matrix may differ greatly. Although this example is contrived, §7 presents two PDEs with induced matrices A that have identical singular values but drastically differing spectra for the corresponding RP matrices.

In §2.4 a heuristic argument was given implying that the eigenvalue distribution for the KACZ and V-RP systems is better for CG acceleration than that of CIMM, which in turn is better than that of CGNE, CG applied to $A^TAx = A^Tb$. For $m = 2$, the eigenvalues of KACZ and V-RP are $\{s_1^2, s_2^2, \cdots, s_{N/2}^2, 1\}$ while those of CIMM are easily seen to be $\{1 - c_1, \cdots, 1 - c_{N/2}, 1 + c_{N/2}, \cdots, 1 + c_1\}$, verifying that KACZ and V-RP have better spectral distribution than CIMM. The next theorem shows that in terms of condition numbers the heuristic argument is valid when $m = 2$.

THEOREM 4.2. *When* $m = 2$, $\kappa(A^TA) \geq \kappa(P_1 + P_2) \geq \kappa(I - (I - P_1)(I - P_2)(I - P_1))$. *Furthermore, let* $c_1 = \cos\theta_1$ *be the canonical cosine corresponding to the smallest angle between* range(A_1) *and* range(A_2). *Then* $\kappa(P_1 + P_2) = (1 + c_1)^2\kappa(I - (I - P_1)(I - P_2)(I - P_1))$.

Proof. Without loss of generality, suppose that range(A_1) and range(A_2) have dimension $N/2$. Let $U_1 = [G_1, G_2]$ and $U_2 = [H_1, H_2]$ be the matrices defined in Theorem 4.1 so that $P_1 = G_1G_1^T$, $P_2 = H_1H_1^T$, and $G_1^TH_1 = C$. Set $X = G_1^TA_1$ and $Y = H_1^TA_2$ so that $A_1 = P_1A_1 = G_1G_1^TA_1 = G_1X$ and $A_2 = H_1Y$. It is easily verified that the eigenvalues of $P_1 + P_2$ are $1 \pm c_i$, corresponding to the eigenvectors $g_i \pm h_i$ where $G_1 = [g_1, g_2, \cdots, g_{N/2}]$ and $H_1 = [h_1, h_2, \cdots, h_{N/2}]$. Furthermore, $G_1g_1 = H_1h_1 = e_1$ and $G_1h_1 = H_1g_1 = c_1e_1$, where e_1 is the first unit vector. Then $A^TA = A_1A_1^T + A_2A_2^T = G_1XX^TG_1^T + H_1YY^TH_1^T$, so $(g_1 + h_1)^T(A^TA)(g_1 + h_1) = (1 + c_1)^2e_1^T(A^TA)e_1$ and $(g_1 - h_1)^T(A^TA)(g_1 - h_1) = (1 - c_1)^2e_1^T(A^TA)e_1$. The minimax characterization of eigenvalues gives

$$\begin{aligned} \lambda_{\max}(A^TA) &\geq (1 + c_1)^2e_1^T(A^TA)e_1/(g_1 + h_1)^T(g_1 + h_1) \\ &= (1 + c_1)^2e_1^T(A^TA)e_1/2(1 + c_1) = (1 + c_1)e_1^T(A^TA)e_1/2 \end{aligned}$$

and $\lambda_{\min}(A^TA) \leq (1 - c_1)e_1^T(A^TA)e_1/2$. Hence $\kappa(A^TA) \geq (1 + c_1)/(1 - c_1) = \kappa(P_1 + P_2)$, proving the first inequality. For the second, from the CS decomposition the eigenvalues of $(I-(I-P_1)(I-P_2)(I-P_1))$ are $\{s_1^2, s_2^2, \cdots, s_{N/2}^2, 1\}$, where $s_i^2 = 1-c_i^2$ are the canonical sines and 1 is of multiplicity $N/2$. Then

$$\frac{\kappa(P_1+P_2)}{\kappa(I-(I-P_1)(I-P_2)(I-P_1))} = \frac{1+c_1}{1-c_1} \div \frac{1}{s_1^2} = (1+c_1)^2. \qquad \square$$

Note that $(1+c_1)^2$ is a measure of the lack of orthogonality between range(A_1) and range(A_2), and so measures the partial orthogonalization effect described in §2.4.

5. Implementation for three-dimensional elliptic PDEs. The primary implementation issue for RP methods is how to partition the rows of A. Since any row partitioning gives a convergent algorithm, the criteria for choosing one can be based on computational considerations. This section describes the set of test problems, outlines the row partitioning criteria, and presents the row partitioning used in the testing.

5.1. Test problems. Test problems are obtained from the seven-point centered difference operator [35] for elliptic partial differential equations

$$(27) \qquad a u_{xx} + b u_{yy} + c u_{zz} + d u_x + e u_y + f u_z + g u = F$$

where $a - g$ are functions of (x, y, z) and the domain is the unit cube $[0,1] \times [0,1] \times [0,1]$. Dirichlet boundary conditions are imposed in a manner described later. When discretized using n grid points in each direction the resulting system is of order $N = n^3$.

This class of test problems is chosen for four reasons. First, it includes important applications such as computational fluid dynamics and structural mechanics. Second, by refining the mesh the problem size grows rapidly with n, allowing test problems that can be scaled up to realistic sizes. Results are shown for problems of order 216000, corresponding to a 60 × 60 × 60 mesh. Third, by selecting the coefficient functions of the partial differential equation, matrices A can be created with or without properties such as diagonal dominance, definite symmetric part, eigenvalues on both sides of the imaginary axis, or extreme ill-conditioning. Fourth, the problems provide a specific example on how to select m, partition the rows into blocks, and analyze the subproblems that define the projectors.

The collection of test problems and their predetermined solutions follows. These are generalized versions of two-dimensional problems from a variety of sources [13], [22], [23], [34]. Each test problem has boundary conditions chosen to give a predetermined solution to the partial differential equation, so that the norm of the error vector as well as that of the residual vector can be checked.

PROBLEM 1. $\Delta u + 1000 u_x = F$.

PROBLEM 2. $\Delta u + 1000 \exp(xyz)(u_x + u_y - u_z) = F$.

PROBLEM 3. $\Delta u + 100 x u_x - y u_y + z u_z + 100(x+y+z)u/xyz = F$.

PROBLEM 4. $\Delta u - 10^5 x^2 (u_x + u_y + u_z) = F$.

PROBLEM 5. $\Delta u - 1000(1+x^2)u_x + 100(u_y + u_z) = F$.

PROBLEM 6. $\Delta u - 1000\left[(1-2x)u_x + (1-2y)u_y + (1-2z)u_z\right] = F$.

Problem 1 has the preassigned solution $u = xyz(1-x)(1-y)(1-z)$, Problem 2 has $u = x+y+z$, and Problems 3–6 have $u = \exp(xyz)\sin(\pi x)\sin(\pi y)\sin(\pi z)$. These test problems exhibit properties that cause difficulties for nonsymmetric iterative solvers, as summarized in Table 2 for a 12 × 12 × 12 grid. "Re $(\lambda) > 0$" indicates whether or not the eigenvalues lie in the right half plane of the complex plane; if this does not

TABLE 2
Properties of A for $N = 1728$

Problem	1	2	3	4	5	6
$\text{Re}(\lambda) > 0$	Yes	Yes	No	Yes	Yes	Yes
$A + A^T$ Definite	Yes	No	No	No	No	No
Estimated $\kappa(A)$	9	57	40000	4786	36	77

hold, then residual polynomial methods will fail. "$A + A^T$ definite" indicates whether or not the symmetric part of A is definite; if the answer is yes, then the sufficient conditions for GMRES(k) to converge hold. $\kappa(A)$ is an estimate of the condition number obtained by applying the LINPACK routine DPOCO to A^TA.

5.2. Row partitioning goals. The first criterion for a row partitioning is that the projections $P_ix = A_i(A_i^TA_i)^{-1}A_i^Tx$ must be efficiently computable. One way to achieve this is through parallelism: if A_i^T is the direct sum of blocks C_j for $j \in S_i$, then P_i is block diagonal [22]. The computation of P_ix can then be done by assigning each block of P_i to a different processor on a multiprocessor machine, or by vectorizing the computations across the blocks on a vector machine. The second criterion is storage efficiency. The additional storage should be $\mathcal{O}(N)$, that is, a few extra vectors, the number of which must not grow with increasing problem size N. The third criterion is that some bound on the condition number of the subproblems induced by the projections should be provided. The need for this is clear when $m = 1$ and A is partitioned into a single block of rows; in this case KACZ simply solves the normal equations $A^TAx = A^Tb$, an approach that can fail if $\kappa(A)$ is large. More generally, when $m > 1$, computing $y = P_ix$ requires solving a system of the form $A_i^TA_iv = w$. Hence the accuracy with which P_i can be computed depends on $\kappa(A_i^TA_i)$, and potentially large errors might occur in the projections if that condition number is large. A practical and cheap numerical way of bounding $\kappa(A_i^TA_i)$ for a given problem should be available. The fourth criterion is that the number m of projectors should be kept as small as possible, and should not depend on N. One reason was presented in Fact 2.1: when the block rows have equal numbers of rows of A, $1/m$ of the eigenvalues of the system matrix are exactly 1. For V-RP the possible reduction in problem size also decreases with increasing m.

In summary, the row partitioning should allow parallelism in the computations, require at most $\mathcal{O}(N)$ storage, give well-conditioned subproblems, and have m a small constant. One of the more useful results of this paper is that all four goals can be achieved simultaneously for important classes of problems.

5.3. Row partitioning of test problems. Row partitioning schemes ranging from $m = 4$ up to $m = 27$ have been tested on the test problems. For brevity, only one, an $m = 9$ partitioning generalized from a scheme in [22], is presented. Consider an $n \times n \times n$ mesh, and for convenience assume that n is a multiple of 3 and that lines in the mesh are numbered within each plane in the natural order. Place equations corresponding to nodes on lines $1, 4, 7, \cdots, n-2$ on the planes numbered $1, 4, 7, \cdots, n-2$ into the first block row A_1^T. Then assign those on lines $2, 5, 8, \cdots, n-1$ on the planes numbered $1, 4, 7, \cdots, n-2$ into the second block row A_2^T. In general, A_i^T consists of nodes on lines $l, l+3, l+6, \cdots$, on planes $p, p+3, p+6, \cdots$, where $l = (i+2) \bmod 3 + 1$ and $p = \lfloor \frac{i-1}{3} \rfloor + 1$. Figure 1 shows this scheme for a $6 \times 6 \times 6$ grid, with each node marked by the number i of its assigned block row A_i^T.

Because each line of nodes assigned to A_i^T is separated from the others by at least

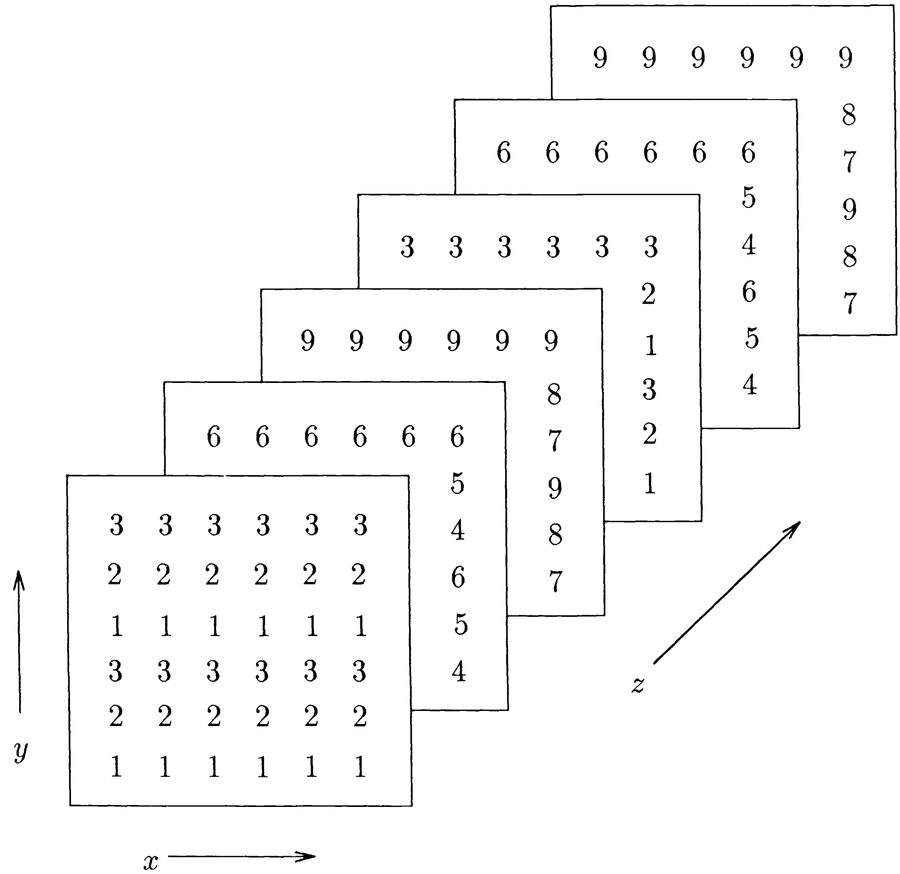


FIG. 1. *Row partitioning for $m = 9$.*

two other lines or planes, their seven point difference stars do not intersect and so A_i^T consists of $n^2/9$ separate subblocks of n coupled equations each. Each projector P_i can then be computed with a parallelism of $n^2/9$. What of the other row partitioning criteria? Let the subblocks of A_i^T be denoted C_j^T, for $j \in S_i$, where S_i is an index set of cardinality $n^2/9$. Dropping the subscripts temporarily, a typical C_j^T has the form

$$C^T = [0, D_1, 0, D_2, T, D_3, 0, D_4, 0] \tag{28}$$

with D_i a diagonal and T a tridiagonal matrix. The matrix C^TC is positive definite because when A is nonsingular each subblock is necessarily of full rank n. More importantly, C^TC is pentadiagonal and so its Cholesky factor R consists of three diagonals, and the Cholesky factors for all of the subblocks for all of the block rows A_i^T can be stored using only three additional vectors.

Surprisingly, the third requirement that each subproblem be well conditioned also is generally satisfied by the $m = 9$ partitioning. The details can be found in [6], but this can be seen intuitively because C^TC consists of the normal equations of T in (28), with the squares of the diagonal blocks D_i added to the main diagonal of T^TT, making it strongly diagonally dominant. This suggests that good bounds for the extremal eigenvalues of C^TC can be obtained from Gerschgorin estimates. However, for some

TABLE 3
$\max_j \kappa(C_j)$ *for* $n = 48$.

Problem	1	2	3	4	5	6
Estimated	7.94	1.92	74.9	1608	12.9	2.19
Actual	4.93	1.87	61.2	518	8.56	2.13

problems the Gerschgorin Disk Theorem provides an estimate $\lambda_{\min}(C^TC) \leq 0$. In these cases a better bound on the smallest eigenvalue comes from an application of the minimax characterization of singular values.

THEOREM 5.1. *Suppose that* $C^T = [D_1\ 0\ D_2\ T\ D_3\ 0\ D_4]$, *where each* $D_i \in \Re^{n\times n}$ *is diagonal, and let* $\delta_i = \min_k |D_i|_{kk}$. *Then*

$$\sigma^2_{\min}(C) = \lambda_{\min}(C^TC) \geq \delta_1^2 + \delta_2^2 + \delta_3^2 + \delta_4^2.$$

Proof. Let $z \in \mathcal{R}^n$, $\| z \| = 1$. Then

$$\begin{aligned} \| Cz \|^2 &= \| D_1z \|^2 + \| D_2z \|^2 + \| Tz \|^2 + \| D_3z \|^2 + \| D_4z \|^2 \\ &\geq \| D_1z \|^2 + \| D_2z \|^2 + \| D_3z \|^2 + \| D_4z \|^2 \geq \delta_1^2 + \delta_2^2 + \delta_3^2 + \delta_4^2. \end{aligned} \tag{29}$$

Minimizing $\| Cz \|$ over all such z gives the stated result. □

When C corresponds to a line of nodes at the top or bottom of a plane, or is within the first or last plane of the grid, one or more of the δ_i's is zero but the theorem is still valid. Table 3 shows results for the test problems on a 48 × 48 × 48 grid with the Gerschgorin estimate of the smallest singular value replaced by that of Theorem 5.1 when it is larger. The actual condition numbers are calculated using routines from Eispack. Clearly the subproblems are well conditioned, so the row partitioning with $m = 9$ satisfies all of the criteria for a suitable row partitioning.

The testing results show that the approach taken here for partitioning A is extremely effective. However, it is of limited use if applicable only to seven-point difference operator matrices. The reasoning for selecting the partitioning is based on a natural decoupling induced by the computational molecule used. This decoupling still occurs if, e.g., the domain is irregular or a 27-point difference operator is applied. Extensions to Neumann or periodic boundary conditions are also possible, again by considering the decoupling available on the mesh. Although more complicated, such a decoupling occurs with finite element methods applied to partial differential equations since usually the elements are chosen to have common support with only a few other elements. What determines the storage requirements of $3N$ for all of the Cholesky factors is that the longest line in the computational star contains three nodes. More generally, if the longest line in the computational star has l nodes, the Cholesky factors require only lN storage. The scheme used here for row partitioning can thus be extended to other, less simple, problems.

5.4. Other approaches. Recently, Arioli, Duff, Noailles, and Ruiz [2] experimented with a block Cimmino method for more general sparse systems. Their approach is to reorder the matrix into a block tridiagonal form, and then to use a row partitioning into two block rows. In this case the goal is not to have equal-sized blocks for load balancing, but rather to have few nonzero columns in the blocks of A_1^T shared with the nonzero columns in the blocks of A_2^T. This allows the use of a novel block diagonal right preconditioner that damps out the overlap between the shared nonzero columns and thereby increases the angles between the range of A_1 and A_2.

This reduces the number of CG iterations (cf. §2.4) but at the cost of making the subproblems more ill conditioned. Arioli et al. then handle this by using a direct method applied to an augmented system for solving the subproblems.

5.5. Implementation details. The crucial operations in the RP algorithms are the preprocessing stage, computation of the modified right-hand side vectors, and forming the matrix-vector products needed for the acceleration schemes. V-RP also requires a postprocessing stage to recover the solution x from the auxiliary unknown z. For convenience, an $n \times n \times n$ grid is used, with n a multiple of 3.

Preprocessing for RP methods consists of computing the necessary Cholesky factors R_j. As §5.3 showed, each C_j is well conditioned, so the procedure is to explicitly form the normal equations and then perform an LDL$^{\mathrm{T}}$ factorization. Recalling that S_i is the set of indices j for which C_j is a subblock of A_i, this stage consists of $n^2/9$ parallel tasks.

Cholesky Factorization:

```
For $i = 1, 2, \cdots, 9$ and $j \in S_i$ (in parallel)
    Form $R_j = (C_j)^T(C_j)$ (vectorized)
    Perform LDL$^{\mathrm{T}}$ factorization on $R_j$, overwriting $R_j$ with the result (sequential)
End For
```

R_j is stored as three vectors, and the diagonal is stored as $(D_j)^{-1}$ so that the backsolves can be performed using multiplication rather than division. The LDL$^{\mathrm{T}}$ factorization is an inherently sequential process, unsuitable for vector machines. For those computers the order of the loops is reversed, that is, the operations are vectorized *across* the blocks C_j rather than parallelized between the blocks.

To compute the KACZ modified right-hand side $\tilde{b}$, let $U_i = \mathrm{diag}(R_{j_1}, R_{j_2}, \cdots, R_{j_{n^2/9}})$, for $j_k \in S_i$. By overwriting, $\tilde{b}$ can be found using a temporary vector $w^T = (w_1^T, w_2^T, \cdots, w_m^T)$ of length N and $4m-2$ triangular solves with U_i or U_i^T, $2m-2$ multiplications by U_i or U_i^T, $3m$ multiplications by A_i, and $m^2 - m$ by A_i^T. The predominant expense is the triangular solves. Furthermore, each of the matrix-vector operations is implemented with $n^2/9$ parallelism.

Computation of $\tilde{b}$ from b for RP-m:

```
$w = b$
For $i = 1, 2, \cdots, m$
    $w_i = (U_i^T U_i)^{-1} w_i$
    $\tilde{b} = A_i w_i$
    For $j = i+1, i+2, \cdots, m$
        $w_j = w_j - A_j^T \tilde{b}$
    End For
End For
For $i = 1, 2, \cdots, m-1$
    $w_i = U_i^T U_i w_i$
End For
For $i = m, m-1, \cdots, 1$
    $\tilde{b} = A_i w_i$
    For $j = 1, 2, \cdots, i-1$
        $w_j = w_j - A_j^T \tilde{b}$
    End For
    If $i > 1$, $w_{i-1} = (U_{i-1}^T U_{i-1})^{-1} w_{i-1}$
```

```
End For
$\tilde{b} = A_1 w_1$
For $i = 2, 3, \cdots, m$
    $\tilde{b} = \tilde{b} + A_i w_i$
End For
```

The V-RP modified right-hand side b_V can be computed at a slightly lower cost, requiring $3m - 2$ triangular backsolves by U_i^T or U_i.

Computation of b_V from b for V-RP:

```
For $i = 1, 2, \cdots, m$
    $(b_V)_i = U_i^{-T} b_i$
End For
$w = A_m U_m^{-1} (b_V)_m$
For $i = m - 1, m - 2, \cdots, 2$
    $(b_V)_i = (b_V)_i - U_i^{-T} A_i^T w$
    $w = w + A_i U_i^{-1} (b_V)_i$
End For
$(b_V)_1 = (b_V)_1 - U_1^{-T} A_1^T w$
```

For KACZ the matrix-vector products needed by CG cost $4m-2$ triangular solves with U_i or U_i^T and $4m-2$ multiplications by A_i or A_i^T. For CIMM, this number is $2m$. V-RP can be implemented with $4m - 5$ of each operation, if the following algorithm is used.

Computation of $v = (I - V)d$ for V-RP:

```
$w = A_m U_m^{-1} d_m$
For $i = m - 1, \cdots, 3, 2$
    $w = w + A_i U_i^{-1} (d_i - U_i^{-T} A_i^T w)$
End For
$w = w - A_1 (U_1^T U_1)^{-1} A_1^T w$
For $i = 2, \cdots, m - 1$
    $v_i = U_i^{-T} A_i^T w$
    $w = w - A_i U_i^{-1} v_i$
End For
$v_m = U_m^{-T} A_m^T w$
```

Note that this only uses the block components numbered from 2 to m of the vectors d and v, reflecting the explicit reduction in the problem size allowed by V-RP. V-RP must also retrieve x from the auxiliary vector z, and this requires $2m - 1$ triangular solves and $2m - 2$ multiplications by A_i or A_i^T.

6. Testing results. Tests were run on three machines to examine different aspects of the algorithms. First, problems of order $N = 13824$ were run on one processor of the University of Illinois' National Center for Supercomputing Application's Cray X-MP in order to test the robustness and vectorization of the methods. Then problems of order $N = 216000$ were tested on one processor of the NCSA Cray-2 to verify that the results are valid for larger problem sizes. Finally the RP methods were tested on an eight-processor Alliant FX/8 at the Center for Supercomputing Research and Development to demonstrate that the algorithms can be implemented efficiently on a shared-memory multiprocessor. This section describes the other nonsymmetric solvers to which the RP methods are compared and the stopping tests used, and briefly summarizes the results comparing the methods. Fuller details of the results, including tables showing numbers of iterations, times, and residual and error norms are provided in [6].

6.1. Description of other methods tested. The RP algorithms are compared to two other basic methods, GMRES(10) (see [31] for references to this and the other Krylov subspace methods of this section) and CGNE, CG applied to $A^TAx = A^Tb$. GMRES(10) is from the PCGPAK library and has been optimized for the Cray-X/MP by Scientific Computing Associates [33], with portions written in Cray Assembly Language. The fixed value of $k = 10$ is chosen because larger values do not improve the robustness of GMRES(k) until k becomes a significant fraction of the problem size, and because iterative algorithms with $\mathcal{O}(N)$ additional storage are of primary interest in this paper. Whenever GMRES is referred to without an argument, GMRES(10) is understood. The reasons for selecting GMRES instead of another Krylov subspace method are the popularity of GMRES, and that Orthomin and GCR give similar results.

GMRES is also implemented with ILU and MILU preconditioners [17]. Because only GMRES is implemented with these preconditioners, the combination of GMRES with ILU preconditioning is abbreviated ILU, and similarly for MILU. PCGPAK allows the user to pass a shift parameter η to the preconditioning routine that helps guard against failure of the preconditioner by factoring $A + \eta I$ instead of A. This parameter is set to 0 in the experiments because in practice several tries at the incomplete factorization may be necessary to find a workable value for the parameter.

CGNE is also implemented with a preconditioner, found by performing an ILU* factorization on A to get $A \approx LU$. Then CG is applied to the normal equations of the left-preconditioned system $(LU)^{-1}Ax = (LU)^{-1}b$. When combined with this preconditioner, CGNE is denoted as ILCG. This is not the same as forming the normal equations A^TA and then performing an incomplete Cholesky factorization of A^TA; earlier work [7] with such a preconditioner for CGNE applied to two-dimensional problems showed that it also suffered robustness problems.

6.2. Stopping tests. The primary stopping test used is

$$\| Ax_k - b \|^2 \leq 10^{-9}. \tag{30}$$

For KACZ, the algorithm checks the *pseudoresidual* $\tilde{b} - (I - Q)x_k$. When it is less than ϵ_ρ, the *true residual* $b - Ax_k$ is checked. If the true residual is small enough, the algorithm stops. Otherwise the CG tolerance is adjusted by the assignment $\epsilon_\rho \leftarrow (0.7\epsilon_\rho / \| Ax_k - b \|^2)\epsilon$. Initially the tolerance for the pseudoresidual is equal to that on the true residual, i.e., $\epsilon_\rho = 10^{-9}$. Thus if the pseudoresidual's norm is 10 times smaller than the true residual's norm, the tolerance is decreased by 0.07, where the additional factor 0.7 is present to prevent the adjustment being made too often. Once this is done, the CG iteration resumes. The same procedure is used for CIMM and V-RP. However, since V-RP works on an auxiliary vector z_k, the corresponding x_k must be retrieved each time the true residual is computed. The preconditioned PCGPAK routines use the residual of the preconditioned system as the stopping test, so the same tolerance adjustment procedure is used with them. When resumed this way, the preconditioner is not recomputed.

Additional stopping tests, corresponding to failure conditions, are also imposed. A maximum number of iterations is set to 4001, except for CGNE which is allowed 8001 iterations. These limits are generous and when a method fails because the maximum allowed iterations are reached, iterating further has little effect.

TABLE 4
Failures among methods.

	$N = 13824$						$N = 216000$					
Method	1	2	3	4	5	6	1	2	3	4	5	6
KACZ												
V-RP			MX						MX			
CIMM									MX			
GMRES			RS	RS					RS	TM		
ILU		RS	RS	RS		RS		UP	UP	UP		
MILU	RS	RS	RS		RS	RS	RS	UP	RS		RS	UP
CGNE			MX	MX					MX	MX		
ILCG		UP	MX	MX		MX		UP	TM	UP		TM

For the test problems there is no guarantee that the preconditioners will exist or provide a better system. The preconditioner is called *unstable* if a zero pivot occurs during the factorization, or if the iterations experience overflow because of the use of a preconditioner.

The PCGPAK routines have one more stopping criterion. If the residual does not change by a difference of at least 10^{-6} over the course of ten successive iterations, the residual is said to *stagnate*. Omitting or relaxing this test simply changes the reason for failure from stagnated residual to maximum iterations reached.

Finally, for the problems with $N = 216000$ a maximum CPU time limit of 20 minutes is set for all of the methods. This is done for budgetary reasons. These error conditions are summarized and given the following codes in the tables of results:

- MX: Maximum iterations reached,
- UP: Unstable preconditioner,
- RS: Residual stagnates,
- TM: Maximum allowed CPU time reached.

6.3. Comparison of RP methods with other methods.

6.3.1. Robustness. Robustness results are shown in Table 4. KACZ is the only algorithm tested that succeeds in all cases, while CIMM and V-RP follow with only one and two failures, respectively. All the other methods fail at least in four cases. Furthermore, KACZ is the only method that solves Problem 3 for $N = 216000$.

Problems with large off-diagonal elements were the source of most of the failures of the preconditioned methods. A possible explanation is that ILU and ILCG ignore fillin positions when forming the incomplete factors and for matrices with large entries far from the central dense band of A, those positions can be large. MILU accounts for the fillin positions by moving their contributions to the main diagonal, but it failed in ten cases. For the test problems MILU still provides incomplete factors that are not good approximations to the factors of A, which is revealed by noting that the initial residual of the preconditioned system often is many orders of magnitude larger than that of the unpreconditioned system. Unpreconditioned GMRES fails on Problems 3 and 4, for which A has an indefinite symmetric part, but it succeeds on Problems 2, 5, and 6, which also have indefinite symmetric parts. This confirms that positive definiteness of $A + A^T$ is sufficient but not necessary for convergence. Unpreconditioned CGNE fails on Problems 3 and 4, which have a poor distribution of eigenvalues for the conjugate gradient algorithm, viz., they have many small eigenvalues.

Can the robustness of the GMRES(k)-based methods be improved by increasing the number k of vectors stored? To answer this, these algorithms were run again on the test problems of size $N = 13824$, for which they failed. Values of k up to 40 give no improvement in the robustness. Furthermore, since for some problems the GMRES(k)-based methods report a stalled residual after only a few iterations, those problems were also tested again by letting them run for as long a time as consumed by the successful RP methods. This resulted in no significant change in the error or residual norms, even when several hundred additional iterations were performed.

Why are RP methods more robust than GMRES algorithms? Each iteration of GMRES(k) minimizes the components of the residual contributed by the eigenvectors of A restricted to a Krylov subspace. If the residual is orthogonal to that subspace, the method stalls. Even when the residual is not orthogonal to the Krylov subspace, the minimization is over a subspace of dimension k, which becomes negligible for large N. However, every projection of KACZ and V-RP minimizes the residual on a subspace of dimension $N/9$, a fixed fraction of the problem size. Although each iteration requires more work than one iteration of GMRES(k), a more robust method results.

It should be noted that the test problems were chosen partly to be difficult for Krylov subspace methods and may not represent a fair mix of problems normally encountered. It should also be noted that some of the problems with robustness could possibly be obviated by using modified upwind differencing. However, the results show that such a change is unnecessary if RP methods are used. Furthermore, if a nonlinear PDE is being solved and a problem similar to the test problems is generated in the nonlinear iteration, it may not be easy or practical to change the differencing scheme adaptively.

6.3.2. Accuracy. The final residual and error norms are shown in Figs. 2 and 3, respectively, where for each problem the three points plotted are the smallest norm from the RP methods, the smallest from the GMRES-based methods, and the smallest from CGNE or ILCG. No point is plotted for the GMRES methods for Problem 3 or for CGNE or ILCG for Problem 4, because the final norms produced are $\mathcal{O}(1)$ or larger. CGNE produces the smallest residual norm on 8 of the 12 cases, while KACZ does so in two cases and CIMM and MILU in one each. V-RP and CGNE produce the smallest error norm in four cases, GMRES in one case, and ILU in three cases. Although V-RP fails on Problem 3 for both problem sizes by using the full number of iterations allowed, it produces a smaller error norm than all of the other methods, even though its residual norm is larger than that of the successful solvers. An explanation of this behavior lies in Theorem 3.3, which states that V-RP minimizes the error, while KACZ minimizes the error in an elliptic norm.

6.3.3. Speed and efficiency. The robustness property is expected, since RP methods were designed to achieve it. The pleasing result is that in spite of the greater amount of work done per iteration, RP methods are the fastest of the solvers tested on 8 of the 12 experiments with the $N = 13824$ problems run on the Cray X-MP and the $N = 216000$ problems run on the Cray-2. In the other four experiments, an RP method is still competitive with the fastest solver. Figure 4 shows the execution times for problems of size $N = 13824$, where for each problem the three points plotted are the fastest of the RP methods, the fastest of the GMRES-based methods, and the fastest of CGNE or ILCG. Since no Krylov method solved problem 3 and CGNE and ILCG failed on both Problems 3 and 4, those data points are omitted.

The speed of RP methods is caused by their parallelism. Roughly 85 percent of the computation time of KACZ is consumed in computing the matrix-vector products needed for the CG acceleration, and this has a natural parallelism or vector length

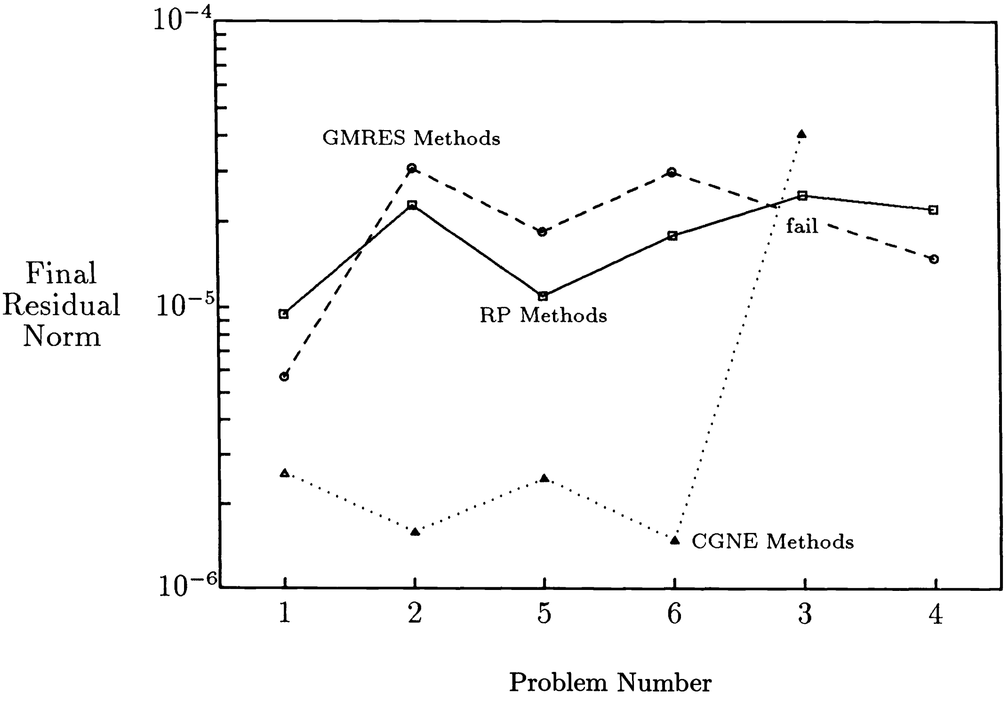


FIG. 2. *Best residual norms for* $N = 13824$.

of $n^2/9$, where $N = n^3$. GMRES computes the matrix-vector product with a vector length of only 7. Even though more parallelism is available for GMRES, since each row can be handled separately, the parallel tasks are small. This is partly because of the data structure that PCGPAK uses to store A, and GMRES can run faster if the matrix-vector product is performed by diagonals rather than rows. However, GMRES benefits by having crucial parts written in Cray Assembly Language, while the RP methods are written exclusively in Fortran. For CGNE, relatively poor performance results primarily from the large number of iterations it requires.

6.3.4. Multiprocessor results. The preceding results were obtained on single processors of Crays using vectorization across the $n^2/9$ block rows C_j^T that comprise a block row A_i^T. The algorithms can also be implemented on a shared-memory multiprocessor by treating the $n^2/9$ subproblems defined by a projector as separate tasks to be assigned to different processors, with vectorizaton available in each subtask from the matrix-vector multiplications. The RP methods were implemented this way on an eight-processor Alliant FX/8. Figure 5 shows the ratio of execution time on one processor of the Alliant FX/8 to that on p processors for KACZ, averaged over all six problems. For comparison, a dashed line with slope 1 is included showing the ratio corresponding to ideal parallelism. On eight processors KACZ runs between five and six times faster, and the efficiency is 63 percent–75 percent.

7. Discussion and summary. Section 2.4 presented an explanation for the properties of RP methods in terms of the ill-conditioning of a matrix arising from

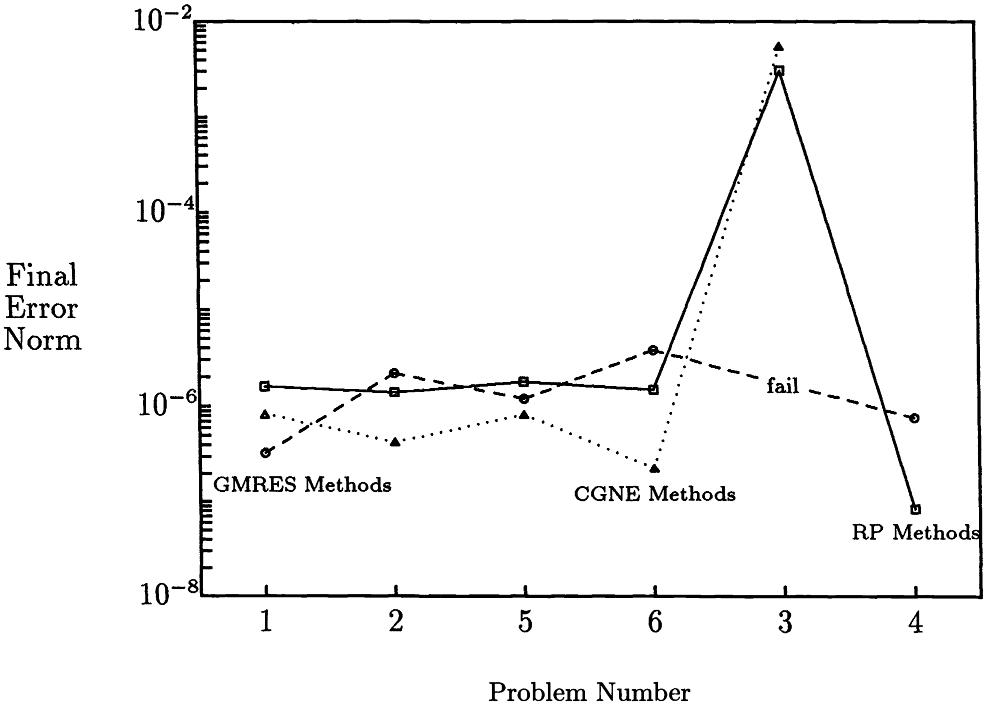


FIG. 3. *Best error norms for* $N = 13824$.

TABLE 5
Number of iterations required for $N = 13824$.

Method	1	2	3	4	5	6
KACZ	8	109	666	715	20	33
V-RP	9	113	4001	813	20	33
CIMM	17	343	572	2000	51	92
CGNE	90	682	8001	8001	325	114

within and across block rows A_i^T. One implication of that argument is proven for the two block row case by Theorem 4.2, which says that KACZ should require fewer CG iterations than CIMM, which in turn should require fewer than CGNE. Table 5 shows that this relationship also holds for the $m = 9$ case tested here; the only exception is that CIMM required fewer iterations than KACZ on Problem 3 with $N = 13824$. Generally, CIMM requires two to three times as many iterations as KACZ, while CGNE requires so many iterations on some problems that it uses the maximum number allowed.

Another implication of the heuristic explanation of RP properties is that fewer iterations are needed for a given RP method when rows that make small angles with each other are placed in the same block row instead of in different block rows. This is confirmed by considering the following problem.

PROBLEM 1′. $\Delta u + 1000 u_y = F$.

This is the same as Problem 1 but with the first derivative term u_y instead of

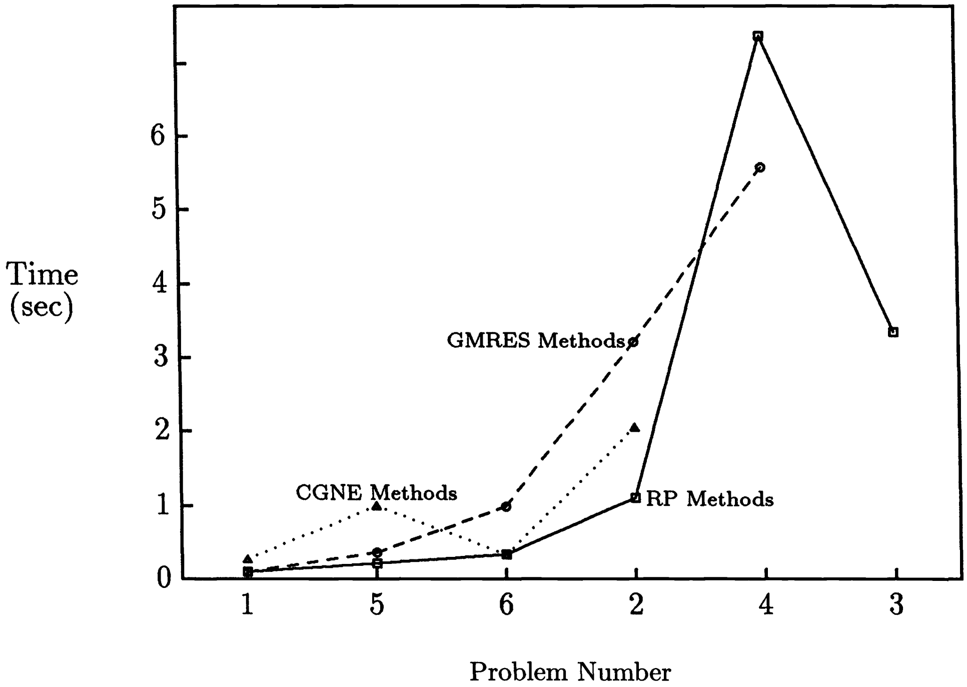


FIG. 4. *Fastest times for $N = 13824$.*

TABLE 6
Number of iterations for Problems 1 and 1′.

Problem	KACZ	V-RP	CIMM	GMRES	CGNE
1	8	9	17	269	90
1′	35	36	73	269	90

u_x. The resulting matrix A has the same spectrum as that of Problem 1, but the number of iterations required increases for the RP methods while remaining constant for GMRES and CGNE, as shown in Table 6 for $N = 13824$. For both Problems 1 and 1′ it can be shown that each row a_i of A makes a small angle with at most two other rows. For Problem 1 these other two rows are located in the same block row as a_i, while for 1′ they are located in different block rows. For Problem 1 the RP methods remove this near linear dependence by implicitly replacing A_i with Q_i, an orthonormal basis matrix for range(A_i). For Problem 1′, the near linear dependence is across block rows and must be handled by the outer CG iteration.

The usefulness of this explanation of RP properties is that it indicates how one should partition the matrix and number the nodes for a given problem. For example, the grid for CFD problems should be arranged so that the lines of nodes that form the block rows A_i^T are aligned in the direction of predominant flow; this occurs in Problem 1 but not in Problem 1′. Furthermore, RP methods like other iterative solvers work well if the system is diagonally dominant. If there are large off-diagonal elements in the diagonal closest to the main diagonal, RP algorithms can still work

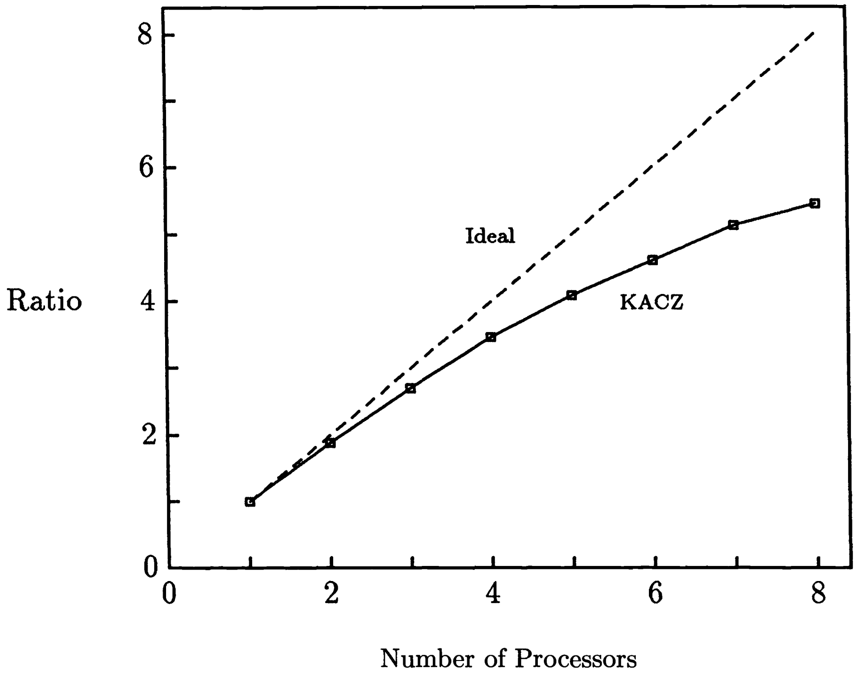


FIG. 5. *Ratio of times for p processors on Alliant FX/8.*

well. Although every worker in the field of RP methods has recognized the importance of having the angles between the block rows large, this seems to be the first guideline on how to achieve it for practical problems without having to actually compute the angles involved.

Although the testing results of this paper are for the structured sparse systems arising from seven-point centered differences, Arioli, Duff, Noailles, and Ruiz [2] have recently applied block Cimmino to more general sparse systems and introduced an innovative preconditioner with good results. In [2] they report on this approach and include tests on some of the problems from this paper.

In summary, CG accelerated row projection methods can have a robustness unmatched by other nonsymmetric solvers, and successfully solve large systems with indefinite real parts and eigenvalues arbitrarily distributed in the complex plane. The row partitioning scheme described here allows large scale parallelism suitable for both vector and multiprocessor machines and yields algorithms competitive in speed with other solvers. The new method V-RP is error minimizing and gives better solutions than the other methods, as well as explicitly reducing the problem size. Finally, the relationship with conjugate gradients applied to the normal equations is shown and an explanation for the behavior of RP algorithms is provided. This explanation is verified both theoretically and numerically.

REFERENCES

[1] R. Ansorge, *Connections between the Cimmino-methods and the Kaczmarz-methods for the solution of singular and regular systems of equations*, Computing, 33 (1984), pp. 367–375.

[2] M. Arioli, I. Duff, J. Noailles, and D. Ruiz, *A block projection method for general sparse matrices*, SIAM J. Sci. Statist. Comput., this issue, pp. 47–70.

[3] S. Ashby, *Chebycode: A Fortran implementation of Manteuffel's adaptive Chebyshev algorithm*, Report No. UIUCDCS-R-85-1203, Department of Computer Science, University of Illinois, Urbana, IL, May 1985.

[4] A. Björck and T. Elfving, *Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations*, BIT, 19 (1979), pp. 145–163.

[5] A. Björck and G. Golub, *Numerical methods for computing angles between linear subspaces*, Math. Comp., 27 (1973), pp. 579–594.

[6] R. Bramley, *Row projection methods for linear systems*, CSRD Tech. Report 881, Center for Supercomputing Research and Development, University of Illinois, Urbana, IL, 1989.

[7] R. Bramley and A. Sameh, *A robust parallel solver for block tridiagonal systems*, CSRD Tech. Report 806, Center for Supercomputing Research and Development, University of Illinois, Urbana, IL, 1988.

[8] ———, *Row projection methods for large nonsymmetric linear systems*, CSRD Tech. Report 957 (revised), Center for Supercomputing Research and Development, University of Illinois, Urbana, IL, 1990.

[9] G. Cimmino, *Calcolo approssimato per le soluzioni dei sistemi di equazioni lineari*, Ric. Sci. Progr. Tecn. Econom. Naz., 9 (1939), pp. 326–333.

[10] P. Eggermont, G. Herman, and A. Lent, *Iterative algorithms for large partitioned linear systems, with applications to image reconstruction*, Linear Algebra Appl., 40 (1881), pp. 37–67.

[11] T. Elfving, *Group-iterative methods for consistent and inconsistent linear equations*, Report LITH-MAT-R-1977-11, Department of Mathematics, Linköping University, Linköping, Sweden, 1977.

[12] ———, *Block iterative methods for consistent and inconsistent linear equations*, Numer. Math., 30 (1980), pp. 1–12.

[13] H. Elman and G. Golub, *Iterative methods for cyclically reduced non-self-adjoint linear systems*, UMIACS-TR-88-87, CS-TR-2145, Department of Computer Science, University of Maryland, College Park, MD, 1988.

[14] H. Elman, *Iterative methods for large, sparse, nonsymmetric systems of linear equations*, Res. Report 229, Department of Computer Science, Yale University, New Haven, CT, 1982.

[15] P. Gilbert, *Iterative methods for the three-dimensional reconstruction of an object from projections*, J. Theoret. Biol., 36 (1972), pp. 105–117.

[16] G. Golub and C. Van Loan, *Matrix Computations*, The John Hopkins University Press, Baltimore, MD, 1983.

[17] I. Gustafsson, *Modified incomplete Cholesky methods*, in Preconditioning Methods: Theory and Applications, D. Evans, ed., Gordon and Breach, New York, 1983, pp. 265–293.

[18] L. Hageman and D. Young, *Applied Iterative Methods*, Academic Press, New York, 1981.

[19] G. Herman, A. Lent, and P. Lutz, *Relaxation methods for image reconstruction*, Comm. ACM, 21 (1978), pp. 152–158.

[20] S. Kaczmarz, *Angenäherte Auflösung von Systemen linearer Gleichungen*, Bull. Intern. Acad. Polonaise Sci. Lettres (Cracouie); Class Sci. Math. Natur.: Seira A. Sci. Math., (1939), pp. 355–357.

[21] C. Kamath, *Solution of nonsymmetric systems of equations on a multiprocessor*, CSRD Tech. Report 591, Center for Supercomputing Research and Development, University of Illinois, Urbana, IL, 1986.

[22] C. Kamath and A. Sameh, *A projection method for solving nonsymmetric linear systems on multiprocessors*, Parallel Computing, 9 (1988/1989) pp. 291–312.

[23] D. Kincaid and D. Young, *Adapting iterative algorithms developed for symmetric systems to nonsymmetric systems*, in Elliptic Problem Solvers, M. Schultz, ed., Academic Press, New York, 1981, pp. 353–359.

[24] D. Kuck, E. Davidson, D. Lawrie, and A. Sameh, *Parallel supercomputing today and the Cedar approach*, Science, 231 (1986), pp. 967–974.

[25] A. Kydes and R. Tewarson, *An iterative method for solving partitioned linear equations*, Computing, 15 (1975), pp. 357–363.

[26] A. Lakshminarayanan and A. Lent, *Methods of least squares and* SIRT *in reconstruction*, J. Theor. Biol., 76 (1979) pp. 267–295.

[27] T. Manteuffel, *The Tchebychev iteration for nonsymmetric linear systems*, Numer. Math., 28 (1977), pp. 307–327.

[28] ———, *Adaptive procedure for estimating parameters for the nonsymmetric Tchebychev iteration*, Numer. Math., 31 (1978), pp. 183–208.

[29] S. Nelson and M. Neumann, *Generalizations of the projection method with applications to SOR theory for Hermitian positive semidefinite linear systems*, Numer. Math., 51 (1987), pp. 123–141.

[30] W. Peters, *Lösung linearer Gleichungssysteme durch Projektion auf Schnitträume von Hyperebenen und Berechnung einer verallgemeinerten Inversen*, Beit. Numer. Math., 5 (1976), pp. 129–146.

[31] Y. Saad and M. Schultz, *Conjugate gradient-like algorithms for solving nonsymmetric linear systems*, Math. Comp., 44 (1985), pp. 417–424.

[32] P. Saylor, *Leapfrog variants of iterative methods for linear algebraic equations*, J. Comp. Appl. Math., 24 (1988), pp. 169–193.

[33] *PCGPAK User's Guide*, Scientific Computing Associates, Inc., New Haven, CT, 1987.

[34] A. Sherman, *An empirical investigation of methods for nonsymmetric linear systems*, in Elliptic Problem Solvers, M. Schultz, ed., Academic Press, New York, 1981, pp. 429–434.

[35] G. Smith, *Numerical Solution of Partial Differential Equations: Finite Difference Methods*, Clarendon Press, Oxford, 1978.

[36] G. Stewart, *On the perturbation of pseudo-inverses, projections, and linear least squares problems*, SIAM Rev., 19 (1977), pp. 634–662.

[37] K. Tanabe, *A projection method for solving a singular system of linear equations*, Numer. Math., 17 (1971), pp. 203–214.

[38] T. Whitney, R. Meany, *Two algorithms related to the method of steepest descent*, SIAM J. Numer. Anal., 4(1967), pp. 109–118.

[39] J. Wilkinson, *Modern error analysis*, SIAM Rev., 13 (1971), pp. 548–568.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 1, pp. 194–226, January 1992

011

A SET OF NEW MAPPING AND COLORING HEURISTICS FOR DISTRIBUTED-MEMORY PARALLEL PROCESSORS*

CLAUDE POMMERELL†, MARCO ANNARATONE†, AND WOLFGANG FICHTNER†

Abstract. New mapping and coloring heuristics are developed to parallelize preconditioned conjugate-gradient-like methods on distributed-memory parallel processors (DMPPs). All these heuristics are fast, with a "quasi" linear time complexity. These schemes are evaluated by solving several systems of linear equations from two-dimensional and three-dimensional semiconductor device simulation on irregular finite-element grids. The benchmarks have been carried out on a simulated 64-processor DMPP with fast communication channels that is under development at the Integrated Systems Laboratory of the Swiss Federal Institute of Technology. The linear systems involved had between 2000 and 75000 unknowns. Depending on the problem under consideration, speedups between 42 and 54 were obtained. Mapping heuristics that use geometric information given by the embedding of the problem graph into physical space were found to be superior to heuristics based solely on the topological information of adjacency structure. Coloring heuristics for vector computers or shared-memory parallel computers were not sufficient for effective parallelization on DMPPs. Coloring strategies for DMPPs had to balance the load for each color, taking into account the mapping and exploiting the locality corresponding to it.

Key words. preconditioned iterative methods, mapping, coloring, distributed-memory parallel processors, finite elements, semiconductor device simulation

AMS(MOS) subject classifications. 65F10, 65W05

1. Introduction.

The efficient solution of large sparse linear systems lies at the heart of many scientific computing problems. Such solutions usually account for the major part of the computing resources needed, both in CPU cycles and memory. The systems themselves arise from the application of appropriate discretization schemes (e.g., finite elements) to partial differential equations, and may exhibit rather irregular sparsity patterns.

While direct solution techniques based on Gaussian elimination have proved to be extremely robust and stable, they are no longer competitive with iterative methods for problems with large numbers of unknowns, as in three-dimensional (3-D) applications [30]. Apart from these reasons, iterative schemes offer unique opportunities to better exploit advanced computer architectures such as parallel and vector machines. Among the wide variety of methods published, preconditioned conjugate-gradient schemes [52], [16] have become popular in recent years.

The vectorization and parallelization of iterative methods applied to irregular problems are more difficult than similar program transformations on dense or regularly structured matrices. While some work on efficient parallelization of these irregular problems has been carried out for shared-memory multiprocessors and for vector processors [54], a similar effort for distributed-memory parallel processors (DMPPs) is still lacking. Moreover, as we shall show in this work, new methodologies are needed when the implementation on DMPPs is considered. The problem grid has to be *mapped* onto the DMPP's interconnection topology to lower the interprocessor communication overhead. As for shared-memory machines, *coloring* allows parallelization of preconditioning and/or vectorization of operations running on one processor.

The paper focuses on aspects of mapping and coloring as they might apply to implementations of iterative methods on DMPPs. First, we present a short overview

* Received by the editors April 5, 1990; accepted for publication (in revised form) July 20, 1990.

† Integrated Systems Laboratory, Swiss Federal Institute of Technology, CH-8092 Zürich, Switzerland.

of the implementation details of sparse matrix solvers; this will allow us to introduce the notation used throughout the study. Second, related results are discussed to review the current state-of-the-art. Third, and before going into the implementation details, we summarize the most important contributions our work gives to the research in this field. Then, the mapping and coloring schemes we developed will be presented, discussed, and analyzed. Finally, we evaluate the efficiency of the methods we devised by applying them to realistic problems stemming from two- and three-dimensional simulation of semiconductor structures.

2. Problem statement.

2.1. Sparse systems of linear equations. A system of N linear equations $Ax = b$ in N unknowns x is called *sparse* if most of the entries a_{ij} of the (square) matrix A are zero. We call M the total number of nonzeros in A, and $D = M/N$ the average number of nonzeros per row in A.

A symmetric or structurally symmetric sparse matrix is often represented as an undirected graph [23]. Each pair of off-diagonal nonzero entries a_{ij} and a_{ji} of the matrix A corresponds to an edge between the vertex i and the vertex j in the graph. There are N vertices and $(M - N)/2$ edges in the graph, and the average degree of a vertex is equal to $D - 1$.

In many applications with sparse systems this graph corresponds to a spatial discretization. Two vertices in the physical discretization that are close to each other are more likely to be connected by an edge than vertices that are physically far apart. The matrices are not always symmetric, but usually structurally symmetric, that is, a_{ij} is nonzero if and only if a_{ji} is nonzero. Graphs derived from finite-element discretizations can be quite irregular, featuring a much finer discretization in some regions than in others. A consequence of Euler's polyhedron formula [29] is that $D < 7$ in two-dimensional triangular grids. Values of D between 7 and 27 are typical for three-dimensional finite-element grids.

2.2. Direct solution methods. A sparse system can be solved by *direct* methods such as Gaussian elimination where A is decomposed into the product of the triangular matrices L and U. The lower and upper triangular matrices L and U are generally much denser than A [23], [49]. This fact is known as the problem of "fill" or "fill-in." Fill causes severe storage requirements that are difficult to estimate because of their strong dependence on the sparsity structure of A. The execution time of a direct solver also depends strongly on this unpredictable fill. Ordering techniques (such as the minimum degree algorithm [51], [25], [41], [24]) have been developed to reduce the fill on sequential computers. However, for very large problems (with several hundreds of thousands of unknowns) the storage requirements make the use of direct methods prohibitive [10], [30].

2.3. Iterative solution methods. Iterative solvers have a known storage overhead (generally, fewer than ten N-vectors and one matrix with the sparsity structure of A). The execution time depends on and can be controlled by the accuracy requirements imposed.

While there is a wide range of iterative methods reported in the literature [28], we focus our results on preconditioned conjugate gradient methods. These schemes are derived from the basic *Conjugate-Gradients algorithm* [31], [26], [27] for symmetric positive-definite systems. Several variants have been developed for nonsymmetric systems, such as Biconjugate Gradients [18], Conjugate Gradients Squared (CGS) [58], ORTHOMIN [60], ORTHODIR [61], GMRES [55], etc.

All the above methods share the same basic operations. Each iteration involves multiplications of the sparse matrix A by a vector, linear operations on vectors (scaling and addition of vectors), vector dot products, scalar divisions, and a test for convergence. The Biconjugate Gradients method requires also a multiplication of the transposed matrix A^T by a vector in each iteration.

2.4. Preconditioning for iterative solvers. *Preconditioning* improves stability and accelerates convergence [27]. A preconditioner Q is an approximation of A, such that $Q^{-1}A \approx I$. In addition to the operations mentioned in the previous section, a system $Qz = r$ must be solved in each preconditioned iteration. Therefore, the preconditioner Q must be easy to invert. Figure 1 shows one way of writing the preconditioned conjugate gradients algorithm [27].

```
$r_0 := b - Ax_0;$
$z_0 := Q^{-1}r_0;$
$k := 0;$
while no convergence loop
    if $k = 0$ then
        $p_0 := z_0;$
    else
        $\beta_k := \frac{z_k^T r_k}{z_{k-1}^T r_{k-1}};$
        $p_k := z_k + \beta_k p_{k-1};$
    end if;
    $\alpha_k := \frac{z_k^T r_k}{p_k^T A p_k};$
    $x_{k+1} := x_k + \alpha_k p_k;$
    $r_{k+1} := r_k - \alpha_k A p_k;$
    $z_{k+1} := Q^{-1} r_{k+1};$
    $k := k + 1;$
end loop;
```

FIG. 1. *The preconditioned conjugate gradient method.*

Incomplete factorizations are often used as preconditioning methods; here, Q is the product of two *sparse* triangular matrices L and U. The sparsity structure of $L + U$ is generally chosen to be identical to that of A. Solving a system with Q then reduces to one forward and one backward substitution step, with the same amount of computation as a sparse matrix-vector product. L and U must be set up before the first iteration starts. ILU, MILU, SSOR, ICCG(0) are examples of such schemes [17], [21].

3. Parallelization and vectorization of preconditioned conjugates-gradient-like methods. An efficient implementation of these methods must focus on the four basic operations mentioned above [54]. Both the sparse matrix-vector products and the substitution steps in the incomplete factorization preconditioner require $O(M)$ operations. The linear operations on vectors and the vector dot products require $O(N)$ operations each.

We summarize below the state-of-the-art in this field.

3.1. Vector and shared-memory parallel computers. Linear operations on vectors and vector dot products can be efficiently implemented on these machines. Sparse matrix-vector multiplication is also efficient on vector computers if gather and scatter hardware supports the execution of vector operations with irregular address patterns [7], [38].

Vectorization of the solution of sparse triangular systems with incomplete factorization preconditioners is more difficult. Figure 2 shows the forward substitution to solve the system $Lx = b$, where L is a sparse lower triangular matrix. The inner loop (the summation) can be vectorized, but it requires on average only $D/2$ multiplications and additions. The outer loop features a data dependence on x: the value of x_i depends on some x_j with $j < i$.

```
$i := 1$;
while $i \leq N$ loop
    $x_i := \frac{1}{\ell_{ii}} \left( b_i - \sum_{j<i,\, \ell_{ij} \neq 0} \ell_{ij} x_j \right)$;
    $i := i + 1$;
end loop;
```

FIG. 2. *Sparse forward substitution.*

The outer loop can only be vectorized if the sparsity structure of L is exploited. A set of contiguous values x_s to x_t can be computed in parallel if they are mutually independent in $Lx = b$, that is, the submatrix

$$\begin{pmatrix} \ell_{ss} & \cdots & \ell_{st} \\ \vdots & \ddots & \vdots \\ \ell_{ts} & \cdots & \ell_{tt} \end{pmatrix}$$

must be diagonal.

In the associated graph the vertices in the set $c = \{s, \cdots, t\}$ must be independent. There is no edge connecting any vertex in c to another vertex in c. In graph theory a *partitioning* of the vertices of a graph into sets of independent vertices is called *coloring.*

After coloring, the outer loop of the substitution can be vectorized *color by color.* The achievable performance depends on the relationship between the size of the color sets and the startup latency of the hardware vector pipeline(s). In general, vector processors benefit from color sets that are as large as possible. Finally, note that finding a coloring scheme that provides minimum number of colors in an arbitrary graph is an NP-complete problem [22].

Similar considerations apply also when parallelizing incomplete factorization preconditioners on shared-memory multiprocessors.

The coloring approach suggested by Schreiber and Tang in 1982 [57] has been used by several researchers to vectorize and parallelize the solution of triangular systems resulting from regular grids, like 5-point or 9-point stencil meshes [34], [1], [45], [50], [6], [10]. Red/black coloring is a well-known technique to create sets of independent vertices in finite difference meshes using a 5-point stencil [28].

Poole and Ortega [50] compare several patterns for 4-coloring regular 9-point stencil meshes. They define a *continuous coloring rule* to obtain maximum vector lengths for a given number of colors on a two- and three-dimensional rectangular grid.

Melhem and Ramarao [47] describe a technique to color irregular meshes of triangular elements. The algorithm proceeds by constructing a *level structure* [23] of the graph. Each level is an outerplanar graph (a planar graph that can be embedded in a plane such that all the vertices lie on a contour and all the edges lie inside this contour) and can be 3-colored. If two sets of three colors are used for all the odd and even levels, a 6-coloring is obtained. The coloring algorithm can be extended by a more extensive backtracking scheme to obtain a 4-coloring.

Bank et al. [10] use a simple greedy algorithm [2] to color irregular finite-element meshes, using at most one color more than the maximum vertex degree in the graph.

Lichnewsky [40], [39] uses ν steps of nested dissection [23] to split the graph into 2^ν subdomains that can be processed in parallel on 2^ν processors. The vertices on the separators are handled serially at the end.

Several of the above authors report a degradation in the convergence rate of multicolor preconditioned iterative methods. However, we are not aware of any generalizable results concerning systems with irregular sparsity patterns. In [30], we give some preliminary results on the influence of coloring and ordering on the convergence rate for problems in semiconductor device simulations.

3.2. Distributed-memory parallel processors. The following aspects arise in the parallelization of preconditioned conjugate-gradient-like methods on distributed-memory parallel processors.

3.2.1. Data distribution. Linear operations on vectors (multiplication of a vector by a scalar and addition of vectors) can be executed without any communication if for each component of the result vector computed on processor P, the corresponding components of the operand vectors are also available on P. It is therefore convenient to partition the components of all vectors in the same way. In terms of graphs, each vertex holds one component of each vector. We can assign at most $\lceil N/P \rceil$ vertices to each of the P processors in order to achieve the optimum load balance for the linear operations.

The vector dot product is computed as follows: First, the processors compute in parallel the dot product on their local vectors. This part of the computation is again load balanced with the vector partitioning above. Then, the partial results are added up globally. Finally, the result is made available to all the processors.

The sparse matrix A, as well as the sparse factors of the preconditioner, should be distributed to parallelize the matrix-vector multiplication and the substitutions. The rows or the columns of the matrices can be partitioned the same way as the vector components. In graph terms, row partitioning corresponds to storing each arc together with its source vertex, and column partitioning corresponds to storing each arc together with its destination vertex. The computational load for the two $O(M)$ operations is balanced if the number of arcs is also distributed equally among the processors.

Because global communication and processor synchronization is required, the vector dot product is dominating the execution time on message-passing DMPPs that feature relatively large message startup time, like the iPSC [8], [42]. Aykanat and Özygüner [8], and Aykanat, Özgüner, Ercal, and Sadayappan [9] propose to overcome this bottleneck by modifying the basic conjugate-gradient algorithm. Their "coarse grain" algorithms require fewer vector dot products, at the expense of some overhead in the local computation. Annaratone, Pommerell, and Rühl [5] showed that the synchronization overhead of the vector dot product can be neglected in DMPPs with low latency communication channels, such as those used in systolic communication.

3.2.2. Mapping. When performing a matrix-vector multiplication, the processors must access some nonlocal components of the operand vector, depending on the sparsity pattern of the local rows. In terms of graphs, the computation for one vertex requires data from all its adjacent vertices.

If the matrix is partitioned row-wise as described above, the whole communication can be performed before the actual computation starts. The processors send the components of their local vectors to all the processors that need them, that is, to those processors with nonzero entries in the corresponding columns of their local sparse matrix. After having received the missing parts of the vectors, a processor can multiply its local matrix by the extended vector (local vector and received nonlocal parts), yielding the local part of the resulting vector.

The assignment of adjacent vertices to processors determines the communication requirements:

1. If both vertices v_1 and v_2 incident to an edge are allocated to the same processor P_1 (Fig. 3), no communication is required by the data dependence represented by this edge.

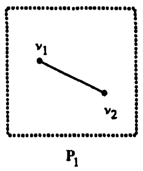


FIG. 3. *Two adjacent vertices on the same processor.*

2. If two adjacent vertices v_1 and v_2 are located on neighboring processors P_1 and P_2 (Fig. 4), their values have to be exchanged through the channel connecting P_1 and P_2.

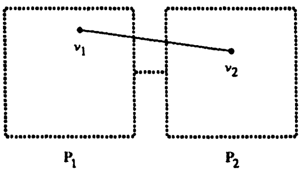


FIG. 4. *Two adjacent vertices on neighboring processors.*

3. If two adjacent vertices v_1 and v_2 are located on non-neighboring processors P_1 and P_3 (Fig. 5), the communication overhead to exchange their respective values increases with the number of hops separating the two processors. The two vertices should therefore be allocated onto neighboring or close processors.
4. The value of a vertex v_1 (located on processor P_1) that is needed by two or more vertices v_2, v_3, ... on the same other processor P_2 (Fig. 6) has to be sent only once to P_2.
5. The value of a vertex v_1 (located on processor P_1) that is needed by two or more vertices v_2, v_3, ... on different processors P_2, P_3, ... (Fig. 7) need

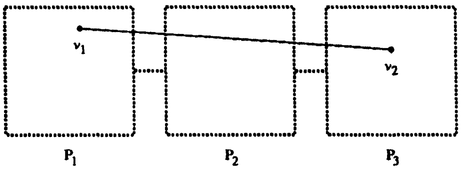


FIG. 5. *Two adjacent vertices on non-neighboring processors.*

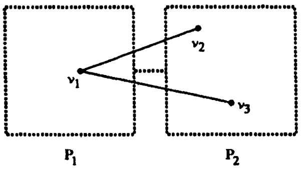


FIG. 6. *Two adjacent vertices to a vertex that are assigned to another processor.*

not be sent directly to all these processors, but can be forwarded by those processors on the shortest path.

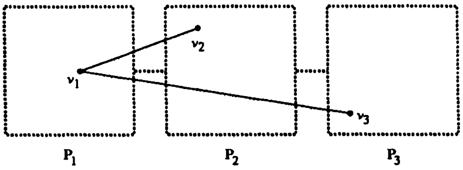


FIG. 7. *Two adjacent vertices to a vertex that are assigned to two other processors.*

6. A set of values to be sent from one processor P_1 to another processor P_2 (Fig. 8) can be packed together into a larger message. This is particularly important on message-passing architectures (e.g., hypercubes) where the overhead for message creation is large.

The problem graph has to be *mapped* onto the interconnection topology to minimize the communication overhead. This mapping still needs to ensure equal load balance as specified above. While this is easy to accomplish for finite-difference graphs, finding an optimal mapping of a complex structure—like a finite-element graph—is NP-complete [22].

The graph is often not mapped onto the full interconnection topology offered by the parallel architecture, but onto a linear array or onto a regular two-dimensional (2-D) mesh (hypercubes or tori contain these topologies). One reason is that the problem domain (the physical space that is discretized by the graph) is usually only two- or three-dimensional.

Sadayappan and Ercal [56] describe a heuristic technique to map square meshes and irregular graphs into a one-dimensional (1-D) array of processors. Vertices are assigned to processors in the order they appear in a level structure partitioning the graph. Load balance is assured, and only local communication is necessary. By using the intersections of two 1-D mappings starting from sets of vertices on orthogonal boundaries of the problem domain, they obtain an unbalanced mapping onto a 2-D mesh of processors. The latter is then rebalanced by local refinement. Aykanat, Ögüner, Ercal, and Sadayappan [9] evaluated these mapping strategies on 1-D and

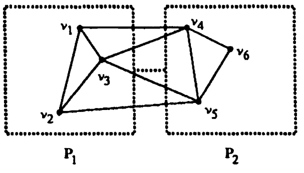


FIG. 8. *Multiple edges between the vertices of two processors.*

2-D topologies embedded into an Intel iPSC/2 hypercube. More values have to be exchanged in 1-D mappings, but they are packed into a smaller number of messages. Because of the large overhead in message creation, the 1-D mappings are preferred for realistic problem sizes.

Fox [20], [19] describes several mapping strategies that use only the topological information given by the graph adjacencies, and no geometric information on the embedding of the graph in the physical problem domain. Simulated annealing and an approach based on neural networks yield the best mappings at the expense of execution time for the mapping heuristic itself. Orthogonal recursive bisection (ORB) recursively cuts the graph into two subgraphs by ordering the vertices according to the difference of their distance to two sets of vertices. The quality of the mapping depends to a large degree on heuristics to obtain orthogonal cuts.

Other strategies have been proposed and tested on small graphs by Bokhari [13], Lee and Aggarwal [37], and Berger and Bokhari [12].

3.2.3. Coloring. The total amount of computation in the two substitution steps for incomplete factorization preconditioning is similar to the matrix-vector product. The total communication requirements are of the same order of magnitude. However, there is much more synchronization required to solve sparse triangular systems.

If two vertices i and j are allocated on two different processors P_i and P_j, and if x_j depends on x_i (that is, if $\ell_{ji} \neq 0$ or $u_{ji} \neq 0$), the processor P_j has to wait until it receives the new value x_i from P_i before it can update x_j. There are up to $2N$ such synchronization points required to perform both forward and backward substitution.

These synchronization points must be grouped together to reduce this unacceptable synchronization overhead. The way to achieve this is again through *coloring.* Each color is distributed to all the processors. Processors handle the same color concurrently. Between the processing of two colors, the needed values from the latest colors are exchanged. If C is the number of colors, a total of $(2C - 2)$ such synchronization points are required to perform the two substitution steps.

Coloring on DMPPs differs from coloring for shared-memory parallelization or vectorization in the following aspects:

1. Dependent vertices allocated on the same processor can be processed in the same color since no communication is required. The edges between local vertices need not be considered for the coloring of the graph. [1]
2. The vertices of each color have to be *distributed* equally onto the processors, in addition to the distribution requirements mentioned in §3.2.1.
3. The number of colors must be as small as possible to reduce the synchronization overhead.

[1] This is not true, however, if the nodes of the DMPP are vector processors and if we want to exploit the vector capabilities of each single node.

4. Coloring *and mapping* (§3.2.2) should be combined to ease the coloring task and to potentially reduce the number of colors that are required.

Because of this, the coloring strategies for shared-memory parallelization and vectorization described in §3.1 can only be partially applied to DMPPs.

Coloring for regular 5-point or 9-point stencil meshes has been described by several researchers, for instance by Saad [53], O'Leary [48], and Lai and Liddell [36].

Tseng [59] used maximum color renumbering to parallelize sparse substitution on a systolic computer. Melhem [43], [46] defines a systolic network for triangular system solution where the sparse matrix is *striped* [44], and each cell handles one stripe. Both these approaches can be applied to irregular sparse systems, and both schemes pass a whole vector through all processors (and through the host in the Warp implementation [3]), so that the communication overhead for each processor is $O(N)$.

4. Mapping and coloring schemes for DMPPs. In this section we describe the mapping and coloring schemes we use to solve sparse systems of equations on a DMPP. The quality of the various schemes for real problems will be compared in §5. In the description of each heuristic we put a shorthand name as (HEUR), which will be used to refer to this method in the tables.

Note that most of the heuristics described in this section are novel approaches for mapping and coloring on DMPPs. Only TOP1-1, GEO2-R, and GREEDY have been used before, albeit in another context.

We are mainly interested in mapping and coloring *irregular* graphs used in 2-D or 3-D finite-element analysis. Besides the requirements developed in §3.2, we want our mapping and coloring heuristics to be *fast*; all our schemes must execute in "almost" linear time, i.e., their time complexity must be smaller than $O(N^{1+\epsilon})$ for any $\epsilon > 0$. In fact, all our heuristics have a time complexity of

$$O(DN + N \log N + N \log P + CN), \tag{1}$$

where N is the number of vertices in the graph, D the average degree of the vertices, P the number of processors, and C the number of colors in the coloring.[2]

4.1. Mapping heuristics. Our mapping heuristics can be classified by the following criteria:

1. The type of information used. The heuristic can be based either only on *topological* information as given by the adjacency structure of the graph, or it can use *geometric* information given by the embedding of the graph into physical space. The latter is available, for instance, in a finite-element graph, where the physical position of the vertices (which are the corner points of elements) is already required to assemble the matrices.
2. The target topology. 1-D mappings map the graph onto a linear array of P processors, while 2-D mappings map it onto a $P_h \times P_v$-mesh of processors. The mappings can be compared for $P = P_h P_v$.

We write a mapping $\mathcal{A}$ as an ordered list of sets $\mathcal{A}_p$, where each $\mathcal{A}_p$ denotes the set of vertices assigned to the processor p. In particular,

$$\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \cdots, \mathcal{A}_{P-1})$$

is a partitioning of the set of vertices of the graph $V(G)$. We define the (*absolute*) *imbalance* $\mathcal{I}(\mathcal{A})$ of a mapping as the maximum number of vertices assigned to a

[2] There is one exception to this rule. If P is not a power of two, one of the heuristics executes in $O(N \log N \log P)$ time. See §4.3.1 for more details.

processor minus the average number of vertices assigned to processors:

$$\mathcal{I}(\mathcal{A}) = \max_p \{|\mathcal{A}_p|\} - \frac{N}{P}.$$

$\mathcal{I}(\mathcal{A})$ corresponds to *load imbalance* in any kind of operation where the amount of computation required for each vertex is the same. If the vertices can be treated independently from each other—e.g., no communication is required—the imbalance can be used to estimate the speedup:

$$S = \frac{N}{\mathcal{I}(\mathcal{A}) + \frac{N}{P}}. \tag{2}$$

Equation (2) gives an estimation for the speedup of linear operations on vectors assigned to the processors of a DMPP according to $\mathcal{A}$. If we assume, moreover, that all rows of the sparse matrix have about the same number of nonzeros, (2) can also be used to estimate the speedup of a matrix-vector product.

A necessary condition for $\mathcal{I}(\mathcal{A})$ to be zero is that the number of processors P divides the number of vertices N. In the general case, we say that a mapping is *perfectly balanced* if its imbalance is smaller or equal to 1:

$$\mathcal{I}(\mathcal{A}) \leq 1.$$

Theorem 4.1 follows immediately.

THEOREM 4.1. *A mapping is perfectly balanced if no processor is assigned more than* $\lceil \frac{N}{P} \rceil$ *vertices.*

All our mapping heuristics try to find a perfectly balanced mapping with minimal communication overhead. In the following, we give a short discussion of the various mapping schemes investigated in this study.

4.1.1. 1-D geometric mapping (GEO1). Geometric mapping onto a linear array of processors is straightforward. We sort the vertices by their x-coordinate and assign the first N/P vertices of the sorted list to the first (leftmost) processor, the next N/P vertices to the second processor, and so on. Instead of sorting according to the x-coordinate, we can choose any direction vector in physical space and sort the vertices according to their projection on this direction.

4.1.2. 2-D geometric mapping (GEO2-S, GEO2-R). We have implemented two schemes to map a graph onto a $P_h \times P_v$-mesh using geometric information.

The first method works on the two dimensions separately (GEO2-S). First, the vertices are sorted according to their x-coordinate. This sorted list of vertices is partitioned by $P_v - 1$ vertical cut lines into P_v sets of N/P_v vertices each, and each of these sets is assigned to one row of the mesh of processors. Then each set is sorted again according to the y-coordinate and partitioned by $P_h - 1$ horizontal cut lines into P_h sets, and each of these sets is assigned to one processor of this row.

The second method proceeds by recursively splitting the graph with dimensions alternating (GEO2-R). Let us assume for a moment that the target mesh is square and the number of processors is a power of two: $P_h = P_v = 2^\nu$. The vertices are first split into two sets by one vertical cut line. Then each of these two sets is split by one horizontal cut line, so that we have now four sets of vertices. Each of these four sets is again partitioned by a vertical cut line and then by horizontal cut lines. This procedure is executed recursively ν times, so that at the end there are P sets of vertices to be assigned to the P processors.

The latter approach is similar to the one that Berger and Bokhari [12] described and tested on some very small graphs. For convenience, we extended it also to the case where the processor mesh is not square and where the number of processors is not a power of two.

Note also that instead of using the x- and the y-coordinate, we can use the projection of the vertex onto any pair of independent vectors in the physical space as partitioning criteria for both our schemes.

4.1.3. 1-D topological mapping (TOP1-1, TOP1-2). If geometric information from the physical embedding of the problem graph is not available, we need another way to extract the concepts of position and distance in the graph.

The *(graphical) distance between a vertex v and a vertex w* is usually defined as the length of the shortest path from v to w [29]. Let S be a (nonempty) subset of the vertices of G. We define the *distance of a vertex v from the set S* as the minimum distance of v to any vertex in S:

$$d(v, S) = \min_{w \in S}\{d(v, w)\}.$$

We can now sort the vertices by their graphical distance from the set S. Then we can map this sorted list of vertices onto a linear array of processors in the way discussed §4.1.1, assigning the first N/P vertices of the sorted list to the first (leftmost) processor, the next N/P vertices to the second processor, and so on (TOP1-1).

This type of 1-D *topological mapping* has been used by Sadayappan and Ercal [56] and by Fox [19] to map sparse graphs onto a ring embedded in a hypercube.

Several vertices will have the same graphical distance from the set S. For a 2-D finite-element graph the number of vertices having the same distance from a small set S is $O(\sqrt{N})$. For a 3-D finite-element graph this number is $O(N^{2/3})$. Thus a variant of the 1-D topological mapping is to use the distance from a second subset of vertices T as a second sorting criterion for all the vertices having the same distance from the first set S (TOP1-2).

4.1.4. 2-D topological mapping (TOP2). The idea of using the distance of each vertex from two starting sets S and T can be used to derive a 2-D mapping of the graph onto a $P_h \times P_v$-mesh. The distances from the two sets S and T form a two-dimensional coordinate system in the graph, attributing to each vertex v its *position* relative to S and T by the pair $(d(v, S), d(v, T))$. Note that different vertices can have the same position.

The same procedure as in 2-D geometric mapping can thus be applied. We first sort the vertices according to their distance to S. We partition this sorted list into P_v sets of N/P_v vertices each and assign each of these sets to one row of the mesh of processors. The vertices of each set are then sorted according to their distance to T, and they are mapped to the row of processors in a similar way to the one shown in §4.1.3.

4.2. Coloring heuristics. Coloring is used to partition the vertices into sets (*colors*) of independent vertices. As pointed out in §3.2.3, local dependences need not be considered, that is, vertices assigned to the same processor of a DMPP can have the same color. We can define three categories of coloring strategies, depending on their relationship with the mapping strategy:

1. Mapping and coloring are done *independently* of each other. In this case, the above property is not exploited, and all vertices in one color must be

independent of each other regardless of their assignment to processors. We can thus use the mapping heuristics that have been developed for vector processors and shared-memory multiprocessors described in §3.1.

2. Coloring is done *after* mapping. After the mapping heuristic has assigned processors to vertices, the edges between vertices mapped on the same processor can be identified and ignored in the coloring heuristic. The first requirement for mapping strategies mentioned in §3.2.2 ensures that as many edges as possible can be deleted this way.
3. Coloring is done *before* mapping. Pairs of adjacent vertices that have received the same color must be assigned to the same processor in the later mapping stage. The mapping scheme has to be adapted to satisfy this additional requirement.

We have implemented the five following coloring heuristics.

4.2.1. Greedy coloring (GREEDY). This scheme colors the graph independently of the assignment of vertices to processors. The algorithm has been described by Aho, Hopcroft, and Ullman [2]. For each color it sweeps once through the whole set of vertices (in the order they are numbered). If a vertex is not yet colored and is not adjacent to any vertex having this color, it is assigned this color. The maximum number of colors is equal to one plus the maximum degree of any vertex, but usually fewer colors are needed to color all the vertices of the graph. Greedy coloring has been used by Bank et al. [10] to color irregular finite-element meshes for vectorization and shared-memory parallelization.

4.2.2. Greedy coloring after mapping (GRAFT). We now map first, then remove all edges between vertices assigned to the same processor, and apply greedy coloring to this reduced graph. As fewer edges restrict the coloring sweeps, possibly fewer colors are needed than in independent greedy coloring.

4.2.3. Balanced greedy coloring (BALANCED1, BALANCED2, BALANCED3). The previous scheme addresses the first and the third points we made about coloring on DMPPs in §3.2.3, ignoring local dependences and reducing the number of colors. The present scheme now addresses point 2 by trying to balance the distribution of the vertices of each color, even if this increases the number of colors.

In extension to the notation introduced in §4.1, let $\mathcal{C}^c$ be the set of vertices colored with the cth color. Let $\mathcal{A}_p^c$ be the set of vertices of color c assigned to processor p: $\mathcal{A}_p^c = \mathcal{A}_p \cap \mathcal{C}^c$. The *coloring imbalance of color* c, $\mathcal{I}^c(\mathcal{A},\mathcal{C})$, is defined as the maximum number of vertices of color c assigned to a processor minus the average number of vertices of color c assigned to processors:

$$\mathcal{I}^c(\mathcal{A},\mathcal{C}) = \max_p \left\{ |\mathcal{A}_p^c| \right\} - \frac{|\mathcal{C}^c|}{P}.$$

In a *balanced coloring scheme*, we put an upper bound $\mathcal{I}_{\max}$ on the coloring imbalance, forcing

$$\forall c : \mathcal{I}^c(\mathcal{A},\mathcal{C}) \leq \mathcal{I}_{\max}.$$

The resulting number of colors might be larger than in a coloring where the coloring imbalance is not limited.

Our balanced greedy coloring is implemented as follows: for each color, we try to color at each stage one vertex of each processor. We accept at most $\mathcal{I}_{\max}$ stages where we could not color vertices from all processors before passing to the next color. This coloring has to be performed after the mapping anyway, so we disregard local dependences in the mapping (BALANCED1 with $\mathcal{I}_{\max} = 1$, BALANCED2 with $\mathcal{I}_{\max} = 2$, and BALANCED3 with $\mathcal{I}_{\max} = 3$).

4.2.4. Level coloring (LEVEL). This and the following coloring schemes are performed before a suitable mapping scheme is carried out.

Let $S = L_0(S)$ be a (nonempty) subset of the vertices of the graph G. For each $k \geq 1$, the set $L_k(S)$ is called *kth level around S*, defined recursively as being the set of all vertices of G that are adjacent to a vertex in previous level $L_{k-1}(S)$, but do not belong to any of the levels $L_0(S), \cdots, L_{k-1}(S)$.
The collection of sets

$$\mathcal{L}(S) = (L_0(S), L_1(S), \cdots, L_{e(S)}(S))$$

is a *level structure around S* of G. S is called the *starting set* of the level structure. See Appendix A for a more rigorous derivation of the properties of level structures.

Level coloring colors all the vertices of the even-numbered levels with a first color and all the vertices of the odd-numbered levels with a second color. Isochromatic vertices in different levels are not connected because of Theorem A.2. Two vertices inside the same level, however, may be connected by an edge. This is why we have to map *whole levels together* so that these dependences inside a level are local to a processor.

A 1-D mapping heuristic on a level colored graph can be derived from our first 1-D topological mapping scheme TOP1-1 described in §4.1.3. We need to assign all the vertices having the same distance from the set S together. Each level that was partitioned on more than one processor in the original mapping is now entirely assigned to the processor that had the most of the vertices of this level (TOP1-1*).

The resulting mapping may not be perfectly balanced, but its imbalance cannot be larger than the largest level of the level structure:

$$\mathcal{I}(\mathcal{A}) \leq \max_i \{|L_i(S)|\}.$$

4.2.5. Level intersection coloring (INODE). Two level structures of the same graph can be combined into a *double level structure* $\mathcal{L}(S,T)$, defined as being the collection of the sets formed by the intersections of the levels of $\mathcal{L}(S)$ and $\mathcal{L}(T)$. The intersections $L_{ij}(S,T) = L_i(S) \cap L_j(T)$ are called *I-nodes.*

Level intersection coloring colors all the vertices that belong to the same I-node (that is, that have the same distances to the sets S and T) with the same color. Two I-nodes that are included into adjacent levels of either $\mathcal{L}(S)$ or $\mathcal{L}(T)$ are colored differently. In Appendix B we show how four colors can be used to obtain this coloring. As all the vertices of an I-node have the same color, we have to assign *whole I-nodes together* to the same processor when mapping the intersection level colored graph to a DMPP.

We can derive 1-D and 2-D mappings for intersection level colored graphs from all our three topological mapping heuristics in §§4.1.3 and 4.1.4. For our first 1-D topological mapping scheme TOP1-1, we can proceed as in §4.2.4, assigning whole levels together, so that I-nodes will certainly not be split (TOP1-1*). For the two schemes based on the graphical distance to two sets TOP1-2 and TOP2, each I-node will now be assigned to the processor that had most of the vertices of this I-node in the original topological mapping (TOP1-2* and TOP2*).

Again, we do not necessarily obtain a perfectly balanced mapping. The imbalance of the mapping based on I-nodes cannot be larger than the size of the largest I-node:

$$\mathcal{I}(\mathcal{A}) \leq \max_{i,j} \{|L_{ij}(S)|\}.$$

Note that no I-node can be larger than any level of the two level structures $\mathcal{L}(S)$ and $\mathcal{L}(T)$ it is derived from. Therefore, this upper bound on the imbalance of the mapping of the intersection level colored graph is certainly less than or equal to the smallest bound on the imbalance of the mappings of the graph level colored by any of these level structures.

4.3. Implementation. We will show now how all the heuristics presented above can be implemented with the complexity bound (1) fixed at the beginning of this section. The construction of data structures that support the data exchange during the parallel execution of the algorithms with fixed mapping and coloring requires an even harder implementation effort than these mapping and coloring heuristics. However, we will not go into these implementation details.

4.3.1. Geometric mappings. The 1-D geometric mapping heuristic (GEO1) only needs to *sort* all the N vertices according to a direction. To achieve this, standard sorting algorithms like *Quicksort* or *Heapsort*, which have a time complexity of $O(N \log N)$ [35], can be used.

The first 2-D geometric mapping heuristic (GEO2-S) also starts by sorting the N vertices according to the first dimension. In a second step, it has to sort P_v arrays of N/P_v vertices each according to the second dimension. The overall time complexity is still $O(N \log N)$.

Calculating the complexity of the second 2-D geometric mapping heuristic, recursive splitting (GEO2-R), is a bit more complicated. Assume that, in the first stage, we split the graph into P_1 subsets of equal size. We can again do this by sorting the N vertices and cutting the sorted array into P_1 pieces. In the second stage, we split each of these P_1 subsets into P_2 subsets, and we do this by sorting P_1 arrays of size N/P_1. This procedure is repeated recursively until we have P sets of vertices that can be assigned to the processors. Let us assume it takes r recursive stages, and we split into $P_1, P_2, P_3, \cdots, P_r$ pieces at each stage. The total number of operations will be a constant multiple of

$$\begin{aligned} n_{\text{oper}} &= N \log N + P_1 \frac{N}{P_1} \log \frac{N}{P_1} + \cdots + P_1 P_2 \cdots P_{r-1} \frac{N}{P_1 P_2 \cdots P_{r-1}} \log \frac{N}{P_1 P_2 \cdots P_{r-1}} \\ &= N(r \log N - (r-1) \log P_1 - (r-2) \log P_2 - \cdots - \log P_{r-1}). \end{aligned}$$

The product of the number of subsets P_k generated at each stage k must be equal to P:

$$P_1 P_2 \cdots P_r = P.$$

The P_k's are integers, and none of them can be smaller than 2. The number of stages can therefore not be larger than the logarithm base two of the number of processors:

$$r \leq \log_2 P.$$

This implies a bound on the complexity of the implementation:

$$n_{\text{oper}} \leq N \log P \log N.$$

This is the case mentioned in footnote 2 in the introduction of §4, and it does not satisfy the bound (1). If P is a power of 2, however, we need not sort the array completely at each stage: We rather use the *find* algorithm [32] to find the median element of the array. This algorithm, which uses the same divide-and-conquer strategy as Quicksort, executes in linear time and has the side effect that it reorders the array such that all the keys that are smaller than the median precede the median, and all the keys that are larger than the median succeed it. The bound then becomes

$$n_{\text{oper}} \leq N \log P,$$

which satisfies (1).

4.3.2. Topological mappings and level structures. The construction of a level structure can be derived directly from Corollary A.5 in Appendix A, as depicted in Fig. 9.

```
$L_0(S) := S$;
$R := V(G) - S$;
$k := 0$;
while $R \neq \emptyset$ loop
    $k := k + 1$;
    $L_k(S) := \emptyset$;
    for each vertex $v \in L_{k-1}(S)$ loop
        for each vertex $w \in Adj(v)$ loop
            if $w \in R$ then
                $L_k := L_k \cup \{w\}$;
                $R := R - \{w\}$;
            end if;
        end loop;
    end loop;
end loop;
$e(S) := k$;
```

FIG. 9. *Construction of a level structure.*

The algorithm in Fig. 9 takes $O(DN)$ time, as the inner loop is executed (on the whole) at most DN times, once for each edge.

If we put the vertices of G into a list in the order we remove them from the set of remaining vertices R, this list can be used directly for both *level coloring* and our first 1-D topological mapping scheme.

The distances to the two starting sets S and T, as they can be read from the two level structures $\mathcal{L}(S)$ and $\mathcal{L}(T)$, form a discrete two-dimensional "coordinate system" in the graph. In this coordinate system we can again use the same 2-D geometric mapping schemes to generate our second 1-D and our 2-D topological mapping scheme as well as our level intersection coloring scheme.

4.3.3. Greedy-type coloring schemes. Figure 10 shows a possible implementation of the greedy coloring algorithm described by Aho, Hopcroft, and Ullman [2]. After execution, the integer C holds the number of colors needed, and the sets C_c hold the vertices for each color c.

```
$R := V(G)$;
$c := 0$;
while $R \neq \emptyset$ loop
    $C_c := \emptyset$;
    $I := R$;
    while $I \neq \emptyset$ loop
        select a vertex $v \in I$
        $C_c := C_c \cup \{v\}$;
        $I := I - \{v\}$;
        $R := R - \{v\}$;
        for each vertex $w \in Adj(v)$ loop
            if $w \in I$ then
                $I := I - \{w\}$;
            end if;
        end loop;
    end loop;
    $c := c + 1$;
end loop;
$C := c$;
```

FIG. 10. *The greedy coloring algorithm.*

The outer loop in Fig. 10 is executed C times (where C is the number of colors), the second loop is executed N times on the whole, and the inner loop is executed DN times on the whole. The set assignment inside the outer loop takes at most $O(N)$ time, so the complexity of the basic greedy algorithm is $O(DN + CN)$.

The algorithm for greedy coloring after mapping is very similar to the simple greedy coloring algorithm, one just has to put an additional guard in the inner condition statement.

Figure 11 shows an implementation of the balanced greedy algorithm with a given mapping $\mathcal{A}$. The complexity analysis is the same as for the simple greedy algorithm, except that the innermost loop on the processors may execute up to $CP\mathcal{I}_{\max}$ times more. We ignored this term in our upper bound on the complexity (1), assuming that it is smaller than the other terms in (1).

5. Evaluation.

5.1. Description of the experiments.

We will now quantify the quality of the different mapping and coloring schemes by comparing the speedups obtained on conjugate-gradients-like methods applied to a set of test problems. The following parameters characterize our experimental environment:

- In each experiment, we applied all possible combinations of our mapping and our coloring schemes and then ran the same *number of iterations* of the same CG-like *algorithm* on the same *DMPP* (same *topology* and same *architecture*) to solve the same *problem.* Speedup comparisons are relative to the same algorithm without coloring on a single processor with the same architecture.
- All our examples are systems of linear equations used for 2-D or 3-D semiconductor device simulation with irregular finite-element grids. We used *conjugate gradients* (CG) to solve the symmetric (not necessarily positive-definite)

```
for each processor $p$ loop
    $R_p := \mathcal{A}_p$;
end loop;
$c := 0$;
while $R_0 \cup R_1 \cup \cdots \cup R_{P-1} \neq \emptyset$ loop
    $C_c := \emptyset$;
    for each processor $p$ loop
        $I_p := R_p$;
    end loop;
    $flag := false$;
    while $I_{\mathrm{curr}} < I_{\max}$ loop
        for each processor $p$ loop
            if $I_p = \emptyset$ then
                $flag := true$;
            else
                select a vertex $v \in I_p$
                $C_c := C_c \cup \{v\}$;
                $I_p := I_p - \{v\}$;
                $R_p := R_p - \{v\}$;
                for each vertex $w \in Adj(v)$ loop
                    find $p_w$ such that $w \in \mathcal{A}_{p_w}$
                    if $p_w \neq p$ AND $w \in I_{p_w}$ then
                        $I_{p_w} := I_{p_w} - \{w\}$;
                    end if;
                end loop;
            end if;
        end loop;
        if $flag = true$ then
            $I_{\mathrm{curr}} := I_{\mathrm{curr}} + 1$;
        end if;
    end loop;
    $c := c + 1$;
end loop;
$C := c$;
```

FIG. 11. *The balanced greedy coloring algorithm.*

systems for the solution of the Poisson equation for the electrostatic potential, and *biconjugate gradients* (BiCG), *conjugate gradients squared* (CGS), or ORTHOMIN(1), all with ILU or MILU preconditioning, to solve the nonsymmetric systems for the solution of the continuity equations for the carrier concentrations. We used the right preconditioned versions of all the algorithms. The preconditioner was ILU. For symmetric positive definite matrices, ILU preconditioning is equivalent to incomplete Cholesky preconditioning, so our CG algorithm is nothing but ICCG(0).

- We measured the execution time for *ten iterations* of the respective algorithm, regardless of the accuracy achieved. We will explain in §5.5 why we selected this standard of comparison rather than comparing the execution times to achieve a given accuracy.

- The topology we used is a 2-D *mesh* of 8 *rows by* 8 *columns of processors.* Each processor communicates with its four neighbors in the mesh via *send* and *receive* statements. Sending or receiving one number to or from a neighbor processor takes half the time of one multiply-add operation on one processor. The processors are AMD 29000 microprocessors with floating-point coprocessor. These numbers reflect the hardware of a DMPP currently under development at our Laboratory [4]. The choice of the topology is justified by the results from the works of Dally [15] and Johnsson [33], who show that a two-dimensional topology represents a natural choice for a DMPP with 64 processors.
- Our experiments were carried out on the *K9 simulator* [11]. Besides reproducible timing measurements, it delivers detailed information about the processor utilization and the communication volume. Although the K9 timing model for the hardware under development is accurate within ±8 percent, we will present here only relative results.

  We are aware of the risks associated with the use of simulators to predict speedup results. Note, however, that DMPPs supporting systolic communication tend to be easier to model than those employing message-passing. This is typically due to the former's high degree of synchronism in the communication and deterministic latency in information transfer.

Section 5.2 presents in detail the results obtained on one particular 2-D problem. Execution times will be broken down into the time required for the different kinds of operations, and also in the contributions of computation, communication, and idle time due to synchronization. Section 5.3 shows in a similar way the results for one particular 3-D problem. Section 5.4 summarizes the results obtained on other experiments.

5.2. A 2-D problem.

5.2.1. The problem. The problem we used in this experiment arises in the two-dimensional simulation of the electrical behavior of a short-channel MOSFET. The system of linear equations (whose sparsity structure is reflected by a rather irregular grid) has $N = 2674$ unknowns and $M = 18404$ nonzeros in the sparse matrix, with an average of 6.88 nonzero elements per row.

5.2.2. Obtained speedups. As an example, Table 1 shows the speedups obtained for preconditioned BiCG using different pairs of a mapping and a coloring. Each column of a table lists the values for the same mapping scheme, and each row lists the values for the same coloring scheme.

The first observation we can make concerns the type of information used for the mapping heuristic. Our geometric mapping strategies achieve significantly better results than our topological ones. The choice of the target topology, that is, whether the graph is mapped onto a 1-D array of 64 processors or onto an 2-D mesh of 8 by 8 processors, influences the speedup only marginally. The same holds true for the two variants for 2-D geometric and 1-D topological mapping. The effort of balancing the coloring pays off, and the improvement in speedup is noticeable. The level and level intersection coloring heuristics deliver some speedup, but are outperformed by the other schemes.

5.2.3. Analysis of relative contributions. We will now justify the speedup results by a detailed analysis. As we stated in §§2.3 and 2.4, the basic operations used

TABLE 1

Speedup obtained on a 64-processor torus for 10 iterations of preconditioned BiCG on a 2-D problem. With the exception of the numbers in italics, *all the results refer to the novel mapping and/or coloring techniques presented in this study.*

Coloring	*Mapping*					
	GEO1	GEO2-S	GEO2-R	TOP1-1	TOP1-2	TOP2
GREEDY	41.04	36.02	*35.99*	*21.00*	21.44	25.39
GRAFT	40.74	35.16	33.40	20.69	21.90	26.02
BALANCED1	44.74	44.68	43.37	22.34	23.11	26.87
BALANCED2	44.63	44.90	43.61	22.66	23.32	27.14
BALANCED3	44.03	44.10	43.45	22.93	23.58	27.14
LEVEL				8.92		
INODE				10.64	16.97	16.12

in all these methods are linear operations on vectors, vector dot products, multiplications of the sparse matrix A or its transpose A^T by a vector, and solutions of systems of the type $Qz = r$ with the preconditioner Q. Table 2 shows the relative percentages of the execution time of the serial implementation of the different algorithms taken by the different types of operations.

TABLE 2

Decomposition of the execution time of the serial algorithms (2-D *problem*) *into linear operations* ($\alpha v + w$)*, vector dot products* ($v^T w$)*, matrix-vector products* (Av)*, transposed matrix-vector products* ($A^T v$)*, and preconditioning operations* ($Q^{-1} v$).

Type of operation	*Contribution to*			
	CG	CGS	BiCG	ORTHOMIN(1)
$\alpha v + w$	13.16%	14.06%	16.72%	16.00%
$v^T w$	8.19%	5.95%	5.99%	13.91%
Av	35.14%	35.74%	17.99%	31.32%
$A^T v$	—	—	16.97%	—
$Q^{-1} v$	43.50%	44.24%	42.32%	38.77%

We will now examine the speedups that can be obtained on each of these basic operations.

5.2.4. Linear operations and vector dot products. For linear operations on vectors and vector dot products, we stated above that the achievable speedup depends only on a single parameter of the mapping, which is the imbalance $\mathcal{I}(\mathcal{A})$ defined in §4.1. Almost all our mappings are perfectly balanced. The only exceptions are the specialized variants of topological mappings used for level and level intersection coloring. Table 3 shows the speedups obtained for a linear operation (the SAXPY operation $v = v + \alpha w$) and for the vector dot product and compares them to the theoretical speedup computed from (2).

Besides confirming our estimation for the speedup of linear operations, Table 2 reinforces the assertion made by Annaratone, Pommerell, and Rühl [5] mentioned in §3.2.1 that the vector dot product does *not* hamper the parallelization of CG-like algorithms on a DMPP with fast interprocessor communication channels.

5.2.5. Sparse matrix-vector products. The efficiency of sparse matrix-vector products and transposed matrix-vector products depends only on the mapping. Table 4 lists the speedup of the matrix-vector product as well as the overhead caused by communication and load imbalance. The communication overhead is expressed by the average percentage of the total execution time that the processors spend actively

TABLE 3

Estimated and obtained speedup of linear operations on vector and vector dot products for balanced mappings and for specialized non-balanced mappings (2-D *problem*).

Mapping	$\mathcal{I}(\mathcal{A})$	*Theoretical speedup*	*Obtained speedup*	
			$v = v + \alpha w$	$v^T w$
(balanced)	0.2	63.67	63.24	49.79
TOP1-1*	120.2	16.51	16.48	15.55
TOP1-2*	22.2	41.78	41.60	36.56
TOP2*	20.2	43.13	42.94	37.14

communicating. The load imbalance overhead is expressed by the average percentage idle time of the processors. Such idle times occur in part during the data exchange if the communication is not perfectly synchronous, and are in part due to the unequal distribution of nonzeros, even if the unknowns are equally distributed.

TABLE 4

Speedup, communication time, and synchronization overhead (*idle time*) *for sparse matrix-vector product* (2-D *problem*).

Mapping	*Speedup*	*Communication*	*Idle*
GEO1	47.57	6.1%	5.3%
GEO2-S	47.06	5.1%	9.6%
GEO2-R	45.37	5.0%	12.7%
TOP1-1	24.07	12.7%	23.2%
TOP1-2	24.49	12.5%	23.1%
TOP2	28.36	8.0%	30.4%
TOP1-1*	12.14	2.3%	73.9%
TOP1-2*	21.77	10.9%	32.4%
TOP2*	24.59	6.8%	40.2%

Here we see the reason for the bad performance of the topological mappings compared to the geometric mappings. The pure communication time is not the only factor that limits the speedup, but the synchronization overhead due to idle time in the communication pattern is substantial. Had we used a DMPP architecture with slower communication and large setup times for message creation, our results would show a clearer advantage in using 1-D mappings, the increase in communication volume being more than compensated for by decomposition of the communication into fewer pieces. Note that these considerations are independent from whether or not the DMPP permits the overlap of communication with computation, as some message-passing architectures do.

The implementation of the transposed matrix-vector products involves the same amount of communication, but some additional arithmetic operations between the different communication steps are required. This increases the synchronization overhead slightly, so that transposed matrix-vector multiplication is slower than the nontransposed version. This explains why the speedup for biconjugate gradients, the only algorithm that requires transposed matrix-vector products, is slightly smaller than that of the other algorithms.

5.2.6. Preconditioning operators. The operator for incomplete factorization preconditioners involves one forward and one backward substitution step to solve sparse triangular systems of linear equations. The parallelization depends both on the mapping and on the coloring. Table 5 shows the speedups obtained for each viable combination of a mapping and a coloring.

TABLE 5
Speedups for the preconditioning operator ($Q^{-1}v$) (2-D *problem*).

Coloring	*Mapping*					
	GEO1	GEO2-S	GEO2-R	TOP1-1	TOP1-2	TOP2
GREEDY	35.16	28.88	28.97	18.40	18.81	22.65
GRAFT	34.78	27.71	25.61	17.69	19.70	23.31
BALANCED1	42.25	42.75	41.67	20.91	21.82	25.48
BALANCED2	42.03	43.33	41.89	21.37	22.17	26.15
BALANCED3	41.03	41.93	41.79	22.05	22.76	25.81
LEVEL				6.80		
INODE				8.84	13.11	10.76

The parallelization of the preconditioning operator turns out to be the least efficient part of the algorithms, thus dominating their total efficiency. Comparing Tables 4 and 5, we see that the quality of the mapping gives a major contribution to the speedup of the preconditioning.

During the execution of the substitution steps, one color is processed after the other, and data have to be exchanged between two colors. Table 6 shows the number of colors obtained for each coloring heuristic.

TABLE 6
Number of colors (2-D *problem*).

Coloring	*Mapping*					
	GEO1	GEO2-S	GEO2-R	TOP1-1	TOP1-2	TOP2
GREEDY	7	7	7	7	7	7
GRAFT	5	4	4	5	5	5
BALANCED1	7	7	6	9	8	8
BALANCED2	6	6	6	7	7	7
BALANCED3	6	5	5	6	5	6
LEVEL				2		
INODE				4	4	4

From the number of colors, we see that the communication inside the preconditioning operator is very fine-grained. On the average, only one to two numbers are exchanged with each neighbor between two colors. The number of colors alone does not explain the impact of the coloring on the performance of the preconditioning. Table 7 shows the total coloring imbalance, which is the sum of the coloring imbalances $\mathcal{I}^c(\mathcal{A},\mathcal{C})$ of all the colors, as defined in §4.2.3.

TABLE 7
Total coloring imbalance $\mathcal{I}(\mathcal{A},\mathcal{C}) = \sum_c \mathcal{I}^c(\mathcal{A},\mathcal{C})$ (2-D *problem*).

Coloring	*Mapping*					
	GEO1	GEO2-S	GEO2-R	TOP1-1	TOP1-2	TOP2
GREEDY	24.2	40.2	45.2	39.2	31.2	29.2
GRAFT	35.2	46.2	53.2	54.2	44.2	44.2
BALANCED1	1.2	1.2	1.2	2.2	2.2	1.2
BALANCED2	3.2	3.2	3.2	4.2	3.2	4.2
BALANCED3	6.2	3.2	4.2	5.2	4.2	4.2
LEVEL				278.2		
INODE				229.2	116.2	138.2

The total coloring imbalance is the final parameter that controls the quality of a coloring with a given mapping. Our balanced greedy coloring scheme (and any other balanced scheme based on another basic coloring heuristic) achieves the best results by minimizing this parameter. The number of colors is a smaller issue. As long as it is chosen small, the parameter $\mathcal{I}_{\max}$ of the heuristic does not influence the efficiency significantly.

5.3. A 3-D problem.

5.3.1. The problem. This sparse system arises in the 3-D simulation of a bipolar transistor. The transistor is discretized using bricks, tetrahedra, pyramids, and prisms [14]. The sparse matrix involved has $N = 20412$ rows and $M = 252074$ nonzeros, or, on the average, 12.35 nonzeros per row.

We will limit the discussion here to the presentation of only those results that may add new insight to the problems discussed so far.

5.3.2. Obtained speedups. Table 8 shows the speedups obtained for CGS with ILU preconditioning (the tables for other algorithms are similar).

TABLE 8

Speedup obtained on a 64-processor torus for 10 *iterations of preconditioned* CGS *on a* 3-D *problem. With the exception of the numbers in* italics, *all the results refer to the novel mapping and/or coloring techniques presented in this study.*

Coloring	*Mapping*					
	GEO1	GEO2-S	GEO2-R	TOP1-1	TOP1-2	TOP2
GREEDY	29.62	47.79	*48.28*	*27.94*	29.91	36.48
GRAFT	29.70	46.03	45.94	27.74	29.85	34.37
BALANCED1	33.26	51.88	51.27	32.14	32.07	40.76
BALANCED2	33.23	51.87	51.30	32.10	32.23	40.75
BALANCED3	33.38	51.77	51.06	32.13	32.19	40.85
LEVEL				7.68		
INODE				9.16	15.92	17.95

While for the 2-D example, 1-D mappings were almost as successful as 2-D mappings, this is not true anymore for this 3-D example. 1-D mappings are now significantly worse than 2-D mappings. Also, the performance gap between topological and geometric mappings is here reduced. Level and level intersection colorings still perform poorly.

5.3.3. Analysis of relative contributions. The relative contributions of the matrix-vector products and the preconditioning operations to the total execution time of the algorithms are 2 to 5 percent higher than for the 2-D problem. This is due to the higher number of nonzeros per row.

5.3.4. Linear operations and vector dot products. The speedups and speedup predictions for the linear operations on vectors are about the same as in Table 3. As the problem size has increased, the vector dot product suffers even less from communication and synchronization overhead and shows a speedup of 61.72 on balanced mappings.

5.3.5. Sparse matrix-vector products. Table 9 shows the speedup of the matrix-vector product and the overhead caused by communication and synchronization in a similar manner as in Table 4.

The reason for the relatively poor performance of 1-D geometric mapping is that each of the 64 "slices" is not dense enough to separate the graph. The values of

TABLE 9

Speedup, communication time, and synchronization overhead (idle time) for sparse matrix-vector product (3-D *problem*).

Mapping	*Speedup*	*Communication*	*Idle*
GEO1	32.32	10.8%	16.9%
GEO2-S	51.71	4.2%	6.4%
GEO2-R	51.49	3.8%	7.4%
TOP1-1	31.23	6.7%	28.0%
TOP1-2	31.80	6.8%	27.7%
TOP2	40.06	5.5%	20.7%
TOP1-1*	10.33	1.0%	80.8%
TOP1-2*	24.23	6.6%	42.4%
TOP2*	30.18	4.0%	40.7%

almost all vertices have to be transmitted to the four processors around the processor to which they are assigned.

5.3.6. Preconditioning operators. Table 10 shows the speedups obtained for each combination of mapping and coloring. The results of this table confirm the results in §5.2.6.

TABLE 10

Speedups for the preconditioning operator ($Q^{-1}v$) (3-D *problem*).

Coloring	*Mapping*					
	GEO1	GEO2-S	GEO2-R	TOP1-1	TOP1-2	TOP2
GREEDY	23.59	41.57	42.59	21.80	24.70	30.06
GRAFT	23.32	38.66	38.82	21.53	24.69	25.77
BALANCED1	29.75	48.96	48.40	28.74	28.21	37.06
BALANCED2	29.70	48.95	48.46	28.71	28.50	36.77
BALANCED3	29.95	48.83	47.99	28.76	28.43	36.90
LEVEL				5.70		
INODE				7.67	10.48	11.40

5.4. More problems. Table 11 summarizes the results from the above and other simulations. All statements made above apply also to these problems. We listed the best speedup we obtained, and the mapping and coloring for which this speedup occurred.

TABLE 11

Summary of the results obtained on a 64*-processor torus for preconditioned* CGS *on a set of problems.* LDD *is the* 2-D *problem discussed in detail before,* BIPOL20K *is the* 3-D *problem discussed in detail before.*

Name	*N*	*D*	*FE mesh*	*Best speedup*	*Best mapping*	*Best coloring*
BBIG	2069	6.8	2-D	44.99	GEO2-R	BALANCED2
LDD	2674	6.9	2-D	47.28	GEO2-S	BALANCED2
BIPOL4K	4913	12.0	3-D	42.60	GEO2-S	BALANCED1
DR15	15564	8.6	3-D	49.42	GEO2-R	BALANCED1
BIPOL20K	20412	12.3	3-D	51.88	GEO2-S	BALANCED1
DRAM0	22680	24.4	3-D	54.25	GEO2-S	BALANCED1
DRAM1	22680	6.6	3-D	54.52	GEO2-S	GREEDY
BP25C	76926	20.1	3-D	43.80	GEO1	BALANCED2

Two-dimensional geometric mapping together with balanced greedy coloring with coloring imbalance 1 or 2 turned out to give best speedups in most cases. On the DRAM1 experiment, greedy coloring performed slightly (less than 1 percent) better than the balanced variant. This experiment was carried out on a regular 7-point stencil grid, where the greedy heuristic was the only one to find the optimal 2-coloring. The BP25C experiment involved three unknowns per grid point on a 3-D grid.

5.5. A note on convergence. Our speedup figures above are computed by running exactly 10 iterations of each method on the problem, regardless of the accuracy achieved. As coloring reorders the rows of the matrix, it modifies the fill of a complete factorization and thus modifies the quality of the incomplete factorization preconditioner. As a consequence, the convergence properties of the preconditioned algorithms change with coloring.

In normal application a user is not interested in how long it takes to perform a certain number of iterations of an iterative solver, but how long it takes to achieve a certain accuracy in the result. Since coloring modifies the convergence behavior of the preconditioned iterative method, the number of iterations to achieve a given accuracy will differ from one coloring scheme to another or to a sequential implementation, so that the user sees a different speedup value than the speedup over a fixed number of iterations.

At first glance, one should assume that we should give speedup results that compare the time to achieve a given accuracy in the result, rather than just comparing a fixed number of iterations. However, what accuracy should be selected?

In (transient) semiconductor device simulation, the application our examples come from, a great number of linear systems with the same sparsity structure have to be solved. These systems fall into four classes: the three types of systems derived from the three semiconductor drift-diffusion equations, and the coupled systems involving the three equations. The first three types of systems involve one unknown per discretization grid point. The latter type involves three unknowns per grid point. Its matrix is written as a block matrix with the same sparsity structure as the matrices of the single systems, but with blocks of size 3×3. The matrices of one of the first three classes are symmetric positive definite and can be solved using Conjugate Gradients, all the other systems are nonsymmetric, and CGS is the fastest solution method (see Heiser, Pommerell, Weis, and Fichtner [30] for details).

In Fig. 12 we plotted the speedup as a function of the desired accuracy. The accuracy is expressed as the relative reduction of the norm of the residual, and is varied from 1 to 10 digits of precision. The speedup is computed by dividing the time required to achieve this reduction of the error norm on a single processor by the time required on 64 processors. The sequential implementation uses the initial ordering of the unknowns produced by our grid generator [14], the parallel implementation uses GEO2 mapping and BALANCED1 coloring. The four curves represent one system out of each of the four classes mentioned above. All these systems stem from the same simulation of a trench DRAM cell on an irregular 3-D grid with 15564 unknowns, and thus all have the same sparsity structure.

The strong variations in the curves of Fig. 12 show the difficulties if we base our speedup calculations on the time to achieve a given accuracy. If we want three digits of precision in the residual for the fourth (coupled) system in Fig. 12, our 64-processor implementation shows a speedup of 130 over the 1-processor implementation. If we want one digit more, the speedup drops to less than 12. Note also how the speedup figure varies as we choose a different system with the same sparsity structure. For very high accuracies (higher than required for the surrounding nonlinear solver), these

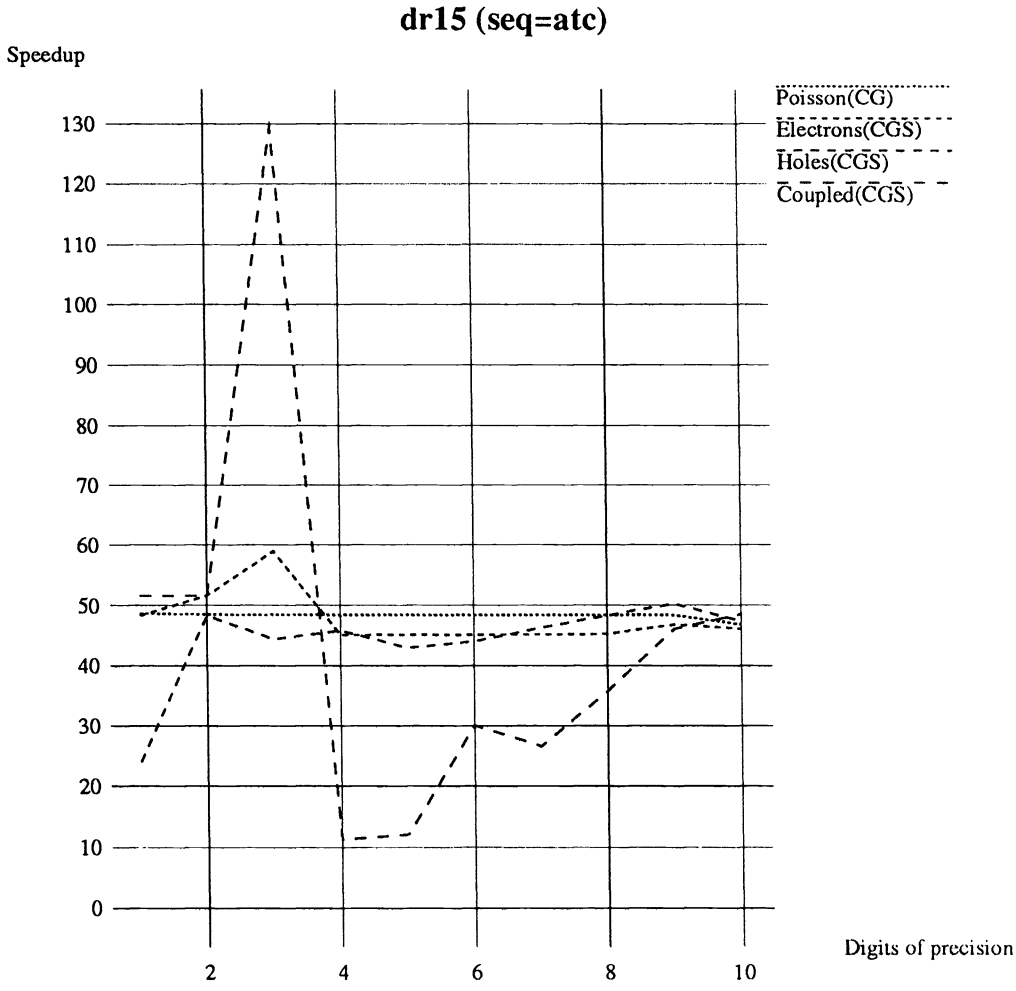


FIG. 12. *Speedup as a function of the desired accuracy, for four different matrices with the same sparsity structure. The sequential implementation uses the ordering as produced by the grid generator.*

speedups approach the value of 49, as predicted from the speedup on 10 iterations.

The fact that sometimes the parallel implementation requires fewer iterations should not lead to precipitate hopes, however. The convergence of the sequential implementation can be improved by other reordering techniques. For instance, using a natural ordering on the grid points (i.e., sorting the grid points by one of their coordinates) can reduce the number of iterations. Figure 13 shows the speedup as a function of the desired accuracy, as in Fig. 12, but uses the sequential implementation with natural ordering as the basis for comparison.

6. Summary of the findings derived from the experiments presented. The findings of the detailed analysis presented in §5 are summarized below:

1. The mapping should be kept perfectly balanced.
2. Geometric mappings perform better than topological mappings.

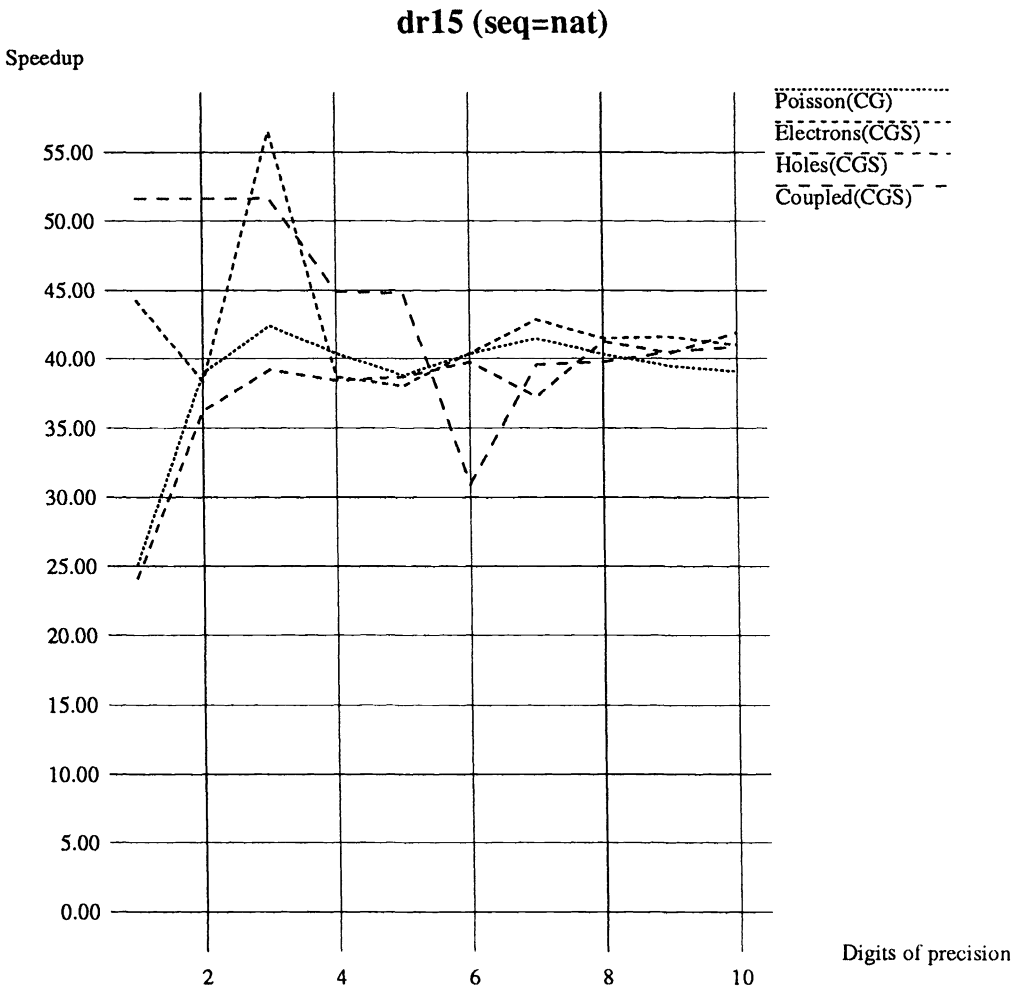


FIG. 13. *Speedup as a function of the desired accuracy, for four different matrices with the same sparsity structure. The sequential implementation uses a natural reordering of the grid points.*

3. For problems on 2-D meshes, 2-D geometric mappings do not bring much improvement over 1-D geometric mappings. On 3-D meshes, however, the improvement is significant.
4. The difference between the two variants for 2-D geometric and for 1-D topological mapping is small.
5. Level coloring and level intersection coloring perform generally much worse than the various greedy schemes.
6. Coloring can speed up the preconditioning operator almost up to the speedup of the matrix-vector multiplication, as long as each color is distributed equally to the processors.

We think that, whenever possible, geometric information on the physical location of a vertex of the sparse graph should be exploited in the mapping heuristic. Extracting good mappings from the adjacency structure only is much more expensive. 2-D mappings are never worse than 1-D mappings.

Load balancing is the most important aspect in parallelizing this type of operation on a DMPP. A good approximation to this is to balance carefully the data involved in each computation step. Balanced coloring schemes are the answer to obtain high speedups on distributed incomplete factorization preconditioning.

We cannot make any statement about the impact of the coloring on convergence. In some of our experiments, some colorings degraded seriously the convergence of the algorithms, while in other experiments, some colorings even improved it. These effects will be the topic of a future study.

7. Further improvements.

7.1. Topological mappings and level structures. Our current heuristic for choosing the starting sets for the level structures is quite simple. The first starting set S simply contains a single vertex at a corner of the problem domain. The second starting set T is then a single vertex lying in the middle level of $\mathcal{L}(S)$.

Further refinement would improve the quality of the topological mappings as well as of the level-based colorings:

- More edges local to the same level, thus reducing the communication volume.
- Smaller maximum size of a level or of an I-node, thus reducing the mapping and the coloring imbalances. For I-nodes this means that the two level structures should be more or less *orthogonal* to each other.

It is not clear whether these refinements can be easily implemented, and more experimentation is therefore necessary.

Sadayappan and Ercal [56] propose to use the vertices of orthogonal boundaries of the problem as starting sets to obtain orthogonal level structures, but this strategy failed on our case studies.

Some of the ideas presented by Fox [19] could be used to achieve these refinements. His ORB scheme could also extend our set of 2-D topological mapping heuristics.

7.2. Geometric mappings. In the experiments we only used the projections of the vertices onto the x- and y-axes as partitioning criteria. We could vary the projection vectors to obtain a slightly better mapping.

7.3. Mappings in general. Our current geometric mapping schemes use *only* geometric information and ignore the adjacency structure of the graph. We could experiment with mixed heuristics that use both topological and geometric sources of information.

We define the imbalance only by looking at the number of vertices assigned to each processor. However, vertices with many neighbors require more computation than vertices with few neighbors. We can include the degree of each vertex into our definition of imbalance.

7.4. Coloring after mapping. We derived our balanced coloring schemes from the greedy coloring algorithm. The ideas we presented here could be included into other general coloring algorithms, like the one proposed by Melhem and Ramarao [47].

8. Conclusions. We proposed and evaluated a set of fast mapping and coloring heuristics for the implementation of preconditioned conjugate-gradients-like methods on a DMPP. Most of these heuristics were new approaches to the problem. We evaluated the quality of the generated mappings and colorings using systems of linear equations stemming from semiconductor device simulation. Depending on the problem under consideration, we achieved speedups between 42 and 54 on the simulator of a 64-processor DMPP currently under development.

We found that mapping heuristics that use geometric information given by the embedding of the problem graph into physical space were superior to heuristics relying only on the topological information given by the nonzero structure of the system matrix. Coloring heuristics devised for vector computers or shared-memory parallel computers were not sufficient for the efficient implementation on a DMPP. Mapping and coloring had to be combined to yield best performance. The best approach was to map first and then to use a coloring heuristic that balances the number of vertices per processor for each color.

A. Level structures. Most of the theory in this section is well known [23]. We included it here as an introduction to the new notions presented in Appendix B.

Recall the notion of distance of a vertex from a set of vertices introduced in §4.1.3. The *eccentricity* of a vertex $v \in V(G)$ is the maximum distance of v from any vertex of G. We call the maximum distance of any vertex of G to S *eccentricity of the set* S:

$$e(S) = \max_{v \in V(G)} \{d(v, S)\}.$$

We call *kth level around* S the set $L_k(S)$ of vertices whose distance to any vertex in S is k:

$$L_k(S) = \{v \in V(G) : d(v, S) = k\}.$$

The following properties derive immediately:

$$L_0(S) = S, \tag{3}$$

$$L_k(S) \neq \emptyset \Leftrightarrow 0 \leq k \leq e(S), \tag{4}$$

$$i \neq j \Rightarrow L_i(S) \cap L_j(S) = \emptyset, \tag{5}$$

$$L_0(S) \cup L_1(S) \cup \cdots \cup L_{e(S)}(S) = V(G). \tag{6}$$

The collection of sets

$$\mathcal{L}(S) = (L_0(S), L_1(S), \cdots, L_{e(S)}(S))$$

is a *level structure around* S of G. It follows from (4), (5), and (6) that $\mathcal{L}(S)$ is a *partitioning* of $V(G)$. S is called the *starting set* of the level structure.

LEMMA A.1. *Any vertex adjacent to a level lies either in the preceding or in the succeeding level:*

$$\forall k : \mathrm{Adj}(L_k(S)) \subset L_{k-1}(S) \cup L_{k+1}(S).$$

Proof. Let $v \in L_i(S)$ and $w \in L_k(S)$ be adjacent to each other. Then we know that $d(v, S) = i$ and $d(w, S) = k$.

- If $i = k$, then $v \in L_k(S)$, so $v \notin \mathrm{Adj}(L_k(S))$ (by definition).
- If $i < k - 1$, there is a path of length $i + 1$ from a vertex in S to w going through v. This leads to a contradiction:

$$d(w, S) \leq (i + 1) < k.$$

- If $i > k+1$, there is a path of length $k+1$ from a vertex in S to v going through w. This also leads to a contradiction:

$$d(v,S) \leq (k+1) < i.$$

It follows that if $v \in \mathrm{Adj}(L_k(S))$, i can only be equal to $k-1$ or $k+1$. □

Theorem A.2 states a fundamental property of level structures, which follows immediately from Lemma A.1.

THEOREM A.2. *Two distinct nonsubsequent levels are not adjacent:*

$$\forall i,j : |i-j| > 1 \Rightarrow \mathrm{Adj}(L_i(S)) \cap L_j(S) = L_i(S) \cap \mathrm{Adj}(L_j(S)) = \emptyset.$$

The level $L_k(S)$ can be viewed as the vertices of the *quotient graph* $G/\mathcal{L}(S)$, where $L_i(S)$ is adjacent to $L_j(S)$ only if some vertex of $L_i(S)$ is adjacent to some vertex of $L_j(S)$ in G [23]. Theorem A.2 can be formulated as the following corollary.

COROLLARY A.3. *$G/\mathcal{L}(S)$ is a path.*

Finally, Theorem A.4 and Corollary A.5 give hints on how to construct a level structure.

THEOREM A.4. *$L_{k+1}(S)$ is the set of vertices adjacent to $L_k(S)$ that are not in $L_{k-1}(S)$.*

$$\forall k : L_{k+1}(S) = \mathrm{Adj}(L_k(S)) - L_{k-1}(S).$$

Proof.

- Subtracting $L_{k-1}(S)$ on both sides of the set inequality in Lemma A.1 leads to

$$\mathrm{Adj}(L_k(S)) - L_{k-1}(S) \subset L_{k+1}(S).$$

- Let $v \in L_{k+1}(S)$. There is a shortest path from v to S of distance $k+1$. The first internal vertex on this path is at distance k from S, thus v is adjacent to $L_k(S)$: $L_{k+1}(S) \subset Adj(L_k(S))$. As L_{k-1} and L_{k+1} are disjunct (5), we can write

$$L_{k+1}(S) \subset \mathrm{Adj}(L_k(S)) - L_{k-1}(S). \qquad \square$$

The same proof applies to Corollary A.5.

COROLLARY A.5. *$L_{k+1}(S)$ is the set of vertices adjacent to $L_k(S)$ that are not in a former level $L_i(S)$ with $i < k$:*

$$\forall k : L_{k+1}(S) = \mathrm{Adj}(L_k(S)) - (L_0(S) \cup L_1(S) \cup \cdots \cup L_k(S)).$$

Theorems A.2 and A.4 justify the term *adjacent levels* for two subsequent levels $L_k(S)$ and $L_{k+1}(S)$. Furthermore, we will call nonadjacent levels *independent.*

The quotient graph $G/\mathcal{L}(S)$ has the interesting property that it is 2*-colorable.* The vertices are assigned the two colors alternatively, as shown in Fig. 14.

B. Double level structures. Two level structures of the same graph can be combined into a *double level structure* $\mathcal{L}(S,T)$, defined as being the collection of the sets formed by the intersections of the levels of $\mathcal{L}(S)$ and $\mathcal{L}(T)$:

$$\mathcal{L}(S,T) = \{L_{ij}(S,T) : 0 \leq i \leq e(S), 0 \leq j \leq e(T)\},$$

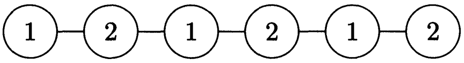


FIG. 14. *2-coloring of a path.*

where $L_{ij}(S,T)$ (hereafter abbreviated L_{ij}) is defined as

$$L_{ij} = L_i(S) \cap L_j(T).$$

As levels are disjunct (5), so are their intersections:

$$(i,j) \neq (k,\ell) \Rightarrow L_{ij} \cap L_{k\ell} = \emptyset. \tag{7}$$

The following property follows from the construction and from (6):

$$L_{00} \cup L_{01} \cup \cdots \cup L_{10} \cup \cdots \cup L_{e(S)e(T)} = V(G). \tag{8}$$

Some intersections in $\mathcal{L}(S,T)$ may be empty, but the set of nonempty intersections in $\mathcal{L}(S,T)$ is a *partitioning* of $V(G)$. The concept of independent levels from Theorem A.2 extends to *independent intersections.*

THEOREM B.1. *Two intersections from independent levels in $\mathcal{L}(S)$ or in $\mathcal{L}(T)$ are independent:*

$$\forall i,j,k,\ell : |i-j| > 1 \vee |k-\ell| > 1 \Rightarrow \text{Adj}(L_{ij}) \cap L_{k\ell} = L_{ij} \cap \text{Adj}(L_{k\ell}) = \emptyset.$$

The intersections L_{ij} can be viewed as vertices (called from now on *I-nodes*) of the quotient graph $G/\mathcal{L}(S,T)$. Theorem B.1 can then be formulated as the following corollary.

COROLLARY B.2. *$G/\mathcal{L}(S,T)$ is a subgraph of a 9-point stencil mesh.*

We define a *9-point stencil mesh* with r rows and c columns $\mathcal{N}_{rc}$ as a graph with the following properties:

- $V(\mathcal{N}_{rc})$ contains rc vertices numbered v_{ij}, with $0 \leq i < r$ and $0 \leq j < c$.
- A pair of distinct vertices v_{ij} and $v_{k\ell}$ is connected by an edge only if

$$(|i-k| \leq 1) \quad \text{and} \quad (|j-\ell| \leq 1).$$

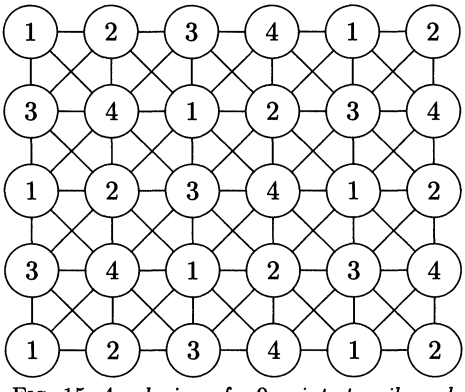


FIG. 15. *4-coloring of a 9-point stencil mesh.*

A 9-*point stencil mesh* is 4-colorable. Figure 15 gives one of the possible color assignments. Of course, any subgraph of such a graph, like the quotient graph $G/\mathcal{L}(S,T)$, is also 4-colorable and can be colored similarly.

Acknowledgments. We thank Josef Bürgler, Paolo Conti, and Gernot Heiser for providing us with the real data used in semiconductor device simulators. Stefan Ludwig implemented one of the mapping heuristics.

REFERENCES

[1] L. ADAMS, *M-step preconditioned conjugate gradient methods*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 452–463.

[2] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *Data Structures and Algorithms*, Addison–Wesley, Reading, MA, 1983.

[3] M. ANNARATONE, E. ARNOULD, T. GROSS, H. T. KUNG, M. LAM, O. MENZILCIOGLU, AND J. A. WEBB, *The Warp computer: Architecture, implementation, and performance*, IEEE Trans. Comput., C-36 (1987), pp. 1523–1538.

[4] M. ANNARATONE, M. FILLO, K. NAKABAYASHI, AND M. VIREDAZ, *The* K2 *parallel processor: Architecture and hardware implementation*, in Proc. 17th IEEE–ACM Symposium on Computer Architecture, Seattle, WA, June 1990.

[5] M. ANNARATONE, C. POMMERELL, AND R. RÜHL, *Interprocessor communication speed and performance in distributed-memory parallel processors*, in Proc. 16th IEEE-ACM Symposium on Computer Architecture, Jerusalem, June 1989, pp. 315–324.

[6] C. C. ASHCRAFT AND R. G. GRIMES, *On vectorizing incomplete factorization and* SSOR *preconditioners*, SIAM J. Sci. Statist. Comput., 8 (1988), pp. 122–151.

[7] C. C. ASHCRAFT, R. G. GRIMES, J. G. LEWIS, B. W. PEYTON, AND H. D. SIMON, *Progress in sparse matrix methods for large linear systems on vector supercomputers*, Internat. J. Supercomput. Appl., 1 (1987), pp. 10–30.

[8] C. AYKANAT AND F. ÖZGÜNER, *Large grain parallel conjugate gradient algorithms on a hypercube multiprocessor*, in Proc. 1987 International Conference on Parallel Processing, S. K. Sahni, ed., Aug. 1987, pp. 641–644.

[9] C. AYKANAT, F. ÖZGÜNER, F. ERCAL, AND P. SADAYAPPAN, *Iterative algorithms for solution of large sparse systems of linear equations on hypercubes*, IEEE Trans. Comput., C-37 (1988), pp. 1554–1568.

[10] R. E. BANK, W. M. COUGHRAN, JR., M. A. DRISCOLL, R. K. SMITH, AND W. FICHTNER, *Iterative methods in semiconductor device simulation*, Comput. Phys. Comm., 53 (1989), pp. 201–212.

[11] P. BEADLE, C. POMMERELL, AND M. ANNARATONE, K9: *A simulator of distributed-memory parallel processors*, in Supercomputing '89, ACM–IEEE, Nov. 1989, pp. 765–774.

[12] M. J. BERGER AND S. H. BOKHARI, *A partitioning strategy for nonuniform problems on multiprocessors*, IEEE Trans. Comput., C-36 (1987), pp. 570–580.

[13] S. H. BOKHARI, *On the mapping problem*, IEEE Trans. Comput., C-30 (1981), pp. 207–214.

[14] P. CONTI, N. HITSCHFELD, AND W. FICHTNER, *Ω—an octree-based mixed element grid allocator for adaptive* 3d *device simulation*, IEEE Trans. Comput.-Aided Design of Integrated Circuits and Systems, CAD-10 (1991).

[15] W. J. DALLY, *Performance analysis of k-ary n-cube interconnection networks*, IEEE Trans. Comput., C-39 (1990), pp. 775–785.

[16] C. DEN HEIJER, *Preconditioned iterative methods for nonsymmetric linear systems*, in Simulation of Semiconductor Devices and Processes, K. Board and D. R. J. Owen, eds., 1984, pp. 267–285.

[17] H. C. ELMAN, *Iterative methods for large, sparse nonsymmetric systems of linear equations*, Res. Report 229, Department of Computer Science, Yale University, New Haven, CT, 1982.

[18] R. FLETCHER, *Conjugate gradient methods for indefinite systems*, in Proc. Dundee Biennial Conference on Numerical Analysis, G. A. Watson, ed., Springer-Verlag, New York, 1975.

[19] G. C. FOX, *A graphical approach to load balancing and sparse matrix vector multiplication on the hypercube*, in Numerical Algorithms for Modern Parallel Computer Architectures,

M. Schultz, ed., Springer Verlag, New York, 1988, pp. 37–61.

[20] ———, *A review of automatic load balancing and decomposition methods for the hypercube*, in Numerical Algorithms for Modern Parallel Computer Architectures, M. Schultz, ed., New York, 1988, Springer-Verlag, pp. 63–76.

[21] G. Gambolati, G. Pini, and G. Zilli, *Numerical comparision of preconditionings for large sparse finite element problems*, Numer. Methods Partial Differential Equations, 4 (1988), pp. 139–157.

[22] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.

[23] A. George and J. W. H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice–Hall, Englewood Cliffs, NJ, 1981.

[24] ———, *The evolution of the minimum degree ordering algorithm*, SIAM Rev., 31 (1989), pp. 1–19.

[25] A. George and E. Ng, *On the complexity of sparse* QR *and* LR *factorization of finite-element matrices*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 849–861.

[26] G. H. Golub and D. P. O'Leary, *Some history of the conjugate gradient and Lanczos algorithms:* 1948-1976, SIAM Rev., 31 (1989), pp. 50–102.

[27] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The John Hopkins University Press, Baltimore MD, 1983.

[28] L. A. Hageman and D. M. Young, *Applied iterative methods*, in Computer Science and Applied Mathematics, Academic Press, New York, 1981.

[29] F. Harary, *Graph Theory*, Addison–Wesley, Reading, MA, 1972.

[30] G. Heiser, C. Pommerell, J. Weis, and W. Fichtner, *Three dimensional numerical semiconductor device simulation: Algorithms, architectures, results*, IEEE Trans. Comput.-Aided Design of Integrated Circuits and Systems, CAD-10 (1991).

[31] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards., 49 (1952), pp. 409–436.

[32] C. A. R. Hoare, *Proof of a program: Find*, Comm. ACM, 14 (1971), pp. 39–45.

[33] S. L. Johnsson, *Communication efficient basic linear algebra computations on hypercube architectures*, J. Parallel Distributed Computing, 4 (1987), pp. 133–172.

[34] D. Kincaid, T. Oppe, and D. Young, *Vector computations for sparse linear systems*, Tech. Report CNA-189, Center for Numerical Analysis, The University of Texas, Austin, TX, 1984.

[35] D. E. Knuth, *Sorting and Searching*, The Art of Computer Programming, Vol. III, Addison–Wesley, Reading, MA, 1973.

[36] C. H. Lai and H. M. Liddell, *Finite element using long vectors of the* DAP, Parallel Comput., 8 (1988), pp. 351–361.

[37] S.-Y. Lee and J. K. Aggarwal, *A mapping strategy for parallel processing*, IEEE Trans. Comput., C-36 (1987), pp. 433–442.

[38] J. G. Lewis and H. D. Simon, *The impact of hardware gather/scatter on sparse Gaussian elimination*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 304–311.

[39] A. Lichnewsky, *Sur la résolution de systèmes linéaires issus de la méthode des éléments finis par une machine "multiprocesseur"*, Rapport de Recherche 119, INRIA, Centre de Rocquencourt, Février, France, 1982.

[40] ———, *Some vector and parallel implementations for preconditioned conjugate gradient algorithms*, in High-Speed Computation, J. S. Kowalik, ed., NATO ASI Series, Vol. F7, Springer-Verlag, Berlin, Heidelberg, 1984, pp. 343–359.

[41] J. W. H. Liu, *Modification of the minimum-degree algorithm by multiple elimination*, ACM Trans. Math. Software., 11 (1985), pp. 141–153.

[42] O. A. McBryan and E. F. van de Velde, *Hypercube algorithms and implementations*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s227–s287.

[43] R. Melhem, *Determination of stripe structures for finite element matrices*, SIAM J. Numer. Anal., 24 (1987), pp. 1419–1433.

[44] ———, *Iterative solution of sparse linear systems on systolic arrays*, in Proc. 1987 Internat. Conference on Parallel Processing, S. K. Sahni, ed., 1987, pp. 560–563.

[45] ———, *Toward efficient implementation of preconditioned conjugate gradient methods on vector supercomputers*, Internat. J. Supercomput. Appl., 1 (1987), pp. 70–98.

[46] ———, *Parallel solution of linear systems with striped sparse matrices*, Parallel Comput., 6 (1988), pp. 165–184.

[47] R. G. Melhem and K. V. S. Ramarao, *Multicolor reordering of sparse matrices resulting from irregular grids*, ACM Trans. Math. Software, 14 (1988), pp. 117–138.

[48] D. O'Leary, *Ordering schemes for parallel processing of certain mesh problems*, SIAM J. Sci.

Statist. Comput., 5 (1984), pp. 620–632.

[49] S. Pissanetzky, *Sparse Matrix Technology*, Academic Press, Orlando, FL, 1984.

[50] E. L. Poole and J. M. Ortega, *Multicolor* ICCG *methods for vector computers*, SIAM J. Numer. Anal., 24 (1987), pp. 1394–1418.

[51] D. J. Rose, *A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations*, in Graph Theory and Computing, R. C. Read, ed., Academic Press, New York, 1972, pp. 183–217.

[52] Y. Saad, *The Lanczos biorthogonalization algorithm and other oblique projection methods for solving large unsymmetric systems*, SIAM J. Numer. Anal., 19 (1982), pp. 485–506.

[53] ———, *On the design of parallel numerical methods in message passing and shared memory environments*, in Supercomputing, A. Lichnewsky and C. Saguez, eds., INRIA, North-Holland, Amsterdam, 1987, pp. 253–274.

[54] ———, *Krylov subspace methods on supercomputers*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 1200–1232.

[55] Y. Saad and M. H. Schultz, GMRES*: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.

[56] P. Sadayappan and F. Ercal, *Nearest-neighbor mapping of finite element graphs onto processor meshes*, IEEE Trans. Comput., C-36 (1987), pp. 1408–1424.

[57] R. Schreiber and W. Tang, *Vectorizing the conjugate gradient method*, in Proc. Symp. CYBER 205 Applic., Control Data Corporation, Colorado Springs, CO, 1982.

[58] P. Sonneveld, CGS*, a fast Lanczos-type solver for nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 36–52.

[59] P. S. Tseng, *Sparse matrix computations on Warp*, CMU, 1988.

[60] P. K. W. Vinsome, *ORTHOMIN—an iterative method for solving sparse sets of simultaneous linear equations*, in Proc. Fourth SPE Symposium on Reservoir Simulation, Los Angeles, CA, Society of Petroleum Engineers, 1976, pp. 149–160.

[61] D. M. Young and K. C. Jea, *Generalized conjugate-gradient acceleration of nonsymmetric iterative methods*, Linear Algebra Appl., 34 (1980), pp. 159–194.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 1, pp. 227–242, January 1992

MULTILEVEL FILTERING PRECONDITIONERS: EXTENSIONS TO MORE GENERAL ELLIPTIC PROBLEMS*

CHARLES H. TONG[†], TONY F. CHAN[‡], AND C. C. JAY KUO[§]

Abstract. The concept of multilevel filtering (MF) preconditioning applied to second-order selfadjoint elliptic problems is briefly reviewed. It is then shown how to effectively apply this concept to other elliptic problems such as the second-order anisotropic problem, biharmonic equation, equations on locally refined grids and interface operators arising from domain decomposition methods. Numerical results are given to show the effectiveness of the MF preconditioners on these problems.

Key words. multilevel preconditioners, elliptic problems, conjugate gradient method, domain decomposition

AMS(MOS) subject classifications. 65F10, 65N30

1. Introduction.

Preconditioned conjugate gradient (PCG) methods have been a very popular and successful class of methods for solving large systems of equations arising from discretizations of elliptic partial differential equations. With the advent of parallel computers in recent years, there has been increased research into effective implementation of these methods on various parallel computers. Since effective preconditioning plays a critical role in the competitiveness of the PCG methods, many classical preconditioners have been proposed and studied, especially for second-order elliptic problems. Among these are the Jacobi preconditioner (diagonal scaling), the SSOR preconditioner, the incomplete factorization preconditioners (ILU and MILU), and polynomial preconditioners. Many such preconditioners have been very successful in giving high performance, especially when implemented on sequential computers.

In the parallel implementation of PCG methods, the major bottleneck is often the parallelization of the preconditioner. The rest of the PCG methods can usually be parallelized in a straightforward way (the inner product computation is also considered a bottleneck but its wide applicability in other methods has prompted many parallel computer manufacturers to develop a highly optimized and efficient code for it). Unfortunately, for many of the classical preconditioners, there is a fundamental trade-off in the ease of parallelization and the rate of convergence. A principal obstacle to parallelization of many preconditioners that are effective in improving the convergence rate (e.g., SSOR, ILU, and MILU) is the sequential manner these preconditioners use in traversing the computational grid—the data dependence implicitly prescribed by the method limits the amount of parallelism available. Reordering the grid traversal (e.g., from natural to red-black ordering) or inventing new methods

* Received by the editors October 4, 1990; accepted for publication (in revised form) October 25, 1990. This work was supported in part by the Department of Energy contract DE-FG-03-87-ER-25037, the Army Research Office contract DAAL03-88-K-0085, and the National Science Foundation grant contract FDP NSF ASC 9003002.

[†] Sandia National Laboratories, Livermore, California 94551-0969 (chtong@snll-arpagw.llnl.gov). The research of this author was supported by a Defense Advanced Research Projects Agency-sponsored graduate research assistantship (award 26947F) through the University of Maryland Institute of Advanced Computer Studies.

[‡] Department of Mathematics, University of California, Los Angeles, California 90024 (chan@math.ucla.edu).

[§] Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, California 90089 (cckuo@sipi.usc.edu). The research of this author was supported by the University of Southern California Faculty Research and Innovation Fund and a National Science Foundation Research Initiation Award.

(e.g., polynomial preconditioners) to improve the parallelization alone often has an adverse effect on the rate of convergence [8].

The fundamental difficulty can be traced to the global dependence of elliptic problems. An effective preconditioner must account for the global coupling inherent in the original elliptic problem. Preconditioners that use purely local information (such as red-black orderings and polynomial preconditioners) are limited in their ability to improve the convergence rate. On the other hand, global coupling through a naturally ordered grid traversal is not highly parallelizable. The challenge is therefore to construct effective global coupling that is highly parallelizable. We are thus led to the consideration of preconditioners that share global information through a multilevel grid structure (ensuring a good convergence rate) but perform only local operations on each grid level (and are hence highly parallelizable). Preconditioners of the multilevel type for second-order selfadjoint operators have been proposed recently by several researchers, including Bramble, Pasciak, and Xu [6]; Axelsson [1]; Vassilevski [25]; Axelsson and Vassilevski [2]; [3]; Kuo, Chan, and Tong [13]; and Kuznetsov [14].

The main goal of our paper is to employ the main ideas in [6] and [13] to develop algorithms for more general problems, such as second-order anisotropic problems, the biharmonic equation, problems on locally refined grids, and interface operators for domain decomposition methods. Our approach can be viewed as adapting the projection operators and eigenvalue estimates in [6] to more general problems. On the other hand, the filtering framework of [13] offers the flexibility in designing the filters (or projection operators), which improves the performance substantially in several cases (e.g., the anisotropic case). In particular, the second-order anisotropic problems and problems on locally refined grids can be solved more efficiently by using different types of filters, while the the biharmonic equation and interface operator can be solved efficiently by using different eigenvalue estimates. To the best of our knowledge, two of these extensions (i.e., the biharmonic and the domain decomposition applications) are novel. While a general theory is lacking at this point, we demonstrate numerically that these algorithms perform very well, at least for model problems.

The multilevel preconditioners mentioned above are similar in spirit to the classical multigrid method. They are designed to capture the mesh-dependent spectral property of a discretized elliptic operator. The variations in the coefficients are handled in most cases by the conjugate gradient method, which also makes the iteration more robust. In [13], we presented some experimental results comparing several multilevel preconditioners with a multigrid cycle as a preconditioner. However, further tests are needed to decide whether this new class of multilevel preconditioners offers practical advantages over the classical multigrid methods.

2. The concept of MF preconditioners. We shall motivate the construction of the MF preconditioner by first considering the following one-dimensional Poisson equation on $\Omega = [0, 1]$:

$$-\triangle u = f(x) \tag{1}$$

subject to zero Dirichlet boundary conditions. A standard second-order discretization of the above equation on a uniform grid with grid size $h = 1/(n+1)$ gives rise to a linear system of equations denoted by $Au = f$, where A, u, and f correspond to the discrete Laplacian, the solution and the forcing functions, respectively, and A is a tridiagonal matrix with diagonal elements $-1/h^2$, $2/h^2$, $-1/h^2$. It is well known that the matrix A can be diagonalized as

$$A = W \Lambda W^T$$

where W is an orthogonal matrix with elements

$$(W)_{ij} = 2\sqrt{h}\sin ij\pi h,$$

and

$$\Lambda = diag(\lambda_k), \qquad \lambda_k = \frac{4}{h^2}\sin^2\frac{k\pi h}{2}.$$

The main idea of the MF preconditioning is to approximate this eigendecomposition of A. First, the eigenfunctions of A are grouped into subsets corresponding to different frequency bands. In matrix form, for $n = 2^L - 1$, we partition W into L bands so that

$$W = [W_1, W_2, \cdots, W_L].$$

where

$$W_l = [w_{2^{l-1}}, \cdots, w_{2^l-1}],$$

with w_j being the jth column of the matrix W. Thus, for example, W_1 and W_L correspond to the lowest and highest frequency bands, respectively.

Using the notations introduced above, we can rewrite

$$A = \sum_{l=1}^{L} W_l \Lambda_l W_l^T$$

where

$$\Lambda = diag(\Lambda_l), \Lambda_i = diag(\lambda_{2^{l-1}}, \cdots, \lambda_{2^l-1}).$$

The first approximation comes in when we replace all the eigenvalues (Λ_l) within each band by a constant c_l. Thus, we have a preconditioner $\hat{M}$ such that

$$\hat{M}^{-1}v = \sum_{l=1}^{L} \frac{B_l v}{c_l}$$

where

$$B_l = W_l W_l^T.$$

Note that we have the following property for B_l :

$$B_l v = \begin{cases} v & \text{if } v \in range\{W_l\} \\ 0 & \text{if } v \in range\{W_l\}^{\perp}. \end{cases}$$

Hence, B_l can be considered as an ideal spatial bandpass filter. Thus, applying the preconditioner $\hat{M}$ to a vector (i.e., $M^{-1}v$) consists of three phases : projection of v into the subspace corresponding to each band (operator B_l), scaling by the corresponding approximate eigenvalues c_l, and synthesizing the scaled components (summation).

Since the implementation of ideal filters is computationally expensive, requiring many global operations (e.g., sine transforms), we seek the approximation of ideal filters with nonideal ones that are computationally more efficient. For the construction of efficient nonideal filters, we borrow ideas from standard digital filtering theory [13]. Typically, a bandpass filter is constructed by taking the difference of two lowpass filters, one that filters out all frequencies higher than the highest ones in the band and the other one that lets through all frequencies lower than all frequencies in the band. In turn, the lowpass filters can be approximated by cascading a sequence of elementary filters H_l's, which are simple averaging operators over a small fixed number of grid points separated by spacing proportional to the wavelength of the band W_l.

Mathematically, the effect of using nonideal filters can be summarized by replacing B_l with approximations $\tilde{B}_l$ in the definition of $\hat{M}$ to get our final preconditioner M :

$$M^{-1}v = \sum_{l=1}^{L} \frac{\tilde{B}_l v}{c_l}.$$

In the rest of the paper, we use the following two filters :

- The first order filter defined by

$$(H_{l,1})_j = \frac{1}{4}(v_{j-2^{L-l}} + 2v_j + v_{j+2^{L-l}})$$

where $(\cdot)_j$ denotes the jth element of the argument, and v is extended periodically by

$$v_{-j} = -v_j, \quad \text{and} \quad v_{n+j} = -v_{n+2-j}.$$

- The filter $H_{l,2}$ obtained by applying $H_{l,1}$ twice :

$$(H_{l,2})_j = \frac{1}{16}(v_{j-2^{L-l+1}} + 4v_{j-2^{L-l}} + 6v_j + 4v_{j+2^{L-l}} + v_{j+2^{L-l+1}}).$$

We call the method introduced above the single grid multilevel filtering (SGMF) preconditioner, which involves computation on the same number of grid points n at all levels (corresponding to the frequency bands). Since there are $L = \log_2(n+1)$ levels and $O(n)$ operations are required per level, the total number of operations required per iteration is thus $O(nL)$.

To further improve the efficiency, we introduce a multigrid version of our preconditioner, which we called the multigrid multilevel filtering (MGMF) preconditioner. This is motivated by the fact that waveforms consisting only of low wavenumber components can be well represented on coarser grids. To incorporate the multigrid structure, the operators I_{l+1}^{l} and I_{l-1}^{l}, which are the down-sampling and up-sampling operators, respectively, are introduced. Note that in the multigrid literatures these operators are commonly known as restriction and interpolation operators. Using the concept of MGMF, we construct a sequence of grids Ω_l of sizes $h_l = O(2^{L-l}h), 1 \le l \le L$, to represent the decomposed components. With MGMF, the total number of operations per iteration is $O(n)$, a reduction by a factor of $\log_2 n$ compared to SGMF.

We allow variations in designing the filtering scheme. Several preconditioners, which will be used in the later sections, are defined specifically as follows:

MGMF1, the MGMF preconditioner with 9-point (27-point) filter for two-dimensional (three-dimensional) problems (i.e., $H_{l,1}$).

MGMF2, a modified version of MGMF in which the 9-point (27-point) filter is applied twice (i.e., $H_{l,2}$).

MGMF3, another modified version of MGMF in which the 9-point (27-point) filter is applied once at the finest grid level (to give a smaller amount of work compared to MGMF2) and twice at other grid levels (to achieve a convergence rate between MGMF1 and MGMF2 but close to MGMF2).

We summarize the MGMF1 preconditioning algorithm as follows:

```
Algorithm MGMF1 : input = $r$, output = $z = M^{-1}r$
Decomposition :
          $v_L := r$
          for $l = L-1, \cdots, 1$
                    $v_l := I_{l+1}^{l} H_{l+1,1} v_{l+1}$
          end for
Scaling :
          for $l = 1, \cdots, L$
                    $w_l := v_l \div c_l$
          end for
Synthesis :
          $z_1 := w_1$
          for $l = 2, \cdots, L$
                    $z_l := w_l + H_{l,1} I_{l-1}^{l} z_{l-1}$
          end for
          $z = z_L$
end MGMF1
```

As it stands, this definition of the preconditioner can be extended to higher dimensions, more general elliptic operators and finite element meshes, as long as we have appropriate definitions for the elementary filters H_l's, the restriction and interpolation operators I_{l+1}^{l} and I_{l}^{l+1}, and the c_l's. For example, filters for the high dimensional cases can be constructed from the tensor product of one-dimensional filters. Moreover, it is well known that the eigenvalues λ_k in the wavenumber band B_l behave like $O(h_l^{-2})$ for general second-order elliptic problems, where h_l denotes the grid spacing for level l [21]. Therefore, a general rule for selecting the scaling constant c_l at grid level l is $c_l = O(h_l^{-2})$. For quasiuniform meshes with a refinement factor of 2 (so that $h_l \approx 2h_{l+1}$), this leads to the recurrence relation $c_{l+1} = 4c_l$.

By appealing to the framework in [6], it is also possible to construct filters for quasi-uniform structured finite element meshes. This relationship was briefly discussed in [13]. Basically, the projection operator from the fine grid onto a coarser grid used in [6] can be interpreted as a low-pass filter on the fine grid. Therefore, on the one hand the elementary filters H_l's can be derived from the basis functions on the grid hierarchy. On the other hand, the projection operators in [6] can be adapted to special features of a particular problem, with the insight provided by the filtering framework (e.g., for anisotropic problems).

The MF preconditioner is designed to capture the mesh-dependent spectral property of a discretized elliptic operator, but not the variation of its coefficients. In order to take badly scaled variable coefficients into account, we use diagonal scaling [10]. Suppose that the coefficient matrix can be written as

$$A = D^{1/2} \tilde{A} D^{1/2}$$

where we choose D to be a diagonal matrix with positive elements in such a way that the diagonal elements of $\tilde{A}$ are of the same order. Then in order to solve $Au = f$, we can solve an equivalent problem $\tilde{A}\tilde{u} = \tilde{f}$, where $\tilde{u} = D^{1/2}u$ and $\tilde{f} = D^{-1/2}f$, with the MF preconditioner.

The SGMF preconditioner on uniform meshes can be easily analyzed exactly using Fourier analysis, and the predictions agree quite well with experimental results [13], [23]. However, the Fourier analysis is only meant to be used as a tool for deriving and gaining insights into the algorithms and cannnot be extended as a basis for a general convergence theory. While the Fourier analysis is rigorously applicable only for model problems, the derived algorithms are applicable in a more general setting. Fourier analysis does provide precise convergence rate estimates and eigenvalue distributions, which supplements the more general theory. For this reason, it has been used by many authors in studying iterative methods [27], [11], [26]. The MGMF preconditioner on uniform and quasi-uniform grids can be analyzed using the same finite element analysis framework used in [6], although we will not pursue that in this paper.

On a uniform mesh there is an obvious connection between our multilevel filtering idea and wavelets [20], [12]. Wavelets are orthonormal basis functions for square-integrable functions and are defined on a multilevel structure. These basis functions have compact support in space and almost compact support in the Fourier domain. Thus, wavelets can be considered as efficient bandpass filters. We are currently exploring the use of wavelets in our multilevel filtering preconditioner framework.

3. MF preconditioners for anisotropic problems. In this section, we extend the concept of multilevel filtering to the second-order anisotropic problems. To achieve a high degree of efficiency, the preconditioning step requires some modifications in the design of filters (or the use of a different multilevel nodal basis). We first provide justification for such modifications and then show the condition number as computed by Fourier analysis. Numerical experiments are also included.

Consider the following two-dimensional second-order anisotropic problem:

$$-\alpha u_{xx} - u_{yy} = f(x,y) \text{ in } \Omega = [0,1]^2, \tag{2}$$

where $\alpha > 1$, with zero Dirichlet boundary conditions. The discretization of the equation using a uniform square mesh with $h = 1/(n+1)$ gives a block-tridiagonal matrix A with an equation of the form $Au = f$. In the Fourier domain, we can express this as

$$\hat{A}(j,k)\hat{u}_{j,k} = \hat{f}_{j,k}, \qquad j,k = 1,2,\cdots,n-1 \tag{3}$$

where

$$\hat{u}_{j,k} = \frac{\sqrt{2}}{n}\sum_{l=1}^{n}\sum_{m=1}^{n} u_{l,m}\sin(j\pi lh)\sin(k\pi mh) \tag{4}$$

and

$$\hat{f}_{j,k} = \frac{\sqrt{2}}{n}\sum_{l=1}^{n}\sum_{m=1}^{n} f_{l,m}\sin(j\pi lh)\sin(k\pi mh) \tag{5}$$

such that

$$\hat{A}(j,k) = (2+2\alpha) - 2(\alpha\cos j\pi h + \cos k\pi h). \tag{6}$$

We can observe from the eigenvalue spectrum of $\hat{A}$ that for $\alpha \gg 1$ the variation in magnitudes of the eigenvalues in the k-direction is relatively small compared to that in the j-direction. To maintain uniform variation of eigenvalues within each band, we divide more wavenumber bands in the j-direction than in the k-direction. We call this technique *directionally adaptive filtering.* This can be done in practice by first performing one-dimensional filtering in the j-direction for a number of levels (say, the number of levels $= \gamma$), then resuming two dimensional filtering. This is in contrast to performing two-dimensional filtering for all the levels for the nearly isotropic problems described in the last section. Here γ depends on α as well as the problem to be solved. For second-order elliptic problems with quasi-uniform grid and $h_l \approx 2h_{l+1}$, it is sufficient to use $\gamma = round(\log_4 \alpha)$. Suppose $\alpha = 4$. Then $\gamma = 1$ and the modified $H_{l,1}$ for the finest grid level takes the following stencil form :

$$\frac{1}{4} \left| \begin{array}{ccc} 1 & 2 & 1 \end{array} \right|$$

while the filters for the other coarse grid levels have a two-dimensional stencil (tensor product of one-dimensional filter, i.e., $H_{l,1} \times H_{l,1}$).

Note that if the finest level is defined on a $(n+2) \times (n+2)$ grid, then for $\gamma \geq 1$ the next coarse level is defined on a $((n+1)/2+1) \times (n+2)$ grid instead of a $((n+1)/2+1) \times ((n+1)/2+1)$ grid for $\gamma = 0$. It should also be noted that this modified filtering scheme is analogous to the idea of semi coarsening in the multigrid literature.

We performed Fourier analysis of the single grid version of this scheme (called SGMF1a) on the two-dimensional anisotropic problem with different α and h. The condition numbers of the preconditioned system are given in Table 1. For comparison purposes, the condition numbers of the preconditioned system using the unmodified SGMF1 preconditioner are also included. Table 1 shows that this modified scheme is quite effective. For example, for $\alpha = 1000$ the condition number grows slowly with n, while this is not true for the unmodified SGMF1 preconditioner.

TABLE 1
Condition number for different α and n.

	$\alpha = 10$			$\alpha = 100$			$\alpha = 1000$		
n	A	SGMF1a	SGMF1	A	SGMF1a	SGMF1	A	SGMF1a	SGMF1
7	25	3.8	13	25	3.8	38	25	3.8	47
15	103	4.3	21	103	4.7	117	103	4.7	216
31	414	5.4	28	414	5.8	233	414	5.9	849
63	1659	6.6	34	1659	6.8	328	1659	6.9	2142
127	6639	8.2	40	6639	7.9	395	6639	8.0	3480
255	26560	9.7	46	26560	9.0	454	26560	9.0	4396

The MGMF1 preconditioning algorithm for the above anisotropic problems can be summarized as follows:

```
**Algorithm** MGMF1a : input = $r$, output = $z = M^{-1}r$
    $v_L := r$
Decomposition :
        $count = \gamma$
        **for** $l = L-1, \cdots, 1$
            **if** ($count = 0$) then
                $t_l :=$ x-filter1($v_{l+1}$)
                $v_l :=$ y-filter1($t_l$)
            **else**
                $count = count - 1$
                $v_l :=$ x-filter1($v_{l+1}$)
            **end if**
        **end for**
Scaling :
        **for** $l = 1, \cdots, L$
            $v_l := v_l \div c_l$
        **end for**
Synthesis :
        $t_1 := v_1$
        **for** $l = 2, \cdots, L$
            $t_l := v_l + H_{l,1} I^l_{l-1} t_{l-1}$
        **end for**
        $z := t_L$
**end** MGMF1a
```

Next we show numerical results using the multigrid MF (MGMF1a) preconditioner in conjunction with the conjugate gradient method. Again, we use the standard 5-point discretization on a uniform square mesh with $h = 1/(n+1)$ and the forcing function $f(x,y)$ is such that the solution is $u = x(x-1)y(y-1)e^{xy}$. The stopping criterion used is $\| r^k \| / \| r^0 \| \leq 10^{-5}$ and the initial guess is 0. The iteration counts for different h and α are shown in Table 2.

TABLE 2
Iteration counts for different α and n

	$\alpha = 10$			$\alpha = 100$			$\alpha = 1000$		
n	A	MGMF1a	MGMF1	A	MGMF1a	MGMF1	A	MGMF1a	MGMF1
7	23	11	18	17	7	19	13	6	20
15	48	13	26	41	10	41	27	9	44
31	97	15	32	90	12	64	63	12	84
63	197	16	36	187	13	83	126	13	140
127	405	19	41	388	15	95	258	15	193
255	839	20	45	812	17	106	608	17	224

The numerical results show that this scheme works very well for a wide range of α. A similar scheme can be applied to the case when $\alpha < 1$. It should be noted that the algorithm can also be applied to more general anisotropic problems (e.g., variable coefficients) in the same way that the semicoarsening technique in MG is used (e.g., by averaging coefficients) [18].

4. MF preconditioners for the biharmonic equation. Consider the following biharmonic equation in two dimensions:

$$-\Delta^2 u = f \quad \text{in } \Omega = [0,1]^2 \tag{7}$$

with first boundary conditions:

$$u(x,y)|_\Gamma = g(x,y) \tag{8}$$

and

$$\frac{\partial u}{\partial n} = b(x,y). \tag{9}$$

We discretize this equation using a 13-point second-order centered finite difference approximation with $h = 1/(n+1)$:

$$\begin{aligned} 20u_{i,j} &- 8(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}) \\ &+ 2(u_{i+1,j+1} + u_{i-1,j+1} + u_{i+1,j-1} + u_{i-1,j-1}) \\ &+ u_{i+2,j} + u_{i-2,j} + u_{i,j+2} + u_{i,j-2} = h^4 f_i, j \end{aligned}$$

for $i, j = 2, n-1$. The difference equation for $i = 1$, and $j = 3, \cdots, n-2$ is:

$$\begin{aligned} 21u_{1,j} &- 8(u_{2,j} + u_{1,j+1} + u_{1,j-1}) + 2(u_{2,j+1} + u_{2,j-1}) + u_{3,j} + u_{1,j+2} + u_{1,j-2} \\ &= h^4(f_i, j + 8g_{0,j} - 2(g_{0,j+1} + g_{0,j-1}) - 2hb_{0,j}) \end{aligned}$$

since

$$\frac{\partial u}{\partial n} = -\frac{\partial u}{\partial x} \quad \text{on } x = 0.$$

Using central differencing, we get

$$-\frac{(u_{1,j} - u_{-1,j})}{2h} = b_{0,j}.$$

Also, at $i = j = 1$, we have

$$\begin{aligned} 22u_{1,1} &- 8(u_{2,1} + u_{1,2}) + 2(u_{2,2}) + u_{3,j} + u_{1,3} \\ &= h^4(f_i, j + 8(g_{0,j} + g_{1,0}) - 2(g_{0,j+1} + g_{0,j-1} + g_{2,0}) - 2h(b_{0,1} + b_{1,0})). \end{aligned}$$

The difference equations for other near boundary grid points can be derived similarly.

To derive MF preconditioners, we have to estimate the eigenvalues of the biharmonic operator. To do so, we apply Fourier analysis to the operator with modified boundary conditions, namely,

$$u(x,y)|_\Gamma = 0 \quad \text{and} \quad \frac{\partial^2 u}{\partial n^2} = 0.$$

Based on analyzing this problem, we can estimate the eigenvalues of $\hat{A}$ by

$$\hat{A}(j,k) \approx 4 - 2(\cos(i\pi h) + \cos(j\pi h))^2 \tag{10}$$

which is the square of that for the Poisson equation.

TABLE 3
Condition number for SGMF *preconditioning for the biharmonic equation.*

n	No preconditioning	SGMF1b	SGMF2b	SGMF3b
7	690	25	5.3	17
15	1.1×10^4	108	5.6	66
31	1.7×10^5	438	7.2	256
63	2.8×10^6	1814	8.7	1017
127	4.4×10^7	7367	10.2	4061
255	7.0×10^8	29705	11.7	16238

Since the eigenvalues in B_l for this equation behave like $O(h_l^{-4})$, a natural extension of the MF preconditioner involves changing the scaling recurrence $c_{l+1} = 4c_l$ to $c_{l+1} = 16c_l$ (again, $h_l \approx 2h_{l+1}$ is assumed). In Table 3, we show the result of the Fourier analysis on the MF-preconditioned biharmonic equation. In the table, SGMF1b, SGMF2b, and SGMF3b represent the original SGMF1, SGMF2, and SGMF3 preconditioners with the new scaling.

We see that the condition number of A grows about 16 times with each halving of h. The use of SGMF1b has effectively helped to reduce the condition number. Nevertheless, SGMF2b helps to reduce the condition number even more dramatically.

To verify the Fourier results, we implement the SGMF1b, SGMF2b and SGMF3b preconditioners for the biharmonic equation where the $f(x,y)$, $g(x,y)$ and $b(x,y)$ are such that the solution is $u = x(x-1)y(y-1)\sin(\pi x)\sin(\pi y)$. The stopping criterion is $\| r^k \| / \| r^0 \| \leq 10^{-6}$ and the initial guess is zero. The iteration counts are shown in Table 4.

TABLE 4
Iteration counts for SGMF*-preconditioned* PCG *for the biharmonic equation.*

n	No preconditioning	SGMF1b	SGMF2b	SGMF3b
7	10	9	10	9
15	42	17	12	16
31	160	36	14	30
63	586	82	17	57
127	2218	177	23	113
255	8587	366	33	220

Next we show (in Table 5) the iteration counts when the multigrid formulation of SGMF1b, SGMF2b, and SGMF3b (i.e., MGMF1b, MGMF2b, and MGMF3b) are applied to the same problem.

TABLE 5
Iteration counts for MGMF*-preconditioned* PCG *for the biharmonic equation.*

n	No preconditioning	MGMF1b	MGMF2b	MGMF3b
7	10	10	10	10
15	42	27	22	24
31	160	40	29	32
63	586	56	30	37
127	2218	80	35	40
255	8587	120	43	48

We observe a close correlation between the numerical and Fourier results for the SGMF preconditioners. Indeed, SGMF2b improves significantly over SGMF1b with only a little increase in cost per iteration. SGMF3b improves somewhat over SGMF1b but is still not good enough compared to SGMF2b. Therefore, SGMF2b requires the fewest operation counts out of the three. Looking into the numerical results for the MGMF preconditioners, we first observe that both MGMF1b and MGMF3b give better convergence rates than their SGMF counterparts. We cannot explain why this is the case, nor can we explain why MGMF3b performs much better than predicted by the corresponding Fourier results. Finally, with a little arithmetic, it is not difficult to show that MGMF3b gives the fewest overall operation counts.

5. MF preconditioners for problems with locally refined grids. In this section, we shall consider the application of the MF preconditioners to second-order elliptic problems with local mesh refinement. Such mesh refinements are necessary for accurate modeling of problems with various types of singular behavior. We consider the discretization scheme for locally mesh refined grids by McCormick and Thomas [16]. This discretization scheme was motivated by the desire to preserve the highly regular grid structure (to maintain efficiency on parallel computer architectures), as well as to satisfy the need for local resolution in many physical models. For example, the mesh in Fig. 1 would be effective if the forcing function $f(x,y)$ behaves like a δ function distribution at the points $(1,1)$ and (n,n) (both lower left and upper right corners).

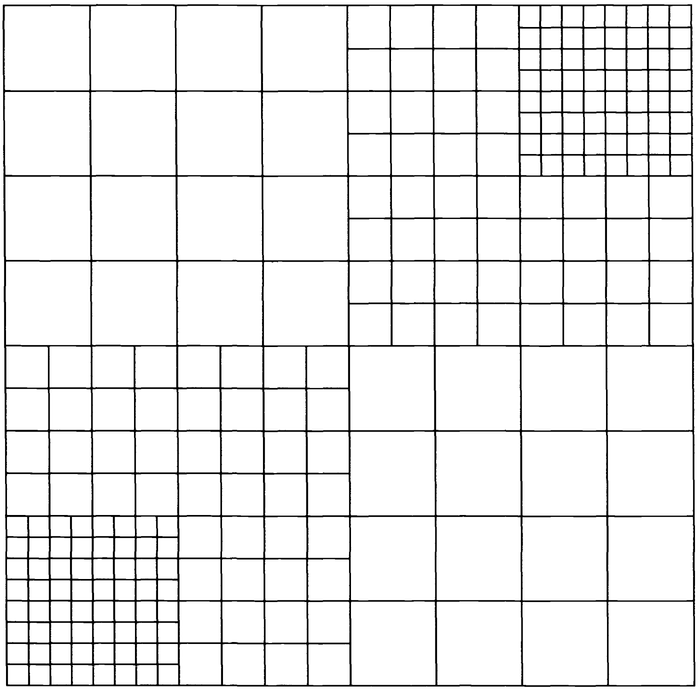


FIG. 1. *Locally refined grids—an example.*

The Fourier analysis cannot be applied here because of the presence of nonuniform grids. However, as was shown in our previous paper [13], the parallel multilevel preconditioner proposed by Bramble, Pasciak, and Xu [6] can be considered as a special case of MF preconditioners with appropriately chosen filters. We can borrow the finite element analysis result from them and we would expect the MGMF preconditioners to be effective also for meshes with local refinement. Below we show the MGMF algorithm for this problem. Here $\hat{I}_i^j$ and $\hat{H}_l$ are restriction (or interpolation) and elementary filtering operators restricted to the locally refined grids only. Moreover, we can use the same recurrence relation $c_l = 4c_{l+1}$ and we have the following algorithm:

```
**Algorithm** MGMF1c : input = $r$, output = $z = M^{-1}r$
Decomposition :
        $v_L := r$
        (* filtering at refined levels *)
        **for** $l = L - 1, \cdots, J - k$
                $v_l := \hat{I}_{l+1}^l \hat{H}_{l+1,1} v_{l+1}$
        **end for**
        (* filtering on uniform grid levels *)
        **for** $l = L - k - 1, \cdots, 1$
                $v_l := I_{l+1}^l H_{l+1,1} v_{l+1}$
        **end for**
Scaling :
        **for** $l = 1, \cdots, L$
                $v_l := v_l \div c_l$
        **end for**
Synthesis :
        $z_1 := v_1$
        **for** $l = 2, \cdots, L - k$
                $z_l := v_l + H_{l,1} I_{l-1}^l z_{l-1}$
        **end for**
        **for** $l = L - k + 1, \cdots, L$
                $z_l := v_l + \hat{H}_{l,1} \hat{I}_{l-1}^l z_{l-1}$
        **end for**
        $z = z_L$
**end** MGMF1c
```

We solve a Poisson equation on the grid

- shown in Fig. 1 but with refinement only at the upper right corner and the forcing function is $f(x,y) = 2^{-l}\delta(1-h, 1-h)$, and
- shown in Fig. 1 and the forcing function $f(x,y) = 2^{-l}(\delta(h,h)+\delta(1-h,1-h))$, where l is the number of level of refinements used and h is the grid size for the nonrefined grid.

We use the discretization scheme for the domain and the interfaces proposed by McCormick and Thomas [16] for aligned grids. The stopping criterion and initial guess are the same as before. The iteration counts for different number of levels and different h are given in Tables 6 and 7. The iteration counts for the unpreconditioned conjugate-gradient (CG) method and the parallel multilevel preconditioner (BPX) [6] are also included for comparison purposes.

The tables show the effectiveness of the MF preconditioner compared to the unpreconditioned CG method and the PCG method with parallel multilevel preconditioner (BPX). The convergence rates seem to be quite insensitive to the number of refinement levels used.

TABLE 6
Iteration counts for the Poisson equation with refinements at upper right corner only.

n	No. of levels	CG	MGMF1c	BPX
15	0	26	9	12
15	1	37	10	14
15	2	45	11	16
15	3	53	12	17
31	0	48	9	13
31	1	70	10	15
31	2	88	11	17
31	3	109	12	18
63	0	84	10	14
63	1	126	11	15
63	2	166	11	17
63	3	210	12	19
127	0	133	10	14
127	1	219	11	15
127	2	309	12	17
127	3	395	13	19

TABLE 7
Iteration counts for the Poisson equation with refinements at both corners.

n	No. of levels	CG	MGMF1c	BPX
15	0	26	9	12
15	1	54	11	15
15	2	63	12	17
15	3	75	16	18
31	0	48	9	13
31	1	86	11	16
31	2	117	13	17
31	3	140	13	19
63	0	84	10	14
63	1	126	12	16
63	2	190	12	18
63	3	235	14	19
127	0	133	10	14
127	1	204	12	16
127	2	297	13	18
127	3	391	14	20

6. MF preconditioners for Schur complement systems. Consider solving a two-dimensional second-order elliptic problem on a domain divided into two subdomains by an interface. If we use a 5-point discretization and order unknowns in the subdomains Ω_1 and Ω_2 first followed by those on the interface Γ_3, we obtain the following linear system:

$$Au = \begin{bmatrix} A_1 & 0 & A_{13} \\ 0 & A_2 & A_{23} \\ A_{31} & A_{32} & A_3 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix}.$$

By applying block Gaussian elimination to eliminate the unknowns u_1 and u_2, we obtain the following system for the interface unknowns u_3:

$$Su_3 = \tilde{f}_3$$

where

$$S = A_3 - A_{31}A_1^{-1}A_{13} - A_{32}A_2^{-1}A_{23}$$

and

$$\tilde{f}_3 = f_3 - A_{31}A_1^{-1}f_1 - A_{32}A_2^{-1}f_2.$$

A standard approach in domain decomposition methods is to solve the Schur complement system $Su_3 = \tilde{f}_3$ with the preconditioned conjugate gradient method. Many preconditioners have been proposed in the literature [7]. A typical one is Dryja's preconditioner [9], which is defined to be the square root of the negative one-dimensional Laplacian and which can be inverted by the use of FFTs in $O(n \log n)$ time, where n is the number of unknowns on the interface. Recently, Smith and Widlund [19] proposed a hierarchical basis preconditioner for S which is cheaper than Dryja's preconditioner, requiring only $O(n)$ work per iteration. Here we propose to use the MF preconditioner for S. To do this, we can retain the multilevel filtering framework and we only need to modify the scaling constants c_l's. We know that the eigenvalues for the Schur complement in the frequency band B_l behaves like $O(h_l^{-1})$ [9]. Therefore, it is sufficient to use the recurrence $c_{l+1} = 2c_l$. In Table 8 we compare the number of iterations to obtain convergence for different n for the Poisson equation on a rectangular $2n \times n$ grid decomposed into two equal subdomains.

TABLE 8
Iteration count versus n.

n	No precond.	Dryja	MGMF1	MGMF2	HB
7	4	4	4	4	4
15	8	6	7	6	7
31	16	6	9	7	8
63	27	6	9	7	10
127	39	6	9	7	12

We observe that MGMF2 performs better than MGMF1 and the hierarchical basis (HB) preconditioner. All but the HB preconditioner seem to show convergence rates independent of n. Although we cannot prove spectral equivalence for the MGMF preconditioners, an $O(\log n)$ upper bound for the condition number for the MGMF1 preconditioned system can be proved and details of such a proof can be found in [24]. We also observe that the MGMF2 preconditioner performs almost as well as Dryja's preconditioner. The MGMF appears to offer convergence rates comparable to Dryja's preconditioner and at the same time is relatively easy to use and costs about the same as the HB preconditioner.

7. Conclusion. In our previous paper [13] and the first part of the present paper we show the competitiveness of the MF preconditioners compared with some other preconditioners such as the hierarchical basis preconditioner, multigrid preconditioner and others. In this paper we have further demonstrated the ease with which we can extend the MF preconditioners to effectively solve other more general elliptic problems. The flexibility of filter and scaling block design offers different ways of achieving a high degree of efficiency for these problems.

REFERENCES

[1] O. Axelsson, *An algebraic framework for multilevel methods,* Report 8820, Department of Mathematics, Catholic University, Toernooiveld, 6525 ED Nijmegen, the Netherlands, 1988.

[2] O. Axelsson and P. Vassilevski, *Algebraic multilevel preconditioning methods*, I, Report 8811, Department of Mathematics, Catholic University, Toernooiveld, 6525 ED Nijmegen, the Netherlands, 1988.

[3] ———, *Algebraic multilevel preconditioning methods*, II, Report 1988-15, Institute for Scientific Computation, University of Wyoming, Laramie, WY, 1988.

[4] R. E. Bank, T. F. Dupont, and H. Yserentant, *The hierarchical basis multigrid method,* Numer. Math., 52 (1988), pp. 427-458.

[5] G. Birkhoff and R. E. Lynch, *Numerical Solutions for Elliptic Problems*, Society for Industrial and Applied Mathematics, Philadelphia, 1984.

[6] J. H. Bramble, J. E. Pasciak, and J. Xu, *Parallel multilevel preconditioners*, Math. Comp., 55 (1990), pp. 1–22.

[7] T. F. Chan, R. Glowinski, J. Periaux, O. B. Widlund, eds., *Domain Decomposition Methods for Partial Differential Equations,* Society for Industrial and Applied Mathematics, Philadelphia, 1989.

[8] T. F. Chan, Jay C. C. Kuo, and C. H. Tong, *Parallel elliptic preconditioners: Fourier analysis and performance on the Connection Machine,* Comput. Phys. Comm., 53 (1989), pp. 237–252.

[9] M. Dryja, *A capacitance matrix method for Dirichlet problem on polygon region*, Numer. Math., 39 (1982), pp. 51–54.

[10] A. Greenbaum, C. Li, and H. Z. Chao, *Parallelizing preconditioned conjugate gradient algorithms*, Comput. Phys. Comm., 53 (1989), pp. 295–309.

[11] W. Hackbusch, *The frequency decomposition multi-grid method*, Numer. Math., 56 (1989), pp. 229-245.

[12] V. E. Henson and W. L. Briggs, *Wavelets: What are they and what do they have to do with Multigrid?*, Copper Mountain Conference on Iterative Methods, April 1-5, Copper Mountain, CO, 1990.

[13] C.-C. J. Kuo, T. F. Chan, and C. H. Tong, *Multilevel filtering elliptic preconditioners*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 403–429.

[14] Y. A. Kuznetsov, *Multigrid domain decomposition methods for elliptic problems*, Proceedings VIII International Conference on Computational Methods for Applied Science and Eng. Vol. 2, Versailles, France, 1987, pp. 605–616.

[15] R. E. Lynch and J. R. Rice, *A high-order difference method for differential equations*, Math. Comp., 34 (1980), pp. 333–372.

[16] S. McCormick and J. Thomas, *The fast adaptive composite grid (FAC) method for elliptic equations*, Math. Comp., 46 (1986), pp. 439–456.

[17] S. McCormick, *Multilevel Adaptive Methods for Partial Differential Equations,* Society for Industrial and Applied Mathematics, Philadelphia, 1989.

[18] W. A. Mulder, *Multigrid Alignment and Euler's Equation,* in Proc. Fourth Copper Mountain Conference on Multigrid Methods, Society for Industrial and Applied Mathematics, Philadelphia, 1989, pp. 348–364.

[19] B. Smith and O. Widlund, *A domain decomposition algorithm using a hierarchical basis*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 1212–1220.

[20] G. Strang, *Wavelets and dilation equations : A brief introduction*, SIAM Rev., 31 (1989), pp. 614–627.

[21] G. Strang and G. J. Fix, *An Analysis of the Finite Element Method*, Prentice-Hall Ser. Automat. Comput., 1973.

[22] Charles H. Tong, *The preconditioned conjugate gradient method on the connection machine*, Internat. J. High Speed Comput., 2nd Issue: Scientific Applications of the Connection Machine, 1989.

[23] C. Tong, T. F. Chan, and C. C. J. Kuo, *Multilevel filtering preconditioners—extension to more general elliptic problems*, UCLA CAM Report 90-12, University of California, Los Angeles, CA, 1990

[24] ———, *A domain decomposition preconditioner based on a change to the multilevel nodal basis*, UCLA CAM Report 90-20, University of California, Los Angeles, CA, 1990.

[25] P. Vassilevski, *Iterative methods for solving finite element equations based on multilevel splitting of the matrix*, Bulgarian Academy of Science, Sofia, Bulgaria, 1987, preprint.

[26] P. Wesseling, *A survey of Fourier smoothing analysis results,* Third European Conference on Multigrid Methods, Oct. 1-3, 1990.

[27] D. M. Young and B. R. Vona, *Parallel multilevel methods*, CNA-243, University of Texas, Austin, TX, May 1990.

[28] H. Yserentant, *On the multi-level splitting of finite element spaces,* Numer. Math., 49 (1986), pp. 379–412.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 1, pp. 243–258, January 1992

DOMAIN DECOMPOSITION ALGORITHMS FOR INDEFINITE ELLIPTIC PROBLEMS*

XIAO-CHUAN CAI† AND OLOF B. WIDLUND‡

Abstract. Iterative methods for linear systems of algebraic equations arising from the finite element discretization of nonsymmetric and indefinite elliptic problems are considered. Methods previously known to work well for positive definite, symmetric problems are extended to certain nonsymmetric problems, which can also have some eigenvalues in the left half plane.

This paper presents an additive Schwarz method applied to linear, second order, symmetric or nonsymmetric, indefinite elliptic boundary value problems in two and three dimensions. An alternative linear system, which has the same solution as the original problem, is derived and this system is then solved by using GMRES, an iterative method of conjugate gradient type. In each iteration step, a coarse mesh finite element problem and a number of local problems are solved on small, overlapping subregions into which the original region is subdivided. The rate of convergence is shown to be independent of the number of degrees of freedom and the number of local problems if the coarse mesh is fine enough. The performance of the method in two dimensions is illustrated by results of several numerical experiments.

Two other iterative methods for solving the same class of elliptic problems in two dimensions is also considered. Using an observation of Dryja and Widlund, it is shown that the rate of convergence of certain iterative substructuring methods deteriorates only quite slowly when the local problems increase in size. A similar result is established for Yserentant's hierarchical basis method.

Key words. Schwarz's alternating method, domain decomposition, nonsymmetric and indefinite, elliptic equations, finite elements

AMS(MOS) subject classifications. 65N30, 65F10

1. Introduction.

Domain decomposition techniques are powerful iterative methods for solving linear systems of equations that arise from finite element problems. In each iteration step, a coarse mesh finite element problem and a number of smaller linear systems, which correspond to the restriction of the original problem to subregions, are solved instead of the large original system. These algorithms can be regarded as divide-and-conquer methods. The number of subproblems can be large and these methods are therefore promising for parallel computation. The central mathematical question is to obtain estimates on the rate of convergence of the iteration by deriving bounds on the spectrum of the iteration operator. We are able to establish quite satisfactory bounds if the coarse mesh is fine enough.

We work with two triangulations of the region: (1) partitioning the region into subregions, also called substructures, which define a coarse, global model; (2) partitioning the region into elements of a finite element model. As in the positive definite case considered previously (see Cai [3], [4]; Dryja [6]; and Dryja and Widlund [7], [8]), the coarse problem provides interchange of information among the different parts of the region. It is known that without such a coarse subproblem the rate of convergence is considerably slower, cf. [24]. This part of the approximate solver plays an additional role in the indefinite case. We can interpret the main results of this paper by saying that if the eigenfunctions corresponding to the eigenvalues in the left half

* Received by the editors April 5, 1990; accepted for publication March 7, 1991.

† Department of Mathematics, University of Kentucky, Lexington, Kentucky 40506. The work of this author was supported in part by National Science Foundation grants NSF-DCR-8521451 and NSF-ECS-8957475, while in residence at Yale University.

‡ Courant Institute of Mathematical Science, New York University, New York, New York 10012. The work of this author was supported in part by National Science Foundation grant NSF-CCR-8903003 and in part by the U.S. Department of Energy contract DE-AC02-76ER03077-V at the Courant Mathematics and Computing Laboratory.

plane are approximated well enough on the coarse mesh, then the spectrum of the preconditioned linear system of equations lies in a fixed bounded subset of the right half plane. This is important for the rate of convergence of the iterative method. The least favorable situation for iterative methods of conjugate gradient type is the case where the origin of the complex plane is surrounded by eigenvalues of the iteration operator. Here we are able to avoid such a situation.

The additive Schwarz algorithms, introduced in [7], cf. also [6], [8], [9], [18], provide a means of constructing preconditioners for many problems in terms of a partition of a given finite element space into a sum of subspaces. The use of such a preconditioner involves solving, exactly or approximately, the restriction of the original problem to the different subspaces. The residual, which plays a central role in the iteration, is computed as a sum of terms from the different subspaces. These terms can be computed in parallel. We note that it has been shown in Dryja and Widlund [9] that many domain decomposition methods can be viewed as additive Schwarz methods. For recent work on the case of more than two levels of triangulation, see Dryja and Widlund [10] and Xu [25].

In the symmetric, positive definite case, the iterative method most commonly used to solve the transformed (preconditioned) equations is the conjugate gradient method. For the cases considered here symmetry is always lost. In our experiments, we have used a generalized conjugate residual method GMRES; see [22]. Since the spectrum of the operator is confined to the right half plane, Manteuffel's Chebyshev algorithm would also be successful, cf. [17]. Since we can show that the symmetric part of the operator is uniformly positive definite, with respect to a suitable inner product, and that the spectrum is uniformly bounded, we can guarantee a rate of convergence that is independent of the mesh size and the number of subregions.

Other methods for indefinite, elliptic problems are discussed in [2], [14], [16], [27], [28].

The paper is organized as follows. In §2, we introduce a class of indefinite, elliptic boundary value problem, the two triangulations of the domain, and a Galerkin finite element method. We briefly review the GMRES method in §3. In §4, we present two variants of the additive Schwarz method and a detailed analysis of their rates of convergence. Our analysis is based on previous work on the positive definite case, see [3], [4], [6], [7], [8], and a result due to Schatz [23]. Schatz's work, in turn, is based on Gårding's inequality and the Aubin–Nitsche trick, see Ciarlet [5] or Nitsche [21]. In §5, we discuss some numerical results. Finally, in §6, we show that, for problems in the plane, our result can be extended to iterative substructuring and hierarchical basis algorithms discussed in Dryja and Widlund [8], [9] and Yserentant [26], respectively.

2. The elliptic problems.

Let Ω be an open, bounded polygonal region in $R^d, d = 2$ or 3, with boundary $\partial\Omega$. Consider the homogeneous Dirichlet boundary value problem:

$$\left\{ \begin{array}{rcll} Lu & = & f & \text{in} \quad \Omega, \\ u & = & 0 & \text{on} \quad \partial\Omega. \end{array} \right. \tag{1}$$

The elliptic operator L has the form

$$Lu(x) = -\sum_{i,j=1}^{d} \frac{\partial}{\partial x_i}\left(a_{ij}(x)\frac{\partial u(x)}{\partial x_j}\right) + 2\sum_{i=1}^{d} b_i(x)\frac{\partial u(x)}{\partial x_i} + c(x)u(x).$$

All the coefficients are, by assumption, sufficiently smooth and the matrix $\{a_{ij}(x)\}$ is symmetric and uniformly positive definite for all $x \in \Omega$. The right-hand side $f \in L^2(\Omega)$. We also assume that the equation has a unique solution in $H_0^1(\Omega)$.

Let $(\cdot,\cdot)$ denote the usual L^2 inner product and $\|\cdot\|$ or $\|\cdot\|_{L^2}$ the corresponding norm. The weak form of (1) is: Find $u \in H_0^1(\Omega)$ such that

$$B(u,v) = (f,v), \qquad \forall v \in H_0^1(\Omega). \tag{2}$$

The bilinear form $B(u,v)$ is defined by

$$B(u,v) \;=\; \sum_{i,j=1}^{d} \int_{\Omega} a_{ij}\frac{\partial u}{\partial x_j}\frac{\partial v}{\partial x_i}dx + \sum_{i=1}^{d} 2\int_{\Omega} b_i \frac{\partial u}{\partial x_i} v dx + \int_{\Omega} cuvdx$$

or

$$B(u,v) \;=\; \sum_{i,j=1}^{d} \int_{\Omega} a_{ij}\frac{\partial u}{\partial x_j}\frac{\partial v}{\partial x_i}dx + \sum_{i=1}^{d} \int_{\Omega} b_i \frac{\partial u}{\partial x_i} v + \frac{\partial (b_i u)}{\partial x_i} v dx \;+ \int_{\Omega} \tilde{c}uvdx.$$

Here, $\tilde{c}(x) = c(x) - \sum_{i=1}^{d} \partial b_i(x)/\partial x_i$.

We also use two other bilinear forms

$$A(u,v) \;=\; \sum_{i,j=1}^{d} \int_{\Omega} a_{ij}\frac{\partial u}{\partial x_j}\frac{\partial v}{\partial x_i}dx$$

and

$$S(u,v) \;=\; \sum_{i=1}^{d} \int_{\Omega} b_i \frac{\partial u}{\partial x_i} v + \frac{\partial (b_i u)}{\partial x_i} v dx \;,$$

which correspond to the second-order terms and the skew-symmetric part of L, respectively. The bilinear form A defines a norm, which we denote by $\|\cdot\|_A$. Under the assumptions on the coefficients a_{ij}, this norm is equivalent to the H_0^1 norm. It is also easy to verify that

$$S(u,v) \;=\; -S(v,u), \;\; \forall u,v \in H_0^1(\Omega).$$

Throughout this paper, c and C, with or without subscripts, denote generic, strictly positive constants. They are independent of the mesh parameters h and H, which will be introduced later in this section.

Using elementary, standard tools, it is easy to establish the following inequalities:

(i) $\mid B(u,v) \mid \leq C\|u\|_A\|v\|_A, \;\; \forall u,v \in H_0^1(\Omega)$.

(ii) Gårding's inequality: There exists a constant C, such that

$$\|u\|_A^2 - C\|u\|_{L^2(\Omega)}^2 \leq B(u,u), \;\; \forall u \in H_0^1(\Omega) \;.$$

(iii) There exists a constant C, such that

$$\mid S(u,v) \mid \leq C\|u\|_A\|v\|_{L^2(\Omega)}, \;\; \forall u,v \in H_0^1(\Omega),$$

$$\mid S(u,v) \mid \leq C\|v\|_A\|u\|_{L^2(\Omega)}, \;\; \forall u,v \in H_0^1(\Omega).$$

We note that the bounds for $B(\cdot,\cdot)$ and $S(\cdot,\cdot)$ are different, since each of the terms in $S(\cdot,\cdot)$ contains a factor that is of zero order. This enables us to control the skew-symmetric term and makes our analysis possible.

We also use the following regularity result, cf. Grisvard [13] and Nečas [19].

(iv) The solution w of the adjoint equation

$$B(\phi, w) \;=\; (g, \phi), \;\; \forall \phi \in H_0^1(\Omega)$$

satisfies

$$\|w\|_{H^{1+\gamma}(\Omega)} \le C\|g\|_{L^2(\Omega)} \;,$$

where γ depends on the interior angles of $\partial\Omega$, is independent of g, and is at least 1/2.

We approximate (2) by a Galerkin conforming finite element method. For simplicity, we consider only continuous, piecewise linear, triangular elements in R^2 and tetrahedral elements in R^3.

To define the additive Schwarz algorithms, we need two levels of triangulation that have already been introduced in [3], [4], [6], [7], [8], and [9]. We first partition Ω into substructures $\{\Omega_i\}, i = 1, \cdots, N$, which provide a regular finite element triangulation of Ω. The Ω_i are nonoverlapping, d-dimensional simplices. They satisfy all the standard rules of finite elements, cf. Ciarlet [5]. This is the coarse mesh and it defines a mesh parameter $H \;=\; \max\{H_1, \cdots, H_N\}$. The triangulation is assumed to be shape regular, i.e., H_i, the diameter of Ω_i is bounded uniformly in terms of the diameter of the largest inscribed ball in Ω_i.

In a second step, we divide each substructure Ω_i into smaller simplices, denoted by $\{\; \tau_i^j, \; j = 1, \cdots \;\}$. They form a shape regular, fine mesh (h-level) finite element triangulation of Ω with the mesh parameter $h \;=\; \max_{i,j}\{h_i^j\}$. Here h_i^j is the diameter of τ_i^j.

We can now define the piecewise linear finite element spaces over the H-level and the h-level triangulations of Ω.

$$V^H = \{v^H \mid \text{ continuous on } \Omega,\; v^H|_{\Omega_i} \text{ linear },\; v^H = 0 \text{ on } \partial\Omega\}$$

and

$$V^h = \{v^h \mid \text{ continuous on } \Omega,\; v^h|_{\tau_i^j} \text{ linear },\; v^h = 0 \text{ on } \partial\Omega\}\;.$$

The Galerkin approximation of (2) is defined by: Find $u^h \in V^h$ such that

$$B(u^h, v^h) \;=\; (f, v^h), \qquad \forall v^h \in V^h. \tag{3}$$

If the mesh size h is small enough, it follows from a result by Schatz [23] that this problem has a unique solution. By using nodal basis functions to span the finite element space, (3) is transformed into a linear system of algebraic equations that is large, sparse, nonsymmetric, indefinite, and relatively ill-conditioned.

3. A brief discussion of the GMRES method.

Among the possible iterative methods to solve the linear system, we have only used one, the GMRES method, cf. Saad and Schultz [22] and Eisenstat, Elman, and Schultz [11]. This is a generalized minimum residual method, which in practice has proven quite powerful for a large class of nonsymmetric problems. The GMRES method is described in [22] and the theory developed in $L^2(\Omega)$ can be found in [11]. Both the algorithm and the theory can easily be extended to an arbitrary Hilbert space, see Cai [3]. In developing our theory and in the numerical results that are discussed in §5, we have exclusively used the A-norm introduced in §2. Here we briefly describe the GMRES algorithm and state a theorem without proof.

Let P be a linear operator in the finite dimensional space R^n with an inner product $[\cdot,\cdot]$, and a corresponding norm $\|\cdot\|$, chosen to take advantage of the special properties of P. (In our applications, P is the preconditioned stiffness matrix and the A-norm is used.) P is not symmetric but is positive definite with respect to $[\cdot,\cdot]$. The GMRES method is used to solve the linear system of equations

$$Px = b,$$

where $b \in R^n$ is given. We begin from an initial approximation $x_0 \in R^n$ and the initial residual $r_0 = b - Px_0$. In the mth iteration, a correction vector z_m is computed from the Krylov subspace

$$\mathcal{K}_m(r_0) = \text{span}\{r_0, Pr_0, \cdots, P^{m-1}r_0\},$$

which minimizes the norm of the residual. In other words, z_m solves

$$\min_{z \in \mathcal{K}_m(r_0)} \|b - P(x_0 + z)\| .$$

The mth iterate is $x_m = x_0 + z_m$.

The exact solution would be reached in no more than n iterations if we use exact arithmetic.

Following Eisenstat, Elman, and Schultz [11], the rate of convergence of the GMRES method can be characterized in terms of the minimal eigenvalue of the symmetric part of the operator and the norm of the operator. They are defined by

$$c_p = \inf_{x \neq 0} \frac{[x, Px]}{[x, x]} \quad \text{and} \quad C_p = \sup_{x \neq 0} \frac{\|Px\|}{\|x\|}.$$

By considering the decrease of the norm of the residual in a single step, the following theorem can be established.

THEOREM (Eisenstat, Elman, and Schultz). *If $c_p > 0$, then the* GMRES *method converges and after m steps, the norm of the residual is bounded by*

$$\|r_m\| \leq \left(1 - \frac{c_p^2}{C_p^2}\right)^{m/2} \|r_0\|.$$

4. Algorithms on overlapping subregions. In this section, we introduce two variants of an additive Schwarz algorithm and provide bounds on their convergence rates, see Theorem 1 in the following discussion. The analysis is valid for both two and three dimensions.

We first form a basic decomposition of the domain Ω into overlapping subregions and then introduce the projections that define our algorithms.

We use the H-level subdivision $\{\Omega_i\}$ of Ω. Each subregion Ω_i is extended to a larger region Ω_i', i.e., $\Omega_i \subset \Omega_i'$. The overlap is generous in the sense that there exists a constant $\alpha > 0$, such that

$$\text{distance}(\partial\Omega_i' \cap \Omega, \partial\Omega_i \cap \Omega) \geq \alpha H_i, \quad \forall i.$$

We assume that $\partial\Omega_i^{'}$ does not cut through any h-level elements. We use the same construction for the subregions that intersect the boundary $\partial\Omega$ except that we cut off the part that is outside Ω.

We also use the notation $\Omega_0^{'} = \Omega$.

We note that the larger α is, the fewer iterations can be expected. However, if we increase the overlap, the size and hence the cost of the subproblems increases. It is an important practical issue to balance the total number of iterations and the cost of solving the subproblems.

For each $\Omega_i^{'}, i > 0$, a regular finite element subdivision is inherited from the h-level subdivision of Ω. The corresponding finite element space is defined by

$$V_i^h = H_0^1(\Omega_i^{'}) \cap V^h.$$

The elements of this subspace of V^h can be extended continuously by zero to the complement of $\Omega_i^{'}$. We also use the subspace

$$V_0^h = V^H.$$

It is easy to see that our finite element function space V^h can be represented as the sum of the $N+1$ subspaces,

$$V^h = V_0^h + V_1^h + \cdots + V_N^h.$$

We can now define the projection operators, which are the main building blocks of our algorithms. These operators map the finite element space V^h onto the subspaces V_i^h and are defined in terms of the bilinear forms $B(\cdot,\cdot)$ and $A(\cdot,\cdot)$.

DEFINITION. For $i = 0, \cdots, N$:
For any $w^h \in V^h$, $Q_i w^h \in V_i^h$ is the solution of the finite element equation

$$B(Q_i w^h, v_i^h) = B(w^h, v_i^h), \quad \forall v_i^h \in V_i^h.$$

For any $w^h \in V^h$, $P_i w^h \in V_i^h$ is the solution of the finite element equation

$$A(P_i w^h, v_i^h) = B(w^h, v_i^h), \quad \forall v_i^h \in V_i^h.$$

We now introduce the two operators that define our transformed equations

$$Q^{(1)} = Q_0 + Q_1 + \cdots + Q_N$$

and

$$Q^{(2)} = Q_0 + P_1 + \cdots + P_N.$$

Our main effort goes into the study of the spectra of these two operators. The only difference between $Q^{(1)}$ and $Q^{(2)}$ is that, for $i > 0$, we replace the projection Q_i, corresponding to $\Omega_i^{'}$, by P_i. The coarse mesh projection is not changed.

The computation of $Q_i w^h$ or $P_i w^h$, for $i > 0$ and for an arbitrary function $w^h \in V^h$, involves the solution of a standard finite element linear system of algebraic equations on the small subregion $\Omega_i^{'}$. The former gives rise to a nonsymmetric linear system of equations and the latter to a positive definite, symmetric problem. For $i = 0$, the problem is a standard finite element equation on the H-level, coarse space. One can view P_i as a preconditioner of Q_i in the subspace V_i^h; cf. the discussion in Dryja and Widlund [8], [9]. The cost of the computation can often be decreased by simplifying the local problems further. We can replace the given second-order elliptic operator by the Laplacian. If it is possible to choose some of the Ω_i' to be rectangular and the corresponding mesh to be uniform, a Fast Poisson solver can then be used to compute the contribution from V_i^h. It is an easy exercise to modify our theory to cover such a case.

We will consider two additive Schwarz algorithms:

ALGORITHM 1. *Obtain the solution of* (3) *by solving the equation*

$$Q^{(1)}u^h = b^{(1)}, \tag{4}$$

and

ALGORITHM 2. *Obtain the solution of* (3) *by solving the equation*

$$Q^{(2)}u^h = b^{(2)}. \tag{5}$$

In order for (4) and (5) to have unique solutions, the operators $Q^{(1)}$ and $Q^{(2)}$ must be invertible. This follows from Theorem 1 given in the following discussion. To obtain the same solution as (3), the right-hand sides $b^{(1)}$ and $b^{(2)}$ must be chosen correctly. The crucial observation is that these right-hand sides can be computed without knowledge of the solution of (3). The following formulas are valid:

$$b^{(1)} = Q^{(1)}u^h = \sum_{i=0}^{N} Q_i u^h$$

and

$$b^{(2)} = Q^{(2)}u^h = Q_0 u^h + \sum_{i=1}^{N} P_i u^h.$$

Each of these terms can be computed by solving a problem in a subspace since, by (3) and the definitions of Q_i and P_i,

$$B(Q_i u^h, v_i^h) = B(u^h, v_i^h) = (f, v_i^h), \quad \forall v_i^h \in V_i^h$$

and

$$A(P_i u^h, v_i^h) = B(u^h, v_i^h) = (f, v_i^h), \quad \forall v_i^h \in V_i^h.$$

The main result of this study is Theorem 1. By combining it with the theorem given in §3, we establish that the two algorithms converge at a rate that is independent of the mesh parameters h and H, if the coarse mesh is fine enough.

THEOREM 1. *There exist constants* $H_0 > 0$, $c(H_0) > 0$ *and* $C(H_0) > 0$, *such that if* $H \le H_0$, *then, for* $i = 1, 2$,

$$c(H_0)C_0^{-2}A(u^h, u^h) \le A(u^h, Q^{(i)}u^h)$$

and

$$A(Q^{(i)}u^h, Q^{(i)}u^h) \le C(H_0)A(u^h, u^h).$$

The special constant C_0 is introduced in Lemma 1.

Remarks. (a) The operator Q_0 is very important, since it provides global transportation of information. All the other projections are local mappings. Without using Q_0, information would travel only from one subregion to its neighbors in each iteration and it would take $O(1/H)$ iterations for the information to propagate across the region. For further details, see [24].

Without such a global mechanism, it would also be impossible to confine the spectrum to the right half plane. To see this, we consider a symmetric, indefinite case. If the subregions are small enough, all the local elliptic problems are positive definite, symmetric and, in the absence of a global part, the preconditioner defined by the Schwarz algorithm is positive definite symmetric. Therefore, by the inertia theorem, the operator P has as many negative eigenvalues as the original discrete elliptic problem.

(b) The constant H_0 determines the minimal size of the coarse mesh problem and it depends on the operator L. In general, H_0 decreases if we increase the coefficients of the skew-symmetric terms, it decreases with $\tilde{c}$, while it increases if we increase the overlap. H_0 also depends on the shape of the domain Ω. If the domain is not convex, the estimate of H_0, implicit in our proof of Lemma 5, depends on the parameter γ in (iv). We do not have an explicit formula for H_0 but we know from experience that it can be determined by numerical experiments.

If the operator L is positive definite, symmetric, there is no restriction on the coarse mesh size H, i.e., $H_0 = \infty$.

The proof of Theorem 1 is based on the following results.

LEMMA 1. *There exists a constant C_0, which is independent of h and H, such that, for all $u^h \in V^h$, there exist $u_i^h \in V_i^h$ with*

$$u^h = \sum_{i=0}^{N} u_i^h$$

and

$$\sum_{i=0}^{N} A(u_i^h, u_i^h) \leq C_0^2 A(u^h, u^h).$$

This lemma is also central in the theory previously developed for positive definite, symmetric problems. For a proof see Dryja and Widlund [8]; cf. also Lions [15] or Nepomnyaschikh [20]. Note that this lemma is independent of the skew-symmetric and zero-order terms of the elliptic operator. In the symmetric, positive definite case, Lemma 1 is combined with an abstract argument to give a lower bound for the spectrum of the iteration operator.

The next lemma is a variation of a result by Schatz, cf. [23]. In his proof, Gårding's inequality, (ii), and the regularity result, (iv), are used. The proof of Lemma 2 follows directly from Schatz's work by replacing the approximate solution by the coarse mesh solution and the exact solution of the continuous problem by the finite element solution in V^h.

LEMMA 2. *There exist constants $H_0 > 0$ and $C(H_0) > 0$ such that if $H \leq H_0$, then,*

$$\|Q_0 u^h\|_A \leq C(H_0)\|u^h\|_A$$

and

$$\|Q_0 u^h - u^h\|_{L^2} \leq C(H_0) H^{\gamma} \|Q_0 u^h - u^h\|_A.$$

LEMMA 3. *The restriction of the quadratic form $B(\cdot,\cdot)$ to the subspaces V_i^h, $i > 0$, is strictly positive definite for H sufficiently small, i.e., there exists a constant $c > 0$ such that*

$$cA(u^h,u^h) \leq B(u^h,u^h), \quad \forall u^h \in V_i^h .$$

Proof of Lemma 3. We have to prove that the second order terms dominate the other symmetric term; the contribution from the skew-symmetric term vanishes. This follows from the fact that the smallest eigenvalue for the Dirichlet problem for $-\triangle$ on the region $\Omega_i^{'}$ is on the order of H_i^{-2}.

LEMMA 4. *Let $v^h = \sum v_i^h$, where $v_i^h \in V_i^h$. Then there exists a constant $C > 0$, such that*

$$\|\sum v_i^h\|_A^2 \leq C \sum \|v_i^h\|_A^2.$$

Proof of Lemma 4. The proof follows from the observation that for each $x \in \Omega$, the number of terms in the sum that differ from zero is uniformly bounded.

LEMMA 5. *There exist constants $H_0 > 0$, $c(H_0) > 0$, and $C(H_0) > 0$ such that if $H \leq H_0$, then,*

$$c(H_0)C_0^{-2}A(u^h,u^h) \leq \sum_{i=0}^{N} A(Q_iu^h,Q_iu^h) \leq C(H_0)A(u^h,u^h)$$

and

$$\begin{aligned} c(H_0)C_0^{-2}A(u^h,u^h) &\leq A(Q_0u^h,Q_0u^h) + \sum_{i=1}^{N} A(P_iu^h,P_iu^h) \\ &\leq C(H_0)A(u^h,u^h). \end{aligned}$$

Proof of Lemma 5. An upper bound for $A(Q_0u^h,Q_0u^h)$ is given in Lemma 2. To obtain an upper bound for the sum of the other terms, we use Lemma 3 and the formula

$$B(Q_iu^h,Q_iu^h) = B(u^h,Q_iu^h)$$

to show that

$$c\sum_{i=1}^{N} A(Q_iu^h,Q_iu^h) \leq B\left(u^h, \sum_{i=1}^{N} Q_iu^h\right).$$

The right-hand side can be estimated by using inequality (i) and Lemma 4. The other upper bound is established in a similar way.

To prove the lower bounds, we begin by using Lemma 2 and the triangle inequality to obtain

$$\|u^h\|_{L^2}^2 \leq C(H^{2\gamma}A(u^h,u^h) + \|Q_0u^h\|_{L^2}^2).$$

Since the eigenvalues of the Dirichlet problem for $-\triangle$ are bounded from below and Lemma 2 holds, the last term can be replaced by $C\|Q_0u^h\|_A\|u^h\|_A$. By using Gårding's inequality, (ii), it follows that

$$(1 - CH^{2\gamma})A(u^h,u^h) \leq B(u^h,u^h) + C\|Q_0u^h\|_A\|u^h\|_A .$$

By the definition of the operators Q_i and Lemma 1, we find that

$$B(u^h, u^h) = \sum_{i=0}^{N} B(u^h, u_i^h) = \sum_{i=0}^{N} B(Q_i u^h, u_i^h).$$

The boundedness of $B(\cdot, \cdot)$, (i), can now be used to obtain

$$\sum_{i=0}^{N} B(Q_i u^h, u_i^h) \le C \sum_{i=0}^{N} \|Q_i u^h\|_A \|u_i^h\|_A,$$

which by Lemma 1 and the Cauchy–Schwarz inequality can be bounded above by

$$CC_0 \left(\sum_{i=0}^{N} \|Q_i u^h\|_A^2 \right)^{1/2} \|u^h\|_A.$$

We finally obtain

$$A(u^h, u^h) \le CC_0^2 \sum_{i=0}^{N} A(Q_i u^h, Q_i u^h),$$

for sufficiently small H.

The proof of the other lower bound is quite similar.

Proof of Theorem 1. The upper bounds on the norms of the operators follow immediately from Lemmas 4 and 5.

To obtain the lower bounds, we first consider

$$A(u^h, Q^{(1)} u^h) = \sum_{i=0}^{N} A(u^h, Q_i u^h).$$

Using Lemma 5, we see that it suffices to show that

$$\left| \sum_{i=0}^{N} \left(A(u^h, Q_i u^h) - A(Q_i u^h, Q_i u^h) \right) \right|$$

can be bounded from above by

$$CHA(u^h, u^h).$$

By the definition of the quadratic forms

$$A(u^h - Q_i u^h, Q_i u^h) = B(u^h - Q_i u^h, Q_i u^h) - S(u^h - Q_i u^h, Q_i u^h) - (\tilde{c}(u^h - Q_i u^h), Q_i u^h).$$

By using the definition of Q_i, the first term of the right-hand side is seen to vanish.

For $i = 0$, the absolute value of the second term can be bounded above by $CH^\gamma A(u^h, u^h)$ using inequality (iii) and Lemma 2. We note that $S(Q_i u^h, Q_i u^h) = 0$. There remains to consider $S(u^h, \sum_1^N Q_i u^h)$. By using the inequality (iii),

$$\left| \sum_{i=1}^{N} S(u^h - Q_i u^h, Q_i u^h) \right| \le C \|u^h\|_A \left\| \sum_{i=1}^{N} Q_i u^h \right\|_{L^2}.$$

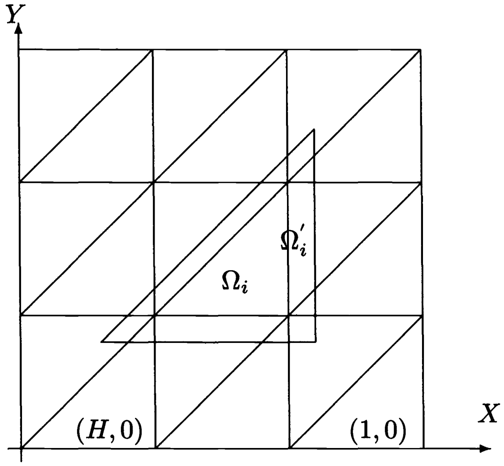


FIG. 1. *An extended subregion.*

Since, for each $x \in \Omega$, the number of terms $Q_i u^h$ that differ from zero is uniformly bounded, the second factor on the right-hand side can be bounded by $C(\sum_{i=1}^{N} \|Q_i u^h\|_{L^2}^2)^{1/2}$. By an elementary estimate, which shows that the smallest eigenvalue of the Dirichlet problem for $-\triangle$ on Ω_i' is on the order of H_i^{-2}, and Lemma 5, the required inequality is established.

The third term is written as the difference of two expressions, which can be handled by exactly the same tools.

The estimate for the operator $Q^{(2)}$ is obtained similarly.

5. Numerical results. In this section, we present some numerical results to demonstrate the behavior of our additive Schwarz algorithms for both symmetric and nonsymmetric indefinite boundary value problems in R^2. Numerical results for positive definite problems, both symmetric and nonsymmetric, have previously been given in [3], [4], [12].

We consider the problem

$$\begin{cases} Lu = f & \text{in } \Omega = [0,1] \times [0,1], \\ u = 0 & \text{on } \partial\Omega. \end{cases}$$

The coefficients of L are specified later for each problem. The right-hand side f is always chosen so that the exact solution is $u = xe^{xy}\sin(\pi x)\sin(\pi y)$.

We use a two-level subdivision of Ω as described in §2. The subregion Ω_i' is obtained by enlarging the triangle Ω_i as in Fig. 1. In this extension, the same number **ovlp** of h-level triangles are added in all directions.

In our experiments all the subproblems are solved exactly by using a band solver from LINPACK. We stop the GMRES method as soon as $\|r_i\|_A/\|r_0\|_A \leq 10^{-3}$. We work with the A-norm, since our theory so far has not been developed for any other norms. However, in our experience, the performance of the algorithm is quite comparable if we replace that norm with the 2-norm. We have found that the overall error is not substantially reduced by a more stringent stopping criterion. In our tables, the **error** denotes the difference between the computed solution and the exact solution of

TABLE 1

Convergence history for Algorithm 1 *and Example* 1. *Here* $h^{-1} = 75$, $H^{-1} = 15$, **ovlp** $= 2$ *and* $\delta = 16.0\pi^2$

Iteration	A – norm; residual	L^2 norm; error	L^∞ norm; error
1	2.50722	0.318660	0.719067
2	1.44028	0.182950	0.405074
3	0.971708	1.63224E-02	4.73967E-02
4	0.218693	7.38034E-03	2.46119E-02
5	5.54836E-02	6.13811E-03	1.94588E-02
6	3.40790E-02	5.16731E-03	1.53261E-02
7	2.60738E-02	3.71766E-03	9.73493E-03
8	1.87841E-02	2.19535E-03	6.23578E-03
9	1.03642E-02	1.67804E-03	4.90165E-03
10	6.81844E-03	1.36607E-03	4.00215E-03
11	5.02644E-03	8.28570E-04	2.39784E-03

TABLE 2

Example 1. *The last two columns give the number of* GMRES *iterations.*

Case #	δ	h^{-1}	H^{-1}	ovlp	Algorithm 1	Algorithm 2
1	$3\pi^2$	15	3	2	11	12
2		30	3	4	11	12
3		45	3	6	12	12
4		60	3	8	12	12
5		15	5	1	10	10
6		30	5	2	12	12
7		45	5	3	12	12
8		60	5	4	12	12
10	$16\pi^2$	45	15	1	10	10
11		60	15	1	11	11
12		75	15	2	11	11
13		60	5	4	44	33
14		60	10	2	17	17
15		60	20	1	8	8
16	$30\pi^2$	60	20	1	16	16
17		80	20	1	17	18

the continuous problem measured in the norms indicated. The programs have been run in single precision on the Multiflow computer at Yale University.

EXAMPLE 1. We consider the symmetric and indefinite Helmholtz equation

$$\left\{ \begin{array}{rl} -\triangle u - \delta u = f & \text{in} \quad \Omega, \\ u = 0 & \text{on} \quad \partial\Omega. \end{array} \right. \tag{6}$$

δ is a constant. The eigenvalues of the operator in (6) are $(i^2 + j^2)\pi^2 - \delta$, where i, j are positive integers. The numerical results are given in Tables 1 and 2. Algorithms 1 and 2, given in (4) and (5), respectively, are used.

EXAMPLE 2. We consider a nonsymmetric and indefinite problem

$$\left\{ \begin{array}{rl} -\triangle u - \eta(\partial u/\partial x + \partial u/\partial y) - \delta u = f & \text{in} \quad \Omega \\ u = 0 & \text{on} \quad \partial\Omega. \end{array} \right. \tag{7}$$

The numerical results are given in Tables 3 and 4.

We note that in a few of the experiments, the rate of convergence is unsatisfactory but that the rate of convergence improves considerably by decreasing H. The rate of convergence varies only marginally with the parameter **ovlp**. Normally, the overall

TABLE 3

Convergence history for Algorithm 1 *and Example* 2. *Here* $h^{-1} = 120$, $H^{-1} = 20$, **ovlp** $= 2$, $\eta = 16.0\pi$, *and* $\delta = 16.0\pi^2$.

Iteration	A-norm; residual	L^2 norm; error	L^∞ norm; error
1	2.99430	0.309833	0.696816
2	1.82397	0.234651	0.529782
3	1.34039	0.136604	0.313758
4	0.905845	7.27307E-02	0.172788
5	0.585598	4.46239E-02	0.108047
6	0.407054	2.78872E-02	6.71409E-02
7	0.288880	1.37858E-02	3.38832E-02
8	0.180577	8.21095E-03	2.06587E-02
9	0.129736	4.44917E-03	1.15359E-02
10	8.77408E-02	2.19873E-03	6.17071E-03
11	5.48599E-02	1.18357E-03	3.88315E-03
12	3.46359E-02	7.18704E-04	2.24515E-03
13	2.29454E-02	4.35330E-04	1.39198E-03
14	1.35519E-02	2.90249E-04	1.03428E-03
15	8.90639E-03	2.18270E-04	6.67672E-04
16	5.98530E-03	1.90636E-04	5.61312E-04
17	3.89341E-03	1.72248E-04	5.31457E-04

TABLE 4

Example 2. *The last two columns give the number of* GMRES *iterations.*

Case #	Parameters	h^{-1}	H^{-1}	ovlp	Algorithm 1	Algorithm 2
1	$\eta = 3\pi$ $\delta = 3\pi^2$	15	5	1	13	12
2		30	5	2	17	14
3		45	5	3	18	14
4		60	5	4	18	14
5		60	6	3	16	14
6		60	10	2	12	11
7	$\eta = 16\pi$ $\delta = 16\pi^2$	45	15	1	17	13
8		60	15	1	18	14
9		75	15	2	25	17
10		60	20	1	13	11
11		80	20	1	14	12
12		100	20	2	18	14
13		120	20	2	17	14
14	$\eta = 30\pi$ $\delta = 30\pi^2$	60	20	1	24	16
15		120	20	2	35	19
16		75	25	1	17	13
17		100	25	1	18	14
18		120	30	1	15	13

cost of the computation is smallest if **ovlp** $= 1$. We also note that, as expected, a smaller H is required when the parameters δ and η are increased to increase the terms that make the operators skew-symmetric and indefinite.

6. Two other methods. We conclude by outlining how some other results, previously analyzed for the positive definite symmetric case, can be extended to the class of elliptic problems described in §2. We confine our discussion to problems in the plane; both of the algorithms considered here need to be modified considerably in order to obtain fast methods for problems in three dimensions.

We first consider a basic iterative substructuring method for problems in two dimensions, cf. Dryja and Widlund [8], [9]. For problems that are nonsymmetric but positive definite, the result to be formulated has previously been obtained by Cai [3], [4].

When iterative substructuring methods are used, the region is divided into substructures and elements as in §2. Though originally derived differently, it has been demonstrated by Dryja and Widlund [8] that these methods can be viewed as additive Schwarz methods. Our work depends heavily on this reinterpretation of the algorithms, see [8] for detailed arguments.

In defining the partition of the finite element space into subspaces, we use the coarse space V^H introduced in §4. We also use subspaces corresponding to the subregions $\Omega_{ij} = \Omega_i \bigcup \Gamma_{ij} \bigcup \Omega_j$. These subregions play the same role as the Ω_i' in §4. Here Ω_i and Ω_j are adjacent substructures with the common edge Γ_{ij}. We note that an interior substructure is covered by three such regions. The local subspaces are $V_{ij}^h = H_0^1(\Omega_{ij}) \cap V^h$.

Compared with the case considered previously, we use less overlap in the sense that only the elements of V^H can differ from zero at the vertices of the substructures. This is reflected in a poorer bound for the constant of Lemma 1,

$$C_0^2 \leq \text{const.}(1+\log(H/h))^2,$$

cf. Dryja and Widlund [8]. Lemma 1 is modified accordingly. The rest of the proof carries over without change. In Theorem 2, we use the notation $\tilde{Q} = Q_0 + \sum Q_{ij}$.

THEOREM 2. *For the iterative substructuring method, introduced as an additive Schwarz method with the subspaces V^H and V_{ij}^h, there exist constants $H_0 > 0$, $c(H_0) > 0$, and $C(H_0) > 0$, such that if $H \leq H_0$,*

$$c(H_0)(1+\log(H/h))^{-2} A(u^h, u^h) \leq A(u^h, \tilde{Q}u^h)$$

and

$$A(\tilde{Q}u^h, \tilde{Q}u^h) \leq C(H_0) A(u^h, u^h).$$

We finally show that the result, obtained by Yserentant [26] for positive definite symmetric problems, can be extended in the same way. We note that Bank and Yserentant [1] have already reported on successful numerical experiments with an accelerated variant of this algorithm for the class of elliptic problems introduced in §2. We also note that our algorithm is different from those proposed by Yserentant [27], [28] for indefinite and nonsymmetric problems. Thus in [27] a reduced system obtained by implicitly eliminating the nodes of the coarsest mesh is solved by an iterative method.

We assume that the region Ω is a plane polygon. A coarse triangulation is introduced as before. Its triangles are recursively subdivided into four congruent triangles a total of j times. The characteristic mesh size for the level k triangulation is h_k. As demonstrated in Yserentant [28], more complicated situations can also be considered where the final triangulation is highly nonuniform, but to simplify our discussion, we only consider the regular case in this paper.

As shown in Dryja and Widlund [10], Yserentant's method can also be viewed as an additive Schwarz method defined by a set of subspaces. Let $I_k v \equiv I_{h_k} v$ be the linear interpolant of $v \in V^h$ onto the space of finite elements on the level k triangulation. The following identity holds

$$v = I_0 v + (I_1 v - I_0 v) + \cdots + (I_j v - I_{j-1} v) \ , \ \forall v \in V^h \ .$$

We represent V^h as

$$V^h = V_0 \oplus V_1 \oplus \cdots \oplus V_j ,$$

where $V_0 = V^H$, and, for $k > 0$, $V_k = R(I_k - I_{k-1})$ is the range of the operator $(I_k - I_{k-1})$. An additive Schwarz method is defined for this set of subspaces. We obtain Yserentant's method by replacing, for $k > 0$, the resulting problems on the subspaces by suitable preconditioners.

The following result holds for the family of elliptic problems introduced in §2. Here $\hat{Q}$ denotes the operator of the transformed equation, which corresponds to Yserentant's method.

THEOREM 3. *For Yserentant's method there exist constants $H_0 > 0$, $c(H_0) > 0$, and $C(H_0) > 0$, such that if $H \leq H_0$,*

$$c(H_0)j^{-2}A(u^h, u^h) \leq A(u^h, \hat{Q}u^h)$$

and

$$A(\hat{Q}u^h, \hat{Q}u^h) \leq C(H_0)A(u^h, u^h).$$

We will only outline how this result can be established. We model our proof on that of Theorem 1. We have to show that the different lemmas hold for the spaces just introduced. It is shown in Yserentant [26] that Lemma 1 holds with $C_0 \leq Cj^2$, cf. [26, Lems. 2.4 and 2.5]. Lemma 2 is still valid since the same coarse operator is used in all the methods considered in this paper. A counterpart of Lemma 3 can be obtained as well, by using [26, Lem. 2.4]. Lemma 4 is modified by using [26, Lem. 2.7], a result that makes it possible to obtain a sharp upper bound in Yserentant's main theorem. Lemma 5 can be modified in a straightforward manner. One change is required in the proof of the theorem. The factor $\| \sum_{i=1}^{N} \hat{Q}_i u^h \|_{L^2}$ must be estimated differently, since, typically, all these terms differ from zero everywhere. By using Yserentant's tools, it is however possible to show that

$$\|\hat{Q}_i u^h\|_{L^2} \ \leq \ Ch_i \|\hat{Q}_i u^h\|_A \ .$$

Since the h_i decay geometrically, the triangle and Cauchy–Schwarz inequalities give

$$\| \sum_{i=1}^{N} \hat{Q}_i u^h \|_{L^2}^2 \ \leq \ CH^2 \sum_{i=1}^{N} \|\hat{Q}_i u^h\|_A^2 \ ,$$

and the proof can be completed.

REFERENCES

[1] R. E. BANK AND H. YSERENTANT, *Some remarks on the hierarchical basis multigrid method*, in Domain Decomposition Methods, T. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989.

[2] J. H. BRAMBLE, J. E. PASCIAK, AND J. XU, *The analysis of multigrid algorithms for nonsymmetric and indefinite elliptic problems*, Math. Comp., 51 (1988), pp. 389–414.

[3] X.- C. CAI, *Some domain decomposition algorithms for nonselfadjoint elliptic and parabolic partial differential equations*, Ph.D. thesis, Courant Institute, New York University, New York, NY, 1989.

[4] ———, *An additive Schwarz algorithm for nonselfadjoint elliptic equations*, in Third International Symposium on Domain Decomposition Methods for Partial Differential Equations, T. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990.

[5] P. Ciarlet, *The Finite Element Method for Elliptic Problems*, North-Holland, Amsterdam, 1978.

[6] M. Dryja, *An additive Schwarz algorithm for two- and three- dimensional finite element elliptic problems*, in Domain Decomposition Methods, T. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989.

[7] M. Dryja and O. B. Widlund, *An additive variant of the Schwarz alternating method for the case of many subregions*, Tech. Report 339, Department of Computer Science, Courant Institute, New York University, New York, NY, 1987.

[8] ———, *Some domain decomposition algorithms for elliptic problems*, in Iterative Methods for Large Linear Systems, L. Hayes and D. Kincaid, eds., Academic Press, San Diego, CA, 1989.

[9] ———, *Towards a unified theory of domain decomposition algorithms for elliptic problems*, in Third International Symposium on Domain Decomposition Methods for Partial Differential Equations, T. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990.

[10] ———, *Multilevel additive methods for elliptic finite element problems*, in Parallel Algorithms for Partial Differential Equations, Proceedings of the Sixth GAMM-Seminar, Kiel, January 19–21, 1990, W. Hackbusch, ed., Braunscweig, Germany, 1991, Vieweg & Son.

[11] S. C. Eisenstat, H. C. Elman, and M. H. Schultz, *Variational iterative methods for nonsymmetric systems of linear equations*, SIAM J. Numer. Anal., 20 (1983), pp. 345–357.

[12] A. Greenbaum, C. Li, and Z. Han, *Parallelizing preconditioned conjugate gradient algorithms*, Comput. Phys. Comm., 53 (1989), pp. 295–309.

[13] P. Grisvard, *Elliptic Problems in Nonsmooth Domains*, Pitman, Boston, 1985.

[14] Yu. A. Kuznetsov and O. D. Trufanov, *Domain decomposition methods for the wave Helmholtz equation*, Soviet. J. Numer. Anal. Math. Modelling, 2 (1987), pp. 113–136.

[15] P. L. Lions, *On the Schwarz alternating method* I, in First International Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988.

[16] J. Mandel, *Multigrid convergence for nonsymmetric, indefinite variational problems and one smoothing step*, Appl. Math. Comput., 19 (1986), pp. 201–216.

[17] T. A. Manteuffel, *The Tchebychev iteration for nonsymmetric linear systems*, Numer. Math., 28 (1977), pp. 307–327.

[18] A. M. Matsokin and S. V. Nepomnyaschikh, *A Schwarz alternating method in a subspace*, Soviet Math., 29 (1985), pp. 78–84.

[19] J. Nečas, *Sur la coercivité des formes sesquilinéaires, elliptiques*, Rev. Roumaine Math. Pures Appl., 9 (1964), pp. 47–69.

[20] S. V. Nepomnyaschikh, *Domain decomposition and Schwarz methods in a subspace for the approximate solution of elliptic boundary value problems*, Ph.D. thesis, Computing Center of the Siberian Branch of the USSR Academy of Sciences, Novosibirsk, USSR, 1986.

[21] J. Nitsche, *Lineare Spline-Funktionen und die Methode von Ritz für Elliptische Randwertprobleme*, Arch. Rational Mech. Anal., 36 (1970), pp. 348–355

[22] Y. Saad and M. H. Schultz, GMRES: *A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 865–869.

[23] A. H. Schatz, *An observation concerning Ritz–Galerkin methods with indefinite bilinear forms*, Math. Comp., 28 (1974), pp. 959–962.

[24] O. B. Widlund, *Iterative substructuring methods: Algorithms and theory for elliptic problems in the plane*, in First International Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988, pp. 113–128.

[25] J. Xu, *Theory of multilevel methods*, Report No. AM 48, Department of Mathematics, Pennsylvania State University, College Park, PA, 1989.

[26] H. Yserentant, *On the multi-level splitting of finite element spaces*, Numer. Math., 49 (1986), pp. 379–412.

[27] ———, *On the multi-level splitting of finite element spaces for indefinite elliptic boundary value problems*, SIAM J. Numer. Anal., 23 (1986), pp. 581–595.

[28] ———, *Hierarchical bases of finite-element spaces in the discretization of nonsymmetric elliptic boundary value problems*, Computing, 35 (1985), pp. 39–49.

SIAM J. Sci. Stat. Comput.
Vol. 13, No. 1, pp. 259–288, January 1992

014

PRECONDITIONING SECOND-ORDER ELLIPTIC OPERATORS: EXPERIMENT AND THEORY*

WAYNE JOUBERT†, THOMAS A. MANTEUFFEL‡, SEYMOUR PARTER¶, AND SZE-PING WONG§

Abstract. In an earlier work Manteuffel and Parter discussed the role of boundary conditions in obtaining elliptic operators B so that the preconditioned operators $B_h^{-1}A_h$ or $A_hB_h^{-1}$ have uniformly bounded L_2 condition number. Here A is the original elliptic operator and A_h and B_h are discretizations. Certain computations, mostly one-dimensional, were undertaken to illustrate and understand these results. These computational experiments provided several surprises. This current work describes those experiments and some subsequent experiments together with theoretical explanations of these surprising results. One of the main points of this report is the discussion of interaction of experimental results with the ensuing development of the theory.

Key words. preconditioning, boundary conditions, conjugate gradient, experiment

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

1. Introduction.

In [MP] Manteuffel and Parter studied the effect of boundary conditions on the preconditioning of an invertible nonsymmetric discrete second-order elliptic operator A_h (i.e., the discretization of an invertible second-order elliptic operator A) by B_h^{-1}, the inverse of another such discrete elliptic operator in the presence of H_2 regularity. They discussed L_2, H_1, and H_2 estimates on $B_h^{-1}A_h$ and L_2 estimates on $A_hB_h^{-1}$. Roughly speaking, the L_2 condition of $A_hB_h^{-1}$ is bounded independent of h if and only if A and B have the same boundary conditions, while the L_2 condition of $B_h^{-1}A_h$ is bounded independent of h if and only if A^* and B^* have the same boundary conditions. Further, the H_1 condition of $B_h^{-1}A_h$ is bounded independent of h if and only if A and B have Dirichlet boundary conditions on the same portion of the boundary. Finally, the H_2 condition number of $B_h^{-1}A_h$ is always bounded independent of h if A and B have H_2 regularity. In order to gain further insight and understanding, we undertook a number of computational experiments. At first, some of the numerical results seemed somewhat puzzling. The purpose of this report is to (i) describe and report on those experiments, (ii) explain their interesting results, and (iii) remark on several related ideas.

In §2 we present one-dimensional computational results. These results show that the condition number associated with various preconditioning strategies is either bounded or grows as the mesh size decreases, as predicted in [MP]. However, the conjugate gradient method applied to these problems converges much faster than bounds based upon the condition number would suggest. In the case where the condition number is growing, we discover in §3 that all but two singular values are bounded

*Received by the editors April 5, 1990; accepted for publication (in revised form) May 12, 1991. This research was supported by Los Alamos National Laboratory under U.S. Department of Energy contract W-7405-ENG-36, Air Force Office of Scientific Research contract AFOSR-86-0163, National Science Foundation grant DMS-8913091 and grant DMS-8704169.

†Center for Numerical Analysis, University of Texas, Austin, Texas 78712.

‡Computational Mathematics Group, University of Colorado, Denver, Colorado 80204, and Computing and Communications Division, Los Alamos National Laboratory, Los Alamos, New Mexico 87544.

¶Department of Computer Sciences, University of Wisconsin, Madison, Wisconsin 53706.

§Department of Mathematics, University of Wisconsin, Madison, Wisconsin 53706.

above and below. This fact allows us to produce bounds that more accurately predict the performance of the conjugate gradient method. In §4, computational results reveal the structure of the bounded, as well as the unbounded, singular values. These results lead to a theory in §5 that explains the behavior of the conjugate gradient method when the condition is bounded.

In §6, two-dimensional computational results are presented. Here again, the various condition numbers behave as predicted in [MP]. Again, the conjugate gradient method behaves better than bounds based upon the condition number would suggest, but not as strikingly as in the one-dimensional case. Now, the rank of the growing part is low but is much greater than 2. In §7, an analysis of a separable problem shows that the number of unbounded singular values grows like the square root of the number of mesh points in the boundary segment where "incorrect" boundary conditions are used for the preconditioning.

2. One-dimensional computational results. In this section, the computational experiments involve simple one-dimensional operators. Let

$$Au = -(a_2(x)u')' + a_1(x)u' + a_0(x)u, \qquad 0 < x < 1, \tag{2.1a}$$

with boundary conditions

$$u(0) = 0, \qquad u'(1) + \alpha u(1) = 0, \tag{2.1b}$$

while

$$Bu = -(b_2(x)u')' + b_1(x)u' + b_0(x)u, \qquad 0 < x < 1, \tag{2.2a}$$

with boundary conditions

$$u(0) = 0, \qquad u'(1) + \beta u(1) = 0. \tag{2.2b}$$

We assume that $a_2(x), b_2(x)$ are smooth bounded positive functions bounded away from zero. That is, there are constraints $0 < \Delta_0 < \Delta_1$ such that

$$0 < \Delta_0 \le a_2(x), b_2(x) \le \Delta_1.$$

The discrete operators are obtained by simple central differences. For example, let $N > 1$ be an integer and let

$$h = \frac{1}{N+1}, \tag{2.3}$$

$$x_j = jh, \qquad j = 0, 1, \cdots, N+1, \tag{2.4}$$

and let $U = \{U_j, j = 0, 1, \cdots, N+1\}$ be a grid vector. Then A_h, the discretization of A, is given by

$$[A_h U]_j = -d_j U_{j-1} + e_j U_j - f_j U_{j+1}, \qquad j = 1, 2, \cdots, N, \tag{2.5a}$$

where

$$d_j = \left[a_2\left(x_j - \frac{1}{2}h\right) - \frac{h}{2}a_1(x_j)\right] h^{-2}, \tag{2.5b}$$

$$e_j = \left[a_2\left(x_j - \frac{1}{2}h\right) + a_2\left(x_j + \frac{1}{2}h\right) + h^2 a_0(x_j)\right] h^{-2}, \tag{2.5c}$$

$$f_j = \left[a_2\left(x_j + \frac{1}{2}h\right) + \frac{h}{2}a_1(x_j)\right] h^{-2}, \tag{2.5d}$$

with the additional (boundary) conditions

$$U_o = 0, \tag{2.6a}$$

$$(U_{N+1} - U_N) + \frac{h}{2}\alpha(U_{N+1} + U_N) = 0. \tag{2.6b}$$

The L_2 adjoint A^* of A is given by

$$A^*u = -(a_2(x)u')' - (a_1u)' + a_ou, \qquad 0 < x < 1, \tag{2.7a}$$

with boundary conditions

$$u(0) = 0, \qquad u'(1) + \alpha^* u(1) = 0, \tag{2.7b}$$

where

$$\alpha^* = \left[\alpha + \frac{a_1(1)}{a_2(1)}\right]. \tag{2.7c}$$

Let

$$\|U\|_h = \left\{h\sum_{j=1}^{N} U_j^2\right\}^{1/2}. \tag{2.8}$$

The theory developed in [MP] asserts: there is a constant $K_L > 0$, independent of h, such that

$$C_h(B_h^{-1}A_h) = \|B_h^{-1}A_h\|_h \cdot \|A_h^{-1}B_h\|_h \leq K_L, \tag{2.9a}$$

if and only if

$$\beta^* = \left[\beta + \frac{b_1(1)}{b_2(1)}\right] = \left[\alpha + \frac{a_1(1)}{a_2(1)}\right] = \alpha^* \tag{2.9b}$$

and there is a constant K_R independent of h, such that

$$C_h(A_hB_h^{-1}) = \|A_hB_h^{-1}\|_h \cdot \|B_hA_h^{-1}\|_h \leq K_R, \tag{2.10a}$$

if and only if

$$\alpha = \beta. \tag{2.10b}$$

Moreover, if these conditions are violated, the corresponding condition number is proportional to h^{-1}.

Numerical experiments were performed to verify the theoretical results. Two approaches were used. In the first, the operators $A_h B_h^{-1}$ and $B_h^{-1} A_h$ were constructed as full matrices and a singular value decomposition was used to determine the appropriate condition numbers. Memory constraints limited this approach to systems of 1024 unknowns.

The second approach was to exploit the relationship between the conjugate gradient method and the Lanczos algorithm. For left preconditioning, the normal equations of the preconditioned system were formed and the conjugate gradient method PCGNE (cf. [AMS]) was performed on the preconditioned system

$$(B_h^{-1}A_h)^*(B_h^{-1}A_h)U = (B_h^{-1}A_h)^* B_h^{-1}F. \tag{2.11}$$

The tridiagonal B_h was factored once and backsolution was used to compute $B_h^{-1}V$ for a given vector V. The PCGNE method is optimal in the ℓ_2-norm of the error, $E_j = U - U_j$, at each step of the iteration. The parameters generated by this iteration can be used to get eigenvalue estimates of the operator $A_h^* B_h^{-*} B_h^{-1} A_h$ and thus estimates of $C_h(B_h^{-1}A_h)$ (cf. [CGO], [AMS]).

For right preconditioning, the preconditioned conjugate gradient (PCG) method was applied to the normal equations. Consider the system

$$A_h B_h^{-1} V = F, \tag{2.12a}$$

$$B_h U = V. \tag{2.12b}$$

The normal equations associated with (2.12a) are

$$(A_h B_h^{-1})^*(A_h B_h^{-1})V = (A_h B_h^{-1})^* F. \tag{2.13}$$

The standard conjugate gradient algorithm (CGHS) may be performed on (2.13) and (2.12b) may be used to find the solution U. However, this process is computationally equivalent to performing the PCG algorithm on

$$(B_h^* B_h)^{-1} A_h^* A_h U = (B_h^* B_h)^{-1} A_h^* F \tag{2.14}$$

and avoids the auxiliary equation (2.12b). This method, known as PCGNR, minimizes the ℓ_2-norm of the residual, $R_i = A_h E_i = A_h(U - U_i)$, at each step of the iteration and yields estimates of the eigenvalues of $(B_h^* B_h)^{-1} A_h^* A_h$, which is similar to $B_h^{-*} A_h^* A_h B_h^{-1}$. Thus estimates of $C_h(A_h B_h^{-1})$ are available.

With this approach, systems of 262,144 unknowns were viable in a convenient amount of time and storage on Cray XMP 4/16. Remarkably, the iterative method gave the same results up to three digits of accuracy as the direct method on the smaller problems. For that reason, only the iterative results are displayed below.

The test problems were given a random starting guess and a zero right-hand side, $F \equiv 0$, so that the error could be easily computed. The algorithms were altered so that direct calculation of the residual replaced the recursive formula. This avoids the stalling that may occur when the residual and direction vector become disassociated due to round-off when the residual is driven to small values. For left preconditioning, the PCGNE iteration was terminated when the ℓ_2-norm of the error, $\| E_i \|_h$, was reduced by a factor of 10^{-9}. For the right preconditioning, the PCGNR iteration was terminated when the ℓ_2-norm of the residual, $\| R_i \|_h$, was reduced by a factor of 10^{-9}. Notice that the stopping criteria were based upon the measure of the error that was optimized by the respective algorithms.

Case (i). $D(A) = D(B)$. In the first set of experiments we consider operators with the same boundary conditions. We denote this by saying the domain of $A, D(A)$, equals the domain of $B, D(B)$. Let

$$A = -u'' + a_1u,' \quad u(0) = 0, \quad u'(1) = 0, \tag{2.15a}$$

$$B = -u'' + cu, \quad u(0) = 0, \quad u'(1) = 0. \tag{2.15b}$$

Table 2.1 shows the results for $a_1 = 10.0$, $c = 0$. The estimate of $C_h(B_h^{-1}A_h)$ grows while the estimate of $C_h(A_hB_h^{-1})$ does not. The rate of growth was measured by assuming that

$$C_h(B_h^{-1}A_h) = K\left(\frac{1}{h}\right)^P, \tag{2.16}$$

where K is independent of h. This yields

$$C_h(B_h^{-1}A_h)/C_{2h}(B_{2h}^{-1}A_{2h}) = 2^P. \tag{2.17}$$

The exponent P was calculated for each successive pair of values of h and appears in the column labeled "Growth Rate."

TABLE 2.1
$D(A) = D(B), a_1 = 10.0, c = 0.$

	Left Preconditioning			Right Preconditioning		
1/h	$C_h(B_h^{-1}A_h)$	Growth Rate	Its	$C_h(A_hB_h^{-1})$	Growth Rate	Its
8	.1834+02	.0000+00	8	.6205+01	.0000+00	8
16	.3803+02	.1051+01	14	.6653+01	.1006+00	12
32	.6748+02	.8274+00	17	.6864+01	.4503-01	12
64	.1117+03	.7282+00	19	.6967+01	.2140-01	12
128	.1898+03	.7636+00	19	.7018+01	.1051-01	12
256	.3501+03	.8832+00	21	.7044+01	.5361-02	12
512	.6782+03	.9539+00	21	.7057+01	.2579-02	13
1024	.1338+04	.9803+00	23	.7063+01	.1276-02	11
2048	.2659+04	.9909+00	23	.7066+01	.6338-03	11
4096	.5302+04	.9956+00	26	.7068+01	.3154-03	12
8192	.1059+05	.9978+00	26	.7068+01	.1573-03	13
16384	.2116+05	.9989+00	39	.7069+01	.7817-04	13
32768	.4231+05	.9994+00	31	.7069+01	.3706-04	12
65536	.8461+05	.9997+00	34	.7069+01	.1026-04	11
131072	.1692+06	.9998+00	38	.7069+01	.2924-04	12
262144	.3383+06	.9997+00	54	.7069+01	.5487-04	11

Finally, the number of iterations required to reduce the error for left preconditioning and the residual for right preconditioning by a factor of 10^{-9} appears in the column labeled "Its."

Numerical experiments were performed with various values of a_1 and c. The complete results appear in [JMPW], where it is shown that $C_h(A_h B_h^{-1})$ grows with a_1. The theory in [MP] predicts that $C_h(A_h B_h^{-1})$ should grow as $a_1^{3/2}$ and the constant K in (2.16) should also grow as $a_1^{3/2}$.

The behavior with respect to c is more complicated. It is possible to choose c so as to tightly cluster the spectrum of $A_h B_h^{-1}$, but the relation to the condition $C_h(A_h B_h^{-1})$ is more complicated. It is true, in any case, that for $a_1 \neq 0$ the optimal choice is not $c = 0$. These issues are explored in [MO], where it is shown that for a class of two-dimensional operators the optimal value of c grows as $(a_1 h)^2$.

Case (ii). $D(A^*) = D(B^*)$. In the next set of experiments we consider operators whose adjoints have the same boundary conditions. We have

$$A = -u'' + a_1 u \quad u(0) = 0, \quad u'(1) = 0, \tag{2.18a}$$

$$B = -u'' + cu \quad u(0) = 0, \quad u'(1) + a_1 u(1) = 0. \tag{2.18b}$$

Table 2.2 shows the results with $a_1 = 10$ and $c = 0$. We see that it is $C_h(B_h^{-1} A_h)$ that remains bounded while $C_h(A_h B_h^{-1})$ grows with h. Again, see [JMPW] for results for other values of a_1 and c.

TABLE 2.2
$D(A^*) = D(B^*)$, $a_1 = 10$, $c = 0$.

	Left Preconditioning			Right Preconditioning		
1/h	$C_h(B_h^{-1} A_h)$	Growth Rate	Its	$C_h(A_h B_h^{-1})$	Growth Rate	Its
8	.3415+01	.0000+00	7	.6031+01	.0000+00	7
16	.5156+01	.5941+00	12	.1135+02	.9123+00	13
32	.6497+01	.3336+00	13	.2016+02	.8286+00	14
64	.7362+01	.1802+00	14	.3611+02	.8408+00	16
128	.7859+01	.9414−01	13	.6717+02	.8955+00	17
256	.8126+01	.4816−01	14	.1291+03	.9424+00	18
512	.8264+01	.2436−01	13	.2529+03	.9703+00	18
1024	.8335+01	.1225−01	13	.5007+03	.9850+00	18
2048	.8370+01	.6146−02	13	.9962+03	.9924+00	21
4096	.8388+01	.3078−02	14	.1987+04	.9962+00	21
8192	.8397+01	.1540−02	13	.3969+04	.9981+00	23
16384	.8402+01	.7702−03	14	.7933+04	.9990+00	26
32768	.8404+01	.3845−03	14	.1585+05	.9991+00	26
65536	.8405+01	.1897−03	14	.3163+05	.9963+00	26
131072	.8405+01	.8317−04	14	.4970+04	−.2670+01	31
262144	.8406+01	.2272−04	14	.1268+06	.4673+01	62

Case (iii). A and B selfadjoint, $D(A) \neq D(B)$. In this set of experiments we let

$$A = -u'' \quad u(0) = 0, \quad u'(1) = 0, \tag{2.19a}$$

$$B = -u'' \quad u(0) = 0, \quad u'(1) + \alpha u(1) = 0. \tag{2.19b}$$

Since $D(A) \neq D(B)$ and $D(A^*) \neq D(B^*)$ neither $C_h(B_h^{-1} A_h)$ nor $C_h(A_h B_h^{-1})$ will be bounded independent of h. On the other hand, since A and B are selfadjoint and positive definite and have Dirichlet boundary conditions on the same portion of the boundary the results of [MP] tell us that A and B are spectrally equivalent. In other words, the spectrum of $B_h^{-1}A_h$ is bounded while $\| B_h^{-1}A_h \|_h$ is unbounded.

We again use PCGNE to yield estimates of $C_h(B_h^{-1}A_h)$. To get estimates of the spectrum of $B_h^{-1}A_h$ (which is the same as the spectrum of $B_h^{-1/2}A_hB_h^{-1/2}$) we use the PCG algorithm, which is optimal in the A_h-norm: $(\langle A_h\cdot,\cdot\rangle_h)^{1/2}$.

Table 2.3 shows the results for $\alpha = 2.0$ in (2.19b). Notice that $C_h(B_h^{-1}A_h)$ grows while $C_h(B_h^{-1/2}\ A_h\ B_h^{-1/2})$ remains bounded. The number of iterations for the symmetric preconditioning reflects the number required to reduce the A_h-norm of the error by a factor of 10^{-9}. Numerical results with $\alpha = 20.0$ in (2.19b) can be found in [JMPW]. The results in [MP] show that, in fact, $B_h^{-1}A_h$ has two eigenvalues:

$$\lambda_1 = 1, \qquad \lambda_2 = 1 + \frac{\alpha}{(1+\frac{\alpha h}{2})(1+h)}.$$

TABLE 2.3
A and B selfadjoint, $D(A) \neq D(B)$, $\alpha = 2.0$, $c = 0$.

	Left Preconditioning			Symmetric Preconditioning		
1/h	$C_h(B_h^{-1}A_h)$	Growth Rate	Its	$C_h(B_h^{-1/2}A_hB_h^{-1/2})$	Growth Rate	Its
8	.1231+01	.0000+00	3	.2444+01	.0000+00	2
16	.1677+01	.4460+00	3	.2705+01	.1465+00	2
32	.2274+01	.4388+00	3	.2848+01	.7409-01	2
64	.3147+01	.4687+00	3	.2923+01	.3729-01	2
128	.5740+02	.4188+01	4	.2961+01	.1871-01	2
256	.1142+03	.9935+00	5	.2980+01	.9374-02	2
512	.2280+03	.9967+00	5	.2990+01	.4691-02	2
1024	.4556+03	.9983+00	5	.2995+01	.2346-02	2
2048	.9105+03	.9989+00	5	.2997+01	.1173-02	2
4096	.1820+04	.9991+00	5	.2998+01	.5869-03	2
8192	.3452+04	.9236+00	5	.2999+01	.2935-03	2
16384	.7282+04	.1076+01	10	.2999+01	.1467-03	2
32768	.1456+05	.9999+00	14	.2999+01	.7337-04	2
65536	.2912+05	.1000+01	15	.2999+01	.3622-04	2
131072	.5826+05	.1000+01	16	.2999+01	.3500-04	2
262144	.1164+06	.9994+00	18	.3000+01	.3455-04	2

3. Analysis of the one-dimensional computations. All conjugate gradient methods are polynomial methods and, thus, yield iterates that obey the equation

$$E_j = p_j(CA)E_0, \tag{3.1}$$

where CA is the preconditioned system and $p_j(z)$ is a polynomial of degree j such that $p_j(0) = 1$ (cf. [AMS]). In fact, if the particular conjugate gradient method is optimal in the $M_h - norm$, $(\langle M_h\cdot,\cdot\rangle_h)^{1/2}$, then we know

$$\| E_i \|_{M_h} = \min_{p_j \in P_j^0} \| p_j(CA)E_0 \|_{M_h} \leq \min_{p_j \in P_j^0} \| p_j(CA) \|_{M_h} \| E_0 \|_{M_h}, \tag{3.2}$$

where $P_j^0 = \{\text{polynomials of degree} \leq j,\ p(0) = 1\}$. If CA is M_h selfadjoint, which is the case in our experiments, then

$$\min_{p_j \in P_j^0} \| p_j(CA) \|_{M_h} = \min_{p_j \in P_j^0} \max_{\lambda \in \Sigma(CA)} | p_j(\lambda) |, \tag{3.3}$$

where $\Sigma(CA)$ is the spectrum of CA. If $\Sigma(CA)$ is assumed to fill the interval $[\lambda_{\min}, \lambda_{\max}]$, then p_j can be replaced in (3.3) by the scaled and translated Chebyshev polynomial based upon $[\lambda_{\min}, \lambda_{\max}]$ to yield the well-known bound

$$(3.4) \qquad \frac{\| E_j \|_{M_h}}{\| E_o \|_{M_h}} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^j ,$$

where $\kappa = (\lambda_{\max}/\lambda_{\min}) = C_{M_h}(CA)$.

In the experiments in §2 we have $M_h = I$ for left preconditioning and

$$(3.5) \qquad \kappa_\ell = C_h((B_h^{-1}A_h)^*(B_h^{-1}A_h)) = [C_h(B_h^{-1}A_h)]^2,$$

while for right preconditioning we have $M_h = A_h^*A_h$ and

$$(3.6) \qquad \kappa_r = C_{A_h^*A_h}((B_h^*B_h)^{-1}(A_h^*A_h)) = [C_h(A_hB_h^{-1})]^2.$$

When the "wrong" boundary conditions are used we have $\kappa = 0(h^{-2})$ for either left and right preconditioning. For κ large, the number of iterations required to reduce the *bound* in (3.4) by a factor of ϵ is approximately

$$(3.7) \qquad j \cong \frac{\log(2) - \log(\epsilon)}{2}\sqrt{\kappa} \ .$$

For $\epsilon = 10^{-9}$ this yields

$$(3.8) \qquad j \cong 10.7\sqrt{\kappa}.$$

A quick glance at Table 2.1 reveals that the required number of iterations is a small fraction of the number suggested by (3.8). Note that the estimated condition number is equal to $\sqrt{\kappa}$ (see (3.5), (3.6)). Clearly, the behavior of the conjugate gradient method is not determined by the worst case analysis but is determined by the distribution of the eigenvalues of the preconditioned matrix CA. For the left preconditioned normal equations these are squares of the singular values of $B_h^{-1}A_h$ while for the right preconditioned normal equation these are the squares of the singular values of $A_hB_h^{-1}$.

We will analyze the behavior of these singular values. Given the differential operator A suppose we choose $B^{(1)}$ and $B^{(2)}$ to be two operators that differ only in the boundary conditions at $x = 1$. Suppose

$$(3.9) \qquad (\beta^{(1)})^* = \alpha^*,$$

while

$$(3.10) \qquad \beta^{(2)} = \alpha.$$

Let us analyze the singular values of $(B_h^{(2)})^{-1}A_h$ in terms of the singular values of $(B_h^{(1)})^{-1}A_h$. The theory of [MP] asserts that there are two positive constants $0 < \Lambda_0 < \Lambda_1$ such that

$$(3.11) \qquad 0 < \Lambda_0 \leq \sigma_j \leq \Lambda_1, \qquad j = 1, 2, \cdots, N,$$

where the σ_j are the singular values of

$$(B_h^{(1)})^{-1}A_h.$$

Observe that

$$B_h^{(2)} = B_h^{(1)} + R_1, \tag{3.12}$$

where R_1 is an operator of rank 1. In the generic case, the Sherman–Morrison formula yields

$$(B_h^{(2)})^{-1} = (B_h^{(1)})^{-1} + T_1, \tag{3.13}$$

where T_1 is an operator of rank 1. Hence

$$(B_h^{(2)})^{-1}A_h = (B_h^{(1)})^{-1}A_h + T_1A_h \quad , \tag{3.14}$$

where T_1A_h is an operator of rank 1. Therefore

$$((B_h^2)^{-1}A_h)^*((B_h^{(2)})^{-1}A_h) = ((B_h^{(1)})^{-1}A_h)^*((B_h^{(1)})^{-1}A_h) + S_2, \tag{3.15}$$

where S_2 is an Hermitian operator of rank 2. Hence we have the following result.

LEMMA 3.1. *Let $\hat{\sigma}_1 \geq \hat{\sigma}_2 \geq, \cdots, \geq \hat{\sigma}_N > 0$ be the singular values of $(B_h^{(2)})^{-1}A_h$. Then, for $j \neq 1$ and $j \neq N$ we have*

$$0 < \Lambda_0 \leq \hat{\sigma}_j \leq \Lambda_1, \qquad j = 2, 3, \cdots, N-1. \tag{3.16}$$

And, while it is possible that for some fixed $h > 0$, $\hat{\sigma}_1$ and/or $\hat{\sigma}_N$ also satisfies (3.16), *there are constants $C_1, C_2, C_3, C_4 > 0$ such that*

$$C_1h^{-1/2} \leq \sigma_1 \leq C_2h^{-1/2}, \tag{3.17a}$$

$$C_3h^{1/2} \leq \sigma_N \leq C_4h^{1/2}. \tag{3.17b}$$

Proof. The inequalities (3.16) follow from the remarks above and (3.11). The estimates (3.17) follow from the results of [MP]. □

Suppose h is fixed and the conjugate gradient method PCGNE applied to

$$Q_h^{(1)} = ((B_h^{(1)})^{-1}A_h)^*((B_h^{(1)})^{-1}A_h) \tag{3.18}$$

requires n_1 iterations to reduce the ℓ_2-norm of the error by a factor of ϵ. (PCGNE is optimal in the ℓ_2-norm.) Let $p_{n_1}(\lambda)$ be the optimal polynomial generated by the iteration. Assume

$$\max_{\lambda\in[\Lambda_0^2,\Lambda_1^2]} \mid p_{n_1}(\lambda) \mid < \epsilon. \tag{3.19}$$

Of course, this need not be true, but if either $\Lambda_1^2 - \Lambda_0^2$ is small or the eigenvalues of $Q_h^{(1)}$ fill the interval $[\Lambda_0^2, \Lambda_1^2]$, then (3.19) will be approximately correct. Consider the polynomial

$$q_{n_1+2}(\lambda) = \frac{(\lambda - \hat{\sigma}_1^2)(\lambda - \hat{\sigma}_N^2)}{\hat{\sigma}_1^2\hat{\sigma}_N^2}p_{n_1}(\lambda). \tag{3.20}$$

Clearly,

$$q_{n_1+2}(0) = 1, \tag{3.21a}$$

$$q_{n_1+2}(\hat{\sigma}_1^2) = q_{n_1+2}(\hat{\sigma}_N^2) = 0, \tag{3.21b}$$

$$\max_{\lambda \in [\Lambda_0^2, \Lambda_1^2]} | \, q_{n_1+2}(\lambda) \, | \leq \epsilon \frac{\Lambda_1^2}{\hat{\sigma}_N^2}. \tag{3.21c}$$

Thus, $n_1 + 2$ iterations of PCGNE applied to

$$Q_h^{(2)} = ((B_h^{(2)})^{-1} A_h)^* ((B_h^{(2)})^{-1} A_h) \tag{3.22}$$

yields

$$\| \, E_{n_1+2}^{(2)} \, \|_h \leq \| \, q_{n_1+2}(Q_h^{(2)}) \, \|_h \leq \epsilon \frac{\Lambda_1^2}{\hat{\sigma}_N^2} \leq \epsilon \frac{\Lambda_1^2}{C_3^2} h^{-1}. \tag{3.23}$$

Now, let r be an integer and observe that

$$\frac{\| \, E_{r(n_1+2)}^{(2)} \, \|_h}{\| \, E_0^{(2)} \, \|_h} \leq \left[\epsilon \frac{\Lambda_1^2}{C_3^2} h^{-1} \right]^r. \tag{3.24}$$

Thus, the number of steps of PCGNE required to reduce the ℓ_2-norm of the error, $\| \, E_i^{(2)} \, \|_h$, by a factor of ϵ can be approximated by setting

$$\left[\epsilon \frac{\Lambda_1^2}{C_3^2} h^{-1} \right]^r = \epsilon, \tag{3.25}$$

and solving for r. This yields

$$r = \frac{\log(\epsilon)}{\log(\epsilon) + 2\log(\Lambda_1) - 2\log(C_3) - \log(h)}. \tag{3.26}$$

In our experiments in §2 we have $\Lambda_1 / C_3 \cong (1 + a_1)/\sqrt{a_1}$, $\epsilon = 10^{-9}$, $h = 2^{-k}$. For $a_1 = 10$ this yields

$$r = \frac{1}{.88 - (.033)k}. \tag{3.27}$$

Table 3.1 displays the actual number of iterations and the number predicted by (3.27) for $h = 2^{-k}, k = 15 - 18$. The results for the "correct" preconditioning, $B_h^{(1)}$, are taken from the left preconditioning of Table 2.2, while the results for the "wrong" preconditioning, $B_h^{(2)}$, are taken from left preconditioning of Table 2.1. The actual numbers are surprisingly close to the predicted numbers.

TABLE 3.1
Predicted number of iterations.

1/h	Its(1)	Actual Its(2)	Predicted Its(2)
$2^{15} = 32768$	14	31	41
$2^{16} = 65536$	14	34	45
$2^{17} = 131072$	14	38	50
$2^{18} = 262144$	14	54	56

Similar arguments can be made for the right preconditioning using the "wrong" boundary conditions. The only difference is that, in this case, the M-norm is optimized where $M = A_h^* A_h$ and the roles of $B_h^{(1)}$ and $B_h^{(2)}$ are reversed, with $B_h^{(2)}$ being the "correct" preconditioning.

4. Further computational results. The analysis of §3 explains why the results of the one-dimensional experiments with the "wrong" boundary conditions are "not so bad" relative to the results for the cases where the "correct" boundary conditions were used. However, it is not yet clear why the results are so good in the case when the "correct" boundary condition is used. For this reason further computational experiments were undertaken. Section 5 will provide an analysis of these results. Recall that the operator A is given by

$$Au = -(a_2(x)u')' + a_1 u' + a_0 u, \tag{4.1a}$$

$$u(0) = 0, \qquad u'(1) = 0, \tag{4.1b}$$

where a_1 and a_0 are constants and $a_2(x)$ may be constant or may be variable. The preconditioners were

$$B^{(1)}v = -v'' + b_0 v, \tag{4.2a}$$

$$v(0) = 0, \qquad v'(1) = -\frac{a_1}{a_2(1)} v(1), \tag{4.2b}$$

and

$$B^{(2)}v = -v'' + b_0 v, \tag{4.3a}$$

$$v(0) = 0, \qquad v'(1) = 0. \tag{4.3b}$$

Here the discretization, A_h, was obtained essentially as the discretization described in §2 with one small change. The boundary condition at $x = 1$ was treated a bit more accurately by choosing

$$h = \frac{2}{(2N+1)}. \tag{4.4a}$$

Then

$$Nh = \frac{2N}{2N+1} = 1 - \frac{1}{2}h, \tag{4.4b}$$

$$(N+1)h = \frac{2(N+1)}{2N+1} = 1 + \frac{1}{2}h. \tag{4.4c}$$

The boundary condition is then expressed as (2.6b), as before.

The operators $(B_h^{(j)})^{-1}$ were given by one cycle multigrid iterations for the solution of

$$B_h^{(j)} V = f.$$

For technical reasons, the choice of (4.4) rather than (2.3) suggested that the multigrid coarsening factor be 3 rather than 2.

While we have done many experiments, for our present purposes we will limit ourselves to three particular operators A.

Case 1.

$$Au = -u'' + 8u' \tag{4.5a}$$

with boundary conditions

$$u(0) = 0, \qquad u'(1) = 0. \tag{4.5b}$$

In this case *all the eigenvalues of A are real and positive.*

The preconditioners chosen are given by

$$B^{(1)}v = -v'' \tag{4.6a}$$

with boundary conditions

$$v(0) = 0, \qquad v'(1) = -8v(1), \tag{4.6b}$$

and

$$B^{(2)}v = -v'' \tag{4.7a}$$

with boundary conditions

$$v(0) = 0, \qquad v'(1) = 0. \tag{4.7b}$$

The theory of [MP] asserts that

$$\|(B_h^{(1)})^{-1}A_h\|_h \le C, \qquad \|A_h^{-1}B_h^{(1)}\|_h \le C \tag{4.8a}$$

and

$$\|(B_h^{(2)})^{-1}A_h\|_h \sim Ch^{-1/2}, \qquad \|A_h^{-1}B_h^{(2)}\|_h \sim Ch^{-1/2}. \tag{4.8b}$$

The results for this case are summarized in Tables 4.1 and 4.2 and Figs. 4.3 and 4.4.

TABLE 4.1

Singular values of $(B_h^{(1)})^{-1}A_h$.

N	$C((B_h^{(1)})^{-1}A_h)$	$\sigma^{(1)}(N)$	$\sigma^{(1)}(N-1)$	$\sigma^{(1)}(1)$
40	6.1493	0.4430	0.9189	2.7239
121	6.3406	0.4339	0.8935	2.7514
364	6.3488	0.4345	0.8901	2.7587
769	6.3438	0.4351	0.8897	2.7604

TABLE 4.2
Singular values of $(B_h^{(2)})^{-1}A_h$.

N	$C((B_h^{(2)})^{-1}A_h)$	$\sigma^{(2)}(N)$	$\sigma^{(2)}(N-1)$	$\sigma^{(2)}(1)$
40	72.416	0.4138	0.6798	29.967
121	158.70	0.3231	0.5218	51.274
364	434.78	0.2033	0.4894	88.397
769	900.73	0.1424	0.4842	128.27

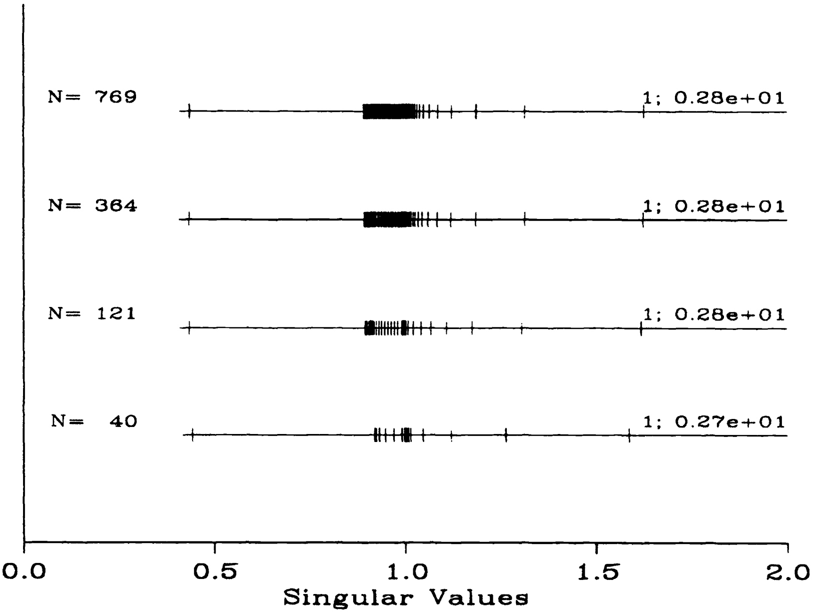


FIG. 4.3. *Singular value distribution of* $(B_h^{(1)})^{-1}A_h$.

In all the figures of this section, the numbers "$j; num$" on the right of the lines are to be read as follows:

$$j = \text{number of singular values larger than 2;}$$
$$num = \text{the largest singular value.}$$

We see that $\sigma^{(1)}(N)$ and $\sigma^{(1)}(1)$ remained fixed as N increases, while $\sigma^{(2)}(N)$ decreases and $\sigma^{(2)}(1)$ increases as N increases. The interior singular values for both preconditionings cluster about 1.0.

Case 2. In this case we introduce a nonconstant $a_2(x)$. Let

$$Au = -(a_2(x)u')' + 8u' \tag{4.9a}$$

with boundary conditions

$$u(0) = 0, \qquad u'(1) = 0, \tag{4.9b}$$

where

$$a_2(x) = 1 + \frac{1}{2}\sin 3\pi x. \tag{4.9c}$$

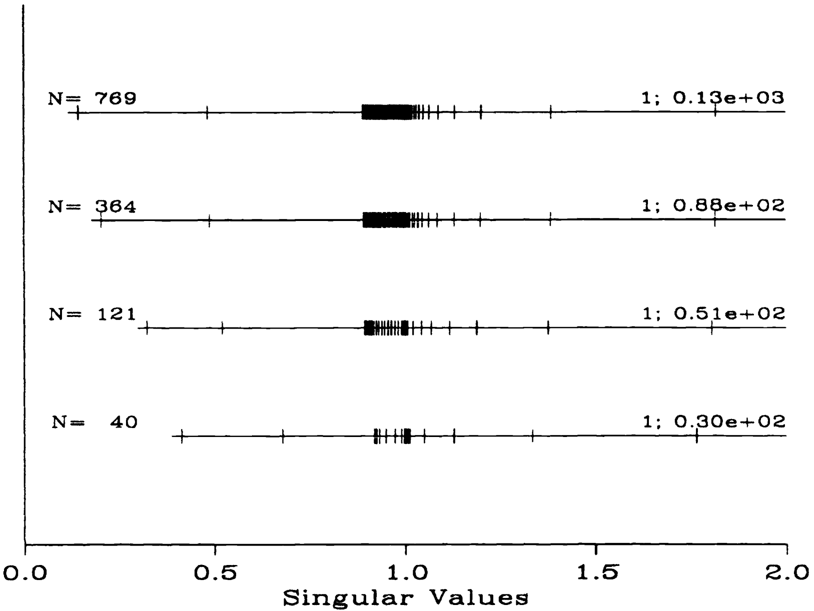


FIG. 4.4. *Singular value distribution of* $(B_h^{(2)})^{-1}A_h$.

As in Case 1, *all the eigenvalues of A are real and positive.* The preconditioners are again $B^{(1)}$ and $B^{(2)}$ as described by (4.6) and (4.7), respectively. Again the theory of [MP] yields (4.8a) and (4.8b). The results for this case are summarized in Tables 4.5 and 4.6 and Figs. 4.7 and 4.8.

TABLE 4.5
Singular values of $(B_h^{(1)})^{-1}A_h$.

N	$C((B_h^{(1)})^{-1}A_h)$	$\sigma^{(1)}(N)$	$\sigma^{(1)}(N-1)$	$\sigma^{(1)}(1)$
40	6.8411	0.4300	0.5008	2.9420
121	7.5987	0.3958	0.4611	3.0078
364	7.7331	0.3914	0.4504	3.0270
769	7.7416	0.3916	0.4476	3.0317

TABLE 4.6
Singular values of $(B_h^{(2)})^{-1}A_h$.

N	$C((B_h^{(2)})^{-1}A_h)$	$\sigma^{(2)}(N)$	$\sigma^{(2)}(N-1)$	$\sigma^{(2)}(1)$
40	75.589	0.3977	0.5008	30.067
121	174.82	0.2936	0.4611	51.325
364	480.02	0.1842	0.4505	88.425
769	994.98	0.1289	0.4476	128.29

The extreme singular values behave as in Case 1. Here, however, the interior singular values become dense in the interval $[.5, 1.5]$.

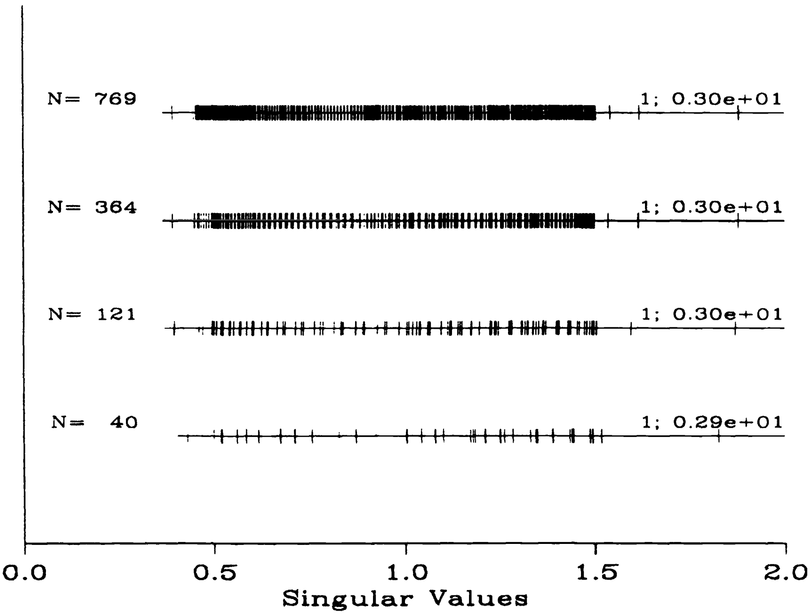


FIG. 4.7. *Singular value distribution of* $(B_h^{(1)})^{-1}A_h$.

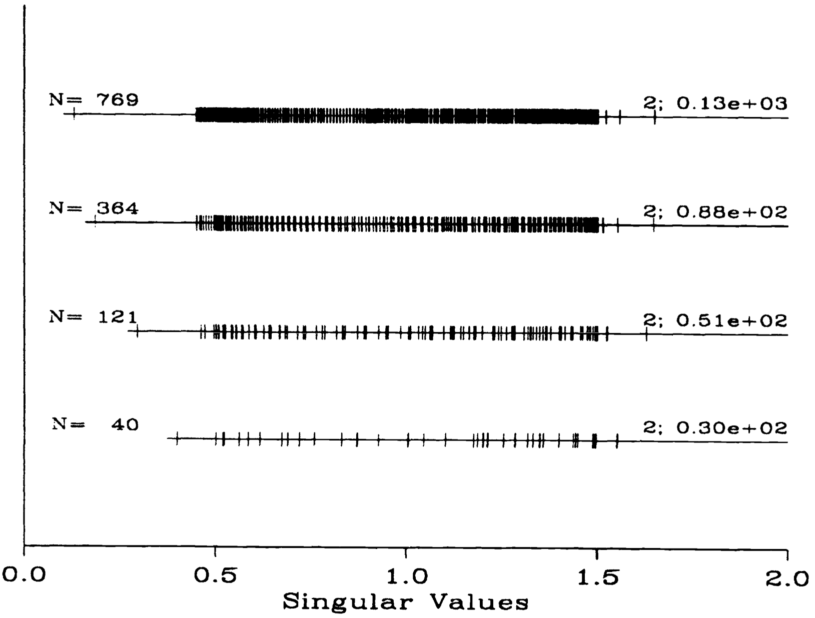


FIG. 4.8. *Singular value distribution of* $(B_h^{(2)})^{-1}A_h$.

Case 3. In this case we examine an indefinite operator. Let

$$Au = -u'' + 8u' - (6.5)\pi^2 u, \tag{4.10a}$$

with boundary conditions

$$u(0) = 0, \qquad u'(1) = 0. \tag{4.10b}$$

Two eigenvalues of A are negative, while all the others are positive. The preconditioners are again $B^{(1)}$ and $B^{(2)}$ as described by (4.6) and (4.7), respectively. Once more, the theory of [MP] yields (4.8a) and (4.8b). The results for this case are summarized in Tables 4.9 and 4.10 and Figs. 4.11 and 4.12.

TABLE 4.9
Singular values of $(B_h^{(1)})^{-1}A_h$.

N	$C((B_h^{(1)})^{-1}A_h)$	$\sigma^{(1)}(N)$	$\sigma^{(1)}(N-1)$	$\sigma^{(1)}(1)$
40	377.46	0.0203	0.8299	7.6628
121	369.12	0.0208	0.8806	7.6898
364	367.70	0.0209	0.8882	7.6945
769	367.50	0.0209	0.8892	7.6954

TABLE 4.10
Singular values of $(B_h^{(2)})^{-1}A_h$.

N	$C((B_h^{(2)})^{-1}A_h)$	$\sigma^{(2)}(N)$	$\sigma^{(2)}(N-1)$	$\sigma^{(2)}(1)$
40	1805.8	0.0203	0.6958	36.662
121	2661.4	0.0208	0.4721	55.443
364	4343.1	0.0209	0.2884	90.875
769	6209.3	0.0209	0.2016	129.99

Observe that

$$\sigma^{(1)}(N) \approx \sigma^{(2)}(N) = \text{constant}$$

but $\sigma^{(2)}(N-1) << \sigma^{(1)}(N-1)$ and is steadily decreasing as a function of N. (This is easily seen from Fig. 4.11.) So, one would expect that for N large enough

$$\sigma^{(2)}(N) < \sigma^{(1)}(N)$$

and $\sigma^{(2)}(N)$ would continue to decrease as a function of N, while $\sigma^{(1)}(N)$ remains constant. Indeed, this is the result predicted by [MP]. Here, as in Case 1, the remaining singular values cluster about 1.0.

5. Distribution of the singular values.

In this section we assume that

$$\alpha^* = \beta^* \tag{5.1}$$

and consider the distribution of the singular values $0 < \sigma_N \leq \sigma_{N-1} \leq, \cdots, \leq \sigma_1$ of $B_h^{-1}A_h$. The theory developed in [MP] asserts only that there exists a pair of fixed numbers $0 < \Lambda_0(L) < \Lambda_0(U)$, independent of h such that

$$0 < \Lambda_0(L) \leq \sigma_N \leq \sigma_j \leq \sigma_1 \leq \Lambda_0(U). \tag{5.2}$$

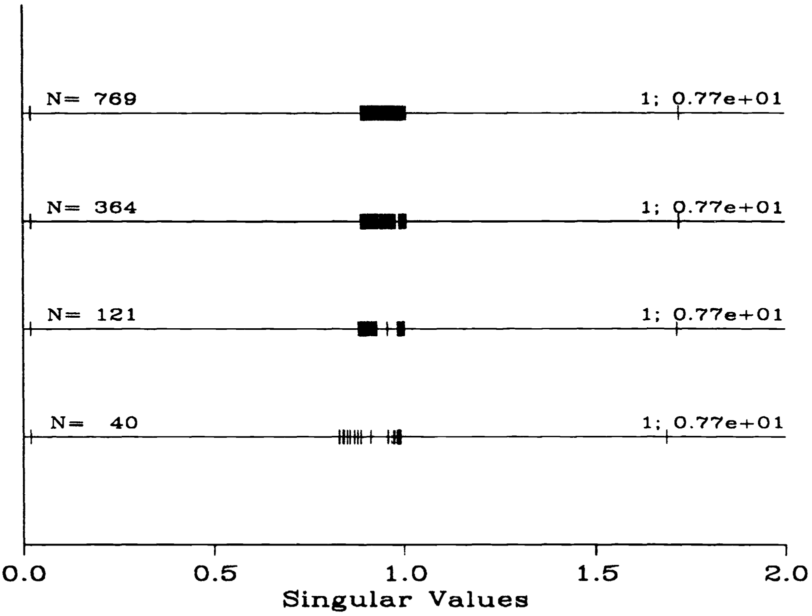


FIG. 4.11. *Singular value distribution of* $(B_h^{(1)})^{-1}A_h$.

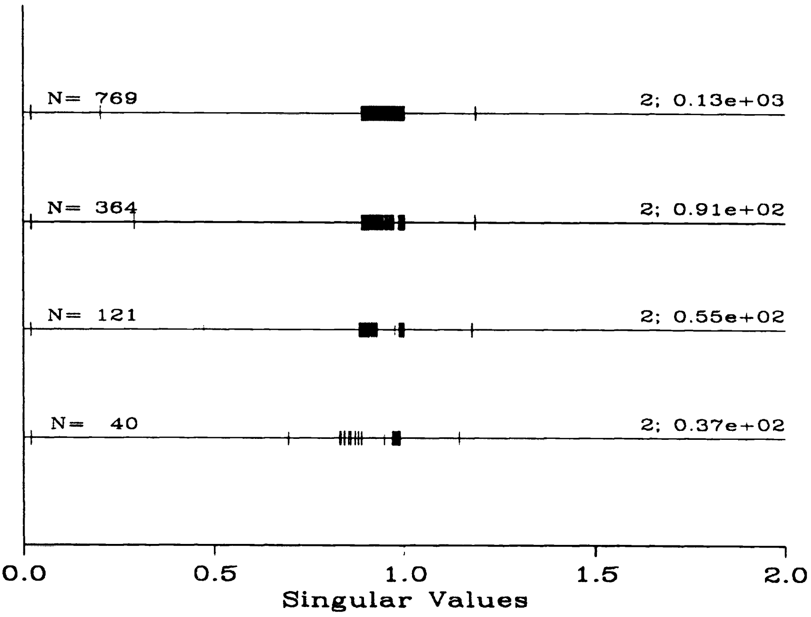


FIG. 4.12. *Singular value distribution of* $(B_h^{(2)})^{-1}A_h$.

The development in this section gives a qualitative explanation of the detailed results on the σ_j seen in §4.

Let A and B be the operators described by (2.1) and (2.2) subject to (5.1). Let u, v satisfy

$$B^{-1}Au = v, \tag{5.3a}$$

that is

$$Au = Bv. \tag{5.3b}$$

We show that

$$v(x) = \frac{a_2(x)}{b_2(x)}u(x) + Ku \tag{5.4a}$$

and the operator K is a linear compact operator. That is,

$$B^{-1}A = \frac{a_2(x)}{b_2(x)}I + K$$

and

$$Q = (B^{-1}A)^*B^{-1}A = \left[\frac{a_2(x)}{b_2(x)}\right]^2 I + \tilde{K}_1, \tag{5.4b}$$

where $\tilde{K}_1$ is another compact operator.

Next, we show that

$$Q_h = (B_h^{-1}A_h)^*(B_h^{-1}A_h) \tag{5.5}$$

is a consistent and convergent approximation to Q. Once this has been done it is an easy matter to prove the main result of this section.

THEOREM 5.1. *Let A, B be given by* (2.1) *and* (2.2), *respectively. Assume that* (5.1) *holds. Let A_h and B_h be constructed as in* (2.5). *Let*

$$m = \min \frac{a_2(x)}{b_2(x)}, \tag{5.6a}$$

$$M = \max \frac{a_2(x)}{b_2(x)}. \tag{5.6b}$$

Let $\epsilon > 0$ be given ($\epsilon < \frac{1}{2}m^2$ is not essential, but is a reasonable additional condition). Then there is an $h_0 > 0$ and an integer $\tilde{n} = \tilde{n}(\epsilon)$ such that for all $h \leq h_0$ at most $\tilde{n}$ of the squared singular values σ_j^2 are outside the interval $[m^2 - \epsilon, M^2 + \epsilon]$.

We return to the proof of (5.4a). From (5.3b) we have

$$-(a_2u')' + a_1u' + a_0u = -(b_2v')' + b_1v' + b_0v. \tag{5.7}$$

We remind the reader that the results of [MP] assert that there is a constant, say, C_0, such that

$$\|v\|_{L_2} \leq C_0\|u\|_{L_2}. \tag{5.8}$$

From (5.7) we obtain

$$(5.9)\qquad -a_2u' + \int_0^x a_1u'dt + \int_0^x a_0udt = (-b_2v') + \int_0^x b_1v'dt + \int_0^x b_0vdt + C_1,$$

where C_1 is a constant depending on u (since we are assuming that u is given and v is determined from (5.3a)).

Integration by parts yields

$$(5.10)\qquad -a_2u' + a_1u + \int_0^x (a_0 - a_1')udt = -b_2v' + b_1v + \int_0^x (b_0 - b_1')vdt + C_1.$$

We observe that the two integrals in (5.10) are compact linear operators applied to u. This is immediately apparent for the integral on the left-hand side of (5.10). However, it is also true of the integral on the right-hand side of (5.10) because (i) (5.3a) shows that v is a linear function of u, and (ii) (5.8) shows that v is a bounded function of u. We shall use the symbol $I(u)$ to denote any such term that is a compact linear operator applied to u. With this understanding, (5.10) can be rewritten as

$$(5.11)\qquad -a_2u' + a_1u + b_2v' - b_1v + I(u) = C_1.$$

Integration and integration by parts now yield

$$-a_2u + b_2v + \int_0^x [(a_2' + a_1)u - (b_2' + b_1)v]dt + \int_0^x I(u)dt = C_1x.$$

That is,

$$(5.12)\qquad v = \frac{a_2}{b_2}u + I(u) + \frac{1}{b_2}C_1x.$$

Since (5.8) must be true, we see that C_1 must be a bounded functional of u. Indeed, a bounded linear functional of u. Hence, $C_1(u)$ is also a compact linear operator. Therefore, we have

$$(5.13)\qquad v = \frac{a_2}{b_2}u + I(u).$$

This is the first goal enunciated at the beginning of this section.

Remark. In the special case where

$$b_1 = b_0 = 0$$

it is easy to obtain an explicit representation of $C_1(u)$.

Consider the difference equations

$$(5.14a)\qquad B_h^{-1}A_hU = V,$$

$$(5.14b)\qquad A_hU = B_hV.$$

A similar computation, based on summation by parts rather than integration by parts, yields

$$V_j = \frac{a_2(x_j - \frac{h}{2})}{b_2(x_j - \frac{h}{2})} U_j + K_h U, \tag{5.15}$$

where $K_h U$ is a discrete analog of the compact operator $K(u)$.

In addition to being a discretization of $K(u)$, these discrete operators are also uniformly compact in the sense that there is a constant C independent of h such that if U is a grid vector and $U(x,h)$ is the associated piecewise linear function, and if

$$K_h U = W, \tag{5.16a}$$

then

$$\|W(\cdot,h)\|_{H_1} \leq C\|U(\cdot,h)\|_{L_2}. \tag{5.16b}$$

Under these circumstances it is very easy to prove Lemma 5.1.

LEMMA 5.1. *Consider the operator*

$$(B^{-1}A)^*(B^{-1}A) = \left(\frac{a_2}{b_2}\right)^2 I + K^* \left(\frac{a_2}{b_2}\right) + \left(\frac{a_2}{b_2}\right) K + K^*K, \tag{5.17}$$

where K *is the operator of* (5.4a), *or, using the notation of* (5.13),

$$Ku = I(u).$$

Then

$$\tilde{K} = K^* \left(\frac{a_2}{b_2}\right) + \left(\frac{a_2}{b_2}\right) K + K^*K \tag{5.18a}$$

is a compact operator. Moreover, the discrete operators

$$\tilde{K}_h = K_h^* \left(\frac{a_2}{b_2}\right)_{j-1/2} + \left(\frac{a_2}{b_2}\right)_{j-1/2} K_h + K_h^* K_h \tag{5.18b}$$

are uniformly compact and

$$\tilde{K}_h \to \tilde{K}. \tag{5.19}$$

In particular, let λ_j *be the eigenvalues of* $\tilde{K}$ *arranged so that*

$$|\lambda_1| \geq |\lambda_2| \geq \cdots \tag{5.20a}$$

and let $\mu_j(h)$ *be the eigenvalues of* $\tilde{K}_h$ *arranged so that*

$$|\mu_1(h)| \geq |\mu_2(h)| \geq \cdots |\mu_k(h)| \geq 0. \tag{5.20b}$$

Then, given a fixed J *and an* ϵ, *there is an* $h_0 > 0$ *such that, for all* $h \leq h_0$, *we have*

$$|\lambda_j - \mu_j(h)| \leq \epsilon, \qquad j = 1, 2, \cdots, J. \tag{5.21}$$

Proof. We omit the proof, which is a standard compactness argument. □

We are now able to prove our main result.

Proof of Theorem 5.1. Let $\epsilon > 0$ be given. Choose J so large that $|\lambda_J| < \frac{\epsilon}{2}$. Choose h_0 so small that

$$|\lambda_j - \mu_j(h)| \le \frac{\epsilon}{4}, \qquad j = 1, 2, \cdots, J; \qquad h \le h_0. \tag{5.22}$$

Then the eigenvalue values $q_j(h)$ of Q_h are characterized by

$$q_j(h) = \max_{S_j} \min_{U \neq 0} \left\{ \frac{U^* Q_h U}{U^* U}, \quad U \in S_j \right\}, \tag{5.23}$$

where dim $S_j = j$. Hence, see [ST, Thm. 5.10],

$$m^2 - \mu_j \le \sigma_j^2 \le M^2 + \mu_j. \tag{5.24}$$

Hence, for $j > J$ we have

$$m^2 - \left(\frac{\epsilon}{2} + \frac{\epsilon}{4}\right) \le \sigma_j^2 \le M^2 + \left(\frac{\epsilon}{2} + \frac{\epsilon}{4}\right).$$

Thus, the theorem is proven. □

With Theorem 5.1 in hand, consider the numerical results of §4. In Case 1 and Case 3 we have $a_2(x) = b_2(x) = 1$. Thus,

$$((B^{(1)})^{-1}A)^*((B^{(1)})^{-1}A) = I + \tilde{K}. \tag{5.25}$$

Figures 4.1 and 4.5 show that the singular values of the discrete operators cluster about 1.0. In Case 2, we have $a_2(x) = 1 + \frac{1}{2}\sin(3\pi x)$ and $b_2(x) = 1$, which yields

$$((B^{(1)})^{-1}A)^*((B^{(1)})^{-1}A) = a_2(x)I + \tilde{K}. \tag{5.26}$$

The spectrum of the multiplication operator $a_2(x)I$ fills the interval $[.5, 1.5]$. In Fig. 4.3 we see that the singular values of the discrete operators fill the interval $[.5, 1.5]$ plus a few isolated values as the theory predicts.

6. Two-dimensional computational results. In this section, we describe computational results on two-dimensional problems. An analysis of these results will appear in §7. Our experiments use centered difference approximations to the operators

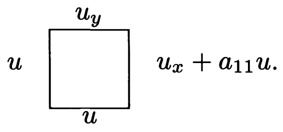
$$Au = -\Delta u + a_{11}u_x + a_{12}u_y, \tag{6.1}$$

$$Bu = -\Delta u, \tag{6.2}$$

for $(x, y) \in [0, 1] \times [0, 1]$. To make notation easier we will denote the boundary conditions schematically. For example, the boundary conditions

(6.3)

$u_y(x, 1) = 0$

$u(0, y) = 0, \qquad u_x(1, y) + a_{11}u(1, y) = 0$ become 

$u(x, 0) = 0$

Again, the conjugate gradient method PCGNE was used on the left preconditioned normal equations with iteration matrix

$$Q_h = (B_h^{-1}A_h)^*(B_h^{-1}A_h). \tag{6.4}$$

For right preconditioning, the conjugate gradient method PCGNR was applied to the iteration matrix

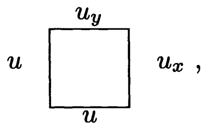
$$\tilde{Q}_h = (B_h^*B_h)^{-1}(A_h^*A_h). \tag{6.5}$$

For A and B selfadjoint, symmetric preconditioning was implemented by using PCG on the iteration matrix

$$G_h = B_h^{-1}A_h. \tag{6.6}$$

Inversion of B_h was accomplished by using a Fast Poisson solver.

Case (i). $D(A) = D(B)$. Let

(6.7a) $Au = -\Delta u + a_{11}u_x + a_{12}u_y,$ 

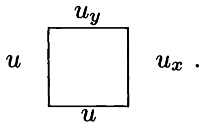
(6.7b) $Bu = -\Delta u,$ 

Table 6.1 shows the results for $a_{11} = a_{12} = 1$, while Table 6.2 shows the results for $a_{11} = a_{12} = 10$. Notice that $C_h(B_h^{-1}A_h)$ grows like $0(h^{-1})$, while $C_h(A_hB_h^{-1})$ is independent of h. Also, notice that this rate is only achieved for the smallest values of h that we were able to compute.

Here "Its" reflects the number of iterations to reduce the ℓ_2-norm of the error for left preconditioning and the ℓ_2-norm of the residual for right preconditioning by a factor of 10^{-6}.

TABLE 6.1
$2 - D, D(A) = D(B), a_{11} = a_{12} = 1.0.$

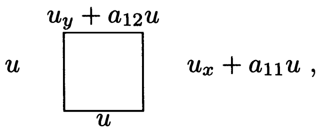
	Left preconditioning			Right preconditioning		
$1/h$	$C_h(B_h^{-1}A_h)$	Growth rate	Its	$C_h(A_hB_h^{-1})$	Growth rate	Its
4	.1511+01	.0000+00	6	.1346+01	.0000+00	5
8	.2115+01	.4855+00	7	.1405+01	.6756−01	6
16	.3044+01	.5251+00	8	.1436+01	.3010−01	5
32	.4709+01	.6291+00	9	.1451+01	.1542−01	5
64	.7890+01	.7445+00	12	.1459+01	.7655−02	5
128	.1413+02	.8409+00	14	.1429+01	−.2964−01	4
256	.2654+02	.9093+00	17	.1379+01	−.5170−01	4
512	.5131+02	.9510+00	22	.1465+01	.8720−01	4

TABLE 6.2
$2 - D, D(A) = D(B), a_{11} = a_{12} = 10.0.$

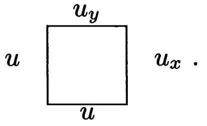
	Left preconditioning			Right preconditioning		
$1/h$	$C_h(B_h^{-1}A_h)$	Growth rate	Its	$C_h(A_hB_h^{-1})$	Growth rate	Its
4	.5736+01	.0000+00	10	.4446+01	.0000+00	9
8	.1216+02	.1084+01	15	.5449+01	.2934+00	14
16	.2356+02	.9536+00	21	.5813+01	.9319−01	14
32	.4098+02	.7985+00	28	.5994+01	.4416−01	14
64	.6953+02	.7626+00	37	.6082+01	.2123−01	13
128	.1237+03	.8319+00	54	.6127+01	.1048−01	13
256	.2379+03	.9427+00	75	.6149+01	.5112−02	12
512	.4663+03	.9078+01	100	.6159+01	.2515−02	11

Case (ii). $D(A^*) = D(B^*)$. Let

(6.8a) $$Au = -\Delta u - a_{11}u_x - a_{12}u_y,$$



(6.8b) $$Bu = -\Delta u,$$



Notice that the A here is the adjoint of the A in Case (i). The operator A was changed rather than B so as to retain the easy-inversion of B_h by a Fast Poisson Solver. Table 6.3 shows the results for $a_{11} = a_{12} = 1$, while Table 6.4 shows the results for $a_{11} = a_{12} = 10$. Now, $C_h(B_h^{-1}A_h)$ is bounded, while $C_h(A_hB_h^{-1})$ grows like h^{-1}. Again, notice that the number of iterations using the "wrong" preconditioning grows very slowly.

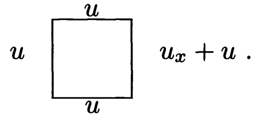
TABLE 6.3
$2 - D, D(A^*) = D(B^*), a_{11} = a_{12} = 1.0.$

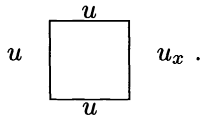
	Left preconditioning			Right preconditioning		
$1/h$	$C_h(B_h^{-1}A_h)$	Growth rate	Its	$C_h(A_hB_h^{-1})$	Growth rate	Its
4	.1314+01	.0000+00	6	.1648+01	.0000+00	6
8	.1393+01	.8334−01	6	.2088+01	.3414+00	7
16	.1421+01	.2909−01	5	.3074+01	.5581+00	8
32	.1444+01	.2285−01	5	.4720+01	.6186+00	9
64	.1455+01	.1144−01	5	.7894+01	.7419+00	10
128	.1461+01	.5677−02	5	.1409+02	.8358+00	13
256	.1464+01	.2826−02	5	.2646+02	.9093+00	16
512	.1465+01	.1408−02	5	.4973+02	.9101+00	21

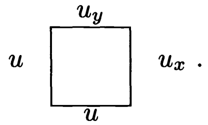
TABLE 6.4
$2-D, D(A^*) = D(B^*), a_{11} = a_{12} = 10.0.$

	Left preconditioning			Right preconditioning		
$1/h$	$C_h(B_h^{-1}A_h)$	Growth rate	Its	$C_h(A_hB_h^{-1})$	Growth rate	Its
4	.5571+01	.0000+00	10	.9392+01	.0000+00	10
8	.6087+01	.1277+00	15	.2186+02	.1219+01	23
16	.5979+01	−.2591−01	15	.3251+02	.5724+00	26
32	.6002+01	.5445−02	15	.4802+02	.5626+00	29
64	.6047+01	.1092−01	14	.7248+02	.5939+00	35
128	.6105+01	.1383−01	14	.1220+03	.7510+00	47
256	.6138+01	.7602−02	14	.2318+03	.9260+00	68
512	.6154+01	.3809−02	13	.4532+03	.9671+00	100

Case (iii). A and B selfadjoint, $D(A) \neq D(B)$. Let

(6.9a) $Au = -\Delta u,$ 

(6.9b) $B^{(1)}u = -\Delta u,$ 

(6.9c) $B^{(2)}u = -\Delta u,$ 

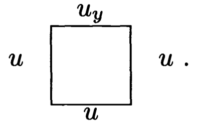
(6.9d) $B^{(3)}u = -\Delta u,$ 

Table 6.5 shows the results for both left and symmetric preconditioning with $(B_h^{(1)})^{-1}$. Although $D(A) \neq D(B)$, both A and $B^{(1)}$ have Dirichlet boundary conditions on the same portion of the boundary. The theory in [MP] says $C_h(B_h^{-1}A_h)$ should be $0(h^{-1})$, while $C_h(B_h^{-1/2}A_hB_h^{-1/2})$ should be bounded. These bounds are supported by the results in Table 6.5.

TABLE 6.5
$2-D$, A and B selfadjoint, $D(A) \neq D(B)$, same Dirichlet part.

	Left preconditioning			Symmetric preconditioning		
$1/h$	$C_h(B_h^{-1}A_h)$	Growth rate	Its	$C_h(B_h^{-1/2}A_hB_h^{-1/2})$	Growth rate	Its
4	.1141+01	.0000+00	5	.1135+01	.0000+00	4
8	.1245+01	.1260+00	5	.1204+01	.8431−01	4
16	.1455+01	.2246+00	5	.1255+01	.5975−01	4
32	.1782+01	.2926+00	7	.1286+01	.3609−01	4
64	.2282+01	.3569+00	7	.1297+01	.1159−01	4
128	.3090+01	.4369+00	8	.1301+01	.4966−02	4
256	.4796+01	.6341+00	9	.1113+01	−.2250+00	3
512	.7924+01	.7242+00	11	.1249+01	.1657+00	3

In Table 6.6 the same results are presented for preconditioning with $(B_h^{(2)})^{-1}$. In this case A has Dirichlet boundary conditions on the top, while $B^{(2)}$ does not. Thus, as predicted, $C_h(B_h^{-1}A_h) = 0(h^{-2})$, while $C_h(B_h^{-1/2}A_hB_h^{-1/2}) = 0(h^{-1})$. The blank line for left preconditioning and $h = 2^{-9}$ indicates that the iteration did not converge in 3000 iterations.

TABLE 6.6
$2-D$, A and B selfadjoint, $D(A) \neq D(B)$, Dirichlet part of A includes Dirichlet part of B.

	Left preconditioning			Symmetric preconditioning		
$1/h$	$C_h(B_h^{-1}A_h)$	Growth rate	Its	$C_h(B_h^{-1/2}A_hB_h^{-1/2})$	Growth rate	Its
4	.4152+01	.0000+00	7	.3875+01	.0000+00	6
8	.1542+02	.1893+01	13	.8506+01	.1134+01	9
16	.6387+02	.2050+01	25	.1782+02	.1067+01	11
32	.2671+03	.2064+01	52	.3649+02	.1033+01	15
64	.1088+04	.2027+01	134	.7386+02	.1017+01	20
128	.4437+04	.2027+01	390	.1486+03	.1008+01	26
256	.1759+05	.1987+01	1221	.2980+03	.1004+01	34
512	—	—	—	.5970+03	.1002+01	43

Finally, Table 6.7 shows the results for preconditioning with $B^{(3)}$. Here A has Dirichlet boundary conditions where $B^{(3)}$ does not, and $B^{(3)}$ has Dirichlet boundary conditions where A does not. As predicted,

$$C_h(B_h^{-1}A_h) = 0(h^{-3}) \quad \text{while} \quad C_h(B_h^{-1/2}A_hB_h^{-1/2}) = 0(h^{-2}).$$

TABLE 6.7
2 − D, A and B selfadjoint, D(A) ≠ D(B), mixed Dirichlet part.

	Left preconditioning			Symmetric preconditioning		
$1/h$	$C_h(B_h^{-1}A_h)$	Growth rate	Its	$C_h(B_h^{-1/2}A_hB_h^{-1/2})$	Growth rate	Its
4	.4196+01	.0000+00	8	.3713+01	.0000+00	6
8	.2435+02	.2536+01	21	.1687+02	.2184+01	13
16	.1805+03	.2890+01	55	.7238+02	.2100+01	21
32	.1460+04	.3015+01	164	.3007+03	.2054+01	34
64	.1049+05	.2845+01	618	.1225+04	.2027+01	61
128	.4011+05	.1934+01	2936	.4923+04	.2005+01	110
256	—	—	—	.1972+05	.2002+01	184
512	—	—	—	.7883+05	.1998+01	328

7. Analysis of the two-dimensional computations. As we have seen, in the one-dimensional computational experiments described in §2, the conjugate gradient algorithm works better than might be expected even in those cases where the "wrong" boundary conditions are used. The reason for this is contained in the analysis of §3 and essentially comes about because

$$B_h^{(1)} = B_h^{(2)} + R_1, \tag{7.1}$$

where R_1 is an operator of rank one. In this section, we turn our attention to the two-dimensional computations and seek to explain their "not so bad" behavior.

The basic reasons for the efficiency of the two-dimensional examples in the cases where $D(A^*) = D(B^*)$, i.e., those cases where $C_{L_2}(B_h^{-1}A_h)$ is bounded independent of h, is essentially explained by the computations of §4 and the analysis of §5, see [GMP] for the two-dimensional extension of the results of §5. As for the case with "wrong" boundary conditions in two-dimensions, we consider a problem with $D((B^{(2)})^*) = D(A^*)$, $D(B^{(1)}) = D(A)$. And, as in all the examples of §6, we may analyze the problems by the method of "separation of variables." We have

$$B_h^{(1)} = B_h^{(2)} + R_t, \tag{7.2}$$

where we might expect R_t to be of rank $t = 0(1/h)$, the number of points next to the boundary. But, as we shall see, in fact the number, T, of unbounded singular values is (at most)

$$T = O\left(\frac{1}{h^{1/2}}\right),$$

a significantly smaller number.

Consider the problem with

$$\Omega = \{(x,y) \mid 0 < x, y < 1\}, \tag{7.3a}$$

$$Au = u_{xx} + u_{yy} + au_x, \tag{7.3b}$$

$$u = 0 \text{ on } x = 0, \quad y = 0, \quad y = 1, \tag{7.3c}$$

$$\frac{\partial u}{\partial x} = 0 \quad \text{on } x = 1. \tag{7.3d}$$

Let the preconditioner be

$$Bv = v_{xx} + v_{yy} \tag{7.4}$$

subject to the same boundary conditions as A. The discretizations A_h and B_h are obtained by straightforward central differences.

We turn immediately to the discrete problem

$$B_h^{-1} A_h U = V, \tag{7.5a}$$

$$A_h U = B_h V. \tag{7.5b}$$

We now apply an argument based on "separation of variables." Let

$$U = \sum_{j=1}^{N} \phi_j(x) \sin \pi j y, \tag{7.6a}$$

$$V = \sum_{j=1}^{N} \psi_j(x) \sin \pi j y, \tag{7.6b}$$

where

$$N = \frac{1}{h}. \tag{7.6c}$$

We find that the functions $\phi_j(x)$ and $\psi_j(x)$ are related by the equations (we drop the subscript j where possible)

$$\begin{gathered} \phi_{x\bar{x}} + a\phi_{\hat{x}} - \mu_j \phi = \psi_{x\bar{x}} - \mu_j \psi, \\ \phi(0) = 0, \qquad (\phi_x)_N = 0, \end{gathered} \tag{7.7a}$$

where

$$\mu_j = \frac{4 \sin^2(\pi j h/2)}{h^2}. \tag{7.7b}$$

The symbols ϕ_x,ψ_x are forward differences. That is,

$$(\phi_x)_k = \frac{\phi_{k+1} - \phi_k}{h}. \tag{7.7c}$$

The symbols $\phi_{x\bar{x}}$, $\psi_{x\bar{x}}$, $\phi_{\hat{x}}$, $\psi_{\hat{x}}$ are defined as central differences. That is,

$$(\phi_{x\bar{x}})_k = \frac{\phi_{k+1} - 2\phi_k + \phi_{k-1}}{h^2}, \tag{7.7d}$$

$$(\phi_{\hat{x}})_k = \frac{\phi_{k+1} - \phi_{k-1}}{2h}. \tag{7.7e}$$

Since

$$\frac{2}{\pi} \le \frac{\sin\theta}{\theta} \le 1, \qquad 0 \le \theta \le \frac{\pi}{2}$$

we have

$$4j^2 \le \mu_j \le \pi^2 j^2. \tag{7.8}$$

Our main results now follow easily.

LEMMA 7.1. *Let ϕ and ψ be connected by* (7.7). *Then*

$$\|\phi - \psi\|_h \le \frac{\|a\|_\infty}{2j^2 h}\|\phi\|_h. \tag{7.9}$$

Proof. We first observe that

$$\|\phi_{\hat{x}}\|_h \le \frac{2}{h}\|\phi\|_h. \tag{7.10}$$

Let $W = \psi - \phi$. Then W satisfies the equation

$$W_{x\bar{x}} - \mu_j W = a\phi_{\hat{x}}. \tag{7.11}$$

Multiplication by W and summation, together with a standard summation by parts argument, yields

$$\left|(-\|W_x\|_h^2 - \mu_j\|W\|_h^2)\right| \le \|a\|_\infty \frac{2}{h}\|\phi\|_h \cdot \|W\|_h, \tag{7.12}$$

thus,

$$4j^2\|W\|_h \le \frac{2\|a\|_\infty}{h}\|\phi\|_h,$$

which implies the lemma. □

LEMMA 7.2. *Let ϕ and ψ be connected by* (7.7). *Then*

$$\|\phi - \psi\|_h \le \frac{\|a\|_\infty}{4j^2}\left[\frac{2}{h}\|\psi\|_h + \frac{\|a\|_\infty}{4}\|\phi - \psi\|_h\right]. \tag{7.13}$$

Proof. Let

$$W = \phi - \psi.$$

Then W satisfies

$$W_{x\bar{x}} + aW_{\hat{x}} - \mu_j W = -a\psi_{\hat{x}}.$$

The summation by parts argument now yields

$$\|W_x\|_h^2 + \mu_j\|W\|_h^2 \le \|a\|_\infty\|W\|_h\left[\|\psi_{\hat{x}}\|_h + \|W_{\hat{x}}\|_h\right].$$

We apply the inequality

$$\|W\|_h \cdot \|W_{\hat{x}}\|_h \le \frac{1}{2}\left[\theta\|W_{\hat{x}}\|_h^2 + \frac{1}{\theta}\|W\|_h^2\right]$$

with $\theta = 2/\|a\|_\infty$. Then

$$\mu_j \|W\|_h^2 \le \|a\|_\infty \|\psi_{\bar{x}}\|_h \cdot \|W\|_h + \frac{\|a\|^2}{4}\|W\|_h^2, \tag{7.14}$$

which implies the lemma. □

This estimate contains $\|\phi - \psi\|_h$ on both sides of the inequality. However, the following corollary is immediate.

COROLLARY. *Suppose*

$$\frac{\|a\|_\infty^2}{8} \le j^2. \tag{7.15a}$$

Then

$$\|\phi - \psi\|_h \le \frac{\|a\|_\infty}{j^2 h}\|\psi\|_h. \tag{7.15b}$$

THEOREM 7.1. *Suppose ϕ and ψ are connected by (7.7). Suppose j is so large that*

$$\frac{\|a\|_\infty^2}{8} \le j^2 \quad \textit{and} \quad j \ge \sqrt{N}. \tag{7.16}$$

Then

$$\|\phi\|_h \le (1 + \|a\|_\infty)\|\psi\|_h, \tag{7.17a}$$

$$\|\psi\|_h \le (1 + \|a\|_\infty)\|\phi\|_h. \tag{7.17b}$$

Proof. Under these hypotheses we have both

$$\|\phi - \psi\|_h \le \frac{\|a\|_\infty}{j^2 h}\|\phi\|_h \le \|a\|_\infty \|\phi\|_h,$$

$$\|\phi - \psi\|_h \le \frac{\|a\|_\infty}{j^2 h}\|\psi\|_h \le \|a\|_\infty \|\psi\|_h.$$

Then the theorem follows from the triangle inequality, e.g.,

$$\|\psi\|_h = \|\psi - \phi + \phi\|_h \le \|\phi\|_h + \|\psi - \phi\|_h. \qquad \square$$

Consider the meaning of this theorem. For each j, $j = 1, 2, \cdots, N$ the separation of variables provides N singular values. However, if $j > \sqrt{N}$ those singular values are all bounded between

$$(1 + \|a\|_\infty)^{-1} \quad \text{and} \quad (1 + \|a\|_\infty).$$

Hence, there are *at most* $\sqrt{N}$ j's which can contribute singular values that tend toward ∞ or tend toward zero. As these are all one-dimensional problems, the analysis of §3 shows that for each such j there is exactly one singular value tending toward ∞ and exactly one singular value tending toward zero. In the worst case one singular value tends toward ∞ as $h^{-1/2}$ and one singular value tends toward zero as $h^{1/2}$. However, in this context many of the "bad" j's may produce singular values that approach 0 and

∞ at slower rates. Hence, the number of "bad" singular values is, at worst, $O(1/\sqrt{h})$ not $O(1/h)$, and the worst of these approach ∞ as $h^{-1/2}$ or 0 as $h^{1/2}$.

REFERENCES

[AMS] S. F. Ashby, T. A. Manteuffel, and P. E. Saylor, *A taxonomy for conjugate gradient methods*, SIAM J. Numer. Anal., 27 (1990), pp. 1542–1568.

[CGO] P. Concus, G. H. Golub, and D. P. O'Leary, *A generalized conjugate gradient iteration for the numerical solution of elliptic partial differential equations*, in Sparse Matrix Computations, J. P. Bunch and D. J. Rose, eds., Academic Press, New York, 1976.

[GMP] C. I. Goldstein, T. A. Manteuffel, and S. V. Parter, *Preconditioning and boundary conditions without H_2 estimates: L_2 condition numbers and the distribution of the singular values*, SIAM J. Numer. Anal., (1990), submitted.

[JMPW] W. D. Joubert, T. A. Manteuffel, S. V. Parter, and S-P. Wong, *Preconditioning second-order elliptic operators: Experiment and theory*, Los Alamos National Laboratories Report LA-UR-90-1615, Los Alamos, NM, April, 1990.

[MO] T. A. Manteuffel and J. S. Otto, *Optimal equivalent preconditionings*, SIAM J. Numer. Anal., (1991), submitted.

[MP] T. A. Manteuffel and S. V. Parter, *Preconditioning and boundary conditions*, SIAM J. Numer. Anal., 27 (1989), pp. 656–694.

[ST] G. W. Stewart, *Introduction to Matrix Computations*, Academic Press, New York, 1973.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 1, pp. 289–306, January 1992

015

FAST ITERATIVE SOLUTION OF CARRIER CONTINUITY EQUATIONS FOR THREE-DIMENSIONAL DEVICE SIMULATION*

O. HEINREICHSBERGER†, S. SELBERHERR†, M. STIFTINGER†, AND K.P. TRAAR‡

Abstract. In this paper the use of iterative methods for the solution of the carrier continuity equations in three-dimensional semiconductor device simulators is summarized. An overview of the derivation of the linear systems from the basic stationary semiconductor device equations is given and the algebraic properties of the nonsymmetric coefficient matrices are discussed. Results from the following classes of iterative methods are presented: The classical conjugate gradient (CG), the symmetrized conjugate gradient (SCG), the generalized minimum residual (GMRES), and the conjugate gradient squared (CGS) method. Preconditioners of incomplete factorization type with partial fill-in are considered. High performance implementations for these algorithms on vector, concurrent, and vector–concurrent computers are presented.

Key words. semiconductor equations, nonsymmetric systems, preconditioned iterative methods, vector computers

AMS(MOS) subject classifications. 65L10, 65F10, 65F20, 35J65

1. Introduction.

The three-dimensional numerical analysis of semiconductor devices by device simulators is increasingly becoming an indispensable tool in design and optimization of micro-miniaturized devices. Such simulators compute the discrete self-consistent solution of the semiconductor device partial differential equations. We restrict ourselves to the decoupled solution method of the three nonlinear device equations on a three-dimensional nonuniform tensor product grid [15]. In this case each single nonlinear (outer) iteration consists of the solution of the Poisson equation for the electrostatic potential ψ and of two carrier continuity equations for the electron and hole concentrations, respectively. The coefficient matrices of the discrete continuity equations in the most practical variable set n (electrons) and p (holes) are nonsymmetric. In this contribution we consider preconditioned iterative methods for the solution of these nonsymmetric linear systems. Related work is found in [12], [19], [21], [28], and [32].

Iterative methods applied to the discrete continuity equations have to cope with high condition numbers of the coefficient matrices [1]. Another problem is the enormous numerical range of the solution vector that has to be computed accurately both in the depletion zones of the device under consideration as well as in high injection regimes. Contrary to the Poisson equation, the discrete continuity equations have to be evaluated much more accurately to guarantee the stability of the nonlinear iteration. This results in substantially higher iteration counts compared to the Poisson equation, and therefore the linear nonsymmetric solvers dominate in the solution process.

We have performed a comparative study of various preconditioned conjugate gradient type solvers of which the conjugate gradient squared (CGS) method [14], [26] was identified as the fastest and most economical. The success of this method (and related ones) depends quite critically on robust preconditioning. We have concentrated on incomplete factorizations of the coefficient matrix. Partial fill-in substantially reduces the iteration count at the expense of more arithmetic work per iteration.

* Received by the editors April 5, 1990; accepted for publication (in revised form) February 26, 1991. This work was supported by Siemens AG, Munich, and Digital Equipment Corporation, Hudson.

† Institute for Microelectronics, Technical University of Vienna, Vienna, Austria.

‡ SIEMENS AG Austria, ETG 215, Vienna, Austria.

Multiple solutions of linear systems of rank $\mathcal{O}(10^5)$ or even larger on vector or vector–concurrent computers with large main memory requires vectorized and/or parallelized iterative procedures. Our implementation has therefore been concentrated on an efficient vectorizable ILU preconditioner. We show how high performance is achieved on supercomputers such as the CRAY 2 (one vector unit), Fujitsu VP200, and on superminicomputers such as the Alliant FX40 (two vector CEs) and a Digital VAX 6260 (one to six scalar processors).

The outline of this paper is as follows. In §2 the semiconductor partial differential equations and the nonlinear expressions for the physical quantities within these equations are summarized, the discretization of which is discussed in §3. The brief consideration of the algebraic matrix properties in §4 provides the preliminaries for the iterative procedures outlined in §5. Robust preconditioning is vital for the iterative solution of the carrier continuity equations. We consider incomplete factorization preconditioners in §6 and their implementation on vector and parallel computers in §7. Numerical experiments and conclusive remarks are found in §8.

2. The semiconductor partial differential equations. We consider the time-invariant case on a three-dimensional rectangular spatial domain using finite difference discretization. The semiconductor equations for the variables (ψ, n, p) consist of the Poisson equation and the carrier continuity equations. Poisson's equation for the electrostatic potential ψ reads

$$\operatorname{div}(\epsilon \cdot \operatorname{grad}\psi) = -\rho \tag{1}$$

with the space charge $\rho = q \cdot (p - n + C)$, where C denotes the net doping density, n the hole, p the electron concentrations, and q the elementary charge. The carrier continuity equations for the electron and hole current densities $\vec{J}_{n,p}$ read

$$\operatorname{div}\vec{J}_n = q \cdot R, \tag{2}$$

$$\operatorname{div}\vec{J}_p = -q \cdot R, \tag{3}$$

where R denotes the carrier generation and recombination rate.

The current densities $\vec{J}_{n,p}$,

$$\vec{J}_n = \mu_n^{LISF} \cdot n \cdot \vec{F}_n, \tag{4}$$

$$\vec{J}_p = \mu_p^{LISF} \cdot p \cdot \vec{F}_p, \tag{5}$$

are assumed to be proportional to the driving forces $\vec{F}_{n,p}$. An extended drift–diffusion approach allows the treatment of hot electron effects in one-band semiconductors such as silicon. This approach for the driving forces reads [13]

$$\vec{F}_n = -q\left(\operatorname{grad}\psi - \frac{1}{n} \cdot \operatorname{grad}\left(\frac{k \cdot T_n}{q} \cdot n\right)\right), \tag{6}$$

$$\vec{F}_p = -q\left(\operatorname{grad}\psi + \frac{1}{p} \cdot \operatorname{grad}\left(\frac{k \cdot T_p}{q} \cdot p\right)\right), \tag{7}$$

where carrier heating is modeled by carrier temperatures $T_{n,p}$. For the mobilities $\mu_{n,p}^{LISF}$ the effect of carrier heating is modeled by a nonlinear dependence on the magnitude of the driving forces $F_{n,p}$:

$$\mu_{n,p}^{LISF} = \frac{2\mu_{n,p}^{LIS}}{1 + \left(1 + \left(2\mu_{n,p}^{LIS} \cdot |\vec{F}_{n,p}|/(q \cdot v_{n,p}^{sat})\right)^{\alpha_{n,p}}\right)^{1/\alpha_{n,p}}} \tag{8}$$

with $\alpha_n = 2$, $\alpha_p = 1$. $\mu_{n,p}^{LIS}$ denotes the zero-field mobility due to lattice (L), impurity (I), and surface (S) scattering mechanisms and $v_{n,p}^{sat}$ the saturation velocity.

Approximations for the carrier temperatures $T_{n,p}$ can be derived by a series expansion of the energy conservation equations

$$(9) \qquad T_{n,p} = T_0 + \frac{2}{3} \cdot \frac{q}{k} \cdot \tau_{n,p}^{\epsilon} \cdot \left(v_{n,p}^{sat}\right)^2 \cdot \left(\frac{1}{\mu_{n,p}^{LISF}} - \frac{1}{\mu_{n,p}^{LIS}}\right)$$

with the Boltzmann constant k, the ambient temperature T_0 and the energy relaxation times $\tau_{n,p}^{\epsilon}$.

The carrier generation and recombination rate R on the right-hand side of the carrier continuity equations represents the sum of the impact ionization rate R^{II}, the Shockley–Read–Hall recombination rate R^{SRH}, and the Auger recombination rate R^{AU}:

$$(10) \qquad R = R^{II} + R^{SRH} + R^{AU}.$$

The impact ionization rate is modeled by the Chynoweth formulae

$$(11) \qquad R^{II} = -\alpha_n \cdot \frac{\left|\vec{J}_n\right|}{q} - \alpha_p \cdot \frac{\left|\vec{J}_p\right|}{q},$$

in which the $\alpha_{n,p}$ depend exponentially on the local electric field:

$$(12) \qquad \alpha_{n,p} = \hat{\alpha}_{n,p} \cdot \exp\left(\frac{-\beta_{n,p}}{\left(\vec{J}_{n,p}/|\vec{J}_{n,p}|\right) \cdot \mathrm{grad}\psi}\right).$$

The SRH recombination rate is expressed by

$$(13) \qquad R^{SRH} = \frac{\left(n \cdot p - n_i^2\right)}{\tau_p\left(n + n_1\right) + \tau_n\left(p + p_1\right)}$$

with positive constants n_1,p_1,$\tau_{n,p}$. Last, the Auger recombination rate is given by

$$(14) \qquad R^{AU} = \left(C_n \cdot n + C_p \cdot p\right) \cdot \left(n \cdot p - n_i^2\right)$$

with $C_{n,p} \geq 0$.

3. Discretization of the nonlinear system of equations. The nonlinear system of equations can be solved either by a full Newton iteration or by decoupling the three partial differential equations (Gummel's algorithm [8]). We restrict ourselves to the latter option. In that case, a nonlinear Gauss–Seidel block iterative scheme is obtained neglecting the nonlinearities in the carrier mobilities and temperatures:

$$(15) \qquad \mathrm{div}\,\mathrm{grad}\psi^{k+1} = -\frac{q}{\epsilon}\left(\frac{\partial\left(p - n + C\right)^k}{\partial\psi}\left(\psi^{k+1} - \psi^k\right) + p^k - n^k + C\right),$$

$$(16) \qquad \mathrm{div}\vec{J}_p\left(\psi^{k+1}, p^{k+1}\right) = -q\left(R\left(\psi^{k+1}, n^k, p^k\right) + \frac{\partial R}{\partial p}\left(p^{k+1} - p^k\right)\right),$$

$$(17) \qquad \mathrm{div}\vec{J}_n\left(\psi^{k+1}, n^{k+1}\right) = q\left(R\left(\psi^{k+1}, n^k, p^{k+1}\right) + \frac{\partial R}{\partial n}\left(n^{k+1} - n^k\right)\right).$$

The set of equations is now discretized on a three-dimensional domain. The boundary value problem is of mixed Dirichlet–Neumann type. For the idealized ohmic contacts, Dirichlet boundary conditions hold. For the artificial interfaces in the deep semiconductor bulk, homogenous Neumann boundary conditions have to be applied. Nonhomogenous Neumann boundary conditions for the electrostatic potential are valid in case of interface charges, e.g., at semiconductor–oxide interfaces. Nonvanishing interface recombination velocities at Schottky contacts yield nonhomogenous Neumann boundary conditions for the carrier concentrations.

The nonlinearities in R have to be treated carefully. The derivatives of R^{II} with respect to the carrier concentrations can be neglected if the carrier generation rate is not updated at every nonlinear iteration but in a superimposed generation *supercycle*. For the recombination rates R^{SRH}, however, the derivatives with respect to n or p are computed, because these contributions increase the diagonal dominance in the resulting linear system. Such a stabilizing effect is not necessarily true for R^{AU}, therefore, negative contributions of the derivatives of R^{AU} to the main diagonal of the coefficient matrix are discarded.

For the finite difference discretization of the carrier continuity equations an exponential interpolation scheme for the carrier concentrations n and p must be used [3], [24]. This is due to an exponential dependence of the carrier concentrations on the electrostatic potential (Scharfetter–Gummel interpolation). Herein the quantities ψ, $\mu_{n,p}$, and $T_{n,p}$ are interpolated linearly. For a one-dimensional nonuniform discretization with mesh spacings h_i, neglecting the derivatives of R and assuming constant carrier temperatures $T_{n,p}$, one obtains for the three-point stencil

$$\begin{aligned} &n_{i-1} D_{n,i-1/2} \frac{\mathcal{B}(-\Delta_{i-1})}{2h_{i-1}} + n_{i+1} D_{n,i+1/2} \frac{\mathcal{B}(\Delta_i)}{2h_i} \\ &-n_i \left(D_{n,i-1/2} \frac{\mathcal{B}(\Delta_{i-1})}{2h_{i-1}} + D_{n,i+1/2} \frac{\mathcal{B}(-\Delta_i)}{2h_i} \right) = R_i \frac{h_{i-1} + h_i}{2}, \end{aligned} \tag{18}$$

$$\begin{aligned} &p_{i-1} D_{p,i-1/2} \frac{\mathcal{B}(\Delta_{i-1})}{2h_{i-1}} + p_{i+1} D_{p,i+1/2} \frac{\mathcal{B}(-\Delta_i)}{2h_i} - \\ &p_i \left(D_{p,i-1/2} \frac{\mathcal{B}(-\Delta_{i-1})}{2h_{i-1}} + D_{p,i+1/2} \frac{\mathcal{B}(\Delta_i)}{2h_i} \right) = R_i \frac{h_{i-1} + h_i}{2}. \end{aligned} \tag{19}$$

In these formulae the diffusivities obey the Einstein relation $D_{n,p} = \mu_{n,p} \cdot Ut$, where $Ut = \frac{k \cdot T_0}{q}$ denotes the thermal voltage. $\mathcal{B}$ is the Bernoulli function

$$\mathcal{B}(x) = \frac{x}{\exp(x) - 1}. \tag{20}$$

The arguments of the Bernoulli function are $\Delta_i = \frac{\psi_{i+1} - \psi_i}{Ut}$.

For nonconstant carrier temperatures $T_{n,p}$, the above expressions are generalized as follows [27]. Let

$$Ut_{(n,p),i} = \frac{k \cdot T_{(n,p),i}}{q} \tag{21}$$

denote the so-called local "electronic voltages" at the meshpoint i. Assume further that the electronic voltages $Ut_{(n,p),i}$ vary linearly between the meshpoints as is the assumption for the electrostatic potential. Then a local one-dimensional continuity equation is solved for the current densities $J_{n,p}$ between neighboring meshpoints.

Assuming constant current densities, the following midpoint interpolation formula replaces the diffusivities

$$(22)\quad D_{(n,p),i+1/2} = \mu_{(n,p),i+1/2} \cdot \left(Ut_{(n,p),i+1} - Ut_{(n,p),i}\right) \cdot \left[\ln\left(\frac{Ut_{(n,p),i+1}}{Ut_{(n,p),i}}\right)\right]^{-1}.$$

The midpoint values for the mobilities $\mu_{(n,p),i+1/2}$ are approximated by linear interpolation. The Bernoulli function argument evaluates to

$$(23)\qquad \Delta_{(n,p),i} = \frac{(\psi_{i+1} - \psi_i) - \left(Ut_{(n,p),i+1} - Ut_{(n,p),i}\right)}{\left(Ut_{(n,p),i+1} - Ut_{(n,p),i}\right)} \cdot \ln\left(\frac{Ut_{(n,p),i+1}}{Ut_{(n,p),i}}\right).$$

4. Algebraic properties of the coefficient matrices. The exponential interpolation scheme outlined in the last section produces a nonsymmetric, diagonally dominant, two-cyclic, seven-band coefficient matrix A.

Nonsymmetry is caused by the inequality $\mathcal{B}(-x) \neq \mathcal{B}(x)$. Diagonal dominance is due to the fact that each negative column sum of the offdiagonal elements is less than (for nonvanishing R) or equal (for vanishing R) to the main diagonal pivot. This implies at least semidefiniteness of A. Consider the case of constant carrier temperatures $T = T_n = T_p$. The equality

$$(24)\qquad \mathcal{B}(-x) = \exp(x) \cdot \mathcal{B}(x)$$

and the fact that the exponentially scaled potential increments can be factored yields that A can be transformed to a symmetric, positive definite matrix $\bar{A}$,

$$(25)\qquad \bar{A} = W^{-1} \cdot A \cdot W,$$

by a diagonal similarity transformation. The diagonal matrix W is positive definite and the elements $w_{(n,p),i}$ are given by $\exp(+\psi_i/2Ut)$ for electrons and $\exp(-\psi_i/2Ut)$ for holes. Since a similarity transformation leaves the spectrum of the matrix A unchanged, we have a nonsymmetric system of linear equations, in which A has a positive real spectrum. For local carrier temperature we are not aware of such a transformation, hence we cannot make statements of the spectrum of A in this case.

We note the enormous numerical range of the W matrices. For a maximum electrostatic potential of 100 Volts and liquid nitrogen temperature (77 K) we may expect exponents of the order $\log_{10}(w_{i,\max}) = 3275$. These numbers make the explicit transformation of the linear system undesirable on standard, double precision computer arithmetics. The very large rank of A gives preference to iterative methods over sparse Gaussian elimination.

5. Selected iterative methods for the linear systems. In this section we discuss the iterative procedures that were used in our computations. In the case of symmetrizability these are the classical conjugate gradient (CG) algorithm, then a variant of CG that circumvents the explicit symmetrization (SCG), and the conjugate gradient squared (CGS) procedure. For the nonsymmetrizable case we consider the generalized minimum residual algorithm (GMRES) and again CGS.

The numerical condition of the discrete carrier continuity equations can be rather poor [1], therefore efficient preconditioning is important. Incomplete (left or split) LU factorizations are used for preconditioning together with (left or symmetric) scaling by the main diagonal pivots $\tilde{D}$ of the preconditioner, and the Eisenstat procedure [7] is used to compute the preconditioned matrix-vector multiply. In §8, where various

TABLE 1
ILU-SCG.

```
Choose $x_0$
$\hat{r}_0 = Q_R^{-1} Q_L^{-1} (b - A x_0)$
$x_{-1} = \hat{r}_{-1} = 0$
$\hat{\rho}_0 = 1$
FOR $n = 0$ STEP 1 UNTIL convergence DO
    $\hat{u} = W^{-2} Q_L Q_R \hat{r}_n$
    $\hat{\sigma}_n = \langle \hat{u}, \hat{r}_n \rangle$
    $\hat{v} = Q_R^{-1} Q_L^{-1} A \hat{r}_n$
    $\hat{\gamma}_n = \frac{\hat{\sigma}_n}{\langle \hat{u}, \hat{v} \rangle}$
    $\hat{\rho}_n = \left(1 - \frac{\hat{\gamma}_n}{\hat{\gamma}_{n-1}} \frac{\hat{\sigma}_n}{\hat{\sigma}_{n-1}} \frac{1}{\hat{\rho}_{n-1}}\right)^{-1}$        (if $n > 0$, $\hat{\rho}_0 = 1$)
    $x_{n+1} = \hat{\rho}_n (x_n + \hat{\gamma}_n \hat{r}_n) + (1 - \hat{\rho}_n) x_{n-1}$
    $\hat{r}_{n+1} = \hat{\rho}_n (\hat{r}_n - \hat{\gamma}_n \hat{v}) + (1 - \hat{\rho}_n) \hat{r}_{n-1}$
END FOR
```

numerical experiments will be presented, estimates for the extremal eigenvalues of the preconditioned matrix, obtained by the conjugate gradient method, illustrate that the preconditioned problem is well conditioned. We use the following notational conventions: The index s denotes the scaled matrix A_s and its strictly triangular parts L_s and U_s. Angle brackets denote the dot-product. Quantities with a hat ($\hat{\ }$) refer to the preconditioned system.

In this section we shall make explicit use of the similarity property, which was derived in the last section. In case of knowledge of the diagonal transformation matrices W, an explicit transformation (e.g., during the sparse matrix assembly) of the linear system into symmetric form is the most straightforward approach provided that the computer arithmetic is sufficiently accurate for the numbers generated by the similarity transformation. This transformation saves matrix storage and the classical preconditioned conjugate gradient algorithm (CG) is the optimal iterative method.

The very large number range in the iterates can be circumvented by a variant of the conjugate gradient algorithm, e.g., proposed in [11]. In this case the diagonal transformation matrices are confined to the inner products in the variant of the CG algorithm given in Table 1. An appropriate scaling can be employed for the inner products, thus avoiding restrictions on computer arithmetics. A three-term recursion is used with left preconditioning. The $W^T W Q^{-1} A$ norm of the solution error is minimized at each iteration step.

If the linear systems are not symmetrizable one must choose from a more general class of iterative methods for nonsymmetric linear systems. We concentrate on two methods: The generalized minimum residual method GMRES, an orthogonalization method, and the conjugate gradient squared method CGS, which has no (known) minimization property. Algorithms that were not considered are methods that require (e.g., dynamically computed) eigenvalue estimates, such as the Manteuffel algorithm, and the more recent hybrid methods, which adaptively switch between different acceleration schemes [6], [16].

The GMRES algorithm minimizes the two-norm of the residual at each iteration

TABLE 2
ILU-GMRES(m).

```
Choose $\hat{x}_0$
$\hat{b}_s = (I + L_s)^{-1} b_s$
Choose $m$
FOR $n = 0$ STEP 1 UNTIL convergence DO
    $\hat{t} = (I + U_s)^{-1} \hat{x}_n$
    $\hat{u} = \hat{t} + (I + L_s)^{-1} \left( (diag\,(A_s) - 2I)\,\hat{t} + \hat{x}_n \right)$
    $\hat{r}_n = \hat{b}_s - \hat{u}$
    $\hat{\beta}_n = \|\hat{r}_n\|$
    $\hat{v}_1 = \frac{\hat{r}_n}{\hat{\beta}_n}$
    FOR $j = 1$ STEP 1 UNTIL $j = m$ DO
        $\hat{t} = (I + U_s)^{-1} \hat{v}_j$
        $\hat{u} = \hat{t} + (I + L_s)^{-1} \left( (diag\,(A_s) - 2I)\,\hat{t} + \hat{v}_j \right)$
        FOR $i = 1$ STEP 1 UNTIL $i = j$ DO
            $\hat{h}_{i,j} = \langle \hat{u}, \hat{v}_i \rangle$
        END FOR
        $\hat{v}_{j+1} = \hat{u} - \sum_{i=1}^{j} \hat{h}_{i,j} \hat{v}_i$
        $\hat{h}_{j+1,j} = \|\hat{v}_{j+1}\|$
        $\hat{v}_{j+1} = \hat{v}_{j+1} / \hat{h}_{j+1,j}$
    END FOR
    Solve least squares problem $\|\hat{\beta}_n e_1 - \hat{\bar{H}}_m \hat{y}\|$ for $\hat{y}$
        with $e_1 = [1, 0, \cdots, 0]^T \in \mathbb{R}^{m+1}$, $\hat{\bar{H}}_m \in \mathbb{R}^{m+1 \times m}$
        (upper Hessenberg matrix consisting of the $\hat{h}_{i,j}$), $\hat{y} \in \mathbb{R}^m$
    $\hat{x}_{n+1} = \hat{x}_n + \hat{V}_m \hat{y}$
        with $\hat{V}_m = [\hat{v}_1, \hat{v}_2, \cdots, \hat{v}_m] \in \mathbb{R}^{N \times m}$
END FOR
$x_{n+1} = \tilde{D}^{-1/2} (I + U_s)^{-1} \hat{x}_{n+1}$
```

step. An orthonormal basis is built by an Arnoldi process, and the Hessenberg least squares problem can be solved, e.g., by Householder transformations. See Table 2. The monotonic convergence of GMRES has to be paid for. Full orthogonalization at iteration step n, which yields optimum convergence speed, requires storage of n vectors. This is prohibitive for large, three-dimensional problems and makes restarting of the iteration necessary, thus abandoning optimality. Nevertheless, the convergence of the restarted GMRES(m) is certainly monotonic. Values for the restarting frequency m less than, say, 6 have been found acceptable. Although the solution vector is updated every m iterations, the residual norm is available at each iteration step at no extra cost. This is a by-product of the QR decomposition for the solution of the least squares problem, if the Q and R matrices are updated at each iteration step.

A way of decreasing storage and arithmetic requirements is the use of Lanczos methods, such as the biconjugate gradient (BiCG) algorithm or the biconjugate gradient squared (CGS) algorithm [26] given in Table 3. Both algorithms construct approximations to the solution in the same Krylov subspace as GMRES, but a biorthogonality condition to the transposed system is used rather than an orthogonality condition to

TABLE 3
ILU-CGS

```
Choose $x_0$
$\hat{r}_0 = (I + L_s)^{-1}(b_s - A_s x_0)$
$\hat{x}_0 = (I + U_s)\tilde{D}^{1/2} x_0$
Choose $\hat{y}_0$ such that $\langle \hat{y}_0, \hat{r}_0 \rangle \neq 0$
$\hat{q}_0 = \hat{p}_{-1} = 0$
$\hat{\rho}_{-1} = 1$
FOR $n = 0$ STEP 1 UNTIL convergence DO
    $\hat{\rho}_n = \langle \hat{y}_0, \hat{r}_n \rangle$
    $\hat{\beta}_n = \frac{\hat{\rho}_n}{\hat{\rho}_{n-1}}$
    $\hat{u} = \hat{r}_n + \hat{\beta}_n \hat{q}_n$
    $\hat{p}_n = \hat{u} + \hat{\beta}_n \left(\hat{q}_n + \hat{\beta}_n \hat{p}_{n-1}\right)$
    $\hat{t} = (I + U_s)^{-1} \hat{p}_n$
    $\hat{v} = \hat{t} + (I + L_s)^{-1}\left((diag\,(A_s) - 2I)\,\hat{t} + \hat{p}_n\right)$
    $\hat{\sigma}_n = \langle \hat{y}_0, \hat{v} \rangle$
    $\hat{\alpha}_n = \frac{\hat{\rho}_n}{\hat{\sigma}_n}$
    $\hat{q}_{n+1} = \hat{u} - \hat{\alpha}_n \hat{v}$
    $\hat{u} = \hat{u} + \hat{q}_{n+1}$
    $\hat{t} = (I + U_s)^{-1} \hat{u}$
    $\hat{v} = \hat{t} + (I + L_s)^{-1}\left((diag\,(A_s) - 2I)\,\hat{t} + \hat{u}\right)$
    $\hat{x}_{n+1} = \hat{x}_n + \hat{\alpha}_n \hat{u}$
    $\hat{r}_{n+1} = \hat{r}_n - \hat{\alpha}_n \hat{v}$
END FOR
$x_{n+1} = \tilde{D}^{-1/2}(I + U_s)^{-1} \hat{x}_{n+1}$
```

construct the direction vectors. The residual does not decrease monotonically, hence the stopping procedure is more difficult compared to GMRES. The initial vector $\hat{y}_0$ may be chosen arbitrarily such that $\langle \hat{y}_0, \hat{r}_0 \rangle \neq 0$. We conform to the common practice to set this vector equal to the initial residual vector $\hat{r}_0$. A well-known property of this algorithm is the possibility of breakdown by vanishing of certain inner products. This phenomenon, which cannot be excluded a priori, is certainly a reason for worry. It is an experimental observation that effective preconditioning makes the event of (near) breakdown unlikely. We have observed that a sufficiently high machine precision can avoid breakdown occurring when the iteration is near the true solution.

Related squared Lanczos algorithms [10] (BIORES^2 and BIODIR^2) exist due to the analogy to the Lanczos-ORTHORES and Lanczos-ORTHODIR algorithms. These algorithms can be implemented to generate the same iterates for the solution vector (in exact arithmetic) if identical vectors x_0 and $\hat{y}_0$ are chosen. Though mathematically equivalent, unavoidable roundoff errors cause the iterates of the three biorthogonalization algorithms to drive apart in the course of the iteration. BIOMIN^2, i.e., CGS, not only proved to perform best under presence of roundoff, but could also be implemented most economically concerning storage.

In Table 4 the arithmetic work of the iterative procedures is listed. These figures refer to one linear iteration. For the restarted GMRES(m) method the iteration counter is incremented after the computation of one new orthogonal basis vector.

TABLE 4
Comparison of arithmetic work/iteration.

Solver	Ax	$\langle x,y\rangle$	$x+y$	αx
CG	1	2	4	10
SCG	1	3	4	10
CGS	2	2	7	6
GMR	1	$\frac{m}{2}+1$	$\frac{m}{2}$	$\frac{m}{2}+1$

6. Efficient preconditioning. We start with a comparison of two preconditioners that have been examined intensively: Block Jacobi preconditioning $P_J = D$, where D is the tridiagonal part of A, and incomplete factorization preconditioning [17], where

$$P_{ILU} = \left(L + \tilde{D}\right)\tilde{D}^{-1}\left(U + \tilde{D}\right) \tag{26}$$

with L the strictly lower and U the strictly upper triangular part of A and $\tilde{D}$ computed such that

$$diag\,(P_{ILU}) = diag\,(A)\,. \tag{27}$$

It is trivial to see that for the matrices under consideration the nonzero pattern of the LU factors of P_J is a proper subset of the nonzero pattern of the factors of P_{ILU}. It can be proven that in this case the ILU preconditioner is superior to the Jacobi preconditioner in the sense that if $A = P_J - R_J$ and $A = P_{ILU} - R_{ILU}$, then

$$\rho\left(P_{ILU}^{-1}R_{ILU}\right) \le \rho\left(P_J^{-1}R_J\right) \tag{28}$$

where ρ denotes the spectral radius. Thus, a stationary iterative method based on the ILU splitting will converge at least as fast as a method based on a Jacobi splitting. Accelerated methods (see §5) will be influenced in a similar way. On the other hand, the arithmetic work for the Jacobi preconditioner compared with the ILU preconditioner is smaller and involves only first order recurrences, which can be vectorized more easily. Summarizing our experimental work, we state that the use of the Jacobi preconditioner seems to be limited to low bias voltage applications, i.e., examples where the diffusion term dominates in the current relations (4)–(6). For higher bias voltages unpleasant numerical effects have been observed (especially with the Lanczos methods), such as convergence stagnation or near breakdown in the Lanczos process at the beginning of the iteration. Another disadvantage is the orientation sensitivity due to the line elimination that forces the swapping of the matrix and vector elements to the most favorable direction, thus causing inconvenient computational overhead. A really clear relationship for both the minority and majority carrier continuity equations and the direction of the main current flow that would facilitate a detection of a favorable preconditioning orientation, however, could not be established. Therefore we cannot recommend this type of preconditioner for general use.

An incomplete factorization preconditioner of alternating direction type, P_T, with tridiagonal matrix factors, has been proposed in [5]. The basic idea is to use tridiagonal factors that lend themselves more to parallelization, rather than triangular factors as with ILU, at the same time maintaining the ILU sparsity pattern. For a seven-point stencil such a factorization would read

$$P_T = T_1 D^{-1} T_2 D^{-1} T_3, \tag{29}$$

where D is the main diagonal of A, and the T_i, $i = 1, 2, 3$, are tridiagonal matrices such that

$$T_1 + T_2 + T_3 = A + 2D. \tag{30}$$

The elements in the factors of P_T are altogether equal to the corresponding elements in A, hence there is no need to compute diagonal pivots $\tilde{D}$ as with ILU. However, there are more error terms outside the nonzero pattern in P_T; therefore it is no surprise that the iteration count in the iterative solver is higher compared with the ILU preconditioner. Computing $y = P_T^{-1}x$ involves the backsolves of the LU factors of three tridiagonal matrices. Since each tridiagonal matrix consists itself of many independent tridiagonal systems this work can be parallelized easily (e.g., by the partitioning method, cyclic reduction, etc.).

Let NX, NY, and NZ denote the number of gridlines in the respective directions. The inversion of T_1, which is assumed to contain the innermost diagonals of A, consists of $NY \cdot NZ$ independent tasks and has stride NX. T_3, which is assumed to contain the outermost diagonals of A, has $NX \cdot NY$ independent tasks and stride 1. For T_2 the situation is different because the stride is constant for NZ data sets only. The authors in [5] have reported that one preconditioning step with P_T can be performed up to three times as fast as one step in the hyperplane-ILU preconditioner on vector-supercomputers. Our numerical experiments, however, indicate that the convergence decrease with respect to ILU(0) is often larger than three even for simple examples. We refrained from an implementation on a supercomputer.

The most commonly used and probably most efficient preconditioner, at least on scalar computers, is incomplete LU factorization with allowable fill-in denoted by ILU(k), see, e.g., [2], [4], [12]. The index k denotes a controllable sparsity pattern along the matrix diagonals, $k = 0$ denoting no fill-in, $k = 1$ denoting fill-in caused by the original nonzero pattern but no further, and so on. As expected, a higher degree of fill-in reduces the iteration count. However, the number of operations for the factorization and for each iteration as well as the memory requirements increase considerably. For example, the ILU(1) preconditioner needs four extra diagonals within the original seven-diagonal nonzero pattern. For the fill-in ILU preconditioners we are not aware of a comparably efficient implementation to compute the preconditioned matrix vector multiply as proposed by Eisenstat for the ILU(0) [7], [29].

The two ILU(0) factorization variants under consideration are split ILU

$$\hat{A}_1\hat{x} \equiv \tilde{D}^{1/2}\left(\tilde{D}+L\right)^{-1}A\left(\tilde{D}+U\right)^{-1}\tilde{D}^{1/2}\hat{x} = \tilde{D}^{1/2}\left(\tilde{D}+L\right)^{-1}b \equiv \hat{b} \tag{31}$$

$$\hat{x} \equiv \tilde{D}^{-1/2}\left(\tilde{D}+U\right)^{-1}x \tag{32}$$

and left ILU

$$\hat{A}_2 x \equiv \left(\tilde{D}+U\right)^{-1}\tilde{D}\left(\tilde{D}+L\right)^{-1}Ax = \left(\tilde{D}+U\right)^{-1}\tilde{D}\left(\tilde{D}+L\right)^{-1}b \equiv \hat{b}. \tag{33}$$

The preconditioned matrix-vector multiply $\hat{A}_{1,2}\hat{p}$ can be simplified in the following manner. For the split ILU(0) one obtains, after having scaled the matrix symmetrically by $\tilde{D}$,

$$A_s = \tilde{D}^{-1/2}A\tilde{D}^{-1/2} \equiv diag\,(A_s) + L_s + U_s, \tag{34}$$

TABLE 5
Hyperplane-ILU(k) *on vectorcomputers.*

	VP-200			Cray-2			Alliant FX40		
k	B	Speedup	MFlop	B	Speedup	MFlop	B	Speedup	MFlop
0	4	13.75	96	14	4.30	27	212	1.34	1.8
1	8	12.12	96	26	4.76	30	327	1.51	2.3
2	8	12.62	128	30	5.93	34	410	1.64	2.5

$$\hat{A}_1\hat{p} = \left[\hat{t} + (I + L_s)^{-1}\left(\hat{p} - (2I - diag\,(A_s))\,\hat{t}\right)\right] \tag{35}$$

with

$$\hat{t} = (I + U_s)^{-1}\hat{p}, \tag{36}$$

and for the left ILU(0) together with left scaling

$$\hat{A}_2\hat{p} = (I + U_s)^{-1}\left[\hat{p} + (I + L_s)^{-1}\left(diag\,(A_s) - I + U_s\right)\hat{p}\right]. \tag{37}$$

In our implementation the split ILU(0) requires two additional scratch vectors and N square-roots, whereas the left ILU(0) requires no extra storage but a triangular matrix vector multiply.

Computing the diagonal pivots of the incomplete factorization such that $P_{ILU} - A$ has zero *column* sums (or row sums in the symmetric case) leads to modified incomplete factorization-type preconditioners (MILU) originated by [9]. A modification factor α in the interval $[0, 1]$ is usually introduced to smoothly sweep between pure ILU and full MILU factorization. Our results concerning the choice of such a modification factor do not admit a clear statement. It seems that $\alpha = 1$ always decreases the performance with respect to $\alpha = 0$ slightly. We found a number of device examples where a choice of $\alpha = 0.5$ yields a performance enhancement of about 10 percent to 30 percent concerning the iteration count. However, this gain is partly compensated by the higher arithmetic work for the factorization.

Parallelizable variants of ILU such as the Neumann polynomial preconditioner [31] were investigated as well. Numerical experiments carried out with the NSPCG (Non-Symmetric Preconditioned Conjugate Gradient) code [18] identified none of them competitive with ILU.

7. Implementation notes for parallel and vector computers. Vectorizing and parallelizing the backsolves of triangular or tridiagonal systems is a good exercise to explore computer architecture. We concentrate on vectorization techniques that are unlikely to degrade the performance of the incomplete factorization preconditioners; hence we do not consider multicolor orderings [20]. We further exclude computers that permit only unity-stride vector operations such as the CYBER 205 and disregard optimization measurements possibly required by memory bank and related conflicts. We aim at a production code that is as generally applicable as possible. This can be achieved by a reordering technique that does not change the preconditioner and produces long vector lengths. The *hyperplane* method, which is a plane-diagonalwise reordering, excellently reported in [2], [29], achieves this goal. The price for the rather general implementation we are aiming at is indirect addressing by list vectors.

If the unknowns as well as the matrix elements are indexed by $(i1, i2, i3)$ in the three spatial directions, then hyperplanes H_m (or "computational wavefronts") are defined by the set of all mesh points in the simulation domain that satisfy the equation

$$i1 + (k+1)(i2 + i3) = m, \tag{38}$$

where m is constant. k denotes the level of fill-in. Obviously the computation of the unknowns in H_m can be carried out in parallel (and hence is a vector operation) since they depend only on the unknowns in H_{m-1} for the lower triangular or H_{m+1} for the upper triangular system, respectively.

For an implementation of the hyperplane method it is desirable to form a unique vector of the unknowns in a particular H_m. This is achieved by initially computing the addresses of the unknowns to be processed in an integer list vector $LIST$ and marking the beginning of each hyperplane by an additional list vector $LPTR$. The vectorlengths increase from 1 to $\mathcal{O}(NX \cdot NY \cdot NZ)^{2/3}$.

Special attention has to be paid to the meshpoints at the simulation boundary. Addressing (nonexisting) elements at the boundary points can be prevented by proper IF statements in the code or by computing the unknowns at the boundary outside the loop. We decided to extend the array of the unknowns such that unallowed addressing at the boundaries cannot happen. This is done by allocating an array of size $NX \cdot NY \cdot (NZ + 2)$ for the vector of the unknowns $X(I)$ and filling the front and back plane of this vector with zeros. Then the code for the solution of the lower triangular system for $k = 0$ is surprisingly simple:

```
      DO 1 L=2,NX+NY+NZ-2
      DO 1 M=LPTR(L-1)+1,LPTR(L)
      I=LIST(M)
1     X(I)=R(I)-B(I)*X(I-1)-D(I)*X(I-NX)-F(I)*X(I-NX*NY).
```

R denotes the right-hand side and B, D, F the strictly lower triangular part of A_s. The inner loop is vectorizable. The implementation for the upper triangular system and the higher order recurrences ($k = 1, 2$) is straightforward. The approach sketched above is used in a similar manner to vectorize the ILU factorization at the beginning of the iteration.

Using the computational front approach, additional parallelism can be achieved by twisting the incomplete factorization in one specific direction [29], [30]. Then the factorization and the backsubstitutions can be performed concurrently from both ends to the center and from the center to both ends. Since such a twisting splits the domains into two equal halves, the mean hyperplane vectorlength in each half is decreased unless the number of meshpoints in the direction of splitting is significantly larger than the number of meshpoints in the remaining directions. It has been reported in [29] that the twisted hyperplane approach tends to decrease the iteration count in the linear solver, however, such an effect could not be verified with our type of equations. We think that the twisted hyperplane method is not advantageous, at least on a two-processor machine. To qualify the performance of the hyperplane-ILU(k) implementation, tests have been carried out on a Fujitsu VP200, a Cray-2, an Alliant FX40, and a Digital VAX 6260 computer. In Table 5 the CPU time for one triangular backsubstitution (B) in milliseconds (ms), the overall achieved speedup over the autovectorized code megaflop (MFlop) rate for this operation is given. This test example uses a 40 × 40 × 40 grid and the measured numbers are mean values for 100 solves. For a larger number of meshpoints the values are even more favorable.

The values in Table 6 show the convergence speed improvement factors the level 1, 2 preconditioners must reach to beat the ILU(0) preconditioner. As can be seen,

TABLE 6
ILU-*level* 1, 2 *preconditioner break-even points.*

k	VP-200	Cray-2	Alliant FX-40
1	1.56	1.4	1.44
2	2.15	2.0	1.58

TABLE 7
Parallel hyperplane ILU(0) *on the VAX* 6260.

# Processors	1	2	3	4	5	6
MFlop	0.58	1.15	1.64	2.06	2.41	2.65
Speedup	1.00	1.98	2.82	3.54	4.14	4.56

these values tend to favour ILU(1), since practical values for the convergence speedups against ILU(0) are about 2. Thus, if memory usage is not too restrictive, ILU(1) is to be preferred to ILU(0).

In Table 7 the performance of ILU(0) on a Digital VAX 6260 with six scalar processors is presented. The megaflop rates and the speedup over one processor for the above test example are shown.

8. Numerical results. In this section the effectiveness of the preconditioned iterative methods outlined in the previous chapters will be demonstrated. We chose two simulations of silicon MOS-transistors. The finite-difference grids of both simulations are comparatively small, thus the "true" solution $\bar{x}$ of the linear systems were obtained by a sparse direct solver and the 2-norm of the relative solution error

$$e_n = \frac{\|\bar{x} - x_n\|}{\|\bar{x}\|} \tag{39}$$

was evaluated. A (heuristic) error threshold of $\zeta = 10^{-8}$ for e_n was found satisfactory in practical simulations. The computations with the CG and the SCG were performed with quad precision, the CGS and GMRES(5) iterations in double precision. These tests were carried out on a Digital VAX 8800, the precision of 1.0 is $1.387 \cdot 10^{-17}$ for double precision and $9.629 \cdot 10^{-35}$ for quad precision arithmetic. All data presented in the following have been extracted from version 5 of the device simulator MINIMOS [25].

Example 1 is a moderately nonplanar N-channel MOSFET with a channel length of 1.5 microns. The gate voltage is $U_g = 0.5$ Volts, the drain voltage is $U_d = 1.0$ Volt, all other terminal voltages are zero. The finite-difference grid is built self-adaptively and is small due to the low biasing: $23 \times 27 \times 16$ in x- (channel length), y- (pointing into the substrate), and z- (channel width) direction. Gummel's decoupling algorithm converges in five iterations. The terminal currents are small: $5.30 \cdot 10^{-9}$ Amperes for the drain current (I_D), and $-1.20 \cdot 10^{-13}$ Amperes for the bulk current (I_B). Carrier temperatures are judiciously neglected in this simulation, hence the nonsymmetric linear equations can conveniently be symmetrized and solved by the classical CG algorithm. The convergence of CG is compared with the symmetrized variant of the CG algorithm (SCG) and the (CGS) method. ILU(0) and ILU(1) are used as preconditioners. The CG algorithm provides cheap estimates of the extremal eigenvalues of the preconditioned matrix and hence for the spectral condition number $\kappa = \lambda_{\max}/\lambda_{\min}$.

TABLE 8
Example 1 : Majority carrier continuity equation.

	ILU(0) $\kappa = 35$			ILU(1) $\kappa = 10$		
N	I-CG	I-SCG	I-CGS	I-CG	I-SCG	I-CGS
1	17	32	25	9	22	12
2	34	29	24	20	22	11
3	29	29	22	18	21	11
4	33	29	24	20	21	11
5	21	41	21	15	26	11

TABLE 9
Example 1 : Minority carrier continuity equation.

	ILU(0) $\kappa = 102$			ILU(1) $\kappa = 70$		
N	I-CG	I-SCG	I-CGS	I-CG	I-SCG	I-CGS
1	50	84	48	41	45	35
2	54	78	43	43	40	32
3	53	78	39	42	39	30
4	54	68	42	42	37	27
5	51	86	47	40	46	35

In Table 8 the iteration history of the majority carriers (holes) are listed, in Table 9 the minorities (electrons) are listed. The spectral condition number estimate κ varies only marginally during the nonlinear iterations.

As can be seen from the tables, the iteration counts of the CG algorithm correspond nicely to the bounds suggested by the spectral condition number. The CGS algorithm converges twice as fast as the CG method for the best conditioned problem (majorities with ILU(1)).

Convergence curves for the three iterative methods with the matrices from the first nonlinear iteration are given in Figs. 1 and 2.

The second example is a P-channel MOSFET with bias voltages of $U_s = 0.0$ Volt, $U_g = -4.0$ Volts, $U_d = -1.0$ Volt, and $U_b = 2.0$ Volts. The drain current is $I_d = -6.3 \cdot 10^{-5}$ Amperes and the bulk current is $I_b = -1.7 \cdot 10^{-9}$ Amperes. This time electron and hole carrier temperatures are simulated self-consistently. First an initial solution is computed, which satisfies the classical semiconductor equations for constant carrier (i.e., ambient) temperature. This requires ten Gummel iterations. Subsequently the solution of the semiconductor equations with the extended drift-diffusion transport equations (see §2) is found by relaxation with respect to the local carrier temperatures. This requires seven further nonlinear iterations.

Within this second relaxation scheme, the linear systems are no longer diagonally similar to a symmetric matrix, hence CG and SCG are no longer applicable, but CGS and GMRES are. The restarting frequency m for GMRES is chosen to be 5 and the results are in columns $GMR(5)$. In Table 10 the iteration counts for the carrier relaxation cycles are given. Convergence curves for the three iterative methods with the matrices from the first nonlinear iteration (for locally varying carrier temperatures) are given in Figs. 3 and 4, respectively.

We report some timing results using ILU(0)-CGS. Both small examples execute quickly on supercomputers: Less than 30 seconds on the Fujitsu VP and slightly more than a minute on the Cray-2. This timing ratio is in good accordance with the megaflop counts (see §7) of the forward and backsolves of the triangular systems on these machines. A computationally complex example requires substantially more time, e.g., on the Fujitsu VP computer: A nonplanar N-channel MOSFET with $U_d =$ 5.0 Volts, $U_g = 1.0$ Volt, requires a grid of 55 x 48 x 30 points. The grid loop and the computation of the initial solution requires 150 seconds. The subsequent nonlinear solution pass takes 80 Gummel iterations and a total of 1500 CPU seconds. The linear systems derived from the majority carrier continuity equation converge in roughly 70, from the minority carrier continuity equation 140 iterations (mean values). The same example computed on a fast scalar computer (NAS XL/80) takes approximately ten times more CPU time. The total time, however, is then increased substantially by references to secondary storage.

TABLE 10
Example 2 : Carrier temperature relaxation.

	Majorities (electrons)				Minorities (holes)			
	ILU(0)		ILU(1)		ILU(0)		ILU(1)	
N	CGS	GMR(5)	CGS	GMR(5)	CGS	GMR(5)	CGS	GMR(5)
1	45	375	34	90	87	250	40	175
2	37	315	15	105	87	270	42	170
3	34	465	16	160	81	235	41	195
4	35	355	15	120	79	290	35	195
5	35	370	15	135	81	270	38	150
6	33	405	15	65	77	285	37	160
7	41	330	25	125	81	270	36	175

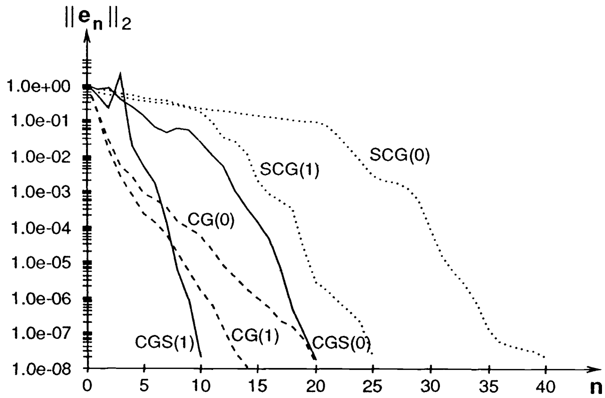


FIG. 1. *Convergence curves for the discrete majority continuity equation of Example* 1.

9. Conclusions. In this paper preconditioned iterative methods for the carrier continuity equations in three-dimensional device simulators have been studied and the performance of these algorithms on various high performance computing systems has been evaluated.

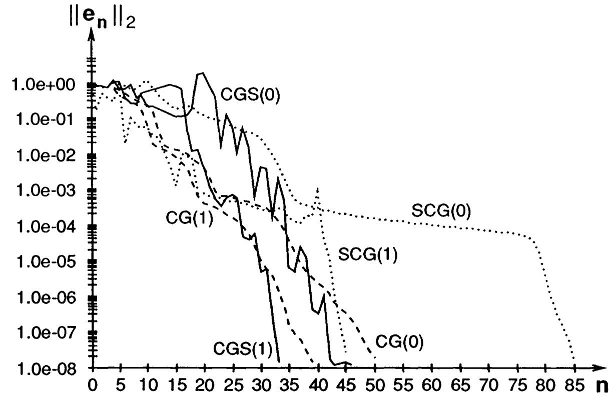


FIG. 2. *Convergence curves for the discrete minority continuity equation of Example* 1.

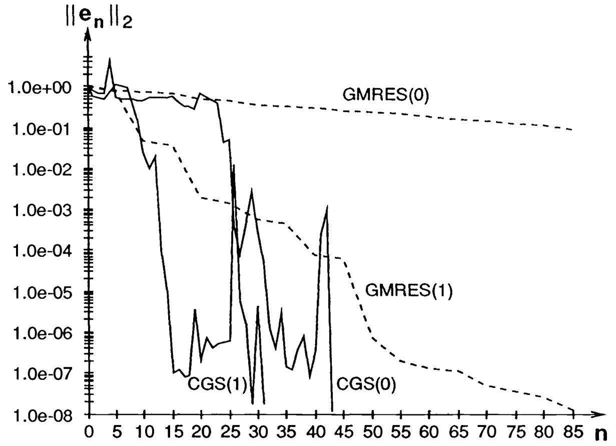


FIG. 3. *Convergence curves for the discrete majority continuity equation of Example* 2.

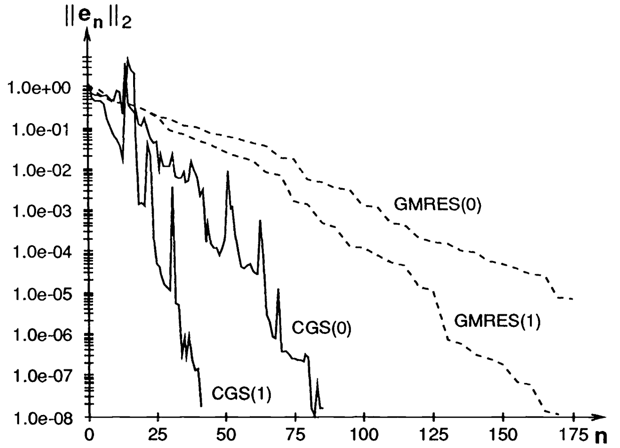


FIG. 4. *Convergence curves for the discrete minority continuity equation of Example* 2.

Among the iterative procedures we feel that the CGS method is the most versatile. It avoids possible numerical problems with the symmetrization matrices and is applicable to "real" nonsymmetric problems. Regarding convergence speed the GMRES(5) algorithm is clearly exceeded.

More decisive than the iterative (acceleration) procedures is the choice of a robust parallelizable preconditioner. We have investigated incomplete LU factorization of levels 0–2 and we have shown that quite a high performance (exceeding 100 megaflops on the Fujitsu VP200 computer) is reachable for the preconditioned matrix-vector multiply.

Acknowledgments. The authors are indebted to Martin Schubert and Hans-Peter Falkenburger from the Institute of Microelectronics, Stuttgart; Dr. Martin Thurner from the Campus-based Engineering Center Vienna; and H. Dietrich, G. Koessl, and H. Wiktorin from the Computer Services of Cooperate Research and Development, Siemens, Munich. The authors wish to thank the anonymous referees for valuable suggestions.

REFERENCES

[1] U. Ascher, P. Markovich, C. Schmeiser, H. Steinrück, and R. Weiss, *Conditioning of the steady state semiconductor device problem*, SIAM J. Appl. Math., 49 (1989), pp. 165–185.

[2] C. C. Ashcraft and R. G. Grimes, *On vectorizing incomplete factorization and* SSOR *preconditioners*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 122–151.

[3] R. E. Bank, D. J. Rose, and W. Fichtner, *Numerical methods for semiconductor device simulation*, IEEE ED-30 (1983), pp. 1031–1041.

[4] P. Concus, G. H. Golub, and G. Meurant, *Block preconditioning for the conjugate gradient method*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 220–252.

[5] S. Doi and N. Harada, *Tridiagonal factorization algorithm: A preconditioner for nonsymmetric system solving on vectorcomputers*, J. Inform. Process., 11 (1987), pp. 38–46.

[6] H. C. Elman, Y. Saad, and P. E. Saylor, *A hybrid Chebyshev Krylov subspace algorithm for solving nonsymmetric systems of linear equations*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 840–855.

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

[8] H. K. Gummel, *A selfconsistent iterative scheme for one-dimensional steady state transistor calculations*, IEEE ED-11 (1964), pp. 455-465.

[9] I. Gustafsson, *A class of first order factorization methods*, BIT, 18 (1978), pp. 142–156.

[10] M. H. Gutknecht, *The unsymmetric Lanczos algorithms and their relations to Padé approximations, continued fractions and the QD algorithm*, in Proceedings of the Second Copper Mountain Conference on Iterative Methods, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990.

[11] L. H. Hageman, F. T. Luk, and D. M. Young, *On the equivalence of certain iterative methods*, SIAM J. Numer. Anal., 17 (1980), pp. 852–873.

[12] K. Hane, *Supercomputing for process/device simulations*, in Proc. Sixth Internat. NASECODE Conference, J. J. H. Miller ed., Trinity College, Boole Press, Ltd., Dublin, Ireland, 1989, pp. 11–21.

[13] W. Hänsch and S. Selberherr, MINIMOS 3: *A* MOSFET *simulator that includes energy balance*, IEEE ED-34 (1978), pp. 1074–1078.

[14] C. den Heijer, *Preconditioned iterative methods for nonsymmetric linear systems*, in Proc. Int. Conf. on Simulation of Semiconductor Devices and Processes, Pineridge Press, Swansea, U.K., 1984, pp. 267–285.

[15] T. Kerkhoven, *On the effectiveness of Gummel's method*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 48–60.

[16] T. A. Manteuffel, *The Tchebychev iteration for nonsymmetric linear systems*, Numer. Math., 28 (1977), pp. 307–327.

[17] H. Meijerink and H. 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.

[18] T. C. Oppe, W. D. Joubert, and D. R. Kincaid, NSPCG *User's Guide*, Center of Numerical Analysis, University of Texas, Austin, TX, 1984.

[19] S. J. Polak, C. den Heijer, W. H. Schilders, and P. Markovich, *Semiconductor device modelling from the numerical point of view*, Internat. J. Numer. Methods Engrg., 24 (1987), pp. 763–838.

[20] E. L. Poole and J. M. Ortega, *Multicolor* ICCG *methods for vector computers*, SIAM J. Numer. Anal., 24 (1987), pp. 1394–1418.

[21] C. S. Rafferty, M. R. Pinto, and R. W. Dutton, *Iterative Methods in Semiconductor Device Simulation*, IEEE ED-32 (1985), pp. 2018–2027.

[22] Y. Saad and M. H. Schultz, GMRES: *A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.

[23] Y. Saad, *Krylov subspace methods on supercomputers*, SIAM J. Sci. Stat. Comput., 10 (1989), pp. 1200–1232.

[24] D. L. Scharfetter and H. K. Gummel, *Large–signal analysis of a silicon read diode oscillator.*, IEEE ED-16 (1969), pp. 64-77.

[25] S. Selberherr, *MINIMOS* 5 *Users's Guide*, Institute for Microelectronics, Technical University of Vienna, Vienna, Austria, 1990.

[26] P. Sonneveld, CGS, *A fast Lanczos-type solver for nonsymmetric systems*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 36–52.

[27] M. Thurner P. Lindorfer and S. Selberherr, *Numerical treatment of nonrectangular field-oxide for* 3D *MOSFET simulation*, IEEE CAD-9 (1990), pp. 1189–1197.

[28] T. Toyabe, H. Masuda, Y. Aoki, H. Shukuri, T. Hagiwara, *Three-dimensional device simulator CADDETH with highly convergent matrix solution algorithms*, IEEE ED-32 (1985), pp. 2038–2044.

[29] H. Vorst, *High performance preconditioning*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 1174–1185.

[30] ———, *Large tridiagonal and block tridiagonal linear systems on vector and parallel computers*, Parallel Comput., 5 (1987), pp. 45-54.

[31] ———, *A vectorizable variant of some* ICCG *methods*, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 350–356.

[32] A. Yoshii, M. Tomizawa, and K. Yokoyama, *Investigation of numerical algorithms in semiconductor device simulation*, Solid-State Electronics, 30 (1987), pp. 913-820.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 1, pp. 307–318, January 1992

THE HIERARCHICAL BASIS EXTRAPOLATION METHOD*

U. RÜDE[†]

Abstract. Traditional extrapolation methods, like Richardson's extrapolation, are based on asymptotic error expansions of the solution, and thus depend on the use of uniform grids. Within the hierarchical basis framework for the finite element method it is natural to consider an extrapolation of the *energy functional.* This idea is based on a purely local element-by-element analysis and is applicable to irregular and unstructured meshes. It will be demonstrated that *higher order* discretization can be obtained by this type of extrapolation without imposing any additional restrictions on the finite element mesh. This method is especially well suited for use in combination with local refinement strategies. If combined with a multilevel solution procedure, the algorithm is closely related to multigrid *tau-extrapolation.* This paper will present the theoretical background and experimental results for the energy extrapolation method in hierarchical basis formulation.

Key words. elliptic partial differential equations, finite element method, hierarchical basis, extrapolation

AMS(MOS) subject classifications. 65N30, 65N50, 65N55, 65F10

1. Introduction.

For truly efficient elliptic solvers four conceptual components will be necessary:

1. adaptive refinement,
2. higher order discretization,
3. multilevel solvers,
4. advanced software techniques.

It is now widely acknowledged that for the effective solution of most *real-life* applications some form of adaptive discretization is needed. Different meshsizes in different parts of the solution domain are necessary to model the locally varying features of the solution effectively.

The need for higher order discretization has been less apparent. However, when high accuracy is desired, then by exploiting the smoothness of the solution in certain regions of the domain with higher order approximations, much better efficiency can be obtained. In most practical applications nonsmooth behavior in elliptic problems occurs at isolated points, typically singularities introduced by source terms, corners in the boundary, or discontinuities in the coefficients. Away from those points, the solutions of elliptic problems are smooth, typically even C^∞. In these parts of the domain higher order should be used.

Even with higher order schemes, the discretization will lead to systems with many unknowns. For these large scale applications we believe that only multilevel techniques can provide satisfactory efficiency.

Finally, the resulting complexity of the software must not be underestimated. We believe that the present software techniques are not powerful enough to provide a sufficient basis for implementing these algorithms effectively.

This paper will describe an extrapolation-based approach to obtain higher order finite element discretizations that is especially well suited to be integrated with adaptive techniques and multilevel solution strategies.

* Received by the editors April 5, 1990; accepted for publication (in revised form) November 4, 1990. This research was partially supported by Deutsche Forschungsgemeinschaft grant Ru 422/1.

[†] Institut für Informatik, Technische Universität München, Arcisstr. 21, D-8000 München 2, Germany (ruede@lan.informatik.tu-muenchen.dbp.de or na.ruede@na-net.ornl.gov).

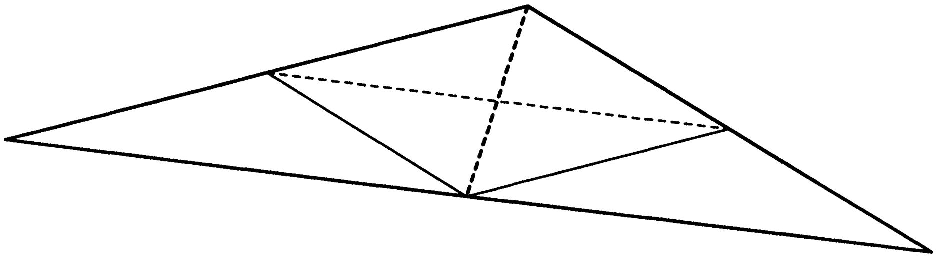


FIG. 1. *Modified refinement in triangles with an obtuse angle.*

After briefly introducing our overall concept, we will derive the *energy extrapolation*, a technique based on purely local elementwise analysis and therefore applicable to all irregular and unstructured meshes. This technique will be shown to be equivalent to the use of higher order finite elements.

2. The adaptive hierarchical algorithm. Our refinement strategy is essentially based on the ideas implemented in Bank's PLTMG [1]. It has been adapted such that the finite element spaces constructed during refinement are truly nested. This supports the multilevel solution procedure on the one side and the energy extrapolation on the other side.

Starting from a user-supplied primary triangulation, finer levels are constructed by partial refinement. Using error estimation techniques, (or a priori knowledge about local meshsizes) certain elements or edges are flagged for refinement.

Next, care must be taken that the new triangulation becomes consistent. This is implemented by a (recursive) procedure, imposing the following rules:

1. If the shortest side of a triangle is flagged for refinement, then all other edges will also be marked.
2. If two edges of a triangle are marked for refinement, the third will also be marked.

Now, a *solution space* with more degrees of freedom than the previous one is created by halving all flagged edges.

In case all three sides of a triangle are flagged, the midpoints of these sides are connected, creating four congruent new triangles. This is called regular refinement.

In case only one side is flagged, the midpoint of the edge is connected to the opposite vertex. It is this step that may cause trouble by creating triangles with large obtuse angles that in turn have unfavorable properties in the finite element solution. Rule 1 above alleviates this problem, but cannot guarantee a limit on the angles that occur. Additionally, it must be assumed that the error indicator behaves reasonably. Furthermore, as a user-selectable option, the regular refinement may be replaced by the refinement shown in Fig. 1 for triangles with an obtuse angle.

In our experiments this refinement strategy has worked well. It is simpler than the refinement algorithms used in Bank's PLTMG, and, in particular, it avoids the complication introduced by temporary edges that must be removed before further refinement levels are created. Our approach has the further advantage that the finite element spaces of coarse levels are true subspaces of finer levels—a feature that is frequently used in theoretical studies of multilevel solution methods.

We assume that the domain and its internal interfaces can be modeled with relatively few (curvilinear) elements, which make up the primary mesh, and that the number of elements necessary to obtain final accuracy is much larger. If the primary mesh contains very few degrees of freedom, direct solvers are feasible, and it can become the coarsest level of a multilevel solver.

If this is not the case, sparse solvers or multileveling "below" the primary triangulation is needed and we suggest using algebraic multigrid (AMG) (see Ruge and Stüben [10]) as a convenient way to construct "coarser" levels relative to the primary mesh. In the following we will assume that the solution on the primary triangulation can be efficiently obtained by some unspecified method and that the amount of work is negligible compared to the work required for the final solution.

Thus we now focus on how to obtain solutions on the increasingly finer meshes. First we notice that coarse grid solutions may be simply interpolated to finer grids, providing good guesses for starting an iterative solver.

To our knowledge two iterative solution techniques are the most promising: the hierarchical basis technique that, together with conjugate gradient acceleration, promises an $O(n \log(n))$ work estimate for n degrees of freedom, see Yserentant [11]. The alternative is a true multigrid-like solution technique, as proposed, e.g., in Brandt [3], or by McCormick [6].

The hierarchical basis technique has the advantage of using smoothing only on the nodes that are *newly added* in each particular step of refinement, which clearly limits the work to $O(n)$ per cycle, no matter how the points are distributed across different levels. The price for this nice feature is a slowly deteriorating convergence rate when the levels get finer—$O(\log(n))$ iterations are necessary to reduce the error by a prescribed factor (in two dimensions).

A multigrid-like approach (cf. MLAT, Brandt [3], or FAC, McCormick [6]) differs from the hierarchical basis algorithm by smoothing not only the newly added points but also the coarse grid points between them. For regular problems this will provide convergence rates independent of the number of levels. Recent results show (see Leinen [12]) that even without regularity assumptions a local refinement algorithm with multigrid-like processing will obtain convergence rates that are at least as good as hierarchical basis rates with conjugate gradient acceleration, and are superior in three dimensions, where the hierarchical basis algorithm's complexity further deteriorates.

Finally, we remark that in contrast to McCormick's FAC method, where processing is based on subgrids, the processing in our algorithms will be based on *global grids*, corresponding to the *composite grids* in FAC. In the most straightforward implementation we thus smooth in the whole domain including subdomains where no refinement has taken place. This is not optimal, because too much smoothing will be applied to points that have not been refined. Such points are smoothed on several levels. For these points coarsening and interpolation become just trivial copying operations for the residuals and solution values, respectively.

But, if the number of *gridpoints* grows geometrically with a factor larger than 1.0 between any two levels, this very simple strategy combined with a V-cycle will already have asymptotically optimal complexity. Furthermore, it of course makes the algorithm a special case of a regular, variational multigrid algorithm on global grids so that the theoretical results for this case become applicable.

3. The energy extrapolation. A multilevel algorithm generates an unusual wealth of information about the solution. Obviously, monitoring the change in the approximate solution from one level to the next provides insight about the local errors that may be used for simple error estimation techniques. On the other hand, an estimate of the local error for any given approximation may be used to improve the accuracy of this approximation. Thus it is quite natural that one attempts to use

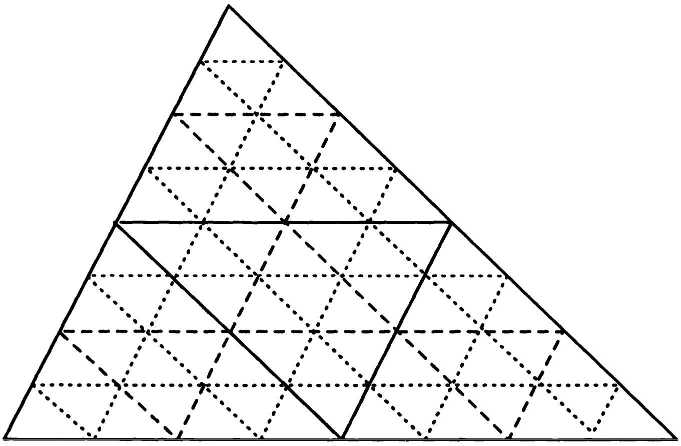


FIG. 2. *Regular refinement of a triangle.*

the extra information provided by multileveling to improve the accuracy. This is the basic idea of extrapolation.

The theoretical foundation of classical extrapolation is based on asymptotic error expansions, which, however, are a feature of using regular grids. The straightforward application of Richardson's extrapolation to the solution does not seem possible on unstructured finite element meshes. General finite element meshes as used in our approach typically only provide *error bounds*, not *error expansions*, as required for extrapolation.

The key for developing a suitable extrapolation technique is based on the variational principle. In a variational setting, the problem is to find a function that minimizes an expression like

$$\min_{u \in H}(E_\Omega[u] - (u, f)) \tag{1}$$

in some function space H. $E_\Omega[u]$ often has a physical interpretation as an energy. In this paper we are concerned with second order linear elliptic selfadjoint problems in two space dimensions, so that we can assume that E_Ω is of the form

$$E_\Omega[f] = \int\int_\Omega a(x,y)(\nabla f(x,y))^2 dxdy. \tag{2}$$

The numerical solution of (1) with finite elements is now closely related to finding numerical approximations for $E_\Omega[u]$.

Consider a single triangle $K = K^0$ of the mesh and a sequence of regular refinements $K^n, n = 1, 2, \cdots$, of this particular triangle, see Fig. 2. For any function $f(x, y)$ on the triangle, a numerical method to evaluate its energy can be defined by projecting f to the space of piecewise linears on K^n (by taking the interpolant), and calculating the energy of the projection. (This can be done exactly.) Thus we define a sequence of numerical methods $E_K^h[f]$ for calculating the energy $E_K[f]$. Note, that E_K^h gives the exact energy if f is piecewise linear (on the corresponding triangulation), and thus the integration method implicitly defines the finite element method (for linear elements).

Next, it can be shown that this sequence of approximations to the energy has an h^2-expansion in the following sense: If $f \in C^{2N+3}(K)$ and $a(x,y) \in C^{2N+2}(K)$, then

$$E_K^h[f] - E_K[f] = h^2 e_1 + h^4 e_2 + \cdots + h^{2N} e_N + h^{2N+2} R_{N+1},$$

where $h = 2^{-n} h_0$, h_0 the diameter of K^0, e_k are constants independent of h, and R_{N+1} a remainder term that satisfies $|R_{N+1}| \leq c$, where c is a constant independent of h. This is related to the fact that the integration rule resembles the trapezoidal rule. Using extrapolation (for h^2-expansions), we can thus calculate the energy of any sufficiently smooth function to $O(h^4)$ by one step of extrapolation, to $O(h^6)$ by two steps, etc. The existence of such an expansion has been proved in Rüde [9].

The remainder term depends on f and a. More precisely, for constant a, it depends on the derivatives of f in such a way that it vanishes if the derivatives of f of order $N+2$ vanish. In particular, if f is a polynomial of degree 1, (linear), then R_1 vanishes, which means that the error expansion degenerates: linear functions are represented correctly on all refinement levels.

Similarly, when f is polynomial of degree 2, then R_2 vanishes, so that the expansion has exactly one term. This means we can integrate quadratics correctly by using one step of extrapolation:

$$E_K[f] = 4/3 E_K^{h/2}[f] - 1/3 E_K^h[f].$$

These results have also been proved in Rüde [9].

Observe now that the extrapolated value is calculated by a linear combination of two bilinear forms based on the nodal values. The linear combination of bilinear forms is a bilinear form.

Considering the refined triangle as a new macro element, we see that the above constructed bilinear form must correspond to the element stiffness matrix for piecewise quadratics on the macro element. This stiffness matrix is uniquely defined for a given finite element space and basis.

Next, this carries over to the collection of all elements, providing us with an alternative method to generate a stiffness matrix for quadratic elements by taking a linear combination of the stiffness matrices for linear elements. In this process it is only required that each element be refined regularly.

This argument is still element by element. The error terms of the expansion are canceled within each element. The higher order terms and remainder terms that are not canceled by the extrapolation are then accumulated with possibly many different values of h and thus cause a perturbation to the solution. This perturbation in general does not have an asymptotic expansion, however, because the low order terms have been canceled elementwise, the perturbation must be of higher order.

To exploit this idea algorithmically we must now look at the algebraic structure of the problem. Assume that we have a nested sequence of triangulations

$$T^1 \subseteq T^2 \subseteq T^3 \cdots, \tag{3}$$

and a corresponding sequence of nested, linear finite element spaces

$$V^1 \subseteq V^2 \subseteq V^3 \cdots. \tag{4}$$

An element $v^k \in V^k$ will interchangeably be interpreted as a piecewise linear finite element function or as a vector of nodal values on T^k. The nesting of meshes induces

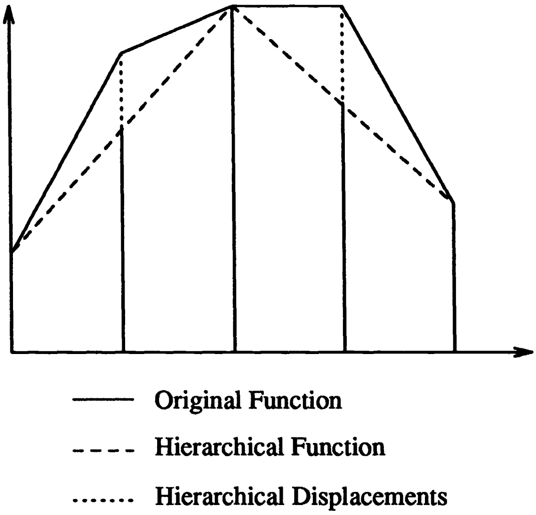

—— Original Function
---- Hierarchical Function
.... Hierarchical Displacements

FIG. 3. *Hierarchical decomposition.*

a partitioning of v^k, $k > 1$:

$$v^k = \begin{bmatrix} v_C^k \\ v_F^k \end{bmatrix}, \tag{5}$$

where v_C^k is associated with the nodes on $T^{k-1} \subseteq T^k$ and v_F^k is associated with the nodes that are in T^k but not in T^{k-1}. Next, define $\bar{P}_{k-1}^k : V^{k-1} \longrightarrow V^k$ as the natural prolongation (linear interpolation). In nodal representation and following the partitioning in (5)

$$\bar{P}_{k-1}^k = \begin{bmatrix} I \\ P_{k-1}^k \end{bmatrix}. \tag{6}$$

P_{k-1}^k is the operator forming the arithmetic average of two values at vertices connected by an edge of the coarse triangulation.

Using this mapping, the *hierarchical transform* $\hat{v}^k$ of $v^k \in V^k$ is defined by

$$v^k = \begin{bmatrix} v_C^k \\ v_F^k \end{bmatrix} \xrightarrow{H_k^{k-1}} \hat{v}^k = \begin{bmatrix} v_C^k \\ \hat{v}_F^k \end{bmatrix} = H_k^{k-1} \begin{bmatrix} v_C^k \\ v_F^k \end{bmatrix} = \begin{bmatrix} I & 0 \\ -P_{k-1}^k & I \end{bmatrix} \begin{bmatrix} v_C^k \\ v_F^k \end{bmatrix}. \tag{7}$$

A one-dimensional example is shown in Fig. 3. Note that the inverse transform reads

$$(H_k^{k-1})^{-1} = \begin{bmatrix} I & 0 \\ P_{k-1}^k & I \end{bmatrix}. \tag{8}$$

A symmetric bilinear form $L^k : V^k \times V^k \longrightarrow \mathcal{R}$ can be partitioned in a compatible way

$$L^k = \begin{bmatrix} C^k & X^k \\ (X^k)^T & F^k \end{bmatrix}, \tag{9}$$

and be transformed to

$$\hat{L}^k = \begin{bmatrix} I & (P_{k-1}^k)^T \\ 0 & I \end{bmatrix} L^k \begin{bmatrix} I & 0 \\ P_{k-1}^k & I \end{bmatrix} = \begin{bmatrix} \hat{C}^k & \hat{X}^k \\ (\hat{X}^k)^T & F^k \end{bmatrix}, \tag{10}$$

where

$$\hat{C}^k = C^k + (P_{k-1}^k)^T (X^k)^T + X^k P_{k-1}^k + (P_{k-1}^k)^T F^k P_{k-1}^k, \tag{11}$$

$$\hat{X}^k = X + (P_{k-1}^k)^T F. \tag{12}$$

Now observe that solving

$$\min_{v^k \in V^k} \left\{ (v^k)^T L^k v^k - 2(v^k)^T f^k \right\} \tag{13}$$

is equivalent to solving the transformed minimization problem

$$\min_{\hat{v}^k \in V^k} \left\{ (\hat{v}^k)^T \hat{L}^k \hat{v}^k - 2(\hat{v}^k)^T \begin{bmatrix} I & (P_{k-1}^k)^T \\ 0 & I \end{bmatrix} f^k \right\}. \tag{14}$$

The *coarse grid block* $\hat{C}^k$ in the transformed matrix is L^{k-1}, that is, the finite element stiffness matrix for a coarser problem. This can be seen by interpreting the coarse grid part of (14) as the problem of minimizing the energy in the coarse grid finite element space.

Based on this formalism, a two-level hierarchical basis algorithm consists of the alternating solution (minimization) for v_C^k (keeping $\hat{v}_F^k$ frozen) and for $\hat{v}_F^k$ (keeping v_C^k frozen). Another interpretation for this is a block Gauss–Seidel procedure.

In this setting, the minimization for v_C is fully equivalent to a multigrid coarse grid correction (for variational multigrid). A good approximate solve for $\hat{v}_F^k$ can be implemented by a few relaxation steps for the corresponding unknowns. This is the form of a hierarchical basis multigrid algorithm as it has been suggested by Bank, Dupont, and Yserentant [2]. Except that smoothing is restricted to the F-unknowns, the hierarchical basis algorithm is a usual multigrid algorithm.

We will go a step further and reintroduce full smoothing making the method fully equivalent to a variational multigrid algorithm. This so-called *hierarchical transformation* multigrid algorithm (HTMG) has been introduced by Griebel [4], see also McCormick and Rüde [7].

A two-level variant of this algorithm is shown in Fig. 4. When step 4 is replaced by a recursive application of the same algorithm we get a V-cycle HTMG method. As mentioned before, it is fully equivalent to a regular (variational) multigrid algorithm. It provides a particularly efficient implementation of a *full approximation storage* (FAS) scheme. In some cases the HTMG implementation is even cheaper than classical *correction storage* schemes. For the details see Griebel [4].

Assume L^{k-1} is the stiffness matrix for a triangulation with linear elements, and L^k is the stiffness matrix for an associated regular (full) refinement. An approximation for the energy of any smooth function u represented by the nodal values $\left[\begin{smallmatrix} u_C \\ u_F \end{smallmatrix}\right]$ is given by $u_C^T L^{k-1} u_C$ and $[u_C^T u_F^T]\, L^k \left[\begin{smallmatrix} u_C \\ u_F \end{smallmatrix}\right]$, respectively. In this notation our results on asymptotic expansions of the energy become

$$[u_C^T u_F^T]\, L^k \begin{bmatrix} u_C \\ u_F \end{bmatrix} = E_\Omega[u] + h^2 e,$$

1. Perform ν_1 relaxations on $L^k v^k = f^k$.
2. Perform the hierarchical transform $v^k \to \hat{v}^k$ (7).
3. Calculate the coarse grid right-hand side $f^{k-1} = f_C^k + (P_{k-1}^k)^T f_F - \hat{X}^k \hat{v}^k$.
4. Solve the coarse grid system $L^{k-1} v^{k-1} = f^{k-1}$.
5. Calculate the corrected fine grid values by reversing (7):
$$v^k = \begin{bmatrix} v_C^k \\ v_F^k \end{bmatrix} \longleftarrow \begin{bmatrix} v^{k-1} \\ P_{k-1}^k v^{k-1} + \hat{v}_F^k \end{bmatrix}.$$
6. Perform ν_2 relaxations on $L^k v^k = f^k$.

FIG. 4. *Basic two-level solution algorithm.*

provided u_C are the nodal values of a function that is piecewise quadratic on the triangulation. Under the same assumptions

$$u_C^T L^{k-1} u_C = E_\Omega[u] + 4h^2 e,$$

such that the usual extrapolation gives

$$\frac{4}{3} \left[u_C^T u_F^T\right] L^k \begin{bmatrix} u_C \\ u_F \end{bmatrix} - \frac{1}{3} u_C^T L^{k-1} u_C = E_\Omega[u]. \tag{15}$$

This is now an *exact* representation of the energy for quadratics based on the energy for linear functions. The formula further simplifies when hierarchically transformed systems are used. Equation (15) is equivalent to

$$\frac{4}{3} \left[u_C^T u_F^T\right] L^k \begin{bmatrix} u_C \\ u_F \end{bmatrix} - \frac{1}{3} u_C^T L^{k-1} u_C = \left[u_C^T \hat{u}_F^T\right] \begin{bmatrix} L^{k-1} & \frac{4}{3}\hat{X}^k \\ \frac{4}{3}(\hat{X}^k)^T & \frac{4}{3}F \end{bmatrix} \begin{bmatrix} u_C \\ \hat{u}_F \end{bmatrix}. \tag{16}$$

In the hierarchical basis representation higher order is simply obtained by multiplying certain matrix entries by the extrapolation factor 4/3.

Finally, we must apply an analogous extrapolation for the inner product

$$(f, u) = \int\int_\Omega f(x, y) u(x, y)\, dxdy$$

to get better representations of the right-hand side. In the basic (linear element) algorithm the inner product is approximated by

$$\int\int_K u(x, y) f(x, y)\, dxdy \approx 1/3\, \text{area}(K)(u(a)f(a) + u(b)f(b) + u(c)f(c)),$$

where a, b, c are the corners of K. Thus in the basic system, the right-hand side value for node i is given by

$$f_i = 1/3 f(x_i, y_i) \sum_{(x_i, y_i) \text{vertex of } K} \text{area}(K).$$

This integration rule can also be shown to have an h^2-expansion, so that the extrapolated system genuinely provides higher order accuracy. The extrapolated equations can be written as

$$\begin{bmatrix} L^{k-1} & \frac{4}{3}\hat{X}^k \\ \frac{4}{3}(\hat{X}^k)^T & \frac{4}{3}F \end{bmatrix} \begin{bmatrix} u_C \\ \hat{u}_F \end{bmatrix} = \begin{bmatrix} \frac{4}{3}(P_{k-1}^k)^T f_F \\ \frac{4}{3} f_F \end{bmatrix}. \tag{17}$$

Note that in the extrapolation of the right-hand side the scaling of the equations has to be taken into account.

This system can now be solved by a standard hierarchical basis procedure. However, when we wish to improve the HB-performance by introducing full grid smoothing, a slight problem occurs. Such a smoothing on the untransformed system is incompatible with the higher order accuracy of the extrapolated system. There are two possible solutions:

- Ignore the problem and smooth anyway. In McCormick and Rüde [7] it has been shown that the resulting method is equivalent to a regular multigrid algorithm using τ-extrapolation. For this case theoretical results have been proven (see Hackbusch [5] or Rüde [8]) that show that the loss in accuracy due to the wrong smoother is not worse than the discretization error itself. This is based on the fact that smoothing affects mainly only high frequency solution components that are not important for the approximation accuracy. If this approach is chosen, the extrapolation only affects step 3 in the basic algorithm of Fig. 4 that must be changed to $f^{k-1} = 4/3(P^k_{k-1})^T f_F - 4/3\hat{X}^k\hat{v}^k$.
- It is also possible to design smoothers fully compatible with the modified system. For this we use a two-level smoothing technique. Here step 1 and step 6 of Fig. 4 are each replaced by a two-level algorithm that has the same basic structure as the basic two-level algorithm itself:

  S1. Perform ν_1 relaxations on the F-nodes of $L^k v^k = f^k$.
  S2. Perform the hierarchical transform $v^k \rightarrow \hat{v}^k$.
  S3. Calculate the coarse grid right-hand side $f^{k-1} = 4/3(P^k_{k-1})^T f_F - 4/3\hat{X}^k\hat{v}^k$ with the extrapolated equations.
  S4. Relax the coarse grid system $L^{k-1}v^{k-1} = f^{k-1}$ using v^k_C as a starting guess.
  S5. Calculate the corrected fine grid values by

$$v^k = \begin{bmatrix} v^k_C \\ v^k_F \end{bmatrix} \longleftarrow \begin{bmatrix} v^{k-1} \\ P^k_{k-1}v^{k-1} + \hat{v}^k_F \end{bmatrix}.$$

  S6. Perform ν_2 relaxations on $L^k v^k = f^k$.

  An algorithm involving this smoother then looks like a hierarchical basis algorithm (with fine point smoothing only) and a nonstandard cycling strategy that involves *short* visits to the coarser levels without full recursion. This more complicated strategy is of course only necessary on the finest level, where the extrapolation applies. On all other levels a standard smoother may be used.

4. Numerical experiments. Our example problem is

$$\begin{aligned} \Delta u &= 0 \text{ on } (0,1)^2, \\ u &= \frac{\cos(4\pi(x-y))\sinh(4\pi(2-x-y))}{\sinh(8\pi)} \quad \text{on } \partial[0,1]^2. \end{aligned} \tag{18}$$

The boundary function also describes the true solution. This function has value 1 at $(0,0)$ and decays exponentially to 0 in the solution domain. Though there is increased activity in the southwest corner of the domain we start out by solving on a regular *square mesh triangulation.* The first of these triangulations is obtained by halving the square with the southwest to northeast diagonal. Then full refinement steps are used to construct a sequence of uniform triangulations. For linear elements this discretization is equivalent to a 5-point difference stencil. In Table 1 the straightforward

TABLE 1
Solution of example 1 with global refinement.

Level	#nd	No extrapol.		Two-level smooth		Regular smooth	
		L^2	energy	L^2	energy	L^2	energy
2	25	7.55e-3	7.08e-2	8.11e-3	7.26e-2	7.66e-3	7.13e-2
3	81	7.78e-3	1.35e-1	7.12e-3	1.32e-1	7.50e-3	1.33e-1
4	289	2.51e-3	6.03e-2	1.32e-3	5.24e-2	1.86e-3	5.37e-2
5	1089	6.63e-4	1.82e-2	1.42e-4	1.18e-2	2.77e-4	1.17e-2
6	4225	1.68e-4	4.81e-3	1.13e-5	1.92e-3	2.74e-5	1.71e-3
7	16641	4.22e-5	1.22e-3	8.23e-7	2.75e-4	2.06e-6	1.97e-4
rate		1.99	1.98	3.79	2.80	3.73	3.11

full multigrid algorithm is compared to a version where extrapolation is applied every time on the finest grid. We further distinguish between a variant using a regular smoother and one that uses the two-level smoothing technique (see the discussion at the end of the previous section). In order to keep algebraic errors low, two V-cycles are used on each level. We are mainly interested in *differential* errors, that is, errors of the discrete solution with respect to the analytic solution. We display discrete Euclidean and discrete energy-norm error estimates. Note that the superiority of the higher order schemes only shows for finer meshes but that the extrapolation saves at least one level of refinement, when we need high accuracy. Further note that both algorithms with extrapolation behave about the same; the algorithm with regular smoothing seems to provide even somewhat lower energy errors. The experimentally determined convergence rates of the Euclidean norm indicate $O(h^2)$ convergence for the linear algorithm and $O(h^4)$ accuracy for the two extrapolated algorithms. In the energy norm the extrapolated algorithms exhibit $O(h^3)$ behavior.

The next example will apply adaptive refinement to the same problem. The algorithm has the structure of a nested iteration starting on the coarsest level.

On any level:

1. Perform a full (regular) refinement step.
2. Interpolate the solution to the new nodes, and perform a few relaxations on the new nodes to dampen high frequency interpolation errors.
3. Using the extrapolated stiffness matrix, calculate the solution with quadratic order accuracy using an extrapolation-V-cycle. (The results of Table 2 refer to regular smoothing. Using the more complicated two-level smoothing technique did not yield significantly better results.)
4. Compare with the coarser level solution to decide which of the new nodes are relevant for the improvement in accuracy.
5. Delete the fine grid nodes that (presumably) have not improved the accuracy.
6. If final accuracy has not been obtained yet, make this level the current level and go to step 1.

Remark. After deleting the extra nodes of a full refinement, there is no new solution phase, assuming that we have only deleted nodes that would cause negligible changes to the solution (relative to the accuracy on that level).

To this end note that asymptotically the exact solution behaves like $u = e^{4\pi(1-r)}$, where r is the distance from $(x, y) = (0, 0)$ and when we only consider the radial dependence. Thus (for a second order discretization) the truncation error asymptotically behaves like $e_t = h^2(4\pi)^4 e^{4\pi(1-r)}$. In order to achieve equally distributed truncation errors, we introduce variable meshsizes in the form

$$h(r) = He^{-4\pi(1-r)/2},$$

TABLE 2
Solution of example 1 *with local refinement.*

Level	#nd	No extrapol.		#nd	With extrapol.	
		L^2	energy		L^2	energy
2	25	7.56e-3	7.09e-2	25	8.11e-3	7.26e-2
3	43	7.68e-3	1.34e-1	43	7.10e-3	1.32e-1
4	77	2.71e-3	1.83e-2	139	1.32e-3	5.25e-2
5	191	7.11e-4	1.83e-2	331	1.44e-4	1.19e-2
6	683	1.80e-4	4.85e-3	1241	1.15e-5	1.95e-3
7	2167	4.84e-5	1.48e-3	4427	8.08e-7	2.77e-4
8	7985	1.22e-5	4.26e-4	17539	5.46e-8	3.73e-5
rate		1.99	1.80		3.88	2.98

where H is a *global* meshsize parameter. For the higher order discretization we use accordingly

$$h(r) = He^{-4\pi(1-r)/4}.$$

We use these relationships to guide the local refinement: The edges of a node are refined, when they are longer than the h associated with the position of that node. The rules of §2 are used to complete the creation of a consistent new level. The results are displayed in Table 2.

It is also instructive to compare Tables 1 and 2. Note that the local refinement algorithm obtains almost the same accuracy on each refinement level as the global refinement algorithm, however with a much reduced number of nodes. This shows that we have really only omitted nodes that are irrelevant for obtaining the solution accuracy.

5. Conclusions. We have introduced the energy extrapolation technique, a method that obtains higher order based on a local extrapolation that can be used on unstructured finite element meshes. This approach is especially well suited for a combination with adaptive refinement and multilevel solution. For model problems we demonstrate the superiority of the integrated algorithm.

Acknowledgments. The author wishes to thank Steve McCormick and the Computational Math. Group at the University of Colorado at Denver for their hospitality and many helpful discussions. The author is also indebted to the referees for some valuable remarks.

REFERENCES

[1] R. BANK, *PLTMG Users' Guide*, Edition 4.0, Department of Mathematics, University of California at San Diego, CA, 1985.

[2] R. BANK, T. DUPONT, AND H. YSERENTANT, *The Hierarchical Basis Multigrid Method*, Konrad Zuse Zentrum Preprint, SC-87-2, April 1987, Berlin.

[3] A. BRANDT, *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics*, GMD Studien 85, 1984.

[4] M. GRIEBEL, *Zur Lösung von Finite-Differenzen- und Finite-Element-Gleichungen mittels der Hierarchischen-Transformations-Mehrgittermethode*, Doctoral Dissertation, Technische Universität München, Germany, 1989.

[5] W. HACKBUSCH, *Multigrid Methods and Applications*, Springer-Verlag, Berlin, 1985.

[6] S. MCCORMICK, *Multilevel Adaptive Methods for Partial Differential Equations*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989.

[7] S. MCCORMICK AND U. RÜDE, *On Local Refinement Higher Order Methods for Elliptic Partial Differential Equations*, Internat. J. High Speed Comput., 2 (1990), pp. 311–334.

[8] U. Rüde, *Multiple tau-Extrapolation for Multigrid Methods*, Institut für Informatik, Technische Universität München, I-8701, 1987.

[9] ———, *Extrapolation Techniques for Constructing Higher Order Finite Element Methods*, Institut für Informatik, Technische Universität München, in preparation.

[10] J. Ruge and K. Stüben, *Efficient Solution of Finite Difference and Finite Element Equations by Algebraic Multigrid (AMG)*, Arbeitspapiere der GMD, 89, 1984.

[11] H. Yserentant, *On the multi-level splitting of finite element spaces*, Numer. Math., 49 (1986), pp 379-412.

[12] P. Leinen and H. Yserentant, *Two fast solvers based on the multi-level splitting of finite element spaces*, Proceedings of the 3rd European Conference on Multigrid Methods, Bonn, Oct. 1-4, 1990, W. Hackbusch and U. Trottenberg eds., Burkhäuser Verlag, Basel, 1991.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 1, pp. 319–338, January 1992

017

FOURIER ANALYSIS OF INCOMPLETE FACTORIZATION PRECONDITIONERS FOR THREE-DIMENSIONAL ANISOTROPIC PROBLEMS*

JUNE M. DONATO[†] [‡] AND TONY F. CHAN[†]

Abstract. To solve three-dimensional elliptic problems using preconditioned conjugate gradient, it is crucial to make a good choice of preconditioner. To facilitate this choice, a Fourier analysis technique has been used by Chan and Elman [*SIAM Rev.*, 31 (1989), pp. 20–49.] and others to study preconditioned systems arising from the discretization of the two-dimensional model elliptic equation. In this paper the same technique is used to analyze relaxed-modified incomplete factorization preconditioned systems that arise from the discretization of a three-dimensional anisotropic elliptic problem. Expressions for the "Fourier eigenvalues" of the preconditioned three-dimensional systems are presented along with estimates of the condition numbers. For MILU, an optimal value for the parameter c is derived. The correlation between the distribution of the eigenvalues and the Fourier results for the preconditioned systems is remarkable. From the expressions for the eigenvalues we prove that $\kappa(M^{-1}A)$ is order h^{-2} for ILU and order h^{-1} for MILU$(c \neq 0)$. Then by examining the distribution of Fourier eigenvalues, the dependence of PCG convergence rate on the clustering of the eigenvalues of an operator, as well as its condition number, can be exemplified. The PCG experiments were performed on an Alliant FX/80.

Key words. Fourier analysis, three-dimensional problems, periodic and Dirichlet boundary conditions, condition numbers

AMS(MOS) subject classifications. primary 65F10, 65N20; secondary 15A06

1. Introduction.

While *preconditioned conjugate gradient* (PCG) is a widely used method of solving systems arising from the discretization of elliptic partial differential equations (PDEs), its performance is highly dependent upon the preconditioner chosen. For two-dimensional discretized elliptic PDEs, much theory and experimental background exists for the use of *incomplete LU* (ILU) and *modified incomplete LU* (MILU) preconditioned systems [3], [4], [6], [12], [13]. However for three-dimensional problems there is considerably less knowledge [1], [2], [14], [15]. Experimental difficulties arise because the discretization of three-dimensional problems leads to extremely large systems. Even though these systems are typically sparse, the space and time requirements are still daunting on most sequential machines. Hence, we see the move to parallel computers. But the choice of a "good" preconditioner still remains.

To facilitate this choice, a Fourier analysis technique has been used by Chan and Elman [7] and by Chan and Meurant [8] to study preconditioned systems arising from the discretization of the two-dimensional model elliptic equation. In this paper we use the same technique to analyze relaxed-modified incomplete factorization preconditioned systems that arise from the discretization of a three-dimensional anisotropic elliptic problem.

Begin by considering the matrix A arising from the discretization of three-dimensional anisotropic elliptic problems. For the preconditioner M we will examine (point) relaxed-modified incomplete LU factorizations. (For experiments using point and block methods, see [2], [15].) Using the Fourier technique of [7], the resulting

* Received by the editors April 5, 1991; accepted for publication (in revised form) January 22, 1991. This work was supported in part by Department of Energy contract DEFG03-87-ER-25037 and Army Research Office contract DAAL03-88-K-0085.

[†] Department of Mathematics, University of California, Los Angeles, California 90024 (na.donato@na-net.ornl.gov and chan@math.ucla.edu or na.tchan@na-net.ornl.gov).

[‡] The research of the first author was also supported by National Aeronautics and Space Administration grant NGT-70125.

preconditioned systems $M^{-1}A$ are analyzed. Expressions for the "Fourier eigenvalues" are given and from these expressions we derive bounds on the condition numbers. For the isotropic problem we find, as in the two-dimensional case [7], that $\kappa(M^{-1}A)$ is order h^{-2} for ILU and h^{-1} for MILU ($c \neq 0$). For MILU, an optimal value of c_{opt} is derived.

To examine the usefulness of these Fourier derivations and calculations, we present the results of a PCG implementation for comparison. Because of the inherent large size of these three-dimensional problems, the PCG algorithm was coded on an Alliant FX/80 using FX/FORTRAN. We examine various grid spacings and their effect upon condition numbers and iterations required for convergence. We find the dependence on h of the preconditioned Dirichlet and periodic operators to be in remarkable agreement. For the anisotropic problem we illustrate the dependence of PCG convergence on the distribution of eigenvalues [3], [4] and show that the clustering of the Fourier eigenvalues mimics that of the Dirichlet eigenvalues.

For many of the experiments, the distribution of the eigenvalues is given for both the preconditioned Dirichlet and the related periodic systems. While true Fourier analysis is not directly applicable to these problems, it is obvious from our numerical results that the Fourier technique is an extremely valuable heuristic method for examining the behavior of these preconditioned systems. The method is easy to apply and can save considerable time by determining initial approximations to optimal parameters. It is worth noting that the use of Fourier methods is a long-standing technique for the analysis of multigrid. A notable example is Brandt's "local mode analysis" [5].

The rest of this paper is outlined as follows. In §2 the stencil and recurrence relation are given for the general relaxed-(M)ILU preconditioner. In §3 the Fourier eigenvalues are derived for the related periodic preconditioned operator for the general anisotropic problem, and theorems for the isotropic problem are stated for ILU and MILU preconditioned systems. In §4 we give background information on the codes used. We also present various experimental results and we compare predicted results to the actual numerical results from the PCG implementation. Section 5 contains a summary of conclusions. Finally, the Appendix contains the proofs of the Fourier theorems and the derivation of c_{opt}.

2. The preconditioner.

We start by considering the following three-dimensional anisotropic equation as our expanded model problem

$$-\left(a_1 u_{xx} + a_2 u_{yy} + a_3 u_{zz}\right) = r \tag{1}$$

posed on the unit cube $\Omega = \{0 \leq x, y, z \leq 1\}$ with $a_1, a_2, a_3 \geq 0$ and Dirichlet boundary conditions

$$u(x, y, z) = 0 \qquad \text{on } \partial\Omega. \tag{2}$$

The problem is then discretized on the interior of the unit cube by the standard second-order finite differences using a uniform $n \times n \times n$ mesh with mesh size $h = \frac{1}{n+1}$. We get a matrix system $Au = b$, where A is represented by a seven-point stencil. The general seven-point stencil for the Dirichlet problem yields a linear equation of the form

$$\begin{aligned} &a_{i,j,k}u_{i,j,k} + b_{i,j,k}u_{i+1,j,k} + c_{i,j,k}u_{i,j+1,k} + d_{i,j,k}u_{i-1,j,k} + e_{i,j,k}u_{i,j-1,k} \\ &\qquad + f_{i,j,k}u_{i,j,k+1} + g_{i,j,k}u_{i,j,k-1} = h^2 r_{i,j,k} \end{aligned} \tag{3}$$

where $1 \leq i, j, k \leq n$ and

$$
\begin{aligned}
b_{i,j,k} &= 0, & i &= n, \\
c_{i,j,k} &= 0, & j &= n, \\
f_{i,j,k} &= 0, & k &= n, \\
d_{i,j,k} &= 0, & i &= 1, \\
e_{i,j,k} &= 0, & j &= 1, \\
g_{i,j,k} &= 0, & k &= 1.
\end{aligned} \tag{4}
$$

Note that the subscripts (i, j, k) correspond to the grid location (ih, jh, kh). For example, the entry $b_{i,j,k}$ represents the coupling between $u_{i,j,k}$ and its neighbor $u_{i+1,j,k}$. Writing the left hand side in stencil form (expanding by planes) we have

$k-1$ plane			k plane			$k+1$ plane		
·	·	·	·	$c_{i,j,k}$	·	·	·	·
·	$g_{i,j,k}$	·	$d_{i,j,k}$	$a_{i,j,k}$	$b_{i,j,k}$	·	$f_{i,j,k}$	·
·	·	·	·	$e_{i,j,k}$	·	·	·	·

Referring back to the anisotropic problem we have the assignments

$$
\begin{aligned}
a_{i,j,k} &= 2(a_1 + a_2 + a_3), \\
b_{i,j,k} &= -a_1, \\
c_{i,j,k} &= -a_2, \\
d_{i,j,k} &= -a_1, \\
e_{i,j,k} &= -a_2, \\
f_{i,j,k} &= -a_3, \\
g_{i,j,k} &= -a_3.
\end{aligned} \tag{5}
$$

A relaxed-modified ILU factorization is an approximate LU factorization M of A based on Gaussian elimination in which nonzero entries (fill-ins) of L and U are dropped (set to zero) if they correspond to a zero element in A (i.e., the sparsity patterns for L and U are the same as for A). Further, the nonzero entries of the resulting matrix $M = LU$ are required to equal the corresponding nonzero entries of A except possibly for those entries along the diagonal. For example, for ILU it is required that $\text{diag}(M) = \text{diag}(A)$. For MILU($c$), the diagonal of M is modified to ensure the condition $rowsum(M) = rowsum(A) + ch^2$. Hence, in MILU a fill-in in a row of M is added onto the diagonal element of that row of M along with the perturbation ch^2. For RILU(w), only a fraction w of the fill-ins are added back onto the diagonal of M. In this paper, we further restrict U to be unit diagonal.

Using the stencil viewpoint, the (relaxed-modified) LU factorization of A has the following structure where L is the lower triangular matrix

$k-1$ plane			k plane			$k+1$ plane		
·	·	·	·	·	·	·	·	·
·	$g_{i,j,k}$	·	$d_{i,j,k}$	$\alpha_{i,j,k}$	·	·	·	·
·	·	·	·	$e_{i,j,k}$	·	·	·	·

and U is the unit upper triangular matrix given by

$k-1$ plane			k plane			$k+1$ plane		
•	•	•	•	$\frac{c_{i,j,k}}{\alpha_{i,j,k}}$	•	•	•	•
•	•	•	•	1	$\frac{b_{i,j,k}}{\alpha_{i,j,k}}$	•	$\frac{f_{i,j,k}}{\alpha_{i,j,k}}$	•
•	•	•	•	•	•	•	•	•

The resulting preconditioner $M = LU$ is then represented by

$k-1$ plane			k plane			$k+1$ plane		
•	$m_{i+1,j,k-1}$	•	$m_{i-1,j+1,k}$	$c_{i,j,k}$	•	•	•	•
•	$g_{i,j,k}$	$m_{i,j+1,k-1}$	$d_{i,j,k}$	$m_{i,j,k}$	$b_{i,j,k}$	$m_{i-1,j,k+1}$	$f_{i,j,k}$	•
•	•	•	•	$e_{i,j,k}$	$m_{i+1,j-1,k}$	•	$m_{i,j-1,k+1}$	•

The center (diagonal) entries $m_{i,j,k}$ of M are given by

$$m_{i,j,k} = \alpha_{i,j,k} + d_{i,j,k}\frac{b_{i-1,j,k}}{\alpha_{i-1,j,k}} + e_{i,j,k}\frac{c_{i,j-1,k}}{\alpha_{i,j-1,k}} + g_{i,j,k}\frac{f_{i,j,k-1}}{\alpha_{i,j,k-1}}.$$

The other six entries of M that correspond to zero elements in A are called the "fill-ins." They are given by

$$\begin{aligned}
m_{i+1,j,k-1} &= g_{i,j,k}\frac{b_{i,j,k-1}}{\alpha_{i,j,k-1}},\\
m_{i,j+1,k-1} &= g_{i,j,k}\frac{c_{i,j,k-1}}{\alpha_{i,j,k-1}},\\
m_{i-1,j+1,k} &= d_{i,j,k}\frac{c_{i-1,j,k}}{\alpha_{i-1,j,k}},\\
m_{i+1,j-1,k} &= e_{i,j,k}\frac{b_{i,j-1,k}}{\alpha_{i,j-1,k}},\\
m_{i-1,j,k+1} &= d_{i,j,k}\frac{f_{i-1,j,k}}{\alpha_{i-1,j,k}},\\
m_{i,j-1,k+1} &= e_{i,j,k}\frac{f_{i,j-1,k}}{\alpha_{i,j-1,k}}.
\end{aligned}\tag{6}$$

To determine the $\alpha_{i,j,k}$ values, the relaxed modified LU factorization is typically augmented by the rowsum condition

$$rowsum(M) = rowsum(A) + ch^2 + (1-w) * (\text{fill-ins in } M).$$

The above formulation includes both the parameter $\delta = ch^2$ of Gustafsson [12], which is added to the main diagonal, and a relaxation parameter w of Axelsson and Lindskog [3]. (See also [1].) This condition yields the ILU factorization for $c = w = 0$ and MILU(c) for $w = 1$. For $c = 0, w \,\epsilon\, (0,1)$ we get RILU(w) [6], [10].

Given that six of the entries in any given row are common to both A and M, the rowsum condition yields a second expression for the diagonal entries of M

$$m_{i,j,k} = a_{i,j,k} + ch^2 - w(\text{fill-ins in } M).$$

Substituting in the expressions for the fill-ins and for $m_{i,j,k}$, the following recurrence for $\alpha_{i,j,k}$ results in

$$\begin{aligned}\alpha_{i,j,k} = a_{i,j,k} + ch^2 - d_{i,j,k}(b_{i-1,j,k} + w(c_{i-1,j,k} + f_{i-1,j,k}))/\alpha_{i-1,j,k} \\ - e_{i,j,k}(c_{i,j-1,k} + w(b_{i,j-1,k} + f_{i,j-1,k}))/\alpha_{i,j-1,k} \\ - g_{i,j,k}(f_{i,j,k-1} + w(b_{i,j,k-1} + c_{i,j,k-1}))/\alpha_{i,j,k-1},\end{aligned} \tag{7}$$

where (4) applies and a term is ignored if it involves an $\alpha_{i,j,k}$ having any of its indices equal to 0 or $n+1$.

3. Fourier analysis. A method of exact analysis of $M^{-1}A$ has not yet presented itself. Yet it is critical to be able to compare the distribution of the resulting eigenvalues and their clustering traits for different preconditioners M. These clustering traits, along with the condition number $\kappa(M^{-1}A)$, affect the convergence rate of a PCG method [3], [4]. So our basic approach here, while not exact, is to find the Fourier transform of $M^{-1}A$. We do this by applying $M^{-1}A$ to the eigenvectors $u^{(s,t,r)}$ composed of Fourier exponential modes. The (i,j,k)th grid component of $u^{(s,t,r)}$ is given by

$$u_{ijk}^{(s,t,r)} = e^{\iota i\theta_s} e^{\iota j\phi_t} e^{\iota k\xi_r}$$

where $\iota = \sqrt{-1}$, $\theta_s = (2\pi/(n+1))s, \phi_t = (2\pi/(n+1))t, \xi_r = (2\pi/(n+1))r$, for $r,s,t = 1,\cdots,n$.

However, this technique is theoretically exact only for constant coefficient problems with periodic boundary conditions [7]. In other words, the $u^{(s,t,r)}$ are not eigenvectors of the matrix $M^{-1}A$ that results from the discretization of the Dirichlet problem. To use the technique, we make the following extensions [7]:

(a) Treat the matrices M and A as if they were periodic. Enforce (5) for $0 \leq i,j,k \leq n+1$ and ignore the Dirichlet constraints (4).

(b) Force M to be a constant diagonal system by treating the $\alpha_{i,j,k}$ as the constant α arising as the asymptotic solution of the recurrence equation (7). So $\alpha_{i,j,k} = \alpha$ is then given by

$$\begin{aligned}\alpha = &\left((a_1+a_2+a_3) + \frac{ch^2}{2}\right) \\ &+\sqrt{\left((a_1+a_2+a_3) + \frac{ch^2}{2}\right)^2 - (a_1^2+a_2^2+a_3^2) - 2w(a_1a_2+a_1a_3+a_2a_3)},\end{aligned} \tag{8}$$

where the positive root has been chosen to agree in magnitude with the Dirichlet values. In the case of very large n, this is the value the $\alpha_{i,j,k}$ would tend toward for those (i,j,k) corresponding to grid points far from the boundaries of Ω. The proof that $\alpha_{i,i,i} \searrow \alpha$ follows analogously to the two-dimensional situation given in [16].

(c) From the theory developed in [7], we then use the formula $h_d = 2h_p$ to relate the mesh sizes used for the Dirichlet problem to that of the corresponding Fourier method (periodic) result.

With the above extensions, we have obtained a periodic constant coefficient ILU factorization preconditioner for which exact Fourier analysis can be used. Note that this related periodic ILU factorization is not the ILU factorization for the periodic version of the Dirichlet problem. It is an artificial operator that we analyze in the hope that the results will apply to the ILU preconditioned Dirichlet system.

Now applying these related periodic extensions of A and M to $u^{(s,t,r)}$ yields

$$\begin{aligned} Au^{(s,t,r)} &= \lambda_{str} u^{(s,t,r)}, \\ Mu^{(s,t,r)} &= \psi_{str} u^{(s,t,r)}, \end{aligned}$$

where

$$\lambda_{str} = \lambda_{str}(A) = 4\left(a_1 \sin^2\left(\frac{\theta_s}{2}\right) + a_2 \sin^2\left(\frac{\phi_t}{2}\right) + a_3 \sin^2\left(\frac{\xi_r}{2}\right)\right) \tag{9}$$

and

$$\begin{aligned} \psi_{str} &= \psi_{str}(M) \\ &= \lambda_{str} + \frac{2}{\alpha}(a_1a_2\cos(\theta_s - \phi_t) + a_1a_3\cos(\xi_r - \theta_s) + a_2a_3\cos(\phi_t - \xi_r)) \\ &\quad - 2w(a_1a_2 + a_1a_3 + a_2a_3)/\alpha + ch^2. \end{aligned} \tag{10}$$

Thus the Fourier transform of $M^{-1}A$ is

$$\mu_{str}(M^{-1}A) = \frac{\lambda_{str}(A)}{\psi_{str}(M)} \tag{11}$$

and the condition number of the preconditioned system is given by

$$\kappa = \kappa(M^{-1}A) = \frac{\max_{str} \mu_{str}}{\min_{str} \mu_{str}}.$$

From (9) and (10) this can be easily computed for a given mesh size h. The n^3 values μ_{str} are also called the Fourier eigenvalues of $M^{-1}A$.

Consider for now the isotropic problem ($a_1 = a_2 = a_3 = 1$). Using the above we get the following results whose proofs are presented in the Appendix.

THEOREM 3.1. *For the* ILU *preconditioned isotropic operator* ($w = 0, c = 0$),

$$\kappa^{(I)} = O(h^{-2}).$$

THEOREM 3.2. *For the* MILU *preconditioned isotropic operator* ($w = 1$),

$$\kappa^{(M)} = \begin{cases} O(h^{-1}), & \textit{if } c > 0; \\ O(h^{-2}), & c = 0. \end{cases}$$

Result 3.1. The optimal value of c occurs near $c_p = 12\pi^2$ for the periodic problem and $c_d = 3\pi^2$ for the Dirichlet problem.

It appears from numerical calculations below that the above results also hold for the Dirichlet ILU and MILU preconditioned systems except for MILU when c is near zero. And in the anisotropic cases, although generalizing these theorems poses some difficulties, we are able to show that the Fourier results are still excellent predictors of the Dirichlet results in terms of dependence on h and c.

4. Numerical results. In order to verify the preceding Fourier results, a PCG routine was implemented on an Alliant FX/80 to solve the system $Au = b$ using equations (3), (4), and (5) where the true solution was chosen to be

$$u(x, y, z) = x(1-x)y(1-y)z(1-z).$$

Except where noted, uniformly distributed random initial data* was used for u on the interior of the unit cube.

In order to compare condition numbers for small h we approximated the extreme eigenvalues of the (M)ILU preconditioned systems from PCG-generated values as outlined in [11]. From values generated during the PCG iterations, a symmetric tridiagonal matrix associated with the Lanczos vectors is generated. An EISPACK routine is then called to determine the eigenvalues of the tridiagonal matrix which will in turn approximate the extreme eigenvalues of the preconditioned system. All computations were done in double precision except for the EISPACK routine. The stopping criterion for the PCG iteration required the following three conditions to be satisfied concurrently

$$\frac{\|r^{(k)}\|}{\|r^{(0)}\|} < 10^{-14}, \quad |\mu_{\min}^{(k)} - \mu_{\min}^{(k-1)}| < 10^{-3}, \quad |\mu_{\max}^{(k)} - \mu_{\max}^{(k-1)}| < 10^{-3}.$$

For some experimental results the full set of eigenvalues was needed for the preconditioned operator. A full set of eigenvalues could only be generated in reasonable time for large h (small n). For this a separate program was implemented wherein the M and A matrices were generated and then a call to another EISPACK routine was made to solve the general eigenvalue problem $Ax = \mu Mx$.

A routine to generate the Fourier eigenvalues and condition numbers via equations (8), (9), and (10) was implemented on a Sun 3/150 workstation. The computations were performed in double precision.

4.1. ILU results ($c = w = 0$)**.** First, we show that the Fourier technique predicts the distribution of the eigenvalues for the ILU preconditioned Dirichlet operator. Figure 1 shows the distribution of the eigenvalues of the preconditioned Dirichlet and periodic operators for $h_d = \frac{1}{8}$ ($h_p = \frac{1}{16}$). The range and clustering of the Dirichlet ILU and the Fourier ILU eigenvalues are extremely close. Table 1 shows the results for various values of h for the ILU preconditioned Dirichlet problem. As h decreases, the Fourier results (Table 2) for the minimum and maximum eigenvalue (and hence condition number) are closer to those for the preconditioned Dirichlet system. As predicted, $\kappa(M^{-1}A) = O(h^{-2})$ in each case.

4.2. MILU results for $w = 1$**.** In Fig. 2 the minimum and maximum eigenvalues and the condition numbers are plotted as a function of c for $h_d = \frac{1}{8}$ for the Dirichlet operator and $h_p = \frac{1}{16}$ for the Periodic MILU results.

For large c, the maximum eigenvalues are indistinguishable. For the minimum eigenvalues (and hence the condition numbers) the values are different, but the trend in the values as functions of increasing c are similar. Also we see from Fig. 2 that the optimal c value for the MILU preconditioned Dirichlet operator does occur at c slightly less than the value $c_d = 3\pi^2$ predicted by Result 3.1.

* Experiments were also performed with zero initial data and normally distributed random initial data. Qualitatively, the results did not depend on the choice of these initial data.

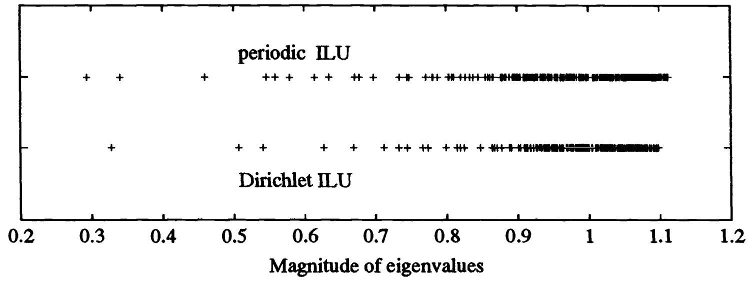


FIG. 1. *Eigenvalues of the isotropic* ILU *preconditioned system,* $h_d = \frac{1}{8}$ $(h_p = \frac{1}{16})$.

TABLE 1
Dirichlet ILU *results for various* h_d *values.*

h_d	$\min \mu$	$\max \mu$	$\kappa(M^{-1}A)$
$\frac{1}{8}$	0.328	1.096	3.341
$\frac{1}{16}$	0.098	1.108	11.281
$\frac{1}{32}$	0.0258	1.111	43.045
$\frac{1}{64}$	0.0065	1.112	170.123

TABLE 2
Periodic ILU *results for corresponding* h_p.

h_d	$\min \mu$	$\max \mu$	$\kappa(M^{-1}A)$
$\frac{1}{8}$	0.293	1.112	3.791
$\frac{1}{16}$	0.095	1.112	11.735
$\frac{1}{32}$	0.026	1.112	43.503
$\frac{1}{64}$	0.0065	1.112	170.574

Tables 3 and 4 contain the results for various values of h_d for $c_d = 3\pi^2$. We see that the Periodic values for MILU are not as close to the Dirichlet values as they were for the ILU case. But we do see that the periodic results display the $O(h^{-1})$ behavior that occurs for the MILU operator for $c_d = 3\pi^2$.

Tables 5 and 6 show the corresponding results for $c = 0$. The periodic MILU condition number displays $O(h^{-2})$ behavior rather than the $O(h^{-1})$ behavior of the Dirichlet MILU operator. This is the same situation that arises for the two-dimensional case [7]. For $c = 0$, it takes a very delicate cancellation to yield the $O(h^{-2})$ results for the Fourier condition number. Away from $c = 0$ the calculations are not as delicate and the Fourier prediction is very good.

Figure 3 plots the condition number of the MILU preconditioned Dirichlet problem and the Fourier condition number results for $h_d = \frac{1}{16}$ (the lower two curves) and for $h_d = \frac{1}{64}$ (the upper two curves). Again we see that, away from $c = 0$, the dependence of conditioning on the parameter c clearly follows the same general pattern for the preconditioned Dirichlet operator (a given h_d) and its corresponding Fourier results $(h_p = h_d/2)$.

4.3. Anisotropic results. To examine results from anisotropic problems we pick the following three sets of data:

Data Set 1: $a_1 = a_2 = a_3 = 1$.

Data Set 2: $a_1 = 1, a_2 = 1, a_3 = 0.01$.

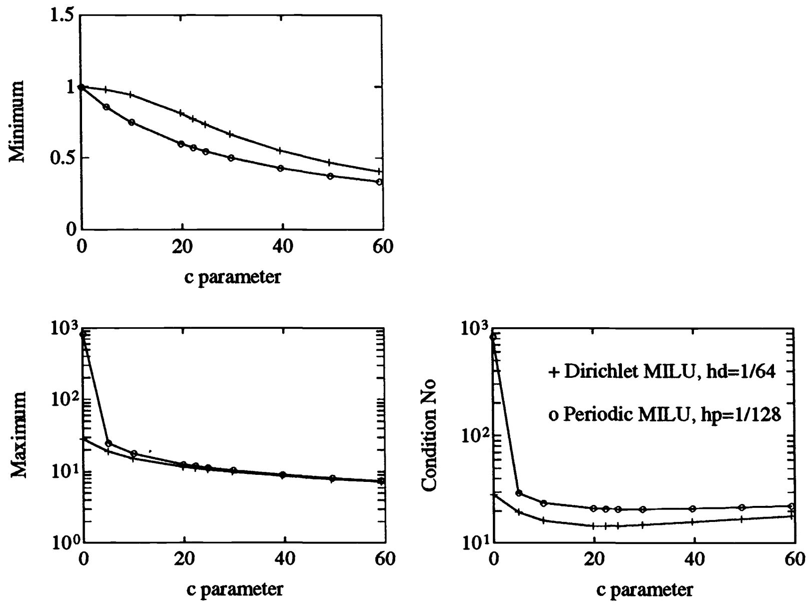


FIG. 2. MILU *results as a function of c for* $h_d = \frac{1}{64}$.

TABLE 3
Dirichlet MILU *results for* $c = 3\pi^2$.

h_d	$\min \mu$	$\max \mu$	$\kappa(M^{-1}A)$
$\frac{1}{8}$	0.537	1.444	2.689
$\frac{1}{16}$	0.585	2.614	4.465
$\frac{1}{32}$	0.629	5.018	7.971
$\frac{1}{64}$	0.664	9.872	14.871

TABLE 4
Periodic MILU *results for* $c = 3\pi^2$.

h_d	$\min \mu$	$\max \mu$	$\kappa(M^{-1}A)$
$\frac{1}{8}$	0.497	1.545	3.110
$\frac{1}{16}$	0.499	2.797	5.603
$\frac{1}{32}$	0.500	5.341	10.687
$\frac{1}{64}$	0.500	10.429	20.859

Data Set 3: $a_1 = 1, a_2 = 0.01, a_3 = 0.01$.

These three data sets are a subset of those used in [2], and to allow us to compare results to [2] we use an initial guess for $u(x, y, z)$ that is zero on the interior of the unit cube.

Tables 7, 8, and 9 present the results for $h = 1/21$ for the ILU preconditioned operators. For each of the three data sets the periodic ILU results are in close approximation to the Dirichlet ILU values. In particular, the behavior of the Dirichlet ILU condition numbers is captured by the periodic ILU condition numbers. What is

TABLE 5
Dirichlet MILU *results for* $c = 0$.

h_d	$\min \mu$	$\max \mu$	$\kappa(M^{-1}A)$
$\frac{1}{8}$	1.000	2.753	2.753
$\frac{1}{16}$	1.000	5.983	5.982
$\frac{1}{32}$	1.000	13.125	13.120
$\frac{1}{64}$	1.001	28.256	28.337

TABLE 6
Periodic MILU *results for* $c = 0$.

h_d	$\min \mu$	$\max \mu$	$\kappa(M^{-1}A)$
$\frac{1}{8}$	1.000	13.252	13.252
$\frac{1}{16}$	1.000	52.156	52.156
$\frac{1}{32}$	1.000	207.784	207.784
$\frac{1}{64}$	1.000	830.301	830.301

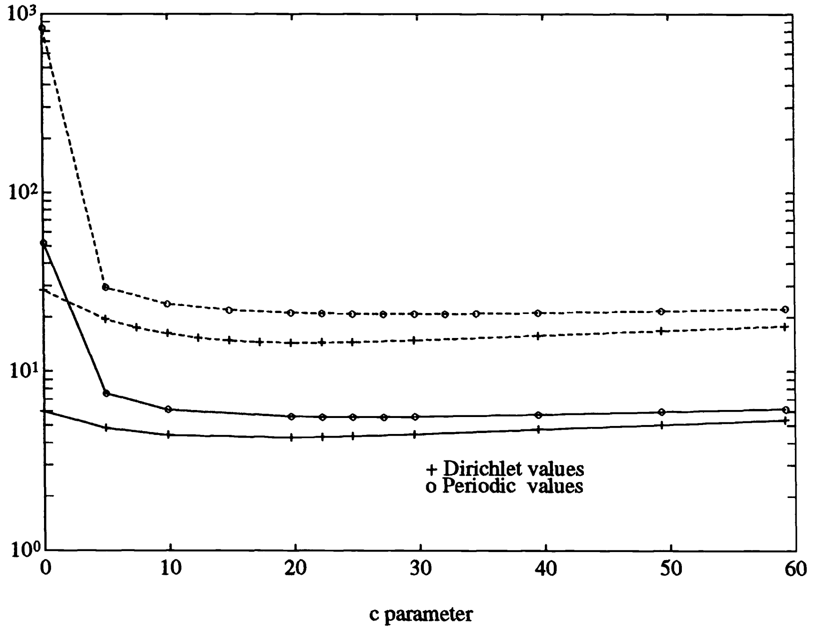


FIG. 3. MILU *condition numbers for* $h_d = \frac{1}{16}$ *(solid lines) and* $h_d = \frac{1}{64}$ *(dashed lines).*

of further interest is the comparison of condition numbers and PCG iteration counts for the Dirichlet ILU operator across the three data sets. $\kappa(M^{-1}A)$ for Data Set 1 is greater than κ for Data Set 2. Yet Data Set 1 requires significantly fewer PCG iterations to converge than Data Set 2. This iteration count variation is also seen in the data presented in [2].

To study this more closely, we redo the calculations for $h_d = \frac{1}{8}$ so that the full set of eigenvalues can be plotted. Tables 10, 11, and 12 present the minimum, maximum, and condition number results for the three data sets and we see that the same situation occurs as for $h_d = \frac{1}{21}$. In Fig. 4, the eigenvalues of the Dirichlet ILU operator are plotted in sorted order for Data Sets 2 and 3 with $h_d = \frac{1}{8}$. These figures also include

TABLE 7
ILU results for Data Set 1.

$h_d = 1/21$	$\min \mu$	$\max \mu$	$\kappa(M^{-1}A)$	Iterations
ILU	0.059	1.108	18.900	37
Periodic	0.057	1.112	19.388	na

TABLE 8
ILU results for Data Set 2.

$h_d = 1/21$	$\min \mu$	$\max \mu$	$\kappa(M^{-1}A)$	Iterations
ILU	0.072	1.198	16.667	57
Periodic	0.070	1.203	17.106	na

TABLE 9
ILU results for Data Set 3.

$h_d = 1/21$	$\min \mu$	$\max \mu$	$\kappa(M^{-1}A)$	Iterations
ILU	0.419	1.436	3.426	27
Periodic	0.479	1.472	3.600	na

the Fourier eigenvalues for $h_p = \frac{1}{16}$.

For the corresponding figure for Data Set 1, refer back to Fig. 1.

First we notice the extreme similarity of the Dirichlet and Fourier eigenvalues. For each data set the behavior of the Fourier eigenvalues corresponds to that of the preconditioned Dirichlet system. And we can see using either the Dirichlet or Fourier spectra that it is the clustering [3], [4] of the eigenvalues that becomes the dominant factor in the number of iterations required.

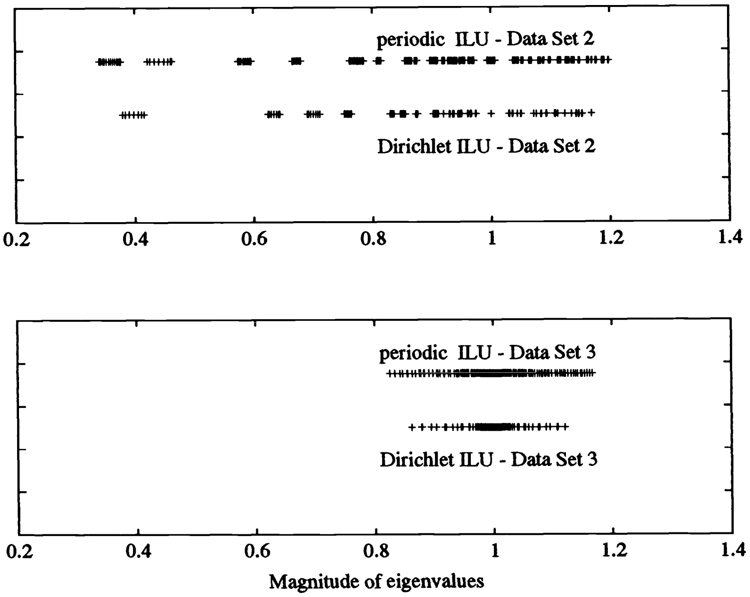


FIG. 4. *Eigenvalues for the anisotropic* ILU *preconditioned systems,* $h_d = \frac{1}{8}$ $(h_p = \frac{1}{16})$.

Data Set 1 yields a larger condition number than Data Set 2 because (in part) of its much smaller minimum eigenvalue. But Data Set 1 has only a few well-isolated minimum eigenvalues, whereas there is a clustering of eigenvalues near the minimum for Data Set 2. The eigenvalues for Data Set 1 have more clustering about 1 than those for Data Set 2. Hence, the systems from Data Set 1 converge more quickly via PCG than those from Data Set 2.

TABLE 10
ILU *results for Data Set* 1.

$h_d = 1/8$	$\min \mu$	$\max \mu$	$\kappa(M^{-1}A)$	Iterations
ILU	0.328	1.095	3.338	16
Periodic	0.293	1.112	3.791	na

TABLE 11
ILU *results for Data Set* 2.

$h_d = 1/8$	$\min \mu$	$\max \mu$	$\kappa(M^{-1}A)$	Iterations
ILU	0.379	1.168	3.079	20
Periodic	0.340	1.199	3.523	na

TABLE 12
ILU *results for Data Set* 3.

$h_d = 1/8$	$\min \mu$	$\max \mu$	$\kappa(M^{-1}A)$	Iterations
ILU	0.863	1.119	1.297	14
Periodic	0.825	1.166	1.413	na

For Data Set 2, Table 13 lists the condition number of the ILU preconditioned system as a function of h. Table 14 similarly reports the results for Data Set 3.

TABLE 13
Data Set 2*:* $\kappa(M^{-1}A)$ *for various* h*.*

h_d	ILU	Periodic
$\frac{1}{8}$	3.079	3.523
$\frac{1}{16}$	10.002	10.446
$\frac{1}{32}$	37.667	38.096

TABLE 14
Data Set 3*:* $\kappa(M^{-1}A)$ *for various* h*.*

h_d	ILU	Periodic
$\frac{1}{8}$	1.297	1.413
$\frac{1}{16}$	2.369	2.546
$\frac{1}{32}$	6.667	6.857

We see the remarkable agreement of the ILU and Fourier results that occurred in the isotropic case for ILU. Hence, the Fourier results remain a good predictor of the dependence of $\kappa(M^{-1}A)$ on h.

We also analyze the MILU preconditioned operators for the anisotropic data sets. Tables 15, 16, and 17 list the condition numbers for each of the three data sets using the MILU preconditioner with $c_d = 2\pi^2$. Each table includes both the Dirichlet MILU and the calculated Fourier condition number for various values of h_d. Again, the similarity in the dependence of $\kappa(M^{-1}A)$ is noticeable. For all three data sets and for both the Dirichlet and Fourier results, $\kappa(M^{-1}A)$ demonstrates $O(h^{-1})$ behavior.

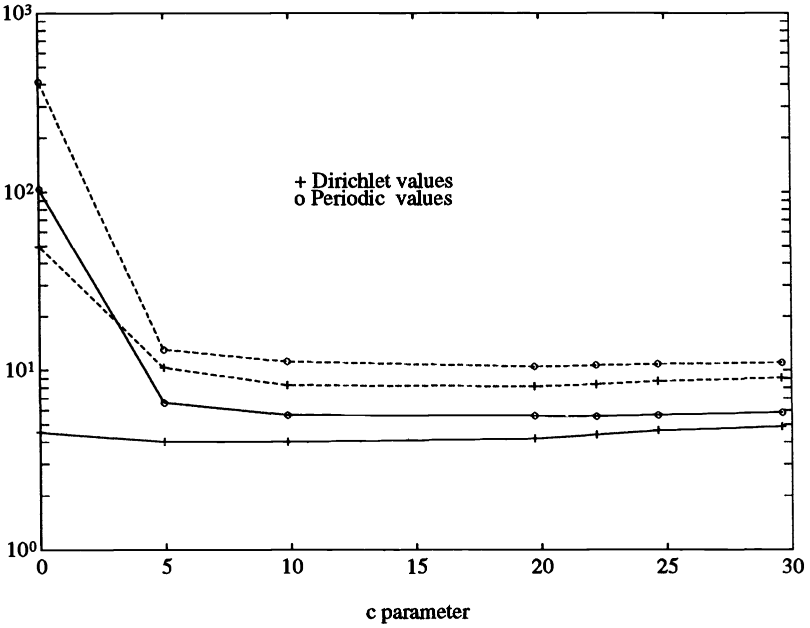


FIG. 5. *$\kappa(M^{-1}A)$ for anisotropic* MILU *for Data Set* 2, *$h_d = \frac{1}{16}$ (solid lines), $h_d = \frac{1}{32}$ (dashed lines).*

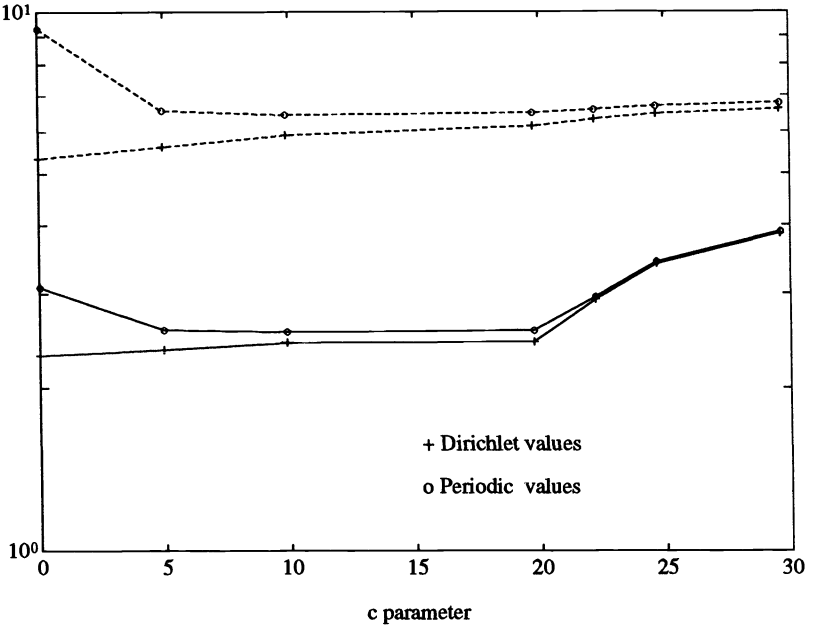


FIG. 6. *$\kappa(M^{-1}A)$ for anisotropic* MILU *for Data Set* 3, *$h_d = \frac{1}{16}$ (solid lines), $h_d = \frac{1}{32}$ (dashed lines).*

TABLE 15
$\kappa(M^{-1}A)$ *results for* MILU ($c_d = 2\pi^2$) *preconditioned system for Data Set* 1.

h_d	MILU	Periodic
$\frac{1}{8}$	2.521	3.032
$\frac{1}{16}$	4.282	5.619
$\frac{1}{32}$	7.721	10.797

TABLE 16
$\kappa(M^{-1}A)$ *results for* MILU ($c_d = 2\pi^2$) *preconditioned system for Data Set* 2.

h_d	MILU	Periodic
$\frac{1}{8}$	2.629	3.075
$\frac{1}{16}$	4.393	5.572
$\frac{1}{32}$	7.987	10.654

TABLE 17
$\kappa(M^{-1}A)$ *results for* MILU ($c_d = 2\pi^2$) *preconditioned system for Data Set* 3.

h_d	MILU	Periodic
$\frac{1}{8}$	2.782	2.801
$\frac{1}{16}$	2.920	2.954
$\frac{1}{32}$	6.322	6.597

In Figs. 5 and 6 we have plotted $\kappa(M^{-1}A)$ as a function of c for MILU for the anisotropic Data Sets 2 and 3, respectively. The upper two curves of each figure correspond to $h_d = \frac{1}{32}$ and the lower two curves to $h_d = \frac{1}{16}$. Again, as in the isotropic MILU results, except when c is near zero, the Fourier curves mimic the dependence on c demonstrated by the Dirichlet MILU preconditioned system.

From Figs. 5 and 6, we can see a difficulty in determining c_{opt} in anisotropic situations. The Dirichlet curves for $\kappa(M^{-1}A)$ are visually flat near the optimal value of c_d. This flatness may indicate that finding c_{opt} is a poorly conditioned numerical task. However, the optimal value for c in the Fourier curves certainly corresponds to a good initial approximation of c_{opt} in the Dirichlet case. By this we mean that by choosing c_d to be the value corresponding to the optimal c determined from the Fourier values, the behavior of $\kappa(M^{-1}A)$ in the Dirichlet problem will be $O(h^{-1})$ rather than $O(h^{-2})$.

5. Conclusions. We note that the Fourier technique used here and in [7] is not exact, but it has been shown to be a powerful tool in the analysis of preconditioned systems. Although the Fourier and Dirichlet condition numbers are not identical, the Fourier method is still capable of predicting the dependence of the Dirichlet condition number on the parameters h and c. In the case of an MILU preconditioner, the Fourier method provides a simple and fast technique to find a first approximation to the optimal c parameter. This makes the method very worthwhile, since there are currently no other "easy" methods to apply that give better results. And this is further emphasized by its easy application to anisotropic problems.

Appendix. In this section, we provide the details of proofs and derivations omitted from the main text: we prove Theorems 3.1 and 3.2 and derive Result 3.1.

THEOREM 3.1. *For the* ILU *preconditioned isotropic operator* $(w = 0, c = 0)$,

$$\kappa^{(I)} = O(h^{-2}).$$

Proof. For the isotropic problem $(a_1 = a_2 = a_3 = 1)$ with ILU preconditioner $(w = 0, c = 0)$ we get the recurrence (7) the expression $6 = \alpha + \frac{3}{\alpha}$ whose solution is $\alpha = 3 + \sqrt{6}$ and we have

$$\begin{aligned}\lambda_{str} &= 4\left(\sin^2\left(\frac{\theta_s}{2}\right) + \sin^2\left(\frac{\phi_t}{2}\right) + \sin^2\left(\frac{\xi_r}{2}\right)\right),\\ \psi_{str} &= \lambda_{str} + \frac{2}{\alpha}(\cos(\theta_s - \phi_t) + \cos(\xi_r - \theta_s) + \cos(\phi_t - \xi_r)).\end{aligned}$$

It holds immediately that

$$\begin{aligned}\lambda_{\min} &\geq \lambda_{l.b.} = 12\sin^2(\pi h) \approx 12(\pi h)^2 + O(h^4),\\ \lambda_{\max} &\leq \lambda_{u.b.} = 12,\\ \psi_{\max} &\leq \psi_{u.b.} = 12 + \frac{6}{\alpha}.\end{aligned}$$

Now, for the lower bound on ψ_{str}: Set $x = \sin\frac{\theta_s}{2}, y = \sin\frac{\phi_t}{2}, z = \sin\frac{\xi_r}{2}$ and use

$$\begin{aligned}\cos(\theta_s - \phi_t) &= 1 - 2(x^2 + y^2) + 4x^2y^2 \pm 4xy\sqrt{(1 - x^2)(1 - y^2)}\\ &> 1 - 2(x^2 + y^2) - 4|xy|,\end{aligned}$$

$$\begin{aligned}\psi_{str} &= \lambda_{str} + \frac{2}{\alpha}(\cos(\theta_s - \phi_t) + \cos(\phi_t - \xi_r) + \cos(\xi_r - \theta_s))\\ &\geq \lambda_{str} + \frac{2}{\alpha}(3 - 2(x^2 + y^2) - 4|xy| - 2(y^2 + z^2) - 4|yz| - 2(x^2 + z^2) - 4|xz|)\\ &= 4(x^2 + y^2 + z^2) + \frac{2}{\alpha}(3 - 4(x^2 + y^2 + z^2) - 4(|xy| + |yz| + |xy|))\\ &= \frac{4}{\alpha}((\alpha - 2)(x^2 + y^2 + z^2) - 2(|xy| + |yz| + |xy|)) + \frac{6}{\alpha}\\ &= \frac{4}{\alpha}((1 + \sqrt{6})(x^2 + y^2 + z^2) - 2(|xy| + |yz| + |xy|)) + \frac{6}{\alpha}\\ &> \frac{4}{\alpha}(2(x^2 + y^2 + z^2) - 2(|xy| + |yz| + |xy|)) + \frac{6}{\alpha}\\ &= \frac{4}{\alpha}((|x| - |y|)^2 + (|y| - |z|)^2 + (|x| - |z|)^2) + \frac{6}{\alpha}\\ &\geq \frac{6}{\alpha}.\end{aligned}$$

In other words,

$$\psi_{\min} \geq \psi_{l.b.} = \frac{6}{\alpha}.$$

So, we can now finally get bounds on the condition number via

$$\begin{aligned}
\mu_{\min}^{(I)} &\geq \frac{\lambda_{\min}}{\psi_{\max}} \geq \frac{12\sin^2(\pi h)}{12+\frac{6}{\alpha}} = \frac{2\sin^2(\pi h)}{2+\frac{1}{\alpha}} \\
\mu_{\max}^{(I)} &\leq \frac{\lambda_{\max}}{\psi_{\min}} \leq \frac{12}{\frac{6}{\alpha}} = 2\alpha \\
\kappa^{(I)} &= \frac{\mu_{\max}^{(I)}}{\mu_{\min}^{(I)}} \leq \frac{2\alpha}{\frac{2\sin^2(\pi h)}{2+1/a}} \\
&= \frac{\alpha(2+\frac{1}{\alpha})}{\sin^2(\pi h)} \approx \frac{\alpha(2+\frac{1}{\alpha})}{(\pi h)^2} \approx 1.2h^{-2}.
\end{aligned}$$

Now we will see that this $O(h^{-2})$ bound is tight.

Let $\theta_s = \phi_t = \xi_r = \pi$, $(r = s = t = (n+1)/2)$, to get $\lambda = 12, \psi = 12 + \frac{6}{\alpha}$. From these we have

$$\mu^{(1)} = \frac{\lambda}{\psi} = \frac{12}{12+\frac{6}{\alpha}} \approx 0.916 = O(1).$$

On the other end, letting $\theta_s = \phi_t = \xi_r = \theta_1 = 2\pi h$, we get for small enough h

$$\begin{aligned}
\lambda &= 12\sin^2(\pi h) \\
\psi &= 12\sin^2(\pi h) + \frac{6}{\alpha} \\
\mu^{(2)} &= \frac{\lambda}{\psi} = \frac{2\sin^2(\pi h)}{2\sin^2(\pi h)+\frac{1}{\alpha}} \approx \frac{2(\pi h)^2 + O(h^4)}{\frac{1}{\alpha} + 2(\pi h)^2 + O(h^4)} \\
&\approx 2\alpha(\pi h)^2 + O(h^4).
\end{aligned}$$

Finally, combining the above, we get

$$\begin{aligned}
\frac{\mu^{(1)}}{\mu^{(2)}} &\approx \frac{1}{(2\alpha+1)\pi^2h^2 + O(h^4)} \\
&= O(h^{-2})
\end{aligned}$$

and so we have that the bound of $O(h^{-2})$ on $\kappa^{(I)}$ is tight. □

THEOREM 3.2. *For the* MILU *preconditioned isotropic operator* $(w = 1)$,

$$\kappa^{(M)} = \begin{cases} O(h^{-1}), & \text{if } c > 0; \\ O(h^{-2}), & c = 0. \end{cases}$$

Proof. For the isotropic problem $(a_1 = a_2 = a_3 = 1)$ with MILU preconditioner $(w = 1, c \neq 0)$ we get for the recurrence (7) the expression $6 + ch^2 = \alpha + \frac{9}{\alpha}$ whose solution is

$$\alpha = 3 + \frac{ch^2}{2} + h\sqrt{3c}\sqrt{1+\frac{ch^2}{12}}.$$

So we have

$$\lambda_{str} = 4\left(\sin^2\left(\frac{\theta_s}{2}\right) + \sin^2\left(\frac{\phi_t}{2}\right) + \sin^2\left(\frac{\xi_r}{2}\right)\right),$$

$$\begin{aligned}\psi_{str} &= \lambda_{str} + \frac{2}{\alpha}(\cos(\theta_s - \phi_t) + \cos(\xi_r - \theta_s) + \cos(\phi_t - \xi_r)) - \frac{6}{\alpha} + ch^2 \\ &= \lambda_{str} - \frac{2}{\alpha}(3 - \cos(\theta_s - \phi_t) - \cos(\xi_r - \theta_s) - \cos(\phi_t - \xi_r)) + ch^2.\end{aligned}$$

We first derive a lower bound on $\mu^{(M)}$. Observe that $\psi_{str} \le \lambda_{str} + ch^2$. Also, for h small enough, there exists $\tilde{c}$ such that $\sin(\pi h) \ge \tilde{c}h$, which yields $\lambda_{str} \ge 12(\tilde{c}h)^2$. Thus we get the lower bound

$$\begin{aligned}\mu_{str}^{(M)} &\ge \frac{\lambda_{str}}{\lambda_{str} + ch^2} = \frac{1}{1 + \frac{ch^2}{\lambda_{str}}} \\ &\ge \frac{1}{1 + \frac{c}{12\tilde{c}^2}} \equiv \mu_{l.b.}.\end{aligned}$$

Next we derive the upper bound on $\mu^{(M)}$. We use that $\lambda_{str} \le 12$. With the aid of the symbolic manipulator Maple [9] we get

$$\frac{3 - \cos(\theta_s - \phi_t) - \cos(\xi_r - \theta_s) - \cos(\phi_t - \xi_r)}{\sin^2\left(\frac{\theta_s}{2}\right) + \sin^2\left(\frac{\phi_t}{2}\right) + \sin^2\left(\frac{\xi_r}{2}\right)} \le 6.$$

Hence,

$$\begin{aligned}\mu_{str} &= \frac{\lambda_{str}}{\lambda_{str} - \frac{2}{\alpha}(3 - \cos(\theta_s - \phi_t) - \cos(\xi_r - \theta_s) - \cos(\phi_t - \xi_r)) + ch^2} \\ &= \frac{1}{1 - \left(\frac{2}{\alpha}\right)\frac{3 - \cos(\theta_s - \phi_t) - \cos(\xi_r - \theta_s) - \cos(\phi_t - \xi_r)}{4(\sin(\theta_s/2) + \sin(\phi_t/2) + \sin(\xi_r/2))} + \frac{ch^2}{\lambda_{str}}} \\ &= \frac{1}{1 - \left(\frac{1}{2\alpha}\right)\frac{3 - \cos(\theta_s - \phi_t) - \cos(\xi_r - \theta_s) - \cos(\phi_t - \xi_r)}{\sin(\theta_s/2) + \sin(\phi_t/2) + \sin(\xi_r/2)} + \frac{ch^2}{\lambda_{str}}} \\ &\le \frac{1}{1 - \frac{1}{2\alpha}(6) + \frac{ch^2}{12}}.\end{aligned}$$

Now we use the approximation $\alpha \approx 3(1 + \frac{1}{3}h\sqrt{3c} + O(ch^2))$ to get for small enough h that $\frac{1}{\alpha} \approx \frac{1}{3}(1 - \frac{1}{3}h\sqrt{3c} + O(ch^2))$, which yields

$$\mu_{str} \le \frac{1}{\frac{1}{3}h\sqrt{3c} + O(h^2)} \equiv \mu_{u.b.}.$$

So, finally,

$$\begin{aligned}\kappa^{(M)} &= \frac{\max \mu_{str}}{\min \mu_{str}} \le \frac{\mu_{u.b.}}{\mu_{l.b.}} = \frac{1 + \frac{c}{12\tilde{c}^2}}{\frac{1}{3}h\sqrt{3c} + O(h^2)} \\ &= \begin{cases} O(h^{-1}), & \text{if } c \ne 0; \\ O(h^{-2}), & c = 0. \end{cases}\end{aligned}$$

Next it is shown that the above bounds are tight.

First consider $\theta_s = \phi_t = \xi_r = \theta_1 = 2\pi h$.

$$
\begin{aligned}
\lambda_{111} &= 12\sin^2(\pi h),\\
\psi_{111} &= 12\sin^2(\pi h) + \frac{2}{\alpha}(3) - \frac{6}{\alpha} + ch^2 = 12\sin^2(\pi h) + ch^2,\\
\mu^{(M)}_{111} &= \frac{\lambda_{111}}{\psi_{111}} = \frac{12\sin^2(\pi h)}{12\sin^2(\pi h) + ch^2} = \frac{1}{1 + (ch^2/12\sin^2(\pi h))} \approx \frac{1}{1 + (ch^2/12(\pi h)^2)}\\
&= \frac{1}{1 + (c/12\pi^2)} = O(1).
\end{aligned}
$$

Now consider $\theta_s = \xi_r = \theta$, and $\phi_t = 2\pi - 2\theta$. Then $\sin(\phi_t/2) = \sin\theta = 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}$ and $\cos(\phi_t - \theta_s) = \cos(\xi_r - \phi_t) = \cos(3\theta_s)$, $\cos(\xi_r - \theta_s) = 1$. In the expression for ψ we will also use the following

$$
\cos(3\theta) = 1 - 18\sin^2\frac{\theta}{2} + 48\sin^4\frac{\theta}{2} - 32\sin^6\frac{\theta}{2}.
$$

$$
\begin{aligned}
\lambda &= 4\left(2\sin^2\left(\frac{\theta}{2}\right) + 4\sin^2\left(\frac{\theta}{2}\right)\cos^2\left(\frac{\theta}{2}\right)\right) = 8\sin^2\left(\frac{\theta}{2}\right)\left(1 + 2\cos^2\left(\frac{\theta}{2}\right)\right)\\
&= 24\sin^2\left(\frac{\theta}{2}\right)\left(1 - \frac{2}{3}\sin^2\left(\frac{\theta}{2}\right)\right),\\
\psi &= \lambda - \frac{4}{\alpha}(1 - \cos(3\theta)) + ch^2\\
&= \lambda - \frac{4}{\alpha}\left(18\sin^2\frac{\theta}{2} - 48\sin^4\frac{\theta}{2} + 32\sin^6\frac{\theta}{2}\right) + ch^2\\
&= 24\sin^2\left(\frac{\theta}{2}\right)\left(1 - \frac{2}{3}\sin^2\left(\frac{\theta}{2}\right)\right)\\
&\quad - \frac{4}{3}\left(1 - \frac{1}{3}h\sqrt{3c} + O(ch^2)\right)\left(18\sin^2\frac{\theta}{2} - 48\sin^4\frac{\theta}{2} + 32\sin^6\frac{\theta}{2}\right) + ch^2\\
&\approx 48\sin^4\frac{\theta}{2} + \frac{4}{9}h\sqrt{3c}\left(18\sin^2\frac{\theta}{2} - 48\sin^4\frac{\theta}{2}\right) + O(ch^2),\\
\mu^{(M)} &= \frac{\lambda}{\psi} \approx \frac{24\sin^2(\frac{\theta}{2})(1 - \frac{2}{3}\sin^2(\frac{\theta}{2}))}{48\sin^4\frac{\theta}{2} + \frac{4}{9}h\sqrt{3c}(18\sin^2\frac{\theta}{2} - 48\sin^4\frac{\theta}{2}) + O(ch^2)}.
\end{aligned}
$$

If $c = 0$, this simplifies to $\mu^{(M)} \approx \frac{1}{2\sin^2(\frac{\theta}{2})}$. Setting $\theta = \theta_1 = 2\pi h$ leads to $\mu^{(M)} = O(h^{-2})$. If $c > 0$, setting $\theta \approx 2\sqrt{h}$ (i.e., $s \approx \sqrt{n+1}/\pi$) leads to $\mu^{(M)} = O(h^{-1})$. Hence, we have the exact bounds

$$
\kappa^{(M)} = \begin{cases} O(h^{-1}), & \text{if } c > 0;\\ O(h^{-2}), & c = 0. \end{cases} \qquad \square
$$

Result 3.1. For the MILU preconditioned isotropic operator, the optimal value of c (the one that minimizes $\kappa^{(M)}$) occurs near $c_p = 12\pi^2$ for the periodic problem and $c_d = 3\pi^2$ for the Dirichlet problem.

In the derivation of this result, we make use of the following empirical results: the maximum value of μ occurs on the plane $\theta + \phi + \xi = 2\pi$ (and symmetric planes) where $\theta = \phi$ is small.

So, as in the latter part of the proof of Theorem 3.2, we have

$$\mu_{str}^{(M)} \approx \frac{24\sin^2(\frac{\theta}{2})(1-\frac{2}{3}\sin^2(\frac{\theta}{2}))}{48\sin^4\frac{\theta}{2}+\frac{4}{9}h\sqrt{3c}(18\sin^2\frac{\theta}{2}-48\sin^4\frac{\theta}{2})+O(h^2)}.$$

Let $x \equiv \sin^2\left(\frac{\theta_s}{2}\right)$.

$$\begin{aligned}\mu_{str}^{(M)} &\approx \frac{24x(1-\frac{2}{3}x)}{48x^2+\frac{4}{9}h\sqrt{3c}(18x-48x^2)+O(h^2)}\\ &= \frac{24(1-\frac{2}{3}x)}{48x+\frac{4}{9}h\sqrt{3c}(18-48x)+\frac{O(h^2)}{x}}\\ &\approx \frac{24}{48x+\frac{4}{9}h\sqrt{3c}(18-48x)+\frac{O(h^2)}{x}}.\end{aligned}$$

In the above and in the rest of the derivation we use that $x << 1$ and hence $x^2 << x$, which can be verified a posteriori from the derivation.

Setting $0 = \partial\mu/\partial x$ we get $x^2 \approx ch^2/48$, which implies $\sin^2\left(\frac{\theta_s}{2}\right) \approx \sqrt{c/48}h$. Substituting this result back into the equation for $\mu_{\max}^{(M)}$ yields

$$\mu_{str}^{(M)} \approx \frac{3}{2\sqrt{3c}h}.$$

For the minimum we use $\theta = \phi = \xi = 2\pi h$ and get

$$\mu_{\min}^{(M)} = \frac{12\sin^2(\pi h)}{12\sin^2(\pi h)+ch^2} = \frac{1}{1+\frac{c}{12\pi^2}}.$$

Thus as a function of c,

$$\kappa^{(M)}(c) = \frac{\mu_{\max}}{\mu_{\min}} = \frac{1+(c/12\pi^2)}{(2\sqrt{3c}h/3)}.$$

We want the c value that yields the minimum $\kappa^{(M)}$, hence we set $0 = (\partial\kappa/\partial c)$ to get $c_p \approx 12\pi^2$.

But this is the parameter c for the periodic approximation. To predict the optimal c_d value for the Dirichlet problem, we use $\delta_p = c_p{h_p}^2 = \delta_d = c_d{h_d}^2$ as in [7], where $h_p = \frac{h_d}{2}$ and find that

$$c_d \approx \frac{1}{4}c_p = 3\pi^2.$$

Acknowledgment. The first author would like to acknowledge the helpful discussions with Rosa Donat on the use of Maple in the proof of Theorem 3.2 and to Richard E. Little for general encouragement. Also, many thanks to the reviewers of this paper. Their comments, large and small, were greatly appreciated.

REFERENCES

[1] C. Ashcraft and R. Grimes, *On vectorizing incomplete factorizations and* SSOR *preconditioners*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 122–151.

[2] O. Axelsson and V. Eijkhout, *Robust vectorizable preconditioners for three dimensional elliptic difference equations with anisotropy*, in Algorithms and Applications on Vector and Parallel Computers, H. J. J. te Riele, Th. J. Dekker, and H. A. van der Vorst, eds., Elsevier Science Publishers B.V., North-Holland, 1987.

[3] O. Axelsson and G. Lindskog, *On the eigenvalue distribution of a class of preconditioning methods,* Numer. Math., 48 (1986a), pp. 479–498.

[4] ———, *On the rate of convergence of the preconditioned conjugate gradient methods*, Numer. Math., 48 (1986b), pp. 499-523.

[5] A. Brandt, *Rigorous Local Mode Analysis of Multigrid,* Tech. Report, Department of Applied Mathematics and Computer Science, Weizmann Institute of Science, Rehovot, Israel, December, 1988.

[6] T. F. Chan, *Fourier analysis of relaxed incomplete factorization preconditioners*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 668–680.

[7] T. F. Chan and H. C. Elman, *Fourier analysis of iterative methods for elliptic problems*, SIAM Rev., 31 (1989), pp. 20–49.

[8] T. F. Chan and G. Meurant, *Fourier analysis of block preconditioners*, University of California, Los Angeles, CA, Computational and Applied Math., CAM Report 90-04, February, 1990.

[9] B. W. Char, G. J. Fee, K. O. Geddes, G. H. Gonnet, and M. B. Monagan, *A tutorial introduction to Maple*, J. Symb. Comput., 2 (1986), pp. 179–200.

[10] H. C. Elman, *Relaxed and stabilized incomplete factorizations for non-self-adjoint linear systems*, BIT, 29 (1989), pp. 890-915.

[11] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1989.

[12] I. Gustafsson, *A class of first order factorization methods*, BIT, 18 (1978), pp. 142–156.

[13] J. A. Meijerink and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix,* Math. Comp., 31 (1977), pp. 148–162.

[14] H. A. van der Vorst, *High performance preconditioning,* SIAM J. Sci. Statist. Comput., 10 (1989), pp. 1174–1185.

[15] ———, ICCG *and related methods for* 3*D problems on vector computers*, Comput. Phys. Comm., 53 (1989), pp. 223–235.

[16] G. Wittum, *On the robustness of* ILU *smoothing*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 699–171.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 1, pp. 339–363, January 1992

LINE ITERATIVE METHODS FOR CYCLICALLY REDUCED DISCRETE CONVECTION-DIFFUSION PROBLEMS*

HOWARD C. ELMAN† AND GENE H. GOLUB‡

Abstract. An analytic and empirical study of line iterative methods for solving the discrete convection-diffusion equation is performed. The methodology consists of performing one step of the cyclic reduction method, followed by iteration on the resulting reduced system using line orderings of the reduced grid. Two classes of iterative methods are considered: block stationary methods, such as the block Gauss–Seidel and SOR methods, and preconditioned generalized minimum residual methods with incomplete LU preconditioners. New analysis extends convergence bounds for constant coefficient problems to problems with separable variable coefficients. In addition, analytic results show that iterative methods based on incomplete LU preconditioners have faster convergence rates than block Jacobi relaxation methods. Numerical experiments examine additional properties of the two classes of methods, including the effects of direction of flow, discretization, and grid ordering on performance.

Key words. iterative methods, line orderings, reduced system, convection-diffusion, elliptic operators

AMS(MOS) subject classifications. primary 65F10, 65N20; secondary 15A06

1. Introduction.

Consider the convection-diffusion equation

$$-\,[(pu_x)_x + (qu_y)_y] + ru_x + su_y = f \quad \text{on } \Omega \tag{1.1a}$$

$$\alpha u + \beta u_n = g \quad \text{on } \partial\Omega, \tag{1.1b}$$

where Ω is a smooth domain in $\mathbf{R}^2$ and $p > 0$, $q > 0$ on Ω. Discretization of (1.1) produces a linear system of equations

$$Au = f, \tag{1.2}$$

where u and f are now vectors in a finite-dimensional space, and A is a nonsymmetric matrix when r and s are nonzero. We are concerned with discretizations (principally, finite difference methods) for which each equation in (1.2) is centered at some mesh point (x_i, y_j), and the associated unknown u_{ij} depends only on its neighbors in the horizontal and vertical directions. That is, the equation centered at (x_i, y_j) has the form

$$a_{ij}u_{ij} = f_{ij} - b_{ij}u_{i,j-1} - c_{ij}u_{i-1,j} - d_{ij}u_{i+1,j} - e_{ij}u_{i,j+1}. \tag{1.3}$$

In this case, we say that (1.2) has a *computational molecule* of the form

$$\begin{array}{ccccc} & & e_{ij} & & \\ & & | & & \\ c_{ij} & \text{——} & a_{ij} & \text{——} & d_{ij}. \\ & & | & & \\ & & b_{ij} & & \end{array}$$

* Received by the editors April 5, 1990; accepted for publication (in revised form) November 26, 1990.

† Department of Computer Science, and Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland 20742. The work of this author was supported by National Science Foundation grants DMS-8607478 and ASC-8958544, and by U. S. Army Research Office grant DAAL-0389-K-0016.

‡ Department of Computer Science, Stanford University, Stanford, California 94305. The work of this author was supported by National Science Foundation grant DCR-8412314.

$\otimes_{15}$	$\times_{25}$	$\otimes_{35}$	$\times_{45}$	$\otimes_{55}$	$\times_{65}$
$\times_{14}$	$\otimes_{24}$	$\times_{34}$	$\otimes_{44}$	$\times_{54}$	$\otimes_{64}$
$\otimes_{13}$	$\times_{23}$	$\otimes_{33}$	$\times_{43}$	$\otimes_{53}$	$\times_{63}$
$\times_{12}$	$\otimes_{22}$	$\times_{32}$	$\otimes_{42}$	$\times_{52}$	$\otimes_{62}$
$\otimes_{11}$	$\times_{21}$	$\otimes_{31}$	$\times_{41}$	$\otimes_{51}$	$\times_{61}$

$\otimes_{13}$	$\times_{28}$	$\otimes_{14}$	$\times_{29}$	$\otimes_{15}$	$\times_{30}$
$\times_{25}$	$\otimes_{10}$	$\times_{26}$	$\otimes_{11}$	$\times_{27}$	$\otimes_{12}$
$\otimes_{7}$	$\times_{22}$	$\otimes_{8}$	$\times_{23}$	$\otimes_{9}$	$\times_{24}$
$\times_{19}$	$\otimes_{4}$	$\times_{20}$	$\otimes_{5}$	$\times_{21}$	$\otimes_{6}$
$\otimes_{1}$	$\times_{16}$	$\otimes_{2}$	$\times_{17}$	$\otimes_{3}$	$\times_{18}$

FIG. 1.1. *A* 6×5 *grid and a red-black ordering. Grid indices are shown on the left, and vector indices for a red-black ordering are shown on the right. Red points are denoted by "*$\otimes$*" and black points by "*$\times$*."*

When the system (1.2) has this property, the mesh points $\{(x_i, y_j)\}$ and unknowns $\{u_{ij}\}$ can be ordered with a red-black ordering so that every equation centered at a "red" point depends only on "black" unknowns, and every equation centered at a "black" point depends only on "red" unknowns. An example of a red-black ordering of a 6×5 grid is shown in Fig. 1.1. If u_{ij} is a black unknown, then by adding appropriate linear combinations of the equations for $u_{i\pm1,j}$ and $u_{i,j\pm1}$ to the equation for u_{ij}, we can eliminate the dependence of u_{ij} on its red neighbors. When this is done for every black equation, the result is a smaller linear system

$$A^{(b)}u^{(b)} = g^{(b)}, \tag{1.4}$$

where $u^{(b)}$ is the set of unknowns associated with black mesh points. In matrix notation, this process corresponds to ordering the rows and columns of A so that (1.2) has the form

$$\begin{pmatrix} D & C \\ E & F \end{pmatrix} \begin{pmatrix} u^{(r)} \\ u^{(b)} \end{pmatrix} = \begin{pmatrix} f^{(r)} \\ f^{(b)} \end{pmatrix},$$

where D and F are nonsingular diagonal matrices. Decoupling of the red points $u^{(r)}$ is equivalent to producing the system (1.4), where $A^{(b)} = F - ED^{-1}C$ and $g^{(b)} = f^{(b)} - ED^{-1}f^{(r)}$.

In [7], [8], we analyzed the convergence behavior of block iterative methods for solving the reduced system (1.4) derived from discretizations of (1.1). We considered block Jacobi, Gauss–Seidel, and successive overrelaxation (SOR) methods [25], [28], where the blockings (of the rows and columns of $A^{(b)}$) are derived from certain *line orderings* of the underlying reduced (black) grid. In particular, the unknown grid values $u^{(b)}$ can be grouped together either by individual lines of the grid, producing a class of *one-line* orderings, or by pairs of lines, producing *two-line* orderings (see §2). These orderings produce matrices with block Property A, so that the classical analysis of Gauss–Seidel and SOR methods [25], [28] can be used. The results of [7], [8] apply to problems with the constant coefficients $p(x, y) = q(x, y) = 1$, $r(x, y) = \sigma$, $s(x, y) = \tau$. They show that convergence is often very fast; in particular, for nonselfadjoint problems (σ or τ nonzero), convergence is typically faster than for selfadjoint problems. They also show that convergence rates for solving the reduced system are often faster than for solving the full system (1.2) by analogous line methods. These observations are in agreement with asymptotic results in [20] and the algebraic analysis of [13]. Related results for point iterative methods are given in [18].

In this paper, we extend the analysis of [7], [8] to separable problems, and we also use it to derive bounds on convergence behavior for stationary methods based on incomplete factorizations [17]. In addition, in a series of numerical experiments, we examine the effect of physically significant properties of the problem (1.1) on the performance of iterative methods applied to (1.4). Here, we consider both block relaxation methods and the preconditioned generalized minimum residual method (GMRES) [23], with preconditioning by incomplete factorizations [17]. We focus on the following issues:

1. For constant coefficient problems, the effect of the signs and magnitudes of r and s in (1.1). These quantities determine the direction and rate of flow associated with the convection in the model. The analysis of [7], [8] is sensitive to magnitudes but not to signs.
2. The effect of variable coefficients r and s. We consider problems both with and without turning points.
3. The effects of the choice of discretization on performance; we consider centered and upwind finite difference discretizations.
4. The first three issues do not address the issue of accuracy of the discrete solution. We also examine the effect of methods designed to improve accuracy in the presence of boundary layers, in particular, local mesh refinement and defect correction methods [12], [15].

An outline of the paper is as follows. In §2, we describe the reduced matrix $A^{(b)}$, and we present the ordering strategies and iterative methods used to solve (1.4), including some block red-black strategies of use for vector and parallel computations. In §3, we extend the analysis of [7], [8] to separable problems and incomplete factorizations. In §4, we describe the results of numerical experiments with constant coefficient problems. For several ordering strategies, we examine how performances of block stationary methods and preconditioned GMRES are affected by direction and rate of flow, choice of difference scheme, and use of local mesh refinement to resolve boundary layers. In §5, we compare experimental results with analytic bounds on convergence, for separable problems. In §6, we consider performance for some problems with nonseparable variable coefficients, i.e., where the flow varies in both direction and magnitude in Ω. Here we consider both centered and upwind finite differences, as well as a difference scheme used to implement defect correction methods. Finally, in §7 we make some concluding remarks.

2. The reduced system and line iterative methods. Let u_{ij} be a black point not next to the boundary $\partial\Omega$. Elimination of the unknowns $u_{i\pm1,j}$ and $u_{i,j\pm1}$ from (1.3) produces an equation in the reduced system with the computational molecule shown in Fig. 2.1. The value "*" in the center is

$$a_{ij} - \frac{b_{ij}e_{i,j-1}}{a_{i,j-1}} - \frac{c_{ij}d_{i-1,j}}{a_{i-1,j}} - \frac{d_{ij}c_{i+1,j}}{a_{i+1,j}} - \frac{e_{ij}b_{i,j+1}}{a_{i,j+1}},$$

and the right-hand side is perturbed by an average of neighboring values,

$$g_{ij}^{(b)} = f_{ij} - \frac{b_{ij}f_{i,j-1}}{a_{i,j-1}} - \frac{c_{ij}f_{i-1,j}}{a_{i-1,j}} - \frac{d_{ij}f_{i+1,j}}{a_{i+1,j}} - \frac{e_{ij}f_{i,j+1}}{a_{i,j+1}}.$$

The line ordering strategies for the reduced grid are outlined as follows (see [7], [8] for further details). In the *natural one-line* ordering, points of the reduced grid are grouped together by diagonal lines, e.g., oriented in the NW–SE direction. The left side of Fig. 2.2 shows an example for a 6 × 5 grid. Here, the kth line consists of all points with grid indices (i, j) such that $i + j = 2k + 1$. (Compare with the left side of Fig. 1.1.) Thus, in Fig. 2.2, the first line consists of the points $\{1, 2\}$, the second line consists of the points $\{3, 4, 5, 6\}$, etc. In the *natural two-line* ordering, points

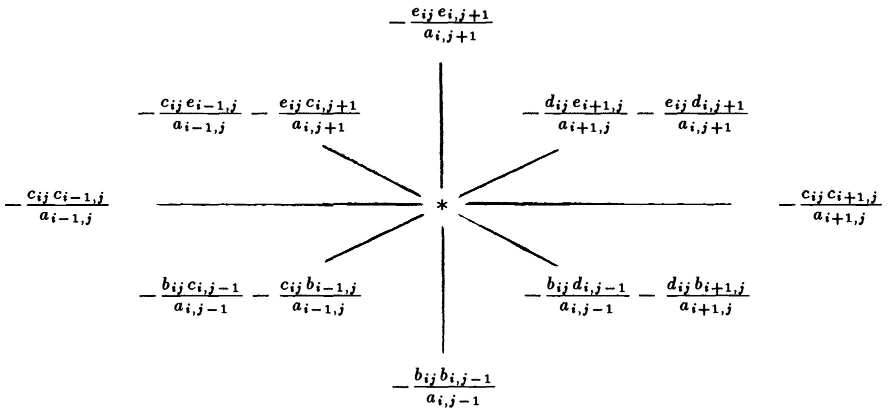

$-\frac{e_{ij}e_{i,j+1}}{a_{i,j+1}}$
$-\frac{c_{ij}e_{i-1,j}}{a_{i-1,j}}$
$-\frac{e_{ij}c_{i,j+1}}{a_{i,j+1}}$
$-\frac{d_{ij}e_{i+1,j}}{a_{i+1,j}}$
$-\frac{e_{ij}d_{i,j+1}}{a_{i,j+1}}$
$-\frac{c_{ij}c_{i-1,j}}{a_{i-1,j}}$
$*$
$-\frac{c_{ij}c_{i+1,j}}{a_{i+1,j}}$
$-\frac{b_{ij}c_{i,j-1}}{a_{i,j-1}}$
$-\frac{c_{ij}b_{i-1,j}}{a_{i-1,j}}$
$-\frac{b_{ij}d_{i,j-1}}{a_{i,j-1}}$
$-\frac{d_{ij}b_{i+1,j}}{a_{i+1,j}}$
$-\frac{b_{ij}b_{i,j-1}}{a_{i,j-1}}$

FIG. 2.1. *The computational molecule for the reduced system.*

•	$\times_{11}$	•	$\times_{14}$	•	$\times_{15}$
$\times_6$	•	$\times_{10}$	•	$\times_{13}$	•
•	$\times_5$	•	$\times_9$	•	$\times_{12}$
$\times_2$	•	$\times_4$	•	$\times_8$	•
•	$\times_1$	•	$\times_3$	•	$\times_7$

•	$\times_{13}$	•	$\times_{14}$	•	$\times_{15}$
$\times_7$	•	$\times_9$	•	$\times_{11}$	•
•	$\times_8$	•	$\times_{10}$	•	$\times_{12}$
$\times_1$	•	$\times_3$	•	$\times_5$	•
•	$\times_2$	•	$\times_4$	•	$\times_6$

FIG. 2.2. *Natural one-line (left) and two-line (right) orderings of the reduced* 6×5 *grid.*

are grouped together by pairs of either horizontal or vertical lines. The right side of Fig. 2.2 shows an example of a horizontal grouping for a 6 × 5 grid. The points in the kth group are those with grid indices (i,j) such that $k-1 < j/2 \le k$. If the number of lines is odd, the last group consists of a single line, as in the group $\{13,14,15\}$. For both these strategies, $A^{(b)}$ is a block tridiagonal matrix; let D denote its block diagonal. For the one-line ordering, each block of D is a tridiagonal matrix, and for the two-line ordering, each block of D is a pentadiagonal matrix (except possibly the last block, which may be tridiagonal). It is also useful (e.g., for parallel computations, see [8]) to define *line red-black* variants of these orderings, in which alternating lines (or line pairs) are assigned opposite colors. For example, for the one-line version, let the sets $\{1,2\}$, $\{7,8,9,10,11\}$, and $\{15\}$ be denoted as "red" lines, and the others as "black" lines. Then every equation centered at a point in a red line depends only on that red line and the neighboring black lines; an analogous statement holds for equations centered on black lines. For the red-black one-line ordering, all red lines are ordered first, followed by all black lines. The red-black two-line ordering is defined in similar fashion.

For any of these line orderings, let

$$A^{(b)} = D - C = (D - L) - U, \tag{2.1}$$

where D is the block diagonal part of $A^{(b)}$ and L and U are the lower and upper triangular parts, respectively, of the block off-diagonal part of $A^{(b)}$. We consider several block stationary methods based on the splittings (2.1). The block Jacobi iteration is given by

$$u_{k+1}^{(b)} = D^{-1}Cu_k^{(b)} + D^{-1}g^{(b)},$$

and the block SOR iteration is

$$u_{k+1}^{(b)} = (D - \omega L)^{-1}[(1-\omega)D + \omega U]u_k^{(b)} + \omega(D - \omega L)^{-1}g^{(b)}. \tag{2.2}$$

The block Gauss–Seidel iteration corresponds to the case $\omega = 1$ in 2.2. In all cases, $A^{(b)}$ is block consistently ordered, so that [28]

$$\rho((D-L)^{-1}U) = [\rho(D^{-1}C)]^2, \tag{2.3}$$

where $\rho(X)$ denotes the spectral radius of a matrix X.

In addition, we consider the use of the ILU(0) incomplete factorization [17] applied to $A^{(b)}$ for each of the orderings. This factorization is defined as

$$M = (\hat{D} - \hat{L})\hat{D}^{-1}(\hat{D} - \hat{U}), \tag{2.4}$$

where $\hat{D}$ is a diagonal matrix; $\hat{L}$ and $\hat{U}$ are strictly lower triangular and upper triangular, respectively; the nonzero structure of $\hat{D} - \hat{L} - \hat{U}$ is the same as that of $A^{(b)}$; and the entries of M are the same as the corresponding entries of $A^{(b)}$ wherever the latter are nonzero. We will examine the use of this factorization as a preconditioner for GMRES.

3. Analysis of separable problems and the ILU(0) factorization. We will be concerned with finite difference discretizations of (1.1). For example, on a uniform grid with mesh size h, let standard second order differences [11] be used for the second derivative terms. If centered differences are used for the first derivative terms, then after scaling by h^2, the values in the computational molecule are given by

$$\begin{aligned} a_{ij} &= p(x_{i+1/2}, y_j) + p(x_{i-1/2}, y_j) + q(x_i, y_{j+1/2}) + q(x_i, y_{j-1/2}), \\ b_{ij} &= -(q(x_i, y_{j-1/2}) + s(x_i, y_j)h/2), \qquad d_{ij} = -(p(x_{i+1/2}, y_j) - r(x_i, y_j)h/2), \\ c_{ij} &= -(p(x_{i-1/2}, y_j) + r(x_i, y_j)h/2), \qquad e_{ij} = -(q(x_i, y_{j+1/2}) - s(x_i, y_j)h/2). \end{aligned}$$

If upwind differencing is used for the first derivatives, then (for the case $r(x_i, y_j) > 0$, $s(x_i, y_j) > 0$) the values are

$$\begin{aligned} a_{ij} &= p(x_{i+1/2}, y_j) + p(x_{i-1/2}, y_j) + q(x_i, y_{j+1/2}) + q(x_i, y_{j-1/2}) \\ &\quad + r(x_i, y_j)h + s(x_i, y_j)h, \\ b_{ij} &= -(q(x_i, y_{j-1/2}) + s(x_i, y_j)h), \qquad d_{ij} = -p(x_{i+1/2}, y_j), \\ c_{ij} &= -(p(x_{i-1/2}, y_j) + r(x_i, y_j)h), \qquad e_{ij} = -q(x_i, y_{j+1/2}). \end{aligned}$$

If instead, $s(x_i, y_j) < 0$, then $b_{ij} = -q(x_i, y_{j-1/2})$, $e_{ij} = -(q(x_i, y_{j+1/2}) - s(x_i, y_j)h)$, and $s(x_i, y_j)h$ is replaced by $-s(x_i, y_j)h$ in the expression for a_{ij}. The case $r(x_i, y_j) < 0$ is handled in an analogous manner.

If Ω is a rectangular domain and the coefficients of (1.1a) satisfy

$$p = p(x), \quad q = q(y), \quad r = r(x), \quad s = s(y),$$

then the differential operator of (1.1) is *separable* [26]. In this case, the discrete coefficients of (1.3) satisfy

$$\begin{aligned} &a_{ij} = a_i^{(x)} + a_j^{(y)} \\ &b_{ij} = b_j, \quad c_{ij} = c_i, \quad d_{ij} = d_i, \quad e_{ij} = e_j. \end{aligned} \tag{3.1}$$

Our convergence analysis is based on symmetrizing the reduced matrix $A^{(b)}$ by a diagonal similarity transformation. The following result gives circumstances under which $A^{(b)}$ can be symmetrized when it comes from a separable operator. In the analysis, matrix entries are referenced using indices from the underlying reduced grid, as shown in Fig. 2.1. That is, every nonzero entry of the row of $A^{(b)}$ associated with the (i, j) grid point is referenced using subscripts i and j. For example, the entry corresponding to the point southwest of the center of the computational molecule (see Fig. 2.1) is denoted by

$$-b_j c_i \left(\frac{1}{a_{i,j-1}} + \frac{1}{a_{i-1,j}} \right),$$

where the numerator is expressed using the notation of (3.1).

THEOREM 1. *If the operator of* (1.1) *is separable and* $c_i d_{i-1}$ *and* $b_j e_{j-1}$ *have the same sign for all* i *and* j, *then the reduced matrix* $A^{(b)}$ *can be symmetrized with a real diagonal similarity transformation.*

Proof. We seek a diagonal matrix Q such that $Q^{-1}A^{(b)}Q$ is symmetric. Let $A^{(b)}$ be ordered by the natural one-line ordering, so that its rows and columns are grouped into l blocks corresponding to l individual lines. Let Q be ordered the same way.

First consider the block diagonal D, which is a tridiagonal matrix. Any two successive rows of a block of D, corresponding to the (i, j) and $(i-1, j+1)$ mesh points, contain the 2×2 sub-block

$$\begin{pmatrix} * & -c_i e_j \left(\frac{1}{a_{i-1,j}} + \frac{1}{a_{i,j+1}}\right) \\ -b_{j+1} d_{i-1} \left(\frac{1}{a_{i-1,j}} + \frac{1}{a_{i,j+1}}\right) & * \end{pmatrix},$$

where "*" denotes a diagonal entry. If q_{ij} is known, then $q_{i-1,j+1}$ must be chosen so that

$$q_{i-1,j+1}^{-1} b_{j+1} d_{i-1} \left(\frac{1}{a_{i-1,j}} + \frac{1}{a_{i,j+1}} \right) q_{ij} = q_{ij}^{-1} c_i e_j \left(\frac{1}{a_{i-1,j}} + \frac{1}{a_{i,j+1}} \right) q_{i-1,j+1}.$$

Thus, within the blocks of Q, successive entries must satisfy

$$q_{i-1,j+1} = \left(\frac{b_{j+1} d_{i-1}}{c_i e_j} \right)^{1/2} q_{ij}. \tag{3.2}$$

For symmetrizing D, the first entry of each block of Q may be arbitrary.

To symmetrize the off-diagonal blocks of $A^{(b)}$, we require

$$Q_k^{-1} A_{k,k-1}^{(b)} Q_{k-1} = (Q_{k-1}^{-1} A_{k-1,k}^{(b)} Q_k)^T, \tag{3.3}$$

where k is a block (or line) index, $2 \le k \le l$. As in [7], there are three cases, corresponding to $2 \le k < l/2 + 1$, $k = l/2 + 1$ (l even), and $l/2 + 1 < k$. In the case $2 \le k \le l/2 + 1$, a careful specification of the entries of Q and $A^{(b)}$ shows that (3.3) is equivalent to the following three scalar relations:

$$q_{ij} = \left(\frac{c_i c_{i-1}}{d_{i-1} d_{i-2}} \right)^{1/2} q_{i-2,j}, \tag{3.4}$$

$$q_{i-1,j+1} = \left(\frac{b_{j+1} c_{i-1}}{d_{i-2} e_j} \right)^{1/2} q_{i-2,j}, \tag{3.5}$$

$$q_{i-2,j+2} = \left(\frac{b_{j+1}b_{j+2}}{e_j e_{j+1}}\right)^{1/2} q_{i-2,j}. \tag{3.6}$$

These relations specify three successive entries of Q_k in terms of a single entry of Q_{k-1} (where $k = (i+j-1)/2$). Since the first entry of Q_k is arbitrary, (3.4) can be used to define it. However, once this entry is defined, all subsequent entries are determined by (3.2). Thus, it is necessary to show that (3.4)–(3.6) are consistent with (3.2). But application of (3.2) and (3.4) in either order results in (3.5), showing that both (3.4) and (3.5) are consistent with (3.2). Similarly, (3.6) follows directly from (3.2) and (3.5).

The arguments for the cases $k = l/2+1$ (l even) and $l/2+1 < k$ are essentially the same and we omit the details. A sufficient condition to guarantee that all the required square roots are well defined is that $c_i d_{i-1}$ and $b_j e_{j-1}$ have the same sign for all i and j.

Finally, note that this analysis is not restricted to the natural one-line ordering. If $A^{(b)}$ is symmetrically permuted into some other order, giving the permuted matrix $\tilde{A}^{(b)}$, then for an analogous permutation of Q to $\tilde{Q}$, $\tilde{Q}^{-1}\tilde{A}^{(b)}\tilde{Q}$ is also symmetric. □

Remark 1. For the centered difference discretization, necessary and sufficient conditions to ensure that all $c_i d_{i-1}$ and $b_j e_{j-1}$ have the same sign are that either

$$\max_i \left[\max\left(\left|\frac{r(x_i)h}{2p(x_{i-1/2})}\right|, \left|\frac{r(x_{i-1})h}{2p(x_{i-1/2})}\right|\right)\right] < 1 \tag{3.7}$$

and

$$\max_j \left[\max\left(\left|\frac{s(y_j)h}{2q(y_{j-1/2})}\right|, \left|\frac{s(y_{j-1})h}{2q(y_{j-1/2})}\right|\right)\right] < 1;$$

or

$$\min_i \left[\min\left(\left|\frac{r(x_i)h}{2p(x_{i-1/2})}\right| \left|\frac{r(x_{i-1})h}{2p(x_{i-1/2})}\right|\right)\right] > 1$$

and

$$\min_j \left[\min\left(\left|\frac{s(y_j)h}{2q(y_{j-1/2})}\right|, \left|\frac{s(y_{j-1})h}{2q(y_{j-1/2})}\right|\right)\right] > 1.$$

In contrast, the full system (1.2) can be symmetrized by a diagonal similarity transformation if and only if the conditions (3.7) hold. For upwind differences, it is always the case that $c_i d_{i-1} > 0$ and $b_j e_{j-1} > 0$ for all i, j.

Let $\hat{A}^{(b)} = Q^{-1}A^{(b)}Q$ denote the symmetrized reduced matrix, when it exists, for any of the strategies under consideration. Fig. 3.1 shows the resulting computational molecule. Let

$$\hat{A}^{(b)} = \hat{D} - \hat{C}$$

denote the block Jacobi splitting, where $\hat{D} = Q^{-1}DQ$, $\hat{C} = Q^{-1}CQ$. Note that $\hat{D}^{-1}\hat{C} = Q^{-1}D^{-1}CQ$, so that the eigenvalues of $D^{-1}C$ are the same as those of $\hat{D}^{-1}\hat{C}$, and in particular they are real. Let $\mathcal{L}_\omega = (D-\omega L)^{-1}[(1-\omega)D + \omega U]$ denote the block SOR iteration matrix. The following result is then a straightforward application of the analysis of the block SOR method [28].

COROLLARY 1. *If $A^{(b)}$ is the reduced matrix derived from a separable operator, and $c_i d_{i-1}$ and $b_j e_{j-1}$ have the same sign for all i and j, then $\rho(D^{-1}C) = \rho(\hat{D}^{-1}\hat{C})$. If $\rho(D^{-1}C) < 1$, then $\rho(\mathcal{L}_{\omega^*}) = \omega^* - 1$, where $\omega^* = 2/(1+\sqrt{1+[\rho(D^{-1}C)]^2})$ minimizes $\rho(\mathcal{L}_\omega)$.*

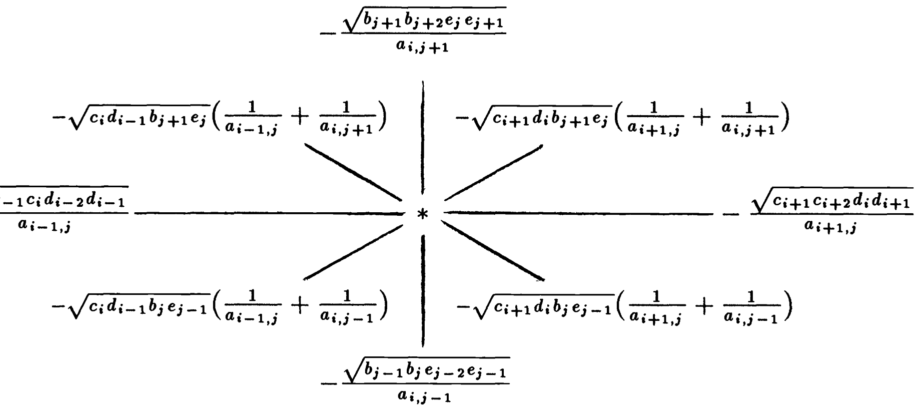

$-\frac{\sqrt{b_{j+1}b_{j+2}e_je_{j+1}}}{a_{i,j+1}}$
$-\sqrt{c_id_{i-1}b_{j+1}e_j}\left(\frac{1}{a_{i-1,j}}+\frac{1}{a_{i,j+1}}\right)$
$-\sqrt{c_{i+1}d_ib_{j+1}e_j}\left(\frac{1}{a_{i+1,j}}+\frac{1}{a_{i,j+1}}\right)$
$-\frac{\sqrt{c_{i-1}c_id_{i-2}d_{i-1}}}{a_{i-1,j}}$
$*$
$-\frac{\sqrt{c_{i+1}c_{i+2}d_id_{i+1}}}{a_{i+1,j}}$
$-\sqrt{c_id_{i-1}b_je_{j-1}}\left(\frac{1}{a_{i-1,j}}+\frac{1}{a_{i,j-1}}\right)$
$-\sqrt{c_{i+1}d_ib_je_{j-1}}\left(\frac{1}{a_{i+1,j}}+\frac{1}{a_{i,j-1}}\right)$
$-\frac{\sqrt{b_{j-1}b_je_{j-2}e_{j-1}}}{a_{i,j-1}}$

FIG. 3.1. *The computational molecule for the symmetrized reduced system in the separable case.*

Remark 2. It may be possible to establish the requirements of Corollary 1 a priori. Sufficient conditions to guarantee that $\rho(D^{-1}C) < 1$ are that the original matrix A be a diagonally dominant M-matrix,[1] which is always the case for upwind differences, and is also true for centered differences for small enough h. (In addition, if A is an M-matrix, then so is $A^{(b)}$ [10].) Even if Corollary 1 cannot be invoked from an a priori examination of matrix entries, it may still be useful as a guideline for practical computation. For example, for constant coefficient problems, empirical evidence and Fourier analysis suggest that $\rho(D^{-1}C) < 1$ in cases where c_id_{i-1} and b_je_{j-1} are both negative but A is not a diagonally dominant M-matrix. A good value for the SOR parameter could be computed from a dynamic estimation of $\rho(D^{-1}C)$, e.g., using the methods of [14, §9]. In addition, note that it is not necessary to compute Q or $\hat{A}^{(b)}$ in order to apply this result, see [7].

The following result contains upper bounds on $\rho(D^{-1}C)$ (for both one-line and two-line splittings), for separable problems.

COROLLARY 2. *Let $A^{(b)}$ come from a separable operator discretized on a uniform square grid of mesh width h, and assume that*

$$a_i^{(x)} \geq \alpha^{(x)}, \quad a_j^{(y)} \geq \alpha^{(y)}, \quad 0 < c_{i+1}d_i \leq \xi, \quad 0 < b_{j+1}e_j \leq \eta, \tag{3.8}$$

for all i, j. If $A^{(b)} = D - C$ is a one-line Jacobi splitting and

$$\alpha^{(x)} + \alpha^{(y)} \geq \sqrt{2}\,(\sqrt{\xi} + \sqrt{\eta}), \tag{3.9}$$

then

$$\rho(D^{-1}C) \leq \frac{2(\sqrt{\xi} + \sqrt{\eta})^2}{(\alpha^{(x)} + \alpha^{(y)})^2 - 2(\sqrt{\xi} + \sqrt{\eta})^2 + 4\sqrt{\xi\eta}\,(1 - \cos \pi h)}. \tag{3.10}$$

If $A^{(b)} = D - C$ is a two-line Jacobi splitting and

$$(\alpha^{(x)} + \alpha^{(y)})^2 \geq 2(\sqrt{\xi} + \sqrt{\eta})^2 + 2\xi, \tag{3.11}$$

[1] A nonsingular matrix X is an M-matrix if $X_{ij} \leq 0$ for $i \neq j$ and $X^{-1} \geq 0$.

then

$$(3.12)\qquad \rho(D^{-1}C) \leq \frac{2\,\eta\,\cos 2\pi h + 4\sqrt{\xi\eta}\cos\pi h}{(\alpha^{(x)}+\alpha^{(y)})^2 - 2(\sqrt{\xi}+\sqrt{\eta})^2 - 2\xi + 4\sqrt{\xi\eta}\,(1-\cos\pi h) + 4\xi\,(1-\cos^2\pi h)} + o(h^2).$$

Proof. Using Corollary 1, we have (for any ordering)

$$\rho(D^{-1}C) = \rho(\hat{D}^{-1}\hat{C}) \leq \|\hat{D}^{-1}\|_2\|\hat{C}\|_2 = \rho(\hat{D})\rho(\hat{C}).$$

Consider the one-line orderings. By (3.8), all nonzero off-diagonal entries of $\hat{D}$ are bounded below by $-2\sqrt{\xi\eta}/(\alpha^{(x)}+\alpha^{(y)})$, and all diagonal entries of $\hat{D}$ are bounded below by

$$\alpha^{(x)}+\alpha^{(y)} - 2\xi/(\alpha^{(x)}+\alpha^{(y)}) - 2\eta/(\alpha^{(x)}+\alpha^{(y)}).$$

Thus, $\hat{D} \geq \widetilde{D}$, where each block of $\widetilde{D}$ is a constant coefficient tridiagonal matrix

$$(3.13)\, tri\left[-\frac{2\sqrt{\xi\eta}}{\alpha^{(x)}+\alpha^{(y)}},\ \alpha^{(x)}+\alpha^{(y)} - \frac{2\xi}{\alpha^{(x)}+\alpha^{(y)}} - \frac{2\eta}{\alpha^{(x)}+\alpha^{(y)}},\ -\frac{2\sqrt{\xi\eta}}{\alpha^{(x)}+\alpha^{(y)}}\right].$$

The size of this block depends on the line from which it is derived. Assumption (3.9) implies that each block (3.13) and, therefore, each corresponding block of $\hat{D}$, is an irreducibly diagonally dominant M-matrix. Hence, the Perron–Frobenius theory implies $\rho(\hat{D}^{-1}) \leq \rho(\widetilde{D}^{-1})$. Similarly, by (3.8), $0 \leq \hat{C} \leq \widetilde{C}$, where $\widetilde{C}$ is a matrix with the same nonzero structure as that of $\hat{C}$ in which all occurences of $c_i d_{i-1}$, $b_j e_{j-1}$, and a_{ij} are replaced by ξ, η, and $\alpha^{(x)}+\alpha^{(y)}$, respectively. Consequently, $\rho(\hat{C}) \leq \rho(\widetilde{C})$, and we have

$$(3.14)\qquad \rho(\hat{D}^{-1})\rho(\hat{C}) \leq \rho(\widetilde{D}^{-1})\rho(\widetilde{C}),$$

where the right side of the inequality contains constant coefficient matrices. The bound (3.10) is determined from the maximum eigenvalue of $\widetilde{D}^{-1}$ and use of Gerschgorin's theorem for $\widetilde{C}$. (See [7, Thm. 4])

For the two-line ordering, the blocks of D and $\hat{D}$ are pentadiagonal matrices, and $\hat{D} \geq \widetilde{D}$, where each block of $\widetilde{D}$ is a constant coefficient pentadiagonal matrix,

$$penta\left[-\frac{\xi}{\alpha^{(x)}+\alpha^{(y)}},\ -\frac{2\sqrt{\xi\eta}}{\alpha^{(x)}+\alpha^{(y)}},\ \alpha^{(x)}+\alpha^{(y)} - \frac{2\xi}{\alpha^{(x)}+\alpha^{(y)}} - \frac{2\eta}{\alpha^{(x)}+\alpha^{(y)}},\right.$$
$$\left.-\frac{2\sqrt{\xi\eta}}{\alpha^{(x)}+\alpha^{(y)}},\ -\frac{\xi}{\alpha^{(x)}+\alpha^{(y)}}\right],$$

which is assumed in (3.11) to be diagonally dominant. In addition, exactly as above, $0 \leq \hat{C} \leq \widetilde{C}$, where $\widetilde{C}$ has the same nonzero structure as $\hat{C}$. The bound (3.12) then follows from [8, Thm. 5]. □

We will examine the use of this result in §5.

Remark 3. In the interest of brevity, we have limited our attention to the natural and red-black variants of the one-line orderings. Other variants, called "torus" one-line orderings, collect some individual lines together into sets of equal sizes; this is useful for parallel and vector computations. (See [8], [16].) All of the analysis of this section also applies to the torus orderings.

We now turn our attention to incomplete LU (ILU) factorizations. Let B be an M-matrix of order N, and let $\mathcal{N} \subseteq \{(i,j)\,|\,1 \le i,j \le N\}$ be an index set containing all diagonal indices (i,i). It is shown in [17] that there is a unique ILU factorization LU such that L is unit lower triangular, U is upper triangular, $l_{ij} = 0$ and $u_{ij} = 0$ for $(i,j) \notin \mathcal{N}$, and $[LU - B]_{ij} = 0$ for $(i,j) \in \mathcal{N}$. The ILU(0) factorization of (2.4) is a particular example. The following result of Beauwens ([2, Thm. 4.4]) can be used to compare the ILU(0) splitting to the block Jacobi splitting.

THEOREM 2. *Let B be a nonsingular M-matrix, and let*

$$B = M_1 - R_1 = M_2 - R_2, \tag{3.15}$$

where $M_1 = L_1U_1$ and $M_2 = L_2U_2$ are incomplete factorizations of B such that the set of matrix indices for which $L_1 + U_1$ is permitted to be nonzero is contained in the set of indices for which $L_2 + U_2$ is permitted to be nonzero. Then

$$\rho(M_2^{-1}R_2) \le \rho(M_1^{-1}R_1). \tag{3.16}$$

The analysis in [2] actually applies to a more general class of factorizations than the standard ILU factorization. Theorem 2 can be proved using the result of Woźnicki [27], that if (3.15) represents two regular splittings of a matrix B for which $B^{-1} \ge 0$, then

$$M_2^{-1} \ge M_1^{-1} \tag{3.17}$$

implies the conclusion (3.16). It is straightforward to establish (3.17) for ILU factorizations.

COROLLARY 3. *Suppose $A^{(b)}$ is an M-matrix, ordered using any of the orderings under consideration. Let $A^{(b)} = M - R$ where M is the* ILU(0) *factorization of $A^{(b)}$, and let $A^{(b)} = D - C$ denote the block Jacobi splitting. Then $\rho(M^{-1}R) \le \rho(D^{-1}C)$.*

Proof. The index set of nonzeros of the block diagonal D is a proper subset of the nonzero index set for the ILU(0) factorization. The result then follows from Theorem 2, where (the factorization of) D is viewed as an incomplete factorization of $A^{(b)}$. □

Thus, we expect convergence of a stationary method based on the ILU(0) splitting to be at least as fast as that for the block Jacobi method, for any ordering. (The work per step for the Jacobi method will be smaller, though.) In particular, as observed in [7], [8], convergence should be faster for mildly nonsymmetric problems arising from nonselfadjoint operators than for symmetric ones derived from selfadjoint operators. Combining the ILU(0) factorization with an acceleration scheme such as GMRES (i.e., using M as a preconditioner) should further improve convergence. Numerical experiments with the ILU(0) preconditioner that support this statement are presented in the following sections.

4. Experimental results: Constant coefficient problems.

In this section, we examine the numerical performance of the block Gauss–Seidel and SOR stationary methods, and GMRES(5) with the ILU(0) preconditioner, for solving the constant coefficient model problem

$$-\Delta u + \sigma u_x + \tau u_y = 0 \tag{4.1}$$

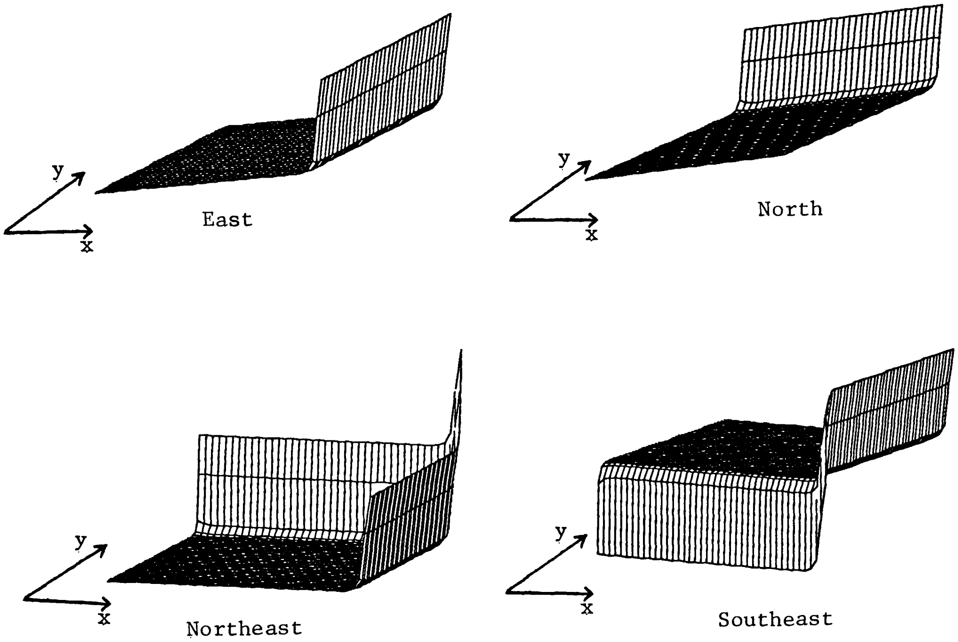


FIG. 4.1. *Plots of the constant coefficient solution for four different directions of flow.*

on $\Omega = (0,1) \times (0,1)$. Dirichlet boundary conditions on $\partial\Omega$ are determined from the exact solution

$$u(x,y) = \frac{e^{\sigma x}-1}{e^{\sigma}-1} + \frac{e^{\tau y}-1}{e^{\tau}-1} \tag{4.2}$$

on $\overline{\Omega}$. The vector (σ, τ) represents a velocity field with the signs of σ or τ determining the direction of flow. We consider eight types of velocity fields, corresponding to eight flow directions in the (x, y)-plane:

East (E):	$\sigma > 0, \tau = 0,$	Northeast (NE):	$\sigma = \tau > 0,$
West (W):	$\sigma < 0, \tau = 0,$	Southeast (SE):	$\sigma = -\tau > 0,$
North (N):	$\sigma = 0, \tau > 0,$	Northwest (NW):	$\sigma = -\tau < 0,$
South (S):	$\sigma = 0, \tau < 0,$	Southwest (SW):	$\sigma = \tau < 0.$

(For $\sigma = 0$ or $\tau = 0$, (4.2) is defined using the limit, i.e., $\lim_{\sigma \to 0}(e^{\sigma x}-1)/(e^{\sigma}-1) = x$.) In addition, the solution (4.2) has a boundary layer at any outflow boundary, i.e., near $x = 1$ for positive σ and $x = 0$ for negative σ, and similarly for y and τ. Plots of the solution for four such (σ, τ) combinations, corresponding to flows in the east, north, northeast, and southeast directions, are shown in Fig. 4.1. Our concern is to determine the effects of direction and magnitude of flow, ordering of unknowns, discretization scheme, and use of local mesh refinement on the performance of reduced system iterative methods.

Details of the numerical experiments are as follows. The experiments were performed on a VAX-8600 in double precision Fortran. Reported iteration counts are averages over three initial guesses consisting of vectors of random numbers in $[-1, 1]$. The stopping criterion for all methods was $\|r_i\|_2/\|r_0\|_2 \leq 10^{-6}$. A maximum of 150 iterations was permitted; an asterisk "*" in any table entry below indicates that for at least one initial guess, the stopping criterion was not met after 150 steps. (We remark that when the block stationary methods failed to meet the stopping criterion, they never "stagnated," i.e., they appeared to be converging.) For red-black SOR, the first iteration was performed with $\omega = 1$, as in [24]. Preconditioned GMRES was performed with right-oriented preconditioning, i.e., GMRES was applied to the preconditioned problem $A^{(b)}M^{-1}\hat{u}^{(b)} = g^{(b)}$, where M is the preconditioning matrix and $u^{(b)} = M^{-1}\hat{u}^{(b)}$. The construction of the reduced matrices and the experiments with GMRES were performed with PCGPAK [21].

TABLE 4.1
Average iteration counts for the natural one-line ordering, for eight flow directions.

		E	W	N	S	NE	SE	NW	SW	
	max	$\sigma > 0$,	$\sigma < 0$,	$\sigma = 0$,	$\sigma = 0$,	$\sigma = \tau > 0$	$\sigma = -\tau > 0$	$\sigma = -\tau < 0$	$\sigma = \tau < 0$	Avg.
	$\lvert\sigma\rvert$, $\lvert\tau\rvert$	$\tau = 0$	$\tau = 0$	$\tau > 0$	$\tau < 0$					
Gauss–Seidel	10	124	148	124	149	63	101	101	117	116
	50	17	35	17	35	5	19	19	35	23
	100	7	26	7	26	8	14	14	40	18
	200	12	31	12	31	32	28	28	71	31
	500	53	75	53	75	124	123	122	150*	97*
	1000	150*	150*	150*	150*	150*	150*	150*	150*	150*
SOR	10	34	47	34	47	22	33	33	44	37
	50	13	30	13	30	4	17	17	32	19
	100					5	15	15	33	17
	200					11	24	23	36	23
	500					27	37	37	42	36
	1000					54	61	60	65	60
GMRES / ILU	10	15	16	14	15	11	16	17	14	15
	50	12	12	8	8	4	16	16	5	10
	100	11	11	6	6	5	15	14	6	9
	200	10	10	4	4	7	14	13	7	9
	500	10	10	4	4	11	17	17	12	11
	1000	9	9	4	4	18	22	21	20	13

TABLE 4.2
Average iteration counts for the red-black one-line ordering, for eight flow directions.

		E	W	N	S	NE	SE	NW	SW	
	max	$\sigma > 0$,	$\sigma < 0$,	$\sigma = 0$,	$\sigma = 0$,	$\sigma = \tau > 0$	$\sigma = -\tau > 0$	$\sigma = -\tau < 0$	$\sigma = \tau < 0$	Avg.
	$\lvert\sigma\rvert$, $\lvert\tau\rvert$	$\tau = 0$	$\tau = 0$	$\tau > 0$	$\tau < 0$					
Gauss–Seidel	10	132	144	133	144	82	103	103	108	119
	50	23	24	23	24	19	18	18	21	23
	100	13	14	13	14	22	11	11	26	15
	200	20	21	20	21	49	27	27	57	30
	500	63	69	63	69	140*	128	128	150*	102*
	1000	150*	150*	150*	150*	150*	150*	150*	150*	150*
SOR	10	33	34	33	34	27	29	30	28	31
	50	23	24	23	24	19	18	18	21	21
	100					18	14	14	19	16
	200					21	23	22	22	22
	500					31	35	34	33	33
	1000					57	58	57	57	57
GMRES / ILU	10	24	28	25	30	27	29	27	32	28
	50	29	35	26	35	37	22	20	51	32
	100	28	33	27	35	38	16	16	53	31
	200	28	34	28	34	37	14	14	53	30
	500	31	34	31	33	35	27	26	49	33
	1000	39	42	39	43	46	52	52	53	46

TABLE 4.3
Average iteration counts for the natural two-line ordering, for eight flow directions.

		E	W	N	S	NE	SE	NW	SW	
	max	$\sigma > 0$,	$\sigma < 0$,	$\sigma = 0$,	$\sigma = 0$,	$\sigma = \tau > 0$	$\sigma = -\tau > 0$	$\sigma = -\tau < 0$	$\sigma = \tau < 0$	Avg.
	$\|\sigma\|, \|\tau\|$	$\tau = 0$	$\tau = 0$	$\tau > 0$	$\tau < 0$					
Gauss–Seidel	10	101	109	92	115	50	84	72	87	89
	50	22	23	9	25	7	22	8	23	18
	100	13	13	8	23	6	21	7	21	14
	200	9	9	15	31	13	28	14	28	19
	500	6	6	52	64	47	63	53	64	44
	1000	5	5	150*	150*	143	150*	148*	150*	117*
SOR	10	30	31	22	33	25	37	26	38	30
	50	19	20	6	20	6	21	8	22	15
	100					9	25	11	25	17
	200					16	29	17	29	23
	500					31	41	31	41	36
	1000					56	64	56	65	60
GMRES / ILU	10	17	16	17	17	12	19	18	18	17
	50	12	13	12	13	5	27	25	5	10
	100	10	10	10	11	5	30	30	5	14
	200	8	8	8	9	10	33	30	10	14
	500	7	7	8	8	22	43	41	22	20
	1000	6	6	8	8	45	49	49	48	28

TABLE 4.4
Average iteration counts for the red-black two-line ordering, for eight flow directions.

		E	W	N	S	NE	SE	NW	SW	
	max	$\sigma > 0$,	$\sigma < 0$,	$\sigma = 0$,	$\sigma = 0$,	$\sigma = \tau > 0$	$\sigma = -\tau > 0$	$\sigma = -\tau < 0$	$\sigma = \tau < 0$	Avg.
	$\|\sigma\|, \|\tau\|$	$\tau = 0$	$\tau = 0$	$\tau > 0$	$\tau < 0$					
Gauss–Seidel	10	100	110	100	109	60	78	78	82	90
	50	19	20	17	18	14	15	15	16	17
	100	10	11	15	16	13	14	13	14	13
	200	8	8	22	24	20	21	21	21	18
	500	6	6	56	58	54	56	59	57	44
	1000	5	5	150*	150*	146	150*	150*	149*	113*
SOR	10	24	26	24	25	28	29	29	29	26
	50	15	16	13	14	13	14	14	15	14
	100					17	17	17	17	17
	200					21	23	22	23	22
	500					34	35	35	35	35
	1000					58	58	58	58	58
GMRES / ILU	10	20	21	20	23	16	23	23	25	21
	50	12	13	25	31	15	23	24	25	21
	100	8	9	26	30	16	22	24	25	20
	200	6	7	26	30	17	23	23	28	20
	500	8	9	34	29	24	30	28	31	24
	1000	7	8	40	43	36	42	41	45	33

The orientation of line orderings was as in §2. That is, for the one-line orderings, lines were oriented in the NW–SE direction, and the natural ordering arranged the lines starting from the SW corner; and for the two-line orderings, line pairs were grouped by horizontal lines and the natural listing is from bottom (south) to top (north). Note that the lines associated with ordering strategies have a relationship with the direction of flow (see also [4]). For example, for the natural one-line ordering, when the flow direction is NE, the lines are perpendicular to the direction of flow, and the Gauss–Seidel and SOR sweeps follow the flow. When the flow direction is SW, the lines are perpendicular to flow, but the sweeps are in the opposite direction of the flow. On the other hand, the sweeps for the red-black orderings do not have a clear relationship to the direction of flow (although the line orientations still do). The ILU(0) preconditioning entails lower and upper triangular solves, so that, for the natural line orderings, the preconditioning operation can be thought of as a pair of bidirectional sweeps.

Tables 4.1–4.4 contain results for centered difference discretizations on a uniform mesh of width $h = 1/32$. For this class of problems, the analysis of §3 is applicable when $|\sigma h/2|$ and $|\tau h/2|$ are both less than one, i.e., when σ or τ are 10 or 50 in the problems considered. In these cases, Corollary 1 is used to choose the SOR parameter ω, where $\rho(D^{-1}C)$ is approximated using the bounds (3.10) and 3.12; here

$$(4.3)\quad a_i^{(x)} = \alpha^{(x)} = a_j^{(y)} = \alpha^{(y)} = 2, \qquad \xi = 1 - (\sigma h/2)^2, \qquad \eta = 1 - (\tau h/2)^2.$$

For the one-line orderings, when both $|\sigma h/2|$ and $|\tau h/2|$ are greater than one, the Fourier analysis of [7] can be used to estimate $\rho(D^{-1}C)$, from which good values of ω are also obtained (i.e., using the formula for ω^* in Corollary 1). These values were also used for the two-line orderings when $|\sigma h/2| > 1$ and $|\tau h/2| > 1$, although there is no theoretical justification for this. We did not examine SOR when one of $|\sigma h/2|$, $|\tau h/2|$ is greater than one and the other is less than one. Table 4.5 shows the choices of ω used for Tables 4.1–4.4. Note that the analysis of §3 and [7], [8], does not distinguish between natural and red-black orderings, or between problems where the magnitudes of σ (or τ) are the same but the signs differ.

TABLE 4.5
Values of SOR *parameters used for Tables* 4.1–4.4.

	One-line orderings		Two-line orderings		
max $\lvert\sigma\rvert, \lvert\tau\rvert$	E/W/N/S	NE/SE/NW/SW	E/W	N/S	NE/SE/NW/SW
10	1.63	1.52	1.52	1.52	1.44
50	1.07	1.02	1.06	1.04	1.01
100		1.05			1.05
200		1.27			1.27
500		1.60			1.60
1000		1.77			1.77

We make the following observations on the data of Tables 4.1–4.4:

1. For the stationary methods (Gauss–Seidel and SOR), performance depends on the relationship between flow direction and sweep direction, but the effects vary depending on the magnitudes of the velocity vectors. For example, for the natural one-line orderings, when the convection terms are small or moderate in size, the best performance of the Gauss-Seidel and SOR methods occurs when the sweeps follow the flow (i.e., when the flow direction is NE). When the convection terms dominate, the stationary methods perform better when the flow direction forms a nonzero acute angle with the sweep direction (flow is N or E), than when the sweeps follow the flow. For the natural two-line ordering, performance for moderate sized convection terms is best when the flow direction forms an acute angle with the sweep direction (i.e., when flow is N, NE, or NW); for convection-dominated systems, performance is best when the sweep is perpendicular to the flow. It is always the case that sweeping in the opposite direction of the flow is a bad choice.

2. Performance of stationary methods for the red-black orderings is much less sensitive to flow directions. In particular, the average iteration counts (over the eight flow directions) are essentially the same for the natural and red-black orderings.

This is significant on parallel architectures, where the red-black orderings can be implemented more efficiently [8]. The minimum iteration counts are typically lower for the natural orderings than for the red-black orderings.

3. Somewhat different conclusions apply for GMRES/ILU. There is no clear correlation between direction of flow and performance, except that for convection-dominated problems, performance for both natural orderings degrades when the directions of flow are not parallel to one of the grid coordinates. We have no simple explanation for this. The average iteration counts for GMRES/ILU are typically higher for the red-black orderings than for the natural orderings. Similar results have been obtained for symmetric problems, with point red-black and natural orderings, e.g., in [1].

4. One step of the block SOR method is approximately as expensive as one matrix-vector product and one scalar-vector product [8]. Thus, its cost per step is approximately $10N_b$ multiply-adds, where N_b is the order of $A^{(b)}$. One step of GMRES(5) with ILU(0) preconditioning entails a preconditioning solve, a matrix-vector product, and approximately $8N_b$ vector operations [23], for a total cost of $26N_b$ multiply-adds. That is, one GMRES/ILU step is about 2.5 times as expensive as one SOR step. Consequently, the performances of the stationary methods and GMRES/ILU are comparable for problems with small and moderate-sized convection terms (where for problems with small convection terms, it is necessary to use a good SOR parameter to achieve good performance). GMRES/ILU is somewhat more effective for convection-dominated systems, especially when there is no simple way of choosing a relaxation parameter. GMRES(5) requires $7N_b$ storage locations [23], plus approximately $9N_b$ for the factors of M. SOR requires essentially one vector of storage for the solution iterates $\{u_b^{(k)}\}$, plus storage for the factors of the block diagonal D. If no pivoting is required, these factors could overwrite the analogous locations of $A^{(b)}$.

TABLE 4.6
Average iteration counts for the block Gauss–Seidel method, upwind differences.

		E	W	N	S	NE	SE	NW	SW	
	max	$\sigma > 0,$	$\sigma < 0,$	$\sigma = 0,$	$\sigma = 0,$	$\sigma = \tau > 0$	$\sigma = -\tau > 0$	$\sigma = -\tau < 0$	$\sigma = \tau < 0$	Avg.
	$\vert\sigma\vert, \vert\tau\vert$	$\tau = 0$	$\tau = 0$	$\tau > 0$	$\tau < 0$					
Natural One-line	10	134	150*	135	150*	77	116	116	133	126*
	50	30	48	30	48	16	34	34	49	36
	100	16	33	16	33	9	24	24	40	24
	200	9	26	9	26	5	19	19	35	19
	500	5	22	5	22	3	17	17	33	15
	1000	4	20	4	20	2	16	16	32	14
Red-black One-line	10	143	150*	144	150*	93	118	118	124	130*
	50	37	39	37	39	31	33	33	36	36
	100	23	24	23	24	23	23	23	26	24
	200	16	17	16	17	20	18	18	21	18
	500	13	13	13	13	17	15	15	19	15
	1000	11	12	11	12	16	14	14	18	14
Natural Two-line	10	104	113	105	129	54	95	84	99	98
	50	27	28	24	41	17	34	20	35	28
	100	16	17	13	29	11	27	13	27	19
	200	11	11	8	23	7	23	9	23	14
	500	7	7	5	20	5	20	6	21	11
	1000	5	6	3	19	4	19	5	20	10
Red-black Two-line	10	103	113	113	124	65	90	90	94	99
	50	24	26	32	34	24	27	27	28	28
	100	14	15	21	22	18	20	13	20	18
	200	9	10	5	16	14	6	6	16	10
	500	6	6	12	13	12	13	13	14	11
	1000	5	5	11	12	11	12	12	13	10

Table 4.6 shows the performance of the block Gauss–Seidel method for solving the same set of problems using the upwind difference scheme for the first derivative terms. The main difference from the results for centered differences is that performance improves as σ or τ increases. This is because $A^{(b)}$ (as well as A) becomes more diagonally dominant in these cases. In addition, for the natural one-line ordering, performance is consistently best when the flow is in the same direction as the sweep (NE), and good performance is achieved when the sweep and flow directions make an acute angle. Similar observations apply for the natural two-line ordering, except that sweeping in the direction of flow (N) is not best when the convection terms are small. As above, the red-black orderings tend to be less sensitive than the natural orderings to flow directions.

The results above do not address the issue of accuracy of the discrete solution. If $|\sigma h/2|$ or $|\tau h/2|$ is greater than one and boundary layers are present in the continuous solution, then the discrete solution tends to be inaccurate near the boundary layers, and it is oscillatory when centered differences are used [22]. If the boundary layer can be located, then one possible remedy is to use local mesh refinement. For the solution (4.2), for nonzero σ or τ, there are boundary layers of width $O(1/\sigma)$ (or $O(1/\tau)$) near the outflow boundary. We consider one local refinement strategy, which we describe in terms of the "horizontal" parameters x and σ. In the interval of width $2/\sqrt{\sigma}$ containing the boundary layer (at either $x = 0$ or $x = 1$), we use a mesh of size $\tilde{h}$ such that $|\sigma\tilde{h}/2| = .75$; away from that interval, we use $h = 1/32$.[2] It was shown in [6] that this strategy does a good job of resolving the boundary layer with the addition

TABLE 4.7
Average iteration counts for the natural one-line ordering, centered differences and local mesh refinement.

	max	E	W	N	S	NE	SE	NW	SW	
		$\sigma > 0$,	$\sigma < 0$,	$\sigma = 0$,	$\sigma = 0$,	$\sigma = \tau > 0$	$\sigma = -\tau > 0$	$\sigma = -\tau < 0$	$\sigma = \tau < 0$	Avg.
	$\|\sigma\|, \|\tau\|$	$\tau = 0$	$\tau = 0$	$\tau > 0$	$\tau < 0$					
	100	7	31	7	31	8	17	17	47	21
Gauss– Seidel	200	12	37	12	37	32	28	28	80	33
	500	46	73	46	73	124	111	109	150*	91*
	1000	134	150*	132	150*					
	100	12	12	6	6	6	17	17	6	10
GMRES	200	10	10	4	4	8	18	17	8	10
/ ILU	500	10	10	4	3	12	18	21	11	11
	1000	9	9	4	2	18	23	24	14	13

[2] Grid points are distributed from left to right within each of these subintervals, so that the rightmost mesh width of either interval may differ from h and $\tilde{h}$.

of a relatively small number of additional mesh points. For example, in the present set of experiments, when $\sigma = 100$ there are 25 coarse grid points and 14 fine grid points in the horizontal direction; when $\sigma = 1000$, there are 29 coarse and 43 fine grid points. (The unrefined mesh contains 31 points in each direction.) Table 4.7 shows the performance of the Gauss–Seidel and GMRES/ILU methods for four problems where mesh refinement is used, for the natural one-line ordering. Comparison with Table 4.1 shows that the behavior of the two iterative methods is essentially the same as that for uniform meshes. Similar conclusions apply for the three other ordering strategies. Thus, we conclude that the behavior on uniform meshes is indicative of behavior where mesh refinement is used to resolve boundary layers. (Experiments with the Gauss–Seidel method for $|\sigma| = |\tau| = 1000$ were not performed because of storage constraints in our implementation.)

5. Experimental results: Separable variable coefficient problems. In this section, we examine the use of Corollary 2 to derive bounds on $\rho(D^{-1}C)$ when $A^{(b)}$ comes from a separable operator. We consider three model problems taken from [3]. Other experiments with these problems are described in [7].

PROBLEM 5.1.

$$\begin{aligned} -\Delta u + \frac{\sigma}{2}\left(1+x^2\right) u_x + \tau u_y &= 0 \quad \text{on } \Omega = (0,1)\times(0,1) \\ u &= 0 \quad \text{on } \partial\Omega. \end{aligned}$$

Discretization by centered differences gives, after scaling by h^2,

$$\begin{aligned} a_i^{(x)} &= \alpha^{(x)} = a_j^{(y)} = \alpha^{(y)} = 2, \\ c_{i+1}d_i &= \left(1 + \frac{\sigma h}{4}(1+x_{i+1}^2)\right)\left(1 - \frac{\sigma h}{4}(1+x_i^2)\right) \\ &\le 1 - \frac{1}{4}\left(\frac{\sigma h}{2}\right)^2 + \sigma h^2 = \xi, \\ b_{j+1}e_j &= 1 - \left(\frac{\tau h}{2}\right)^2 = \eta. \end{aligned} \tag{5.1}$$

For $\sigma \ge 0$ and $\tau \ge 0$, upwind discretization gives

$$\begin{aligned} a_i &= 2 + \frac{\sigma h}{2}\left(1 + x_i^2\right) \ge 2 + \frac{\sigma h}{2} = \alpha^{(x)}, \\ a_j &= 2 + \tau h = \alpha^{(y)}, \\ c_{i+1}d_i &= 1 + \frac{\sigma h}{2}\left(1 + x_{i+1}^2\right) \le 1 + \sigma h = \xi, \\ b_{j+1}e_j &= 1 + \tau h = \eta. \end{aligned}$$

TABLE 5.1

Comparison of computed spectral radii and bounds for the block Gauss–Seidel iteration matrices, for Problem 5.1 *with* $h = 1/32$.

	Centered differences				Upwind differences			
	One-line		Two-line		One-line		Two-line	
$\sigma = \tau$	Computed	Bound	Computed	Bound	Computed	Bound	Computed	Bound
20	.741	.809	.674	.731	.817	1.298	.772	1.379
40	.323	.385	.236	.275	.611	1.182	.544	1.212
60	.047	.062	.015	.018	.455	.961	.386	.985

Table 5.1 compares the bounds for $\rho(\mathcal{L}_1) = \rho(D^{-1}C)^2$ obtained from Corollary 2 with the corresponding computed values of $\rho(\mathcal{L}_1)$, for $h = 1/32$. For this problem, as well as the others considered below, we examine several choices of σ and τ where for the largest such choice, $\max_{x_i} |r(x_i)h/2|$ and $\max_{y_j} |s(y_j)h/2|$ are both close to one.

PROBLEM 5.2.

$$\begin{aligned} -\Delta u + \sigma x^2 u_x &= 0 \quad \text{on } \Omega = (0,1) \times (0,1), \\ u &= 0 \quad \text{on } \partial\Omega. \end{aligned}$$

Centered difference discretization gives

$$\begin{aligned} a_i^{(x)} &= \alpha^{(x)} = a_j^{(y)} = \alpha^{(y)} = 2, \\ c_{i+1}d_i &= \left(1 + \frac{\sigma h}{2} x_{i+1}^2\right)\left(1 - \frac{\sigma h}{2} x_i^2\right) \le 1 + \frac{\sigma h^2}{2} - \frac{\sigma^2 h^4}{2} = \xi, \\ b_{j+1}e_j &= 1 = \eta. \end{aligned}$$

Upwind difference discretization gives

$$\begin{aligned} a_i &= 2 + \sigma x_i^2 h \ge 2 + \sigma h^3 = \alpha^{(x)}, \\ a_j &= 2 = \alpha^{(y)}, \\ c_{i+1}d_i &= 1 + \sigma x_{i+1}^2 h \le 1 + \sigma h = \xi, \\ b_{j+1}e_j &= 1 = \eta. \end{aligned}$$

Table 5.2 compares bounds for $\rho(\mathcal{L}_1)$ with corresponding computed values for Problem 5.2. An entry "–" means that the analysis is not applicable because (3.11) is not satisfied.

TABLE 5.2

Comparison of computed spectral radii and bounds for the block Gauss–Seidel iteration matrices, for Problem 5.2 *with* $h = 1/32$.

	Centered differences				Upwind differences			
	One-line		Two-line		One-line		Two-line	
$\sigma = \tau$	Computed	Bound	Computed	Bound	Computed	Bound	Computed	Bound
20	.963	1.014	.951	.987	.964	3.077	.951	6.630
40	.953	1.033	.939	1.011	.955	10.37	.939	–
60	.945	1.051	.928	1.035	.947	56.22	.928	–

PROBLEM 5.3.

$$\begin{aligned} -\Delta u + \sigma(1 - 2x)u_x + \tau(1 - 2y)u_y &= 0 \quad \text{on } \Omega = (0,1) \times (0,1), \\ u &= 0 \quad \text{on } \partial\Omega. \end{aligned}$$

Centered difference discretization gives

$$\begin{aligned}
a_i^{(x)} &= \alpha^{(x)} = a_j^{(y)} = \alpha^{(y)} = 2, \\
c_{i+1}d_i &= \left(1 + \frac{\sigma h}{2}(1 - 2x_{i+1})\right)\left(1 - \frac{\sigma h}{2}(1 - 2x_i)\right) \\
&= 1 - 2h\left(\frac{\sigma h}{2}\right) - \left(\frac{\sigma h}{2}\right)^2 (1 - 2x_i)(1 - 2x_{i+1}) \\
&\leq 1 - \sigma h^2 + \left(\frac{\sigma h}{2}\right)^2 (2h^3 - h^4) = \xi, \\
b_{j+1}e_j &= \left(1 + \frac{\tau h}{2}(1 - 2y_{j+1})\right)\left(1 - \frac{\tau h}{2}(1 - 2y_j)\right) \\
&\leq 1 - \tau h^2 + \left(\frac{\tau h}{2}\right)^2 (2h^3 - h^4) = \eta.
\end{aligned}$$

For $\sigma \geq 0$ and $\tau \geq 0$, upwind discretization gives

$$\begin{aligned}
a_i &= 2 + \sigma|1 - 2x_i|h \geq 2 = \alpha^{(x)}, \\
a_j &= 2 + \tau|1 - 2y_j|h \geq 2 = \alpha^{(y)}, \\
c_{i+1}d_i &= 1 + \sigma|1 - 2x_{i+1}|h \leq 1 + \sigma h = \xi, \\
b_{j+1}e_j &= 1 + \tau|1 - 2y_{j+1}|h \leq 1 + \tau h = \eta.
\end{aligned}$$

Table 5.3 compares bounds and computed values of $\rho(\mathcal{L}_1)$ for Problem 5.3; the entry "–" indicates that either (3.9) or (3.11) is not satisfied.

TABLE 5.3
Comparison of computed spectral radii and bounds for the block Gauss–Seidel iteration matrices, for Problem 5.3 with $h = 1/32$.

	Centered differences				Upwind differences			
	One-line		Two-line		One-line		Two-line	
$\sigma = \tau$	Computed	Bound	Computed	Bound	Computed	Bound	Computed	Bound
20	.854	.921	.813	.869	.871	3.611	.833	6.986
40	.733	.852	.669	.785	.780	–	.723	–
60	.629	.788	.553	.710	.703	–	.634	–

To understand these results, it is useful to recall the constant coefficient problem (4.1). For that problem, the parameters associated with centered differences are given by (4.3). As shown in [7], [8], if both $\sigma h/2 < 1$ and $\tau h/2 < 1$, then the bounds from Corollary 2 essentially have the form $1 - O(\sigma^2 h^2) - O(\tau^2 h^2)$. In particular, if either $\sigma h/2$ or $\tau h/2$ are near 1, then ξ or η are close to 0, and the bounds from Corollary 2 are very small. For Problem 5.1, $r(x)$ (the coefficient of u_x) is bounded below away from 0, so that for large σ, the contribution $hr(x_i)/2$ cannot be small for any x_i. Consequently, the bounding value ξ is qualitatively like its constant coefficient counterpart (compare (5.1) and (4.3)). Moreover, $\alpha^{(x)}$, $\alpha^{(y)}$ and η have the same values as in the constant coefficient case. (This is true for $\alpha^{(x)}$ and $\alpha^{(y)}$ with all three problems considered here.) Thus, the bounds from Corollary 2 behave like their constant coefficient analogues. For Problem 5.2, the upper bound ξ corresponds to a value for $x_i(= h)$ for which the differential operator is locally nearly selfadjoint; the resulting bounds typically do not even guarantee convergence, and they are larger than what would be obtained in the selfadjoint case. For Problem 5.3, $\xi = 1 - O(\sigma h^2)$ and $\eta = 1 - O(\sigma h^2)$, which lead to asymptotic bounds of the form $1 - O(\sigma h^2) - O(\tau h^2)$; these are larger than those occurring for Problem 5.1 but smaller than for Problem 5.2. Note that for all three problems, the bounding values are qualitatively similar to the behavior of $\mathcal{L}_1$.

The parameters for upwind differences applied to the constant coefficient problem are

$$a_i^{(x)} = \alpha^{(x)} = 2 + \sigma h, \quad a_j^{(y)} = \alpha^{(y)} = 2 + \tau h, \quad \xi = 1 + \sigma h, \quad \eta = 1 + \tau h.$$

In this case, the bounds on $\rho(D^{-1}C)$ from Corollary 2 are less than one, and they decrease with increasing σ or τ (see [7], [8]). However, the extra inequalities required to define $\alpha^{(x)}$ and $\alpha^{(y)}$ decrease the size of the denominators in (3.10) and (3.12) and limit the usefulness of the corollary. For Problem 5.1, σh is replaced by $\sigma h/2$ in $\alpha^{(x)}$, and the bounds on $\rho(D^{-1}C)$ are less than one only when σh is large. The bounds for Problems 5.2 and 5.3, where they are defined, do not provide any useful information.

6. Experimental results: Nonseparable variable coefficient problems. We now examine the performance of the iterative methods for solving some nonseparable problems. Our goals are to examine the effectiveness of the block Gauss–Seidel and SOR methods, and ILU-preconditioned GMRES, for solving such problems; and to determine whether the analytic results of [7], [8], and §3 are of use in predicting behavior.

The following problem, from [19], models the circular flow of a cold fluid with a hot wall at the right boundary.

PROBLEM 6.1.

$$\begin{aligned} -\epsilon\Delta u + 2y(1-x^2)u_x - 2x(1-y^2)u_y &= 0 && \text{on } \Omega = (-1,1)\times(0,1),\\ u &= 0 && \text{on } 0 \le y \le 1, x = -1,\\ u &= 100 && \text{on } 0 \le y \le 1, x = 1,\\ u &= 0 && \text{on } -1 \le x < 0, y = 0,\\ u_n &= 0 && \text{on } 0 \le x \le 1, y = 0,\\ u &= 0 && \text{on } -1 \le x \le 1, y = 1. \end{aligned}$$

The velocity vectors have turning points in the vertical component, and their magnitudes vary throughout the domain of definition. The solution contains a boundary layer at $x = 1$. Figure 6.1 shows the boundary conditions and streamlines, and the general shape of the solution, for $\epsilon = 1/100$.[3] A related problem, differing from Problem 6.1 only in the boundary conditions, was also considered in [9]; experimental results were qualitatively similar to those presented below.

As above, we consider both centered difference and upwind difference discretizations. At the outflow boundary $x \ge 0$, $y = 0$, we used first order upwind differences

$$0 = u_n(x_i, 0) = u_y(x_i, y_0) \approx \frac{u(x_i, y_1) - u(x_i, y_0)}{h},$$

i.e., $u(x_i, 0) = u(x_i, y_1)$. For the centered difference scheme, we consider both a square 31×31 mesh, and a uniform mesh of width $h = 1/32$. The first choice

[3] The discrete solution was computed using centered differences with 31 interior grid points in each direction; the figure includes the exact solution values at $x = \pm 1$ and $y = 1$, but not at $y = 0$.

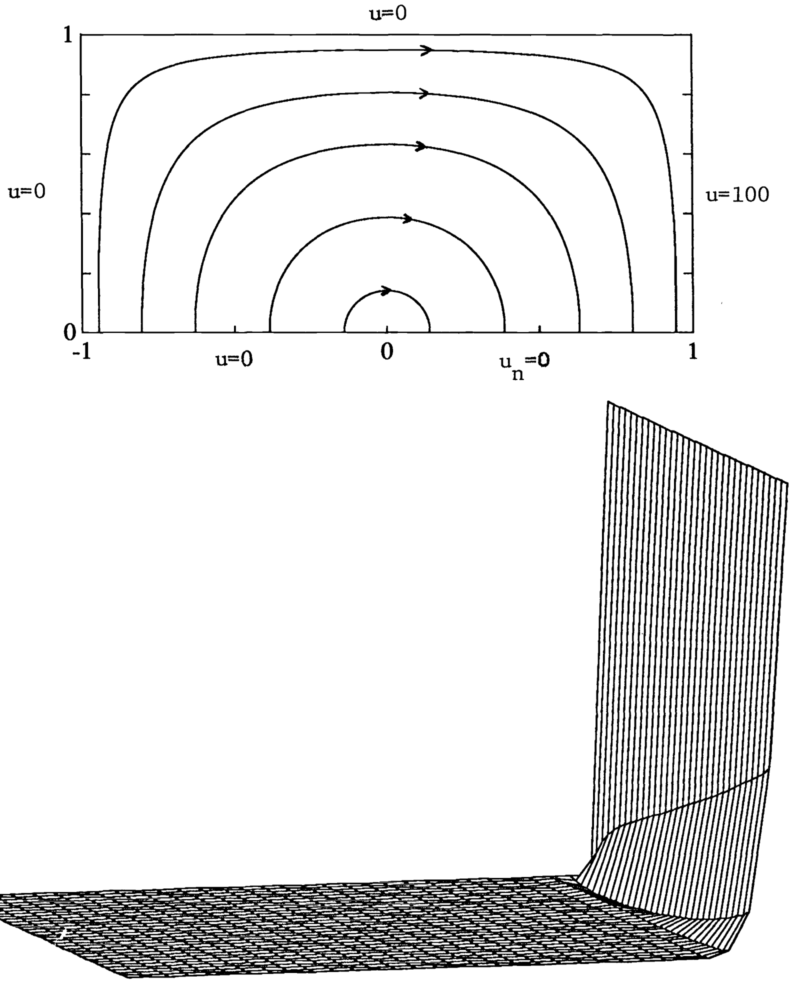


FIG. 6.1. *Boundary conditions and solution for Problem* 6.1.

produces matrices with the same algebraic structure as those considered in §§4–5, but the horizontal mesh width is twice that of the vertical width; the second choice leads to lines of different length in the grid. We also consider a strategy for improving the accuracy of the solution, based on defect correction methods. For all tests, the initial guesses and stopping criteria are as in §4.

Table 6.1 shows average iteration counts for solving the reduced system derived when centered differences are applied on a square 31 × 31 grid. Here, the grid sizes for the full system are uniform in each of the x and y coordinates, with $h_x = 1/16$ and $h_y = 1/32$. As in the constant coefficient case (§4), block relaxation is most effective for intermediate values of ϵ^{-1}, where it is competitive with GMRES/ILU. The latter method is more effective when ϵ^{-1} is either small or large. The performance of the stationary methods is fairly insensitive to the choice of ordering. This is consistent with the fact that, because of variable directions of flow, there is no clear correspondence between lines and flow direction. On the other hand, as in §4, the performance of GMRES/ILU is typically better with the natural orderings than with the red-black orderings. We remark that in a few experiments with Orthomin [5], we

TABLE 6.1

Average iteration counts for Problem 6.1 on a 31×31 *grid* $(h_x = 1/16, h_y = 1/32)$*, with centered differences. Numbers in parentheses are approximate number of digits of accuracy when methods did not meet the stopping criterion.*

		$1/\epsilon$					
	Ordering	10	50	100	200	500	1000
Gauss–Seidel	Natural one-line	122	22	27	57	150 (4)	150 (1)
	Red-black one-line	119	26	29	63	150 (4)	150 (1)
	Natural two-line	114	24	26	54	150 (4)	150 (1)
	Red-black two-line	111	25	26	54	150 (4)	150 (1)
GMRES / ILU	Natural one-line	10	7	7	8	15	33
	Red-black one-line	27	27	34	37	46	74
	Natural two-line	14	10	10	10	19	87
	Red-black two-line	24	21	26	26	42	71

found Orthomin(5) to be somewhat less robust than GMRES(5).

TABLE 6.2

Average iteration counts for Problem 6.1, with the natural one-line and two-line orderings, on a uniform grid with mesh size $h = 1/32$ *and centered differences. Numbers in parentheses are approximate number of digits of accuracy when methods did not meet the stopping criterion.*

	$1/\epsilon$					
Method	10	50	100	200	500	1000
G.S. Natural one-line	150 (5)	28	22	35	122	150 (3)
G.S. Natural two-line	129	27	22	34	101	150 (3)
GMRES/ILU Natural one-line	17	11	10	10	16	150 (3)
GMRES/ILU Natural two-line	20	14	12	12	17	64

Table 6.2 shows iteration counts for solving the reduced system derived from an underlying uniform mesh of width $h = 1/32$, for block Gauss–Seidel and GMRES/ILU, with the two natural line orderings. The lines are oriented as in Fig. 2.2. These results are similar to those of Table 6.1, except that GMRES/ILU has trouble with one problem class ($\epsilon = 1/1000$ with the natural one-line ordering). In this case, the iteration "stagnates," in the sense that the residual norm $\|g^{(b)} - A^{(b)}u_i^{(b)}\|_2$ remains constant over many iterations.[4] In contrast, whenever the block relaxation methods fail to meet the stopping criterion, they appear to be converging.

TABLE 6.3

Average iteration counts for Problem 6.1 on a 31×31 *grid* $(h_x = 1/16, h_y = 1/32)$*, with upwind differences.*

		$1/\epsilon$					
	Ordering	10	50	100	200	500	1000
Gauss–Seidel	Natural one-line	142	31	24	21	18	17
	Red-black one-line	139	37	29	26	24	23
	Natural two-line	132	32	25	23	20	19
	Red-black two-line	131	35	27	22	20	19
GMRES / ILU	Natural one-line	10	8	8	7	7	6
	Red-black one-line	29	25	28	32	36	37
	Natural two-line	15	10	10	9	8	7
	Red-black two-line	28	20	20	21	26	25

Table 6.3 shows average iteration counts for solving the reduced system derived when upwind differences are applied to Problem 6.1. Note that the mesh points used

[4] Stagnation of this type also occurs for GMRES(10) and GMRES(15).

for discretization depend on the direction of flow (see §2), and the reduced matrices $A^{(b)}$ are always diagonally dominant. The results of Table 6.3 (for the stationary methods) are consistent with those for constant coefficient problems.

A methodology for improving accuracy that does not require a priori knowledge about the solution is the class of defect correction methods. A description of this approach can be found in [12], which contains several other references. For the operator $L_\epsilon u \equiv -\epsilon\Delta u + ru_x + su_y$, let $A_{\epsilon,h}$ denote the matrix associated with the (second order) centered difference discretization on a uniform mesh of width h. For $\hat{\epsilon} > \epsilon$, let $A_{\hat{\epsilon},h}$ denote the analogous matrix derived from $L_{\hat{\epsilon}}$. In its simplest form, the defect correction iteration consists of the following steps, where f is the discrete right-hand side.

```
Solve $A_{\hat{\epsilon},h}u^{(m)} = f$.
For $m = 0, 1, \cdots$, Do
    $r^{(m)} = f - A_{\epsilon,h}u^{(m)}$
    Solve $A_{\hat{\epsilon},h}d^{(m)} = r^{(m)}$
    $u^{(m+1)} = u^{(m)} + d^{(m)}$
End
```

The idea is to compensate for instabilities associated with high order operators using lower order operators. For the choice $\hat{\epsilon} = \epsilon + ch$ where $c > 0$ is a fixed constant, $A_{\hat{\epsilon},h}$ is a first order discretization. At every step of the iteration, $A_{\epsilon,h}$ is used only to calculate the residual, and a linear system with coefficient matrix $A_{\hat{\epsilon},h}$ must be solved. Thus, the cost of this method is highly dependent on the cost of solving the linear system.

Any $c > 0$ prevents the convection terms from dominating the discrete problem, for arbitrarily small ϵ. For $c \geq \max\{|r(x,y)|/2, |s(x,y)|/2\}$, $A_{\hat{\epsilon},h}$ and the resulting reduced matrix $A^{(b)}_{\hat{\epsilon},h}$ are diagonally dominant M-matrices. For Problem 6.1, this gives $c = 1$. However, Hemker [15] has observed that (using a variant of the algorithm above) better accuracy is obtained with smaller c. Following [15], we use $c = \frac{1}{2}$. The differential operator $L_{\hat{\epsilon}}$ for Problem 6.1 is then equivalent to

$$-\Delta u + \frac{2y(1-x^2)}{\epsilon + h/2}u_x - \frac{2x(1-y^2)}{\epsilon + h/2}u_y.$$

We refer to the discretization of this operator by centered difference as the "defect correction discretization." Table 6.4 shows the performance of the various iterative methods for solving the resulting reduced linear systems. (See [15] for a discussion of the overall iteration.) These results are qualitatively similar to performance for upwind differences.

7. Concluding remarks. In this paper, we have continued the study of line iterative methods for solving reduced systems begun in [7], [8]. We have extended the analysis in two ways. First, for matrices that arise from variable coefficient separable differential operators, we derived conditions under which the reduced matrices can be symmetrized via diagonal similarity transformations; previous results applied only to constant coefficient problems. Symmetrization is the key to the analysis of convergence behavior for the constant coefficient case. In the present analysis, it determines conditions under which the classical analysis of SOR applies, from which the optimal SOR parameter can be expressed as a simple function of the maximum eigenvalue of the line Jacobi iteration matrix, and it leads to some analytic bounds on performance for separable problems. In addition, we used regular splitting results to show that the

TABLE 6.4

Average iteration counts to solve the linear systems arising from the defect correction method, for the natural one-line and two-line orderings on a uniform grid with mesh size $h= 1/32$. *Numbers in parentheses are approximate number of digits of accuracy when methods did not meet the stopping criterion.*

	$1/\epsilon$					
Method	10	50	100	200	500	1000
G.S. Natural one-line	150 (4)	42	32	28	26	25
G.S. Natural two-line	150 (5)	38	30	27	25	25
GMRES/ILU Natural one-line	17	12	12	11	11	11
GMRES/ILU Natural two-line	21	16	14	14	13	13

analysis of line Jacobi splittings can be extended to splittings based on incomplete LU factorizations, for various line orderings of the reduced grid. The results help explain the good performance of IC preconditioners applied to the nonsymmetric matrix problems arising from the convection-diffusion equation.

We have also performed an extensive set of numerical experiments that examine the effects of direction of flow, discretization, and grid ordering on performance of the line iterative methods. For constant coefficient problems, the results reveal correlations between relaxation sweep direction and direction of flow that are not displayed by any analytic results. They also show that for block relaxation methods, red-black orderings are less sensitive to flow directions than natural orderings, whereas for IC-preconditioned GMRES, convergence is faster for natural orderings than for red-black orderings. In addition, both block relaxation and IC preconditioned GMRES are effective for many problems where the analysis does not apply. In general, IC-preconditioned GMRES is more robust than block relaxation. Finally, experimental results for problems with variable coefficients or locally refined grids are largely consistent with analysis and experiments for constant coefficients and uniform grids.

REFERENCES

[1] C. C. ASHCRAFT AND R. G. GRIMES, *On vectorizing incomplete factorization and SSOR preconditioners*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 122–151.

[2] R. BEAUWENS, *Factorization iterative methods, M-operators and H-operators*, Numer. Math., 31 (1979), pp. 335–357, 1979.

[3] E. F. 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] R. C. Y. CHIN AND T. A. MANTEUFFEL, *An analysis of block successive overrelaxation for a class of matrices with complex spectra*, SIAM J. Numer. Anal., 25 (1988), pp. 564–585.

[5] S. C. EISENSTAT, H. C. ELMAN, AND M. H. SCHULTZ, *Variational iterative methods for nonsymmetric systems of linear equations*, SIAM J. Numer. Anal., 20 (1983), pp. 345–357.

[6] H. C. ELMAN, *Relaxed and stabilized incomplete factorizations for non-self-adjoint linear systems*, BIT, 29 (1989), pp. 890–915.

[7] H. C. ELMAN AND G. H. GOLUB, *Iterative methods for cyclically reduced non-self-adjoint linear systems*, Math. Comp., 54 (1990), pp. 671–700.

[8] ———, *Iterative methods for cyclically reduced non-self-adjoint linear systems* II, Report UMIACS-TR-89-45, Department of Computer Science, University of Maryland, College Park, MD, 1989; Math. Comp., 56 (1991), pp. 215–242.

[9] ———, *Line iterative methods for cyclically reduced discrete convection-diffusion problems*, Report UMIACS-TR-90-16, Department of Computer Science, University of Maryland, College Park, MD, 1990.

[10] K. FAN, *Note on M-matrices*, Quart. J. Math. Oxford Ser. (2), 11 (1960), pp. 43–49.

[11] G. E. FORSYTHE AND W. R. WASOW, *Finite Difference Methods for Partial Differential Equations*, John Wiley and Sons, New York, 1960.

[12] W. HACKBUSCH, *Multi-Grid Methods and Applications*, Springer-Verlag, Berlin, 1985.

[13] L. A. Hageman and R. S. Varga, *Block iterative methods for cyclically reduced matrix equations*, Numer. Math., 6 (1964), pp. 106–119.

[14] L. A. Hageman and D. M. Young, *Applied Iterative Methods*, Academic Press, New York, 1981.

[15] P. W. Hemker, *Mixed defect correction iteration for the accurate solution of the convection diffusion equation*, in Multi-grid Methods, W. Hackbusch and U. Trottenberg, eds., Lecture Notes in Mathematics 960, Springer-Verlag, Berlin, 1982.

[16] L. Lamport, *The parallel execution of DO loops*, Comm. ACM., 17 (1974), pp. 83–93.

[17] 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.

[18] J. P. Milaszewicz, *Improving Jacobi and Gauss–Seidel iterations*, Lin. Alg. Appl., 93 (1987), pp. 161–170.

[19] K. W. Morton, *Generalised Galerkin methods for steady and unsteady problems*, in Numerical Methods for Fluid Dynamics, K. W. Morton and M. J. Baines, eds., Academic Press, Orlando, FL, 1982.

[20] S. V. Parter and M. Steuerwalt, *Block iterative methods for elliptic and parabolic difference equations*, SIAM J. Numer. Anal., 19 (1982), pp. 1173–1195.

[21] *PCGPAK User's Guide, Version* 1.04, Scientific Computing Associates, New Haven, CT, 1987.

[22] P. J. Roache, *Computational Fluid Dynamics*, Hermosa Publishers, Albuquerque, 1982.

[23] Y. Saad and M. H. Schultz, *GMRES: A generalized minimual residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.

[24] J. Sheldon, *On the spectral norms of several iterative processes*, J. Assoc. Comput. Mach., 6 (1959), pp. 494–505.

[25] R. S. Varga, *Matrix Iterative Analysis*, Prentice–Hall, Englewood Cliffs, NJ, 1962.

[26] H. F. Weinberger, *A First Course in Partial Differential Equations with Complex Variables and Transform Methods*, Blaisdell, New York, 1965.

[27] Z. Woźnicki, *Two-sweep iterative methods for solving large linear systems and their application to the numerical solution of multi-group multi-dimensional neutron diffusion equation*, Ph.D. thesis, Report N^{0}1447/CYFRONET/PM/A, Institute of Nuclear Research, Swierk, Poland, 1973.

[28] D. M. Young, *Iterative Solution of Large Linear Systems*, Academic Press, New York, 1971.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 1, pp. 364–378, January 1992

AN OPTIMAL DOMAIN DECOMPOSITION PRECONDITIONER FOR THE FINITE ELEMENT SOLUTION OF LINEAR ELASTICITY PROBLEMS*

BARRY F. SMITH†

Abstract. For linear elasticity problems the finite element method is an extremely successful method to model complicated structures. The successful implementation requires the solution of very large, sparse, positive definite linear systems of algebraic equations. A new technique for solving these systems using the preconditioned conjugate gradient method is proposed. Using ideas from both additive Schwarz methods and iterative substructuring methods, it is proven that the condition number of the resulting system does not grow as the substructures are made smaller and the mesh is refined. This result holds for two and three dimensions. Numerical experiments have been performed to demonstrate the power of this method. For linear elasticity problems in two dimensions the condition numbers are observed numerically to be less than four when using a regular mesh.

Key words. additive Schwarz methods, domain decomposition, elliptic equations, finite elements, iterative substructuring, linear elasticity, preconditioned conjugate gradient, Schur complement

AMS(MOS) subject classifications. 65F10, 65N30

1. Introduction.

We introduce a new domain decomposition method of solving the linear systems arising from the finite element discretization of coupled elliptic systems using a preconditioned conjugate gradient method. The method incorporates certain features of both additive Schwarz methods and iterative substructuring methods. We prove that for second-order, selfadjoint, uniformly elliptic systems the condition number of the resulting preconditioned linear system is bounded independently of the number of substructures and the mesh refinement. The method is highly parallelizable.

Very early, Sobolev [31] showed that the Schwarz alternating method converges for the equations of linear elasticity. More recent work using the Neumann–Dirichlet algorithm was done by Bjørstad and Hvidsten [4]. De Roeck [15] has implemented an iterative substructuring type algorithm for elasticity problems. Hughes and others have been using element-by-element preconditioning [21], [33], on large structural problems. The preconditioned problems for these latter two algorithms still can require hundreds of conjugate gradient iterations.

For the p-version finite elements Mandel has analyzed iterative substructuring type algorithms for elasticity [23], [24], [25], [26]. Other work using the p-version finite elements has been done by Babuška, Craig, Mandel, and Pitkäranta [1] and Babuška, Griebel, and Pitkäranta [2].

Much work using domain decomposition has focused on the scalar elliptic problem, including Bjørstad and Widlund [5], [6]; Bramble, Pasciak, and Schatz [7], [9], [10]; Chan and Resasco [11], [12]; Dryja [16], [17]; Dryja and Widlund [18], [19]; and Widlund [34]. The work by Matsokin and Nepomnyaschikh [27] discusses a Schwarz alternating algorithm that has some similarities to that presented in this paper.

* Received by the editors April 5, 1990; accepted for publication (in revised form December 2, 1990. This work was supported in part by the National Science Foundation under grant NSF-CCR-8903003 and in part by the Norwegian Research Council for Science and the Humanities under grant D.01.08.054 and The Royal Norwegian Council for Scientific and Industrial Research under grant IT2.28.28484 at the University of Bergen, Bergen, Norway.

† Argonne National Laboratory, 9700 South Cass Ave., Argonne, Illinois 60439.

2. The discrete problem. We consider a second order, symmetric, coercive, bilinear form $a_\Omega(u,v)$ on a bounded polyhedral (polygonal) domain Ω and, for simplicity, impose a homogeneous Dirichlet condition on $\partial\Omega$. The variational problem is then to find $u \in (H_0^1(\Omega))^q$, such that,

$$a_\Omega(u,v) = f(v), \qquad \forall v \in (H_0^1(\Omega))^q.$$

Our specific applications are various linear elasticity models, where q is from 2 to 6, cf. [14]. Our results also hold for the scalar case $q=1$.

We perform two levels of triangulations into substructures Ω_i and then triangulate the substructures into elements, and assume shape regularity and that the substructures and elements satisfy the usual rules of finite element triangulations; see, e.g., Ciarlet [13]. $V^H(\Omega) \subset (H_0^1(\Omega))^q$ and $V^h(\Omega) \subset (H_0^1(\Omega))^q$ are the spaces of continuous, piecewise linear functions, on the two triangulations, which vanish on the boundary $\partial\Omega$. We will be working only with piecewise linear finite elements but note that the theory may also hold for higher order elements.

The discrete variational problem is then of the form, find $u_h \in V^h(\Omega)$, such that,

$$a_\Omega(u_h,v_h) = f(v_h), \qquad \forall v_h \in V^h(\Omega). \tag{1}$$

This finite dimensional variational problem is turned into a linear system of equations by introducing a basis $\{\phi_i\}$ for the space V^h. We use the standard nodal basis functions in V^h. If we express the solution as $u_h = \sum x_i\phi_i$, we obtain the linear system

$$Kx = f.$$

Here x is the vector of unknowns x_i, f the vector of components $f(\phi_i)$, and K the stiffness matrix $K_{ij} = a_\Omega(\phi_i,\phi_j)$.

Using subscript notation explained below, for any block matrix

$$K = \begin{pmatrix} K_{II} & K_{IB} \\ K_{BI} & K_{BB} \end{pmatrix},$$

with K_{II} nonsingular, the Schur complement of K is given by

$$S = K_{BB} - K_{BI}K_{II}^{-1}K_{IB}.$$

This is the block that remains after the variables associated with K_{II} have been eliminated by Gaussian elimination. We give, without proof, the important lemma.

LEMMA 2.1. *If K is symmetric, positive definite and if x is partitioned in the same manner as $K, x = (x_I, x_B)^T$, then for any given x_B,*

$$x_B^T S x_B = \min_{x_I} x^T K x.$$

The minimizing extension x_I is referred to as the discrete harmonic extension. This is a direct analogy to the continuous case for the Laplacian. The discrete harmonic extension satisfies the relation $K_{II}x_I + K_{IB}x_B = 0$.

With each substructure Ω_k, we can associate the portion of the stiffness matrix K arising from the integration over that substructure,

$$K_{ij}^{(k)} = a_{\Omega_k}(\phi_i,\phi_j).$$

Similarly, the subvector of unknowns associated with this substructure is called $x^{(k)}$. Since we are working with nodal basis functions the support of ϕ_i associated with any node in the interior of a substructure lies completely in that substructure. Therefore there are no couplings between unknowns associated with the interiors of two different substructures. For each substructure, we partition the variables into those associated with a node in the interior of the substructure, $x_I^{(k)}$, and those on the boundary of the substructure, $x_B^{(k)}$. To simplify notation, we pad the subvectors $x^{(k)}$ and submatrices $K^{(k)}$ by zeros so that the following equation makes sense. This is called the subassembly process; cf. [19].

$$\begin{pmatrix} K_{II} & K_{IB} \\ K_{IB}^T & K_{BB} \end{pmatrix} \begin{pmatrix} x_I \\ x_B \end{pmatrix} = \sum_k \begin{pmatrix} K_{II}^{(k)} & K_{IB}^{(k)} \\ K_{IB}^{(k)^T} & K_{BB}^{(k)} \end{pmatrix} \begin{pmatrix} x_I^{(k)} \\ x_B^{(k)} \end{pmatrix}.$$

The Schur complement of the global stiffness matrix K is then given by

$$\begin{aligned} Sx_B &= \sum_k (K_{BB}^{(k)} - K_{IB}^{(k)^T} K_{II}^{(k)^{-1}} K_{IB}^{(k)}) x_B^{(k)} \\ &= \sum_k S^{(k)} x_B^{(k)}. \end{aligned}$$

Each of the Schur complements, $S^{(k)}$, can be formed independently and in parallel. We are now left with the still large linear system

$$Sx_B = g.$$

This system will be solved using the preconditioned conjugate gradient method. The preconditioner is constructed using the ideas developed in the theory of additive Schwarz methods.

3. Schwarz methods: An abstract framework. Suppose that we wish to solve the following finite dimensional variational problem. Find $u \in V$, such that,

$$a(u,v) = f(v), \qquad \forall v \in V. \tag{2}$$

Here $a(\cdot,\cdot)$ is a symmetric, positive definite bilinear form. For our purposes $a(\cdot,\cdot)$ will be the bilinear form induced by the matrix S. Let V_i be a set of subspaces of V so that $V = V_1 + \cdots + V_N$. With each subspace there is a corresponding projection operator P_i, which is the projection in the $a(u,v)$ inner product onto the subspace V_i, i.e.,

$$a(P_i u, w) = a(u,w) = f(w), \qquad \forall w \in V_i, \qquad P_i u \in V_i. \tag{3}$$

$P_i u$ can be determined by introducing a basis $\{\psi_j^{(i)}\}$ for V_i and expanding $P_i u$ in that basis, $P_i u = \sum_j \alpha_j^{(i)} \psi_j^{(i)}$. This results in the linear system

$$\tilde{K}^{(i)} \alpha^{(i)} = f^{(i)},$$

where $\tilde{K}_{jl}^{(i)} = a(\psi_j^{(i)}, \psi_l^{(i)})$, and $f^{(i)}$ is the vector defined by $f(\psi_j^{(i)})$.

The additive Schwarz method, see Dryja and Widlund [19], of solving (2) is to introduce an auxiliary problem

$$Pu = \sum_i P_i u = \hat{f}, \tag{4}$$

which has the same solution as (2). Since $P_i u$ can be found by (3) without knowing the solution of (2), we first compute $\hat{f}$ and then solve (4) using the conjugate gradient method.

The conjugate gradient method can be an effective method for the solution of a symmetric (in any inner product) positive definite linear system. The decrease in the energy norm of the error after m steps can be bounded by

$$2\left(\frac{\sqrt{\kappa(P)}-1}{\sqrt{\kappa(P)}+1}\right)^m,$$

where $\kappa(P)$ is the condition number of the matrix P, see, e.g., Golub and Van Loan [20].

The reason for going from problem (2) to problem (4) is that by a suitable choice of the subspaces V_i we can turn a large ill-conditioned system into a very well conditioned problem at the expense of solving many small independent linear systems. The following two lemmas allow us to develop bounds on the condition number of P.

LEMMA 3.1. *Consider the undirected graph with a node for each subspace V_i, and an edge between node i and node j if and only if $V_i \cap V_j \neq 0$. Let p be the number of colors needed to color the graph so that no two nodes connected by an edge have the same color. Then*

$$\lambda_{\max}(P) \leq p.$$

Proof. All the subspaces for a particular color are disjoint, hence their corresponding projection operators are mutually orthogonal. Therefore the sum of the projection operators of a particular color is itself a projection operator. P then consists of p of these composite projection operators each with a maximum eigenvalue of one; thus $\lambda_{\max}(P) \leq p$. □

LEMMA 3.2 ([18], [22], [28]). *Assume that for all $u \in V$, there exists a representation $u = \sum_i u_i$ with $u_i \in V_i$ such that*

$$\sum_i a(u_i, u_i) \leq C_0^2 a(u, u),$$

then

$$\lambda_{\min}(P) \geq C_0^{-2}.$$

Throughout this paper c, C, and C_0 represent generic constants independent of h and H.

Proof.

$$|u|_a^2 = \sum_i a(u, u_i) = \sum_i a(u, P_i u_i) = \sum_i a(P_i u, u_i).$$

Therefore,

$$|u|_a^2 \leq \left(\sum_i |P_i u|_a^2\right)^{1/2} \left(\sum_i |u_i|_a^2\right)^{1/2}.$$

By the assumption of the lemma and a property of projections,

$$|u|_a^2 \leq C_0^2 \sum_i |P_i u|_a^2 = C_0^2 \sum_i a(P_i u, u) = C_0^2 a(Pu, u). \qquad □$$

4. The new algorithm. The variational problem we are solving, after the unknowns in the interior of the substructures have been eliminated, is to find $\tilde{u}_h \in \tilde{V}^h$, such that,

$$a_\Omega(\tilde{v}_h, \tilde{u}_h) = f(\tilde{v}_h), \qquad \forall\, \tilde{v}_h \in \tilde{V}^h,$$

where $\tilde{V}^h$ is the subspace of V^h of discrete harmonic functions. The matrix formulation of the problem is to find x_B, such that,

$$y_B^T S x_B = y_B^T g, \qquad \forall y_B.$$

The vectors y_B and x_B are, as in §2, the coefficients of the finite element functions restricted to Γ.

For our algorithm we use the following subspaces: a global coarse space, and spaces associated with overlapping regions of Γ. We can regard the boundaries of the substructures as consisting of three pieces: the substructure vertices, the edges between substructure vertices, and the faces of the substructures. In two dimensions there are no faces, only vertices and edges. To obtain the overlapping regions for the Schwarz method we introduce the faces Γ^{F_i}. We choose Γ^{E_j} to be regions consisting of an edge and an overlap onto the adjacent faces to a distance of order H from the edge. The Γ^{V_k} are regions consisting of a vertex and an overlap onto all adjacent faces and edges to a distance of order H from the vertex. We constrain the overlap so that no portion of Γ is covered more than four times.

The subspaces of $\tilde{V}^h$ are given by,

$$V^H,$$

$$\tilde{V}^h_{F_i} = \{\phi \in \tilde{V}^h : \mathrm{supp}(\phi|_\Gamma) \subseteq \Gamma^{F_i}\},$$

$$\tilde{V}^h_{E_i} = \{\phi \in \tilde{V}^h : \mathrm{supp}(\phi|_\Gamma) \subseteq \Gamma^{E_i}\},$$

and

$$\tilde{V}^h_{V_i} = \{\phi \in \tilde{V}^h : \mathrm{supp}(\phi|_\Gamma) \subseteq \Gamma^{V_i}\}.$$

It is easy to see that each of these spaces is a subspace of $\tilde{V}^h$ and that

$$\tilde{V}^h = \left(\sum_i \tilde{V}^h_{F_i}\right) + \left(\sum_j \tilde{V}^h_{E_j}\right) + \left(\sum_k \tilde{V}^h_{V_k}\right).$$

The matrix projection onto the coefficients of each "face" subspace is given by

$$P_{F_i} = R^T_{F_i} S^{-1}_{F_i} R_{F_i} S,$$

where R_{F_i} is the restriction operator that returns only those coefficients associated with Γ^{F_i}, and S_{F_i} is the principal minor of the Schur complement S that is associated with that same set of coefficients. Note that P_{F_i} is a matrix projection acting on the coefficient vector x_B. Corresponding to each P_{F_i} is a projection in the space $\tilde{V}^h$. The projections onto "edge" and "vertex" subspaces are formed in the same manner.

Naturally the restriction operators R need never be explicitly represented, instead we would use something like hardware scatter-gather.

For the coarse space V^H, the projection operator is

$$P_H = R_H^T S_H^{-1} R_H S.$$

The operator R_H^T maps the coefficients of V^H to the coefficients of $\tilde{V}^h$ and represents linear interpolation from V^H to $\tilde{V}^h$. S_H is defined as $R_H S R_H^T$. This results in P_H being a projection matrix in the S inner product. The matrix $R_H S R_H^T$ may be regarded as a submatrix of the Schur complement after a partial change to hierarchical basis has been made, see Smith and Widlund [30]. In Smith [29] it is shown how this calculation may be made one subdomain at a time.

The preconditioned problem, corresponding to the abstract theory in §2, is given by

$$Px_B = \hat{S}^{-1} S x_B = \hat{S}^{-1} g.$$

The explicit form of the preconditioner is given by

$$\hat{S}^{-1} = R_H^T S_H^{-1} R_H + \sum_i R_{F_i}^T S_{F_i}^{-1} R_{F_i} + \sum_j R_{E_j}^T S_{E_j}^{-1} R_{E_j} + \sum_k R_{V_k}^T S_{V_k}^{-1} R_{V_k}.$$

We now present a detailed description of the algorithm. We note that many opportunities exist for parallelism between and within each step. Naturally, in each step we would take advantage of the band and sparse structures of the matrices.

Iterative substructuring/additive Schwarz algorithm

1. Form the stiffness matrices

$$K^{(j)} = \begin{pmatrix} K_{II}^{(j)} & K_{IB}^{(j)} \\ K_{IB}^{(j)^T} & K_{BB}^{(j)} \end{pmatrix}$$

for each substructure by integration.
2. Factor $K_{II}^{(j)}$ for each substructure.
3. Form the Schur complements $S^{(j)} = K_{BB}^{(j)} - K_{IB}^{(j)^T} K_{II}^{(j)^{-1}} K_{IB}^{(j)}$.
4. Form and factor the coarse stiffness matrix S_H.
5. Form, by subassembly, S_{F_i}, S_{E_i}, and S_{V_i}.
6. Factor S_{F_i}, S_{E_i}, and S_{V_i}.
7. Form the right-hand sides $b^{(j)}$ for each substructure by integration.
8. Modify the right-hand sides; $b_B^{(j)} = b_B^{(j)} - K_{IB}^{(j)^T} K_{II}^{(j)^{-1}} b_I^{(j)}$.
9. Solve $Sx_B = b_B$ using a preconditioned conjugate gradient method with the preconditioner

$$\hat{S}^{-1} = R_H^T S_H^{-1} R_H + \sum_i R_{F_i}^T S_{F_i}^{-1} R_{F_i} + \sum_j R_{E_j}^T S_{E_j}^{-1} R_{E_j} + \sum_k R_{V_k}^T S_{V_k}^{-1} R_{V_k}.$$

10. Form the right-hand sides for the problems on the interiors of the subdomains

$$b_I^{(j)} = b_I^{(j)} - K_{II}^{(j)^{-1}} K_{IB}^{(j)} x_B^{(j)}.$$

11. Solve for the interior unknowns $x_I^{(j)} = K_{II}^{(j)^{-1}} b_I^{(j)}$.

Steps 1 though 6 can be regarded as a preprocessing stage independent of the particular loads; they need not be repeated for different loads.

The main result of this paper is now given in Theorem 4.1.

THEOREM 4.1. *The condition number of the preconditioned linear system is bounded independently of the size of the substructures H and the size of the elements h, i.e.,*

$$\kappa(P) = \kappa(\hat{S}^{-1}S) \leq C.$$

Proof. We note that no node on Γ is contained in more than four of the regions Γ^{F_i}, Γ^{E_j}, and Γ^{V_k}. Therefore the graph as described in Lemma 3.1 can be colored by a fixed finite number of colors. Hence by Lemma 3.1 we have $\lambda_{\max}(P) \leq C$.

We first use the coerciveness of our bilinear form $a_\Omega(u,u)$,

$$c||u||^2_{H^1(\Omega)} \leq a_\Omega(u,u) \leq C||u||^2_{H^1(\Omega)},$$

to make it possible to work in the $(H^1(\Omega))^q$ norm instead of the equivalent norm induced by $a_\Omega(u,u)$. To obtain a lower bound, we must demonstrate that for all $\tilde{u}^h \in \tilde{V}^h$, there exists a representation

$$\tilde{u}^h = \tilde{u}^H + \sum_i \tilde{u}^h_{E_i} + \sum_j \tilde{u}^h_{F_j} + \sum_k \tilde{u}^h_{V_k},$$

with $\tilde{u}^h_{F_i} \in \tilde{V}^h_{F_i}$, $\tilde{u}^h_{E_i} \in \tilde{V}^h_{E_i}$, $\tilde{u}^h_{V_i} \in \tilde{V}^h_{V_i}$, and $\tilde{u}^H \in V^H$, such that,

$$\begin{aligned}(5)\qquad ||\tilde{u}^H||^2_{H^1(\Omega)} + \textstyle\sum_i ||\tilde{u}^h_{E_i}||^2_{H^1(\Omega)} + \sum_j ||\tilde{u}^h_{F_j}||^2_{H^1(\Omega)} + \sum_k ||\tilde{u}^h_{V_k}||^2_{H^1(\Omega)} \\ \leq C_0^2||\tilde{u}^h||^2_{H^1(\Omega)},\end{aligned}$$

where C_0 is independent of $\tilde{u}^h$, h, and H.

We construct this representation as follows. We extend the boundary regions $\Gamma^{E_i}, \Gamma^{F_j}, \Gamma^{V_k}$ into the neighboring substructures. That is, define the following domains, $\Omega^{E_j} = \Gamma^{E_j} \cup_i \Omega_i$ for all Ω_i that satisfy $\bar{\Omega}_i \cap \Gamma^{E_j} \neq \emptyset$. Define similar domains for the face regions, Γ_{F_j}, and vertex regions Γ_{V_j}. This results in a large collection of overlapping domains. Now apply the result of Appendix A to each component of the vector-valued function $\tilde{u}^h$ separately to obtain a representation of $\tilde{u}^h$,

$$\tilde{u}^h = u^H + \sum_i u^h_{E_i} + \sum_j u^h_{F_j} + \sum_k u^h_{V_k},$$

which satisfies

$$||u^H||^2_{H^1(\Omega)} + \sum_i ||u^h_{E_i}||^2_{H^1(\Omega)} + \sum_j ||u^h_{F_j}||^2_{H^1(\Omega)} + \sum_k ||u^h_{V_k}||^2_{H^1(\Omega)} \leq C_0^2||\tilde{u}^h||^2_{H^1(\Omega)}.$$

The equivalence of the $H^1(\Omega)$ norm and seminorm on V^h follows from Friedrichs' inequality and allows us to apply the result in Appendix A. The continuous, piecewise linear functions $u^h_{F_j}$, $u^h_{E_j}$, $u^h_{V_j}$, and u^H are in the spaces $V^h(\Omega) \cap H^1_0(\Omega^{F_j})$, $V^h(\Omega) \cap H^1_0(\Omega^{E_j})$, $V^h(\Omega) \cap H^1_0(\Omega^{V_j})$, and V^H, respectively.

We now define our representation by taking the restrictions of these new functions to Γ and then extending them as discrete harmonic.

$$\tilde{u}^H|_\Gamma = u^H|_\Gamma, \qquad \tilde{u}^h_{F_i}|_\Gamma = u^h_{F_i}|_\Gamma,$$

$$\tilde{u}^h_{E_j}|_\Gamma = u^h_{E_j}|_\Gamma, \qquad \tilde{u}^h_{V_k}|_\Gamma = u^h_{V_k}|_\Gamma.$$

The resulting discrete harmonic, continous piecewise linear functions

$$\tilde{u}^h_{F_i}, \tilde{u}^h_{E_j}, \tilde{u}^h_{V_k}, \text{and } u^H$$

are in the spaces $\tilde{V}^h_{F_j}$, $\tilde{V}^h_{E_j}$, $\tilde{V}^h_{V_j}$, and V^H, respectively. The definition of discrete harmonic then gives us the needed bound in (5). We then apply Lemma 3.2 to conclude the proof of the theorem. □

If we exclude from the algorithm all of the overlap and the "vertex" spaces, we obtain the iterative substructuring algorithm presented in Dryja and Widlund [19]. That is, in the definition of $\hat{S}^{-1}$, we drop the terms

$$\sum_k R^T_{V_k} S^{-1}_{V_k} R_{V_k}$$

and restrict Γ_{E_j} to be the edge extended out to but not including the first nodes on the adjacent faces, i.e., this implies that the restriction operator R_{E_j} retrieves only those coefficients associated with the nodes along the edge. Previously it would also retrieve some coefficients associated with nodes on the adjacent faces. The iterative substructuring algorithm is also very similar to that introduced by Bramble, Pasciak, and Schatz [7]. In two dimensions these algorithms have condition numbers that grow like $(1 + \log(H/h))^2$. In three dimensions the condition numbers for these algorithms grow faster than (H/h) [8], [16].

The algorithm considered here is not intended for the same class of problems as the algorithm presented in Bramble, Pasciak, and Schatz [7]. It is intended for elliptic systems, generally in three dimensions, which result in extremely ill-conditioned linear systems. The construction of the preconditioner, specifically step 3 in the algorithm, is expensive. However, for this more difficult class of problems, we believe the additional expense is needed to produce a well-conditioned problem.

5. Numerical results. We have performed numerous experiments with problems in two dimensions. The problems considered are

- The Laplacian using the usual five-point stencil.
- The equations of linear elasticity using four node square membrane elements with two degrees of freedom per node; cf. [3];
- The equations of linear elasticity using four node square shell elements with three degrees of freedom per node; cf. [3].

Experiments have been performed on square and L-shaped regions; since there was no appreciable difference between the two cases, results are only given for the square regions. The substructures are squares. For the elasticity problems the stiffness matrices were generated using the SESAM code [3], a large, reliable commercial structural analysis code, using a Poisson ratio of .3.

The experiments were run twice, once using all the subspaces as indicated in the algorithm and once excluding the "vertex" spaces. That is, we drop the terms $\sum_k R^T_{V_k} S^{-1}_{V_k} R_{V_k}$ from the preconditioner $\hat{S}^{-1}$. As expected the condition number remains bounded by a constant independent of H and h when all the spaces were included. When the "vertex" spaces were excluded the condition number appears to grow like $(1 + \log(H/h))^2$, as expected.

The selection of an appropriate stopping condition for the preconditioned conjugate gradient method is crucial. A stopping criteria based only on a norm of the

TABLE 1
Condition numbers and iteration counts for the Laplacian.

Number of subdomains	Nodes along edge	Number of unknowns on Γ	No preconditioner		Without "vertex" spaces		With "vertex" spaces	
16	3	81	35.26	14	4.92	7	2.45	6
	7	177	75.10	24	7.59	9	2.55	7
	15	369	155	37	10.89	10	2.82	7
	31	753	315	51	14.84	11	2.99	7
64	3	385	137	32	5.35	9	2.60	8
	7	833	290	49	8.19	10	2.68	8
	15	1729	598	70	11.54	12	2.78	8
	31	3521	1216	*	15.62	13	2.87	8
256	3	1665	545	62	5.46	9	2.63	8
	7	3585	1151	91	8.27	10	2.70	7
	15	7425	2372	*	11.77	12	2.80	7
	31	15105	4766	*	15.90	13	2.89	7

TABLE 2
Condition number as function of overlap for the Laplacian.

Overlap in nodes	0	1	2	3	4	5	6	7	8
Condition number	15.62	4.49	4.01	3.78	3.52	3.38	3.01	2.92	2.81
Iterations	13	8	8	8	8	8	8	8	8

residual can make comparisons between preconditioned and unpreconditioned results misleading since the eigenvalues of the two operators can be of completely different orders of magnitude. For instance, for elasticity problems the eigenvalues of the original stiffness matrices can be of order 10^{12}, while the eigenvalues of the preconditioned problems generally are of order 1. We have therefore chosen to use the stopping condition

$$||\text{residual}||_{L^2} \leq \epsilon||\text{approx. solution}||_{L^2}\lambda_{\min}(\hat{S}^{-1}S).$$

$\lambda_{\min}(\hat{S}^{-1}S)$ is calculated using the Lanczos method at very little extra expense. We have chosen to use $\epsilon = 10^{-5}$; this assures that roughly five digits of the solution are correct and not many more, regardless of the preconditioner used.

In Table 1 the experiments are conducted for the Laplacian. The overlap of the "vertex" spaces onto the "edge" spaces is chosen to be $H/4$. In Table 2, we examine the effect of varying the amount of overlap of the "vertex" spaces onto the "edge" spaces for the case with 64 substructures and 31 nodes along the edge of each substructure. We see that the overlap is very important but a small overlap has almost as much effect as a larger overlap. We give a sample of the convergence behavior in Table 3, showing the discrete L^2 norm of the error as a function of the number of iterations. This is again for the case of 64 substructures and 31 nodes along the edge of each substructure. The linear elasticity problems are considered in Tables 4–9.

TABLE 3
Errors and convergence rates for Laplacian.

Iter.	No preconditioning		Without "vertex" Spaces		With "vertex" Spaces	
i	$\|\|e^i\|\|_{L^2}$	$(\frac{\|\|e^i\|\|_{L^2}}{\|\|e^0\|\|_{L^2}})^{1/i}$	$\|\|e^i\|\|_{L^2}$	$(\frac{\|\|e^i\|\|_{L^2}}{\|\|e^0\|\|_{L^2}})^{1/i}$	$\|\|e^i\|\|_{L^2}$	$(\frac{\|\|e^i\|\|_{L^2}}{\|\|e^0\|\|_{L^2}})^{1/i}$
1	9.8×10^{-1}	.99	3.0×10^{-2}	.03	3.0×10^{-2}	.03
2	9.6×10^{-1}	.98	9.9×10^{-3}	.10	1.0×10^{-2}	.10
3	9.5×10^{-1}	.98	2.2×10^{-3}	.13	4.1×10^{-3}	.16
4	9.3×10^{-1}	.98	1.2×10^{-3}	.19	9.0×10^{-4}	.17
5	9.2×10^{-1}	.98	6.3×10^{-4}	.23	7.0×10^{-5}	.15
6	9.1×10^{-1}	.98	4.5×10^{-4}	.28	2.5×10^{-5}	.17
7	8.9×10^{-1}	.98	1.0×10^{-4}	.27	9.4×10^{-6}	.19
8	8.8×10^{-1}	.98	3.6×10^{-5}	.28	3.3×10^{-6}	.21
9	8.6×10^{-1}	.98	9.5×10^{-6}	.28	3.8×10^{-7}	.19
10	8.4×10^{-1}	.98	6.3×10^{-6}	.30	6.7×10^{-8}	.19
11	8.2×10^{-1}	.98	7.7×10^{-6}	.34	2.4×10^{-8}	.20
12	7.9×10^{-1}	.98	3.0×10^{-6}	.35	5.9×10^{-9}	.21

TABLE 4
Condition numbers and iteration counts for membrane elements.

Number of subdomains	Nodes along edge	Number of unknowns on Γ	No preconditioner		Without "vertex" spaces		With "vertex" spaces	
16	3	162	22.16	19	10.52	15	3.51	10
	7	354	46.63	28	14.84	17	3.51	10
	15	738	96.34	41	19.83	19	3.56	10
	31	1506	196	60	25.51	20	3.62	10
64	3	770	84.82	37	12.13	18	3.85	10
	7	1666	178	55	16.92	19	3.85	10
	15	3458	368	79	22.37	22	3.84	10
	31	7042	747	*	28.52	25	3.89	10
256	3	3330	334	75	12.42	18	3.91	10
	7	7170	705	*	17.31	19	3.90	10
	15	14850	1453	*	22.88	22	3.89	10
	31	30210	2921	*	29.15	25	3.94	10

TABLE 5
Condition number as function of overlap for membrane elements.

Overlap in nodes	0	1	2	3	4	5	6	7	8
Condition number	28.55	5.59	4.85	4.36	4.01	3.89	3.88	3.89	3.89
Iterations	25	12	11	10	10	10	10	10	10

TABLE 6
Errors and convergence rates for membrane elements.

Iter.	No preconditioning		Without "vertex" spaces		With "vertex" spaces	
i	$\|\|e^i\|\|_{L^2}$	$(\frac{\|\|e^i\|\|_{L^2}}{\|\|e^0\|\|_{L^2}})^{1/i}$	$\|\|e^i\|\|_{L^2}$	$(\frac{\|\|e^i\|\|_{L^2}}{\|\|e^0\|\|_{L^2}})^{1/i}$	$\|\|e^i\|\|_{L^2}$	$(\frac{\|\|e^i\|\|_{L^2}}{\|\|e^0\|\|_{L^2}})^{1/i}$
1	9.7×10^{-1}	.98	8.0×10^{-2}	.08	8.7×10^{-2}	.09
2	9.5×10^{-1}	.98	3.7×10^{-2}	.19	4.2×10^{-2}	.20
3	9.3×10^{-1}	.98	1.1×10^{-2}	.22	1.6×10^{-2}	.25
4	9.2×10^{-1}	.98	6.3×10^{-3}	.28	3.9×10^{-3}	.25
5	9.0×10^{-1}	.98	3.7×10^{-3}	.33	1.1×10^{-3}	.26
6	8.8×10^{-1}	.98	3.4×10^{-3}	.39	5.1×10^{-4}	.28
7	8.6×10^{-1}	.98	2.0×10^{-3}	.41	1.6×10^{-4}	.29
8	8.3×10^{-1}	.98	1.4×10^{-3}	.44	4.4×10^{-5}	.29
9	8.1×10^{-1}	.98	5.1×10^{-4}	.43	1.7×10^{-5}	.30
10	7.8×10^{-1}	.98	2.3×10^{-4}	.43	5.4×10^{-6}	.30
11	7.5×10^{-1}	.97	2.5×10^{-4}	.47	1.5×10^{-6}	.30
12	7.2×10^{-1}	.97	2.6×10^{-4}	.50	5.1×10^{-7}	.30

TABLE 7
Condition numbers and iteration counts for shell elements.

Number of subdomains	Nodes along edge	Number of unknowns on Γ	No preconditioner		Without "vertex" spaces		With "vertex" spaces	
16	3	243	452	47	10.51	17	3.48	10
	7	531	442	66	14.83	18	3.48	10
	15	1107	970	*	19.81	21	3.53	10
	31	2259	*	*	25.48	23	3.60	10
64	3	1155	1751	*	11.99	18	3.72	10
	7	2499	1707	*	15.93	19	3.74	10
	15	5187	3800	*	22.12	23	3.83	10
	31	10563	*	*	28.24	26	3.88	10
256	3	4995	*	*	11.63	17	3.73	10
	7	10755	*	*	16.19	19	3.81	10
	15	22275	*	*	22.58	22	3.85	10

A. Appendix: A partitioning result. For two or three dimensions let Ω be a polyhedral (polygonal) domain that has been triangulated into substructures that are shape regular, with diameter $O(H)$. Continue the triangulation to obtain a triangulation with elements of diameter $O(h)$. Furthermore assume that Ω has been covered with N shape regular overlapping regions Ω_i (not necessarily related to the coarse triangulation above), each with a diameter $O(H)$, each of which overlaps all its neighbors with an overlap of $O(H)$. We require that the domains Ω_i be aligned with the fine triangulation. Let $V^H(\Omega) \subset H_0^1(\Omega)$ and $V^h(\Omega) \subset H_0^1(\Omega)$ be the spaces of continuous, piecewise linear functions, on the two triangulations, which vanish on the boundary $\partial\Omega$. We then construct the following spaces

$$V_0^h = V^H, \qquad V_i^h = V^h \cap H_0^1(\Omega_i).$$

The following theorem is a variation of a result given in Dryja and Widlund [19].

THEOREM A.1. *For all $u^h \in V^h$ there exists $u_i^h \in V_i^h$ with $u^h = \sum_{i=0}^N u_i^h$ such that*

$$\sum_{i=0}^{N} |u_i^h|^2_{H^1(\Omega)} \leq C_0^2 |u^h|^2_{H^1(\Omega)},$$

where C_0 is independent of $u^h, h,$ and H.

Proof. From Strang [32], we know that there exists a linear map $\hat{I}_H : V^h \rightarrow V^H$ that satisfies

$$||u^h - \hat{I}_H u^h||^2_{L_2(\Omega)} \leq CH^2 |u^h|^2_{H^1(\Omega)} \tag{6}$$

and

$$|u^h - \hat{I}_H u^h|^2_{H^1(\Omega)} \leq C|u^h|^2_{H^1(\Omega)}. \tag{7}$$

We then define

$$u_0^h = \hat{I}_H u^h, \qquad w^h = u^h - u_0^h$$

and

$$u_i^h = I_h(\theta_i w^h).$$

I_h is the linear interpolation operator onto the space V^h, and the θ_i form a partition of unity with $\theta_i \in C_0^\infty(\Omega_i), 0 \leq \theta_i \leq 1$, and $\sum_{i=1}^N \theta_i = 1$. Since I_h is a linear operator, it is immediate that

$$u^h = \sum_{i=0}^{N} u_i^h.$$

Because of the generous overlap between subregions, we can ensure that the gradients of θ_i are well behaved. That is, θ_i can be constructed so that its gradients never grow faster than $|\nabla\theta_i|^2_{L^\infty} \leq C/H^2$. If we let K represent any single element in the triangulation, this can be expressed as

$$||\theta_i - \bar{\theta}_i||^2_{L^\infty(K)} \leq C(h/H)^2. \tag{8}$$

TABLE 8
Condition number as function of overlap for shell elements.

Overlap in nodes	0	1	2	3	4	5	6	7	8
Condition number	28.24	5.62	4.81	4.33	3.98	3.87	3.87	3.88	3.88
Iterations	23	12	11	10	10	10	10	10	10

TABLE 9
Errors and convergence rates for shell elements.

Iter.	No preconditioning		Without "vertex" spaces		With "vertex" spaces	
i	$\|\|e^i\|\|_{L^2}$	$(\frac{\|\|e^i\|\|_{L^2}}{\|\|e^0\|\|_{L^2}})^{1/i}$	$\|\|e^i\|\|_{L^2}$	$(\frac{\|\|e^i\|\|_{L^2}}{\|\|e^0\|\|_{L^2}})^{1/i}$	$\|\|e^i\|\|_{L^2}$	$(\frac{\|\|e^i\|\|_{L^2}}{\|\|e^0\|\|_{L^2}})^{1/i}$
1	*	*	6.8×10^{-2}	.07	7.4×10^{-2}	.07
2	*	*	3.2×10^{-2}	.18	3.6×10^{-2}	.19
3	*	*	9.4×10^{-3}	.21	1.4×10^{-2}	.24
4	*	*	5.3×10^{-3}	.27	3.4×10^{-3}	.24
5	*	*	3.1×10^{-3}	.32	1.0×10^{-3}	.25
6	*	*	3.1×10^{-3}	.38	4.7×10^{-4}	.28
7	*	*	1.9×10^{-3}	.41	1.4×10^{-4}	.28
8	*	*	1.4×10^{-3}	.44	3.9×10^{-5}	.28
9	*	*	5.5×10^{-4}	.43	1.6×10^{-5}	.29
10	*	*	3.3×10^{-4}	.45	4.9×10^{-6}	.29
11	*	*	2.5×10^{-4}	.47	1.3×10^{-6}	.29
12	*	*	2.5×10^{-4}	.50	4.6×10^{-7}	.30
13	*	*	2.4×10^{-4}	.53	1.5×10^{-7}	.30

Here $\bar{\theta}_i$ is the average of θ_i on element K.

We now estimate the H^1 norm of u_i^h over a single element.

$$\begin{aligned} |u_i^h|^2_{H^1(K)} &= |I_h(\bar{\theta}_i w^h + (\theta_i - \bar{\theta}_i)w^h)|^2_{H^1(K)} \\ &\leq 2|\bar{\theta}_i w^h|^2_{H^1(K)} + 2|I_h(\theta_i - \bar{\theta}_i)w^h|^2_{H^1(K)}, \end{aligned}$$

which can be bounded using an inverse inequality by

$$|u_i^h|^2_{H^1(K)} \leq 2|\bar{\theta}_i w^h|^2_{H^1(K)} + Ch^{-2}||I_h(\theta_i - \bar{\theta}_i)w^h||^2_{L^2(K)}.$$

We now use (8) and the trivial inequality $||\bar{\theta}_i||_{L^\infty} \leq 1$ to obtain

$$|u_i^h|^2_{H^1(K)} \leq 2|w^h|^2_{H^1(K)} + CH^{-2}||w^h||^2_{L^2(K)}.$$

Since a finite bounded number of u_i^h are nonzero for any element K, we obtain, when summing over i,

$$\sum_{i=1}^{N} |u_i^h|^2_{H^1(K)} \leq C|w^h|^2_{H^1(K)} + CH^{-2}||w^h||^2_{L^2(K)}.$$

Next sum over the elements K,

$$\sum_{i=1}^{N} |u_i^h|^2_{H^1(\Omega)} \leq C|w^h|^2_{H^1(\Omega)} + CH^{-2}||w^h||^2_{L^2(\Omega)}.$$

To finish the argument, we use (6) and (7) to obtain

$$\sum_{i=0}^{N} |u_i^h|^2_{H^1(\Omega)} \leq C_0^2|u^h|^2_{H^1(\Omega)}. \qquad \square$$

Acknowledgments. I would like to thank Petter Bjørstad and Anders Hvidsten for their assistance in using the SESAM code and for many useful discussions. I would also like to thank VERITAS SESAM SYSTEMS, Høvik, Norway, for the use and access to SESAM for the generation of test stiffness matrices which were used in the numerical experiments reported in this paper.

REFERENCES

[1] I. Babuška, A. Craig, J. Mandel, and J. Pitkäranta, *Efficient preconditioning for the p-version finite element method in two dimensions*, Tech. Rep. 41098, University of Colorado, Denver, CO, 1989; SIAM J. Numer. Anal., submitted.

[2] I. Babuška, M. Griebel, and J. Pitkäranta, *The problem of selecting the shape functions for a p-type finite element*, Internat. J. Numer. Methods Engrg., 18 (1989), pp. 1891–1908.

[3] K. Bell, B. Hatlestad, O. E. Hansteen, and P. O. Araldsen, *NORSAM, a programming system for the finite element method*, Users manual, Part 1, General description, NTH, Trondheim, 1973.

[4] P. E. Bjørstad and A. Hvidsten, *Iterative methods for substructured elasticity problems in structural analysis*, in Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988.

[5] P. E. Bjørstad and O. B. Widlund, *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.

[6] ———, *Iterative methods for the solution of elliptic problems on regions partitioned into substructures*, SIAM J. Numer. Anal., 23 (1986), pp. 1093–1120.

[7] J. H. Bramble, J. E. Pasciak, and A. H. Schatz, *The construction of preconditioners for elliptic problems by substructuring,* I, Math. Comp., 47 (1986), pp. 103–134.

[8] ———, *The construction of preconditioners for elliptic problems by substructuring,* II, Math. Comp., 49 (1987), pp. 1–16.

[9] ———, *The construction of preconditioners for elliptic problems by substructuring,* III, Math. Comp., 51 (1988), pp. 415–430.

[10] ———, *The construction of preconditioners for elliptic problems by substructuring,* IV, Math. Comp., 53 (1989), pp. 1–24.

[11] T. F. Chan and D. C. Resasco, *A survey of preconditioners for domain decomposition*, Tech. Rep. /DCS/RR-414, Yale University, New Haven, CT, 1985.

[12] ———, *Analysis of domain decomposition preconditioners on irregular regions*, in Advances in Computer Methods for Partial Differential Equations, R. Vichnevetsky and R. Stepleman, eds., IMACS, 1987.

[13] P. G. Ciarlet, *The Finite Element Method for Elliptic Problems*, North-Holland, Amsterdam, 1978.

[14] ———, *Lectures on three-dimensional elasticity*, Springer-Verlag, Berlin, 1983.

[15] Y. De Roeck, *A local preconditioner in a domain-decomposed method*, Tech. Rep., Centre Europée de Recherche et de Formation Avancée en Calcul Scientifique, Toulouse, France, 1989.

[16] M. Dryja, *A method of domain decomposition for 3-D finite element problems*, in First International Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1988.

[17] ———, *An additive Schwarz algorithm for two- and three- dimensional finite element elliptic problems*, in Domain Decomposition Methods, T. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia,1989.

[18] M. Dryja and O. B. Widlund, *An additive variant of the Schwarz alternating method for the case of many subregions*, Tech. Report 339, also Ultracomputer Note 131, Department of Computer Science, Courant Institute, New York, NY, 1987.

[19] ———, *Some Domain Decomposition Algorithms for Elliptic Problems*, Proceedings of the Conference on Iterative Methods for Large Linear Systems, held in Austin, TX, October 1988, to celebrate the sixty-fifth birthday of David M. Young, Jr., Academic Press, Orlando, FL, 1989.

[20] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Second Edition, Johns Hopkins Univ. Press, Baltimore, MD, 1989.

[21] T. R. Hughes and R. M. Ferencz, *Fully vectorized EBE preconditioners for nonlinear solid mechanics: applications to large-scale three-dimensional continuum, shell and contact/impact problems*, in First International Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1988.

[22] P. L. Lions, *On the Schwarz alternating method.* I., in First International Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1988.

[23] J. Mandel, *Iterative solvers by substructuring for the p-version finite element method*, Comput. Methods Appl. Mech. Engrg., (1989). To appear in a special issue as Proceedings of an International Conference on Spectral and High Order Methods, Como, Italy, 1989.

[24] ———, *On block diagonal and Schur complement preconditioning*, Tech. Report University of Colorado, Denver, CO, 1989.

[25] ———, *Two-level domain decomposition preconditioning for the p-version finite element version in three dimensions*, Internat. J. Numer. Methods Engrg., (1989), to appear.

[26] ———, *Hierarchical preconditioning and partial orthogonalization for the p-version finite element method*, in Proceedings of the Third International Symposium on Domain Decomposition Methods for Partial Differential Equations, Houston, Texas, 1989, T. F. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1990.

[27] A. M. Matsokin and S. V. Nepomnyaschikh, *A Schwarz alternating method in a subspace*, Soviet Mathematics, 29 (1985), pp. 78–84.

[28] S. V. Nepomnyaschikh, *Domain decomposition and Schwarz methods in a subspace for the approximate solution of elliptic boundary value problems.* Ph.D. thesis, Computing Center of the Siberian Branch of the USSR Academy of Sciences, Novosibirsk, USSR, 1986.

[29] B. F. Smith, *Domain decomposition algorithms for the partial differential equations of linear elasticity*, Ph.D. thesis, Courant Institute, New York, NY, 1990.

[30] B. F. Smith and O. B. Widlund, *A domain decomposition algorithm using a hierarchical basis*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 1212–1220.

[31] S. L. Sobolev, *L'Algorithme de Schwarz dans la Théorie de l'Elasticité*, Comptes Rendus (Doklady) de l'Académie des Sciences de l'URSS, IV (1936), pp. 243–246.

[32] G. Strang, *Approximation in the finite element method*, Numer. Math., 19 (1972), pp. 81–98.

[33] T. E. Tezduyar and J. Liou, *Element-by-element and implicit-explicit finite element formulations for computational fluid dynamics*, in First International Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1988.

[34] O. B. Widlund, *Iterative substructuring methods: Algorithms and theory for elliptic problems in the plane*, in First International Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1988.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 1, pp. 379–396, January 1992

020

AN UNCONVENTIONAL DOMAIN DECOMPOSITION METHOD FOR AN EFFICIENT PARALLEL SOLUTION OF LARGE-SCALE FINITE ELEMENT SYSTEMS*

CHARBEL FARHAT† AND FRANCOIS-XAVIER ROUX‡

Abstract. A domain decomposition algorithm based on a hybrid variational principle is developed for the parallel finite element solution of selfadjoint elliptic partial differential equations. The spatial domain is partitioned into a set of totally disconnected subdomains, each assigned to an individual processor. Lagrange multipliers are introduced to enforce compatibility at the interface points. Within each subdomain, the singularity due to the disconnection is resolved in a two-step procedure. First, the null space component of each local operator is eliminated from the local problem. Next, its contribution to the local solution is related to the Lagrange multipliers through an orthogonality condition. Finally, a conjugate *projected* gradient algorithm is developed for the solution of the coupled system of local null space components and Lagrange multipliers. When implemented on local memory multiprocessors, the proposed hybrid method requires fewer interprocessor communications than conventional Schur methods. It is also suitable for parallel/vector computers with shared memory. Moreover, unlike parallel direct solvers, it exhibits a degree of parallelism that is not limited by the bandwidth of the finite element system of equations. In this paper, it is applied to the solution of large-scale structural and solid mechanics problems.

Key words. domain decomposition, finite elements, parallel processing

AMS(MOS) subject classifications. 65N20, 65N30, 65W05

1. Introduction.

In this paper, we present a parallel finite element subdomain-based computational method for the solution of selfadjoint elliptic partial differential equations. The method blends direct and iterative solution schemes. Its unique feature is that it requires fewer interprocessor communications than conventional domain decomposition algorithms, while still offering the same amount of parallelism. Roux [9], [10] has presented an early version of this work that is limited to a very special class of problems where a finite element domain can be partitioned into a set of disconnected but nonfloating (nonsingular) subdomains. Here, we generalize the method for arbitrary finite element problems and arbitrary mesh partitions.

In §2, we partition the finite element domain into a set of totally disconnected subdomains and derive a computational strategy from a hybrid variational principle where the intersubdomain continuity constraint is removed by the introduction of a Lagrange multiplier. An arbitrary mesh partition typically contains a set of floating subdomains that induce local singularities. The handling of these singularities is treated in §3. First, the null space components are eliminated in parallel from each local problem and a direct scheme is applied concurrently to all subdomains in order to recover each partial local solution. Next, the contributions of these null space components are related to the Lagrange multipliers through an orthogonality condition.

*Received by the editors April 5, 1990; accepted for publication (in revised form) January 30, 1991.

†Department of Aerospace Engineering Sciences, and Center for Space Structures and Controls, University of Colorado, Boulder, Colorado 80309-0429. This work was supported in part by the National Science Foundation under grant ASC-8717773, and by the Air Force Office of Scientific Research under grant AFOSR-89-0422.

‡O. N. E. R. A. Groupe Calcul Parallele, 29 Av. de la Division Leclerc, BP72 92322 Chatillon Cedex, France.

A parallel preconditioned conjugate *projected* gradient algorithm is developed in §4 for the solution of the coupled system of local zero energy modes and Lagrange multipliers, which completes the solution of the problem. Section 5 emphasizes the parallel characteristics of the proposed method and contrasts it with conventional Schur methods. Section 6 illustrates the method with the parallel solution of structural examples on the iPSC/1 hypercube. Finally, §7 concludes this paper.

2. A hybrid variational principle. For the sake of clarity, we consider first the case of two subdomains, then generalize the method for an arbitrary number of subdomains.

The variational form of the three-dimensional boundary-value problem to be solved goes as follows. Given f and h, find the function u that is a stationary point of the functional:

$$J(v) = \tfrac{1}{2}a(v,v) - (v,f) - (v,h)_\Gamma$$

where

$$\begin{aligned} a(v,w) &= \int_\Omega v_{(i,j)} c_{ijkl} w_{(k,l)}\, d\Omega \\ (v,f) &= \int_\Omega v_i f_i\, d\Omega \\ (v,h)_\Gamma &= \int_{\Gamma_h} v_i h_i\, d\Gamma. \end{aligned} \tag{2.1}$$

In the above, the indices i, j, k take the values 1 to 3, $v_{(i,j)} = (v_{i,j} + v_{j,i})/2$ and $v_{i,j}$ denotes the partial derivative of the ith component of v with respect to the jth spatial variable, c_{ijkl} are tensorial coefficients defining a symmetric positive definite operator, Ω denotes the volume of the body, Γ its piecewise smooth boundary, and Γ_h the piece of Γ where the functions h_i are prescribed.

If Ω is subdivided into two regions Ω_1 and Ω_2 (Fig. 1), solving the above problem is equivalent to finding the two functions u_1 and u_2 that are stationary points of the functionals:

$$\begin{aligned} J_1(v_1) &= \tfrac{1}{2}a(v_1,v_1)_{\Omega_1} - (v_1,f)_{\Omega_1} - (v_1,h)_{\Gamma_1} \\ J_2(v_2) &= \tfrac{1}{2}a(v_2,v_2)_{\Omega_2} - (v_2,f)_{\Omega_2} - (v_2,h)_{\Gamma_2} \end{aligned}$$

where

$$\begin{aligned} a(v_1,w_1)_{\Omega_1} &= \int_{\Omega_1} v_{1(i,j)} c_{ijkl} w_{1(k,l)}\, d\Omega \\ a(v_2,w_2)_{\Omega_2} &= \int_{\Omega_2} v_{2(i,j)} c_{ijkl} w_{2(k,l)}\, d\Omega \\ (v_1,f)_{\Omega_1} &= \int_{\Omega_1} v_{1i} f_i\, d\Omega \\ (v_2,f)_{\Omega_2} &= \int_{\Omega_2} v_{2i} f_i\, d\Omega \\ (v_1,h)_{\Gamma_1} &= \int_{\Gamma_{h1}} v_{1i} h_i\, d\Gamma \\ (v_2,h)_{\Gamma_2} &= \int_{\Gamma_{h2}} v_{2i} h_i\, d\Gamma \end{aligned} \tag{2.2}$$

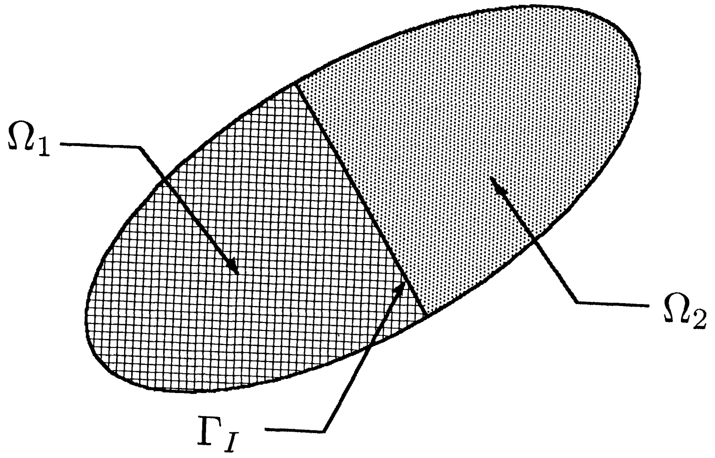


FIG. 1. *A two-subdomain decomposition.*

and that satisfy on the interface boundary Γ_I the continuity constraint:

$$u_1 = u_2 \quad \text{on } \Gamma_I. \tag{2.3}$$

Solving the two above variational problems (2.2) with the subsidiary continuity condition (2.3) is equivalent to finding the saddle point of the Lagrangian:

$$\begin{aligned} J^*(v_1, v_2, \mu) &= J_1(v_1) + J_2(v_1) + (v_1 - v_2, \mu)_{\Gamma_I}, \quad \text{where} \\ (v_1 - v_2, \mu)_{\Gamma_I} &= \int_{\Gamma_I} \mu(v_1 - v_2)\, d\Gamma, \end{aligned} \tag{2.4}$$

that is, finding the two displacement fields u_1 and u_2 and the Lagrange multiplier λ that satisfy

$$J^*(u_1, u_2, \mu) \leq J^*(u_1, u_2, \lambda) \leq J^*(v_1, v_2, \lambda) \tag{2.5}$$

for any admissible v_1, v_2, and μ. Clearly, the left inequality in (2.5) implies that $(u_1 - u_2, \mu)_{\Gamma_I} \leq (u_1 - u_2, \lambda)_{\Gamma_I}$, which imposes that $(u_1 - u_2, \mu)_{\Gamma_I} = 0$ for any admissible μ and therefore $u_1 = u_2$ on Γ_I. The right inequality in (2.5) imposes that $J_1(u_1) + J_2(u_2) \leq J_1(v_1) + J_2(v_2)$ for any pair of admissible functions (v_1, v_2). This implies that among all admissible pairs (v_1, v_2) that satisfy the continuity condition (2.3), the pair (u_1, u_2) minimizes the sum of the functionals J_1 and J_2 defined, respectively, on Ω_1 and Ω_2. Therefore, u_1 and u_2 are the restriction of the solution u of the nonpartitioned problem (2.1) to, respectively, Ω_1 and Ω_2. Indeed, (2.4) and (2.5) correspond to a hybrid variational principle where the intersubdomain continuity constraint (2.3) is removed by the introduction of a Lagrange multiplier (see, for example, Pian [8]).

If now the functions u_1 and u_2 are expressed by suitable shape functions as:

$$u_1 = \mathbf{Nu}_1 \quad \text{and} \quad u_2 = \mathbf{Nu}_2$$

and the continuity equation is enforced for the discrete problem, a standard Galerkin procedure transforms the hybrid variational principle (2.4) in the following algebraic system:

$$\begin{aligned} \mathbf{K}_1\mathbf{u}_1 &= \mathbf{f}_1 + \mathbf{B}_1^T\boldsymbol{\lambda} \\ \mathbf{K}_2\mathbf{u}_2 &= \mathbf{f}_2 - \mathbf{B}_2^T\boldsymbol{\lambda} \\ \mathbf{B}_1\mathbf{u}_1 &= \mathbf{B}_2\mathbf{u}_2 \end{aligned} \tag{2.6}$$

where $\mathbf{K}_j$, $\mathbf{u}_j$, and $\mathbf{f}_j$, $j = 1, 2$, are, respectively, the generalized stiffness symmetric matrix, the generalized displacement vector, and the prescribed generalized force vector associated with the finite element discretization of Ω_j. The vector of Lagrange multipliers $\boldsymbol{\lambda}$ represents the generalized interaction forces between the two subdomains Ω_1 and Ω_2 along their common boundary Γ_I. Within each subdomain Ω_j, we denote the number of interior nodal unknowns by n_j^s and the number of interface nodal unknowns by n_j^I. The total number of interface nodal unknowns is denoted by n_I. Note that $n_I = n_1^I = n_2^I$ in the particular case of two subdomains. If the interior degrees of freedom are numbered first and the interface ones are numbered last, each of the two connectivity matrices $\mathbf{B}_1$ and $\mathbf{B}_2$ takes the form

$$\mathbf{B}_j = [\,\mathbf{O}_j \quad \mathbf{I}_j\,], \qquad j = 1, 2,$$

where $\mathbf{O}_j$ is an $n_j^I \times n_j^s$ null matrix and $\mathbf{I}_j$ is the $n_j^I \times n_j^I$ identity matrix. The vector of Lagrange multipliers $\boldsymbol{\lambda}$ is n_I long.

If both $\mathbf{K}_1$ and $\mathbf{K}_2$ are nonsingular, (2.6) can be written as

$$\begin{aligned} (\mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{B}_1^T + \mathbf{B}_2\mathbf{K}_2^{-1}\mathbf{B}_2^T)\boldsymbol{\lambda} &= \mathbf{B}_2\mathbf{K}_2^{-1}\mathbf{f}_2 - \mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{f}_1 \\ \mathbf{u}_1 &= \mathbf{K}_1^{-1}(\mathbf{f}_1 + \mathbf{B}_1^T\boldsymbol{\lambda}) \\ \mathbf{u}_2 &= \mathbf{K}_2^{-1}(\mathbf{f}_2 - \mathbf{B}_2^T\boldsymbol{\lambda}) \end{aligned} \tag{2.7}$$

and the solution of (2.6) is obtained by solving the first of equations (2.7) for the Lagrange multipliers $\boldsymbol{\lambda}$, then substituting these in the second of (2.7) and backsolving for $\mathbf{u}_1$ and $\mathbf{u}_2$.

For an arbitrary number of subdomains Ω_j, the method goes as follows. First, the finite element mesh is decomposed into a set of totally disconnected meshes (Fig. 2). For each submesh, the matrix $\mathbf{K}_j$ and the vector $\mathbf{f}_j$ are formed. If a_j and N_s denote, respectively, the number of subdomains Ω_k that are adjacent to Ω_j and the total number of subdomains, the finite element variational interpretation of the saddle-point problem (2.4) generates the following algebraic system:

$$\begin{aligned} \mathbf{K}_j\mathbf{u}_j &= \mathbf{f}_j + \sum_{k=1}^{k=a_j} \mathbf{B}_{jk}{}^T\boldsymbol{\lambda} \qquad j = 1, N_s \\ \mathbf{B}_{jk}\mathbf{u}_j &= \mathbf{B}_{kj}\mathbf{u}_k \qquad\qquad\quad j = 1, N_s \text{ and } \Omega_k \text{ connected to } \Omega_j \end{aligned} \tag{2.8}$$

where $\mathbf{B}_{jk}$ is a boolean matrix that interconnects Ω_j with its neighbors Ω_k. In general, $\mathbf{B}_{jk}$ is $n_I \times (n_j^s + n_j^I)$ and has the following pattern:

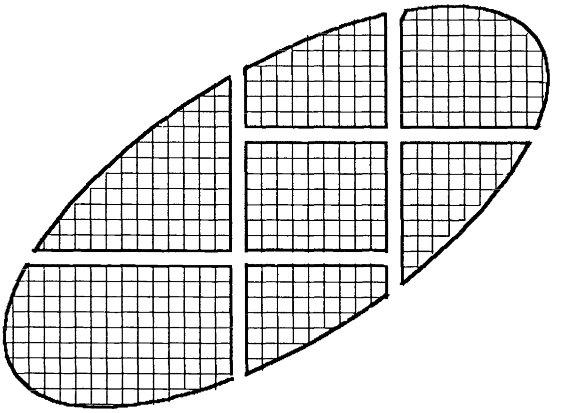


FIG. 2. *A multiple subdomain decomposition.*

$$\mathbf{B}_{jk} = \begin{bmatrix} \mathbf{O}_1(j,k) \\ \mathbf{C}_{jk} \\ \mathbf{O}_2(j,k) \end{bmatrix}$$

where $\mathbf{O}_1(j,k)$ is an $m_1(j,k) \times (n_j^s + n_j^I)$ zero matrix, $\mathbf{O}_2(j,k)$ is another $m_2(j,k) \times (n_j^s + n_j^I)$ zero matrix and $\mathbf{C}_{jk}$ is an $m_c(j,k) \times (n_j^s + n_j^I)$ connectivity matrix, $m_c(j,k)$ is the number of Lagrange multipliers that interconnect Ω_j with its neighbor Ω_k, and $m_1(j,k)$ and $m_2(j,k)$ are two nonnegative integers that satisfy $m_1(j,k) + m_c(j,k) + m_2(j,k) = n_I$. The connectivity matrix $\mathbf{C}_{jk}$ can be written as

$$\mathbf{C}_{jk} = \mathbf{O}_3(j,k) \quad \mathbf{I}_{jk} \quad \mathbf{O}_4(j,k)$$

where $\mathbf{O}_3(j,k)$ is an $m_c(j,k) \times m_3(j,k)$ zero matrix, $\mathbf{I}_{jk}$ is the $m_c(j,k) \times m_c(j,k)$ identity matrix, $\mathbf{O}_4(j,k)$ is another $m_c(j,k) \times m_4(j,k)$ zero matrix, and $m_3(j,k)$ and $m_4(j,k)$ are two nonnegative integers that verify $m_3(j,k) + m_c(j,k) + m_4(j,k) = n_j^s + n_j^I$.

If $\mathbf{K}_j$ is nonsingular for all $j = 1, N_s$, the solution procedure (2.7) can be extended to the case of an arbitrary number of subdomains. However, the finite element tearing process described in this section may produce some "floating" subdomains Ω_f that are characterized by a singular matrix $\mathbf{K}_f$. When this happens, the above solution algorithm (2.7) breaks down and a special computational strategy is required to handle the local singularities.

At this point, we note that the utility of Lagrange multipliers specifically for domain decomposition has also been previously recognized by other investigators (Dihn, Glowinski, and Periaux [2]; Dorr [3]).

3. Removing local singularities.

Again, we focus on the two-subdomain decomposition. The extrapolation to $N_s > 2$ is straightforward. For example, consider the elastostatic problem and suppose that Ω corresponds to a cantilever beam and that Ω_1 and Ω_2 are the result of a vertical partitioning (Fig. 3).

In this case, $\mathbf{K}_1$ is positive definite and $\mathbf{K}_2$ is positive semidefinite, since no boundary condition is specified over Ω_2. Therefore, the second of equations (2.6)

$$\mathbf{K}_2\mathbf{u}_2 = \mathbf{f}_2 - \mathbf{B}_2^T\boldsymbol{\lambda} \tag{3.1}$$

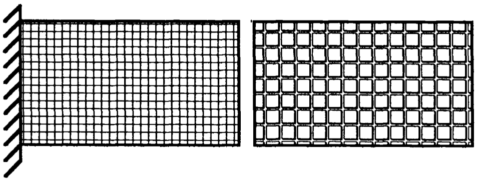


FIG. 3. *Decomposition resulting in a singular subdomain.*

requires special attention. If the singular system (3.1) is consistent, a generalized inverse of $\mathbf{K}_2$ can be found, that is, a matrix $\mathbf{K}_2^+$ that verifies $\mathbf{K}_2\mathbf{K}_2^+\mathbf{K}_2 = \mathbf{K}_2$, and the general solution of (3.1) is given by

$$\mathbf{u}_2 = \mathbf{K}_2^+(\mathbf{f}_2 - \mathbf{B}_2^T\boldsymbol{\lambda}) + \mathbf{R}_2\boldsymbol{\alpha} \tag{3.2}$$

where $\mathbf{R}_2$ is an $(n_2^s + n_2^I) \times n_2^r$ rectangular matrix whose columns form a basis of the null space of $\mathbf{K}_2$, and $\boldsymbol{\alpha}$ is a vector of length n_2^r. Physically, $\mathbf{R}_2$ represents the rigid body modes (zero energy modes) of Ω_2 and $\boldsymbol{\alpha}$ specifies a linear combination of these. Consequently, we have $n_2^r \leq 6$ for three-dimensional problems, and $n_2^r \leq 3$ for two-dimensional problems. Substituting (3.2) into (2.7) leads to

$$\begin{aligned}(\mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{B}_1^T + \mathbf{B}_2\mathbf{K}_2^+\mathbf{B}_2^T)\boldsymbol{\lambda} &= -\mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{f}_1 + \mathbf{B}_2(\mathbf{K}_2^+\mathbf{f}_2 + \mathbf{R}_2\boldsymbol{\alpha})\\ \mathbf{u}_1 &= \mathbf{K}_1^{-1}(\mathbf{f}_1 + \mathbf{B}_1^T\boldsymbol{\lambda})\\ \mathbf{u}_2 &= \mathbf{K}_2^+(\mathbf{f}_2 - \mathbf{B}_2^T\boldsymbol{\lambda}) + \mathbf{R}_2\boldsymbol{\alpha}.\end{aligned} \tag{3.3}$$

The following points should be noted.

1. Because $\mathbf{B}_j$ is a boolean operator, the result of its application to a matrix or vector quantity should be interpreted as an extraction process rather than a matrix-matrix or matrix-vector product. For example, $\mathbf{B}_2\mathbf{R}_2$ is the restriction of the null space basis $\mathbf{R}_2$ of $\mathbf{K}_2$ to the interface unknowns. In the sequel we adopt the notation:

$$\mathbf{R}_2^I = \mathbf{B}_2\mathbf{R}_2.$$

2. The generalized inverse $\mathbf{K}_2^+$ does not need to be explicitly computed. For a given input vector $\mathbf{v}$, the output vector $\mathbf{K}_2^+\mathbf{v}$ and the null space basis $\mathbf{R}_2$ can be obtained at almost the same computational cost as the response vector $\mathbf{K}_1^{-1}\mathbf{v}$, where $\mathbf{K}_1$ is nonsingular (See Appendix A).
3. System (3.3) is underdetermined. Both $\boldsymbol{\lambda}$ and $\boldsymbol{\alpha}$ need to be determined before $\mathbf{u}_1$ and $\mathbf{u}_2$ can be found, but only three equations are available so far.

Since $\mathbf{K}_2$ is symmetric, the singular equation (3.1) admits at least one solution if and only if the right-hand side $(\mathbf{f}_2 - \mathbf{B}_2^T\boldsymbol{\lambda})$ has no component in the null space of $\mathbf{K}_2$. This can be expressed as

$$\mathbf{R}_2^T(\mathbf{f}_2 - \mathbf{B}_2^T\boldsymbol{\lambda}) = 0. \tag{3.4}$$

The above orthogonality condition provides the missing equation for the complete solution of (3.3). Combining (3.3) and (3.4) yields after some algebraic manipulations:

$$
\begin{aligned}
\begin{bmatrix} \mathbf{F}_I & -\mathbf{R}_2^I \\ -\mathbf{R}_2^{I^T} & \mathbf{O} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \boldsymbol{\alpha} \end{bmatrix} &= \begin{bmatrix} \mathbf{B}_2\mathbf{K}_2^+\mathbf{f}_2 - \mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{f}_1 \\ -\mathbf{R}_2^T\mathbf{f}_2 \end{bmatrix} \\
\mathbf{u}_1 &= \mathbf{K}_1^{-1}(\mathbf{f}_1 + \mathbf{B}_1^T\boldsymbol{\lambda}) \\
\mathbf{u}_2 &= \mathbf{K}_2^+(\mathbf{f}_2 - \mathbf{B}_2^T\boldsymbol{\lambda}) + \mathbf{R}_2\boldsymbol{\alpha}, \quad \text{where} \\
\mathbf{F}_I &= (\mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{B}_1^T + \mathbf{B}_2\mathbf{K}_2^+\mathbf{B}_2^T)
\end{aligned}
\tag{3.5}
$$

Clearly, $\mathbf{F}_I$ is symmetric positive and $\mathbf{R}_2^I$ has full column rank. Therefore, the system of equations in $(\boldsymbol{\lambda}, \boldsymbol{\alpha})$ is symmetric and nonsingular. It admits a unique solution $(\boldsymbol{\lambda}, \boldsymbol{\alpha})$ which uniquely determines $\mathbf{u}_1$ and $\mathbf{u}_2$.

It is important to note that since $n_2^r \leq 6$, systems (3.5) and (2.7) have almost the same size. For an arbitrary number of subdomains N_s, of which N_f are floating, the additional number of equations introduced by the handling of local singularities is bounded by $6N_f$. For large-scale problems and relatively coarse mesh partitions, this number is a very small fraction of the size of the global system. On the other hand, if a given tearing process does not result in any floating subdomain, $\boldsymbol{\alpha}$ is zero and the systems of equations (3.5) and (2.7) are identical.

Next, we present a numerical algorithm for the solution of (3.5).

4. A preconditioned conjugate projected gradient algorithm. Here we focus on the solution of the nonsingular system of equations:

$$
\begin{aligned}
\begin{bmatrix} \mathbf{F}_I & -\mathbf{R}_2^I \\ -\mathbf{R}_2^{I^T} & \mathbf{O} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \boldsymbol{\alpha} \end{bmatrix} &= \begin{bmatrix} \mathbf{B}_2\mathbf{K}_2^+\mathbf{f}_2 - \mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{f}_1 \\ -\mathbf{R}_2^T\mathbf{f}_2 \end{bmatrix}, \quad \text{where} \\
\mathbf{F}_I &= \mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{B}_1^T + \mathbf{B}_2\mathbf{K}_2^+\mathbf{B}_2^T.
\end{aligned}
\tag{4.1}
$$

We seek an efficient solution algorithm that does not require the explicit assembly of $\mathbf{F}_I$.

Clearly, the nature of $\mathbf{F}_I$ makes the solution of (4.1) inadequate by any technique that requires this submatrix explicitly. This implies that a direct method or an iterative method of the SOR type cannot be used. The only efficient method of solving (4.1) in the general sparse case is that of conjugate gradients, because once $\mathbf{K}_1$ and $\mathbf{K}_2$ have been factorized, matrix-vector products of the form $\mathbf{F}_I\mathbf{v}$ can be performed very efficiently using only forward and backward substitutions. Unfortunately, the Lagrangian matrix

$$
\mathbf{L} = \begin{bmatrix} \mathbf{F}_I & -\mathbf{R}_2^I \\ -\mathbf{R}_2^{I^T} & \mathbf{O} \end{bmatrix}
$$

is indefinite so that a straightforward conjugate gradient algorithm cannot be directly applied to the solution of (4.1). However, the conjugate gradient iteration with the *projected* gradient (see, for example, Gill and Murray [7]) can be used to obtain the sought-after solution. In order to introduce the latter solution algorithm, we first note that solving (4.1) is equivalent to solving the equality constraint problem:

$$
\begin{aligned}
&\text{minimize} \quad \Phi(\boldsymbol{\lambda}) = \tfrac{1}{2}\boldsymbol{\lambda}^T\mathbf{F}_I\boldsymbol{\lambda} + (\mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{f}_1 - \mathbf{B}_2\mathbf{K}_2^+\mathbf{f}_2)^T\boldsymbol{\lambda} \\
&\text{subject to} \quad \mathbf{R}_2^{I^T}\boldsymbol{\lambda} = \mathbf{R}_2^T\mathbf{f}_2.
\end{aligned}
\tag{4.2}
$$

Since $\mathbf{F}_I$ is symmetric positive, a conjugate gradient algorithm is most suitable for computing the unique solution to the unconstrained problem. Therefore, this algorithm will converge to the solution to (4.1) if and only if it can be modified so that the constraint $\mathbf{R}_2^{I^T}\boldsymbol{\lambda} = \mathbf{R}_2^T\mathbf{f}_2$ is satisfied at each iteration. This can be achieved by projecting all the search directions onto the null space of $\mathbf{R}_2^{I^T}$.

The result is a conjugate gradient algorithm with the projected gradient. It is of the form:

Initialize

Pick $\boldsymbol{\lambda}^{(0)}$ such that $\mathbf{R}_2^{I^T}\boldsymbol{\lambda}^{(0)} \;=\; \mathbf{R}_2^T\mathbf{f}_2$

$$\mathbf{r}^{(0)} \;=\; [\mathbf{I} - \mathbf{R}_2^I(\mathbf{R}_2^{I^T}\mathbf{R}_2^I)^{-1}\mathbf{R}_2^{I^T}]\;(\mathbf{B}_2\mathbf{K}_2^{+}\mathbf{f}_2 - \mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{f}_1) \tag{4.3}$$

Iterate $k = 1, 2, \cdots$ until convergence

$$\begin{aligned}
\beta^{(k)} &= \mathbf{r}^{(k-1)^T}\mathbf{r}^{(k-1)}/\mathbf{r}^{(k-2)^T}\mathbf{r}^{(k-2)} \quad (\beta^{(1)} = 0)\\
\mathbf{s}^{(k)} &= \mathbf{r}^{(k-1)} + \beta^{(k)}\mathbf{s}^{(k-1)} \quad (\mathbf{s}^{(1)} = \mathbf{r}^{(0)})\\
\gamma^{(k)} &= \mathbf{r}^{(k-1)^T}\mathbf{r}^{(k-1)}/\mathbf{s}^{(k)^T}\mathbf{F}_I\mathbf{s}^{(k)}\\
\boldsymbol{\lambda}^{(k)} &= \boldsymbol{\lambda}^{(k-1)} + \gamma^{(k)}\mathbf{s}^{(k)}\\
\mathbf{r}^{(k)} &= \mathbf{r}^{(k-1)} - \gamma^{(k)}[\mathbf{I} - \mathbf{R}_2^I(\mathbf{R}_2^{I^T}\mathbf{R}_2^I)^{-1}\mathbf{R}_2^{I^T}]\;\mathbf{F}_I\mathbf{s}^{(k)}.
\end{aligned} \tag{4.4}$$

Given the projection operator $[\mathbf{I} - \mathbf{R}_2^I(\mathbf{R}_2^{I^T}\mathbf{R}_2^I)^{-1}\mathbf{R}_2^{I^T}]$, it is easily checked that the projected residual vector $\mathbf{r}^{(k)}$ is parallel to the null space of $\mathbf{R}_2^{I^T}$ at each iteration k, and that $\mathbf{R}_2^{I^T}\mathbf{s}^{(k)} = 0$ for all $k \geq 1$. Therefore, $\mathbf{R}_2^{I^T}\boldsymbol{\lambda}^{(k)} = \mathbf{R}_2^{I^T}\boldsymbol{\lambda}^{(0)}$, which indicates that the approximate solution $\boldsymbol{\lambda}^{(k)}$ satisfies the linear equality constraint of problem (4.1) at each iteration k. It is also important to note that within each iteration, only one projection is performed. This projection is relatively inexpensive since the only implicit computations that are involved are associated with the matrix $\mathbf{R}_2^{I^T}\mathbf{R}_2^I$, which is at most 6×6. This matrix is factored once, before the first iteration begins. Except for this small overhead, algorithm (4.4) above has the same computational cost as the regular conjugate gradient method. After $\boldsymbol{\lambda}$ is found, $\boldsymbol{\alpha}$ is computed as

$$\boldsymbol{\alpha} \;=\; (\mathbf{R}_2^{I^T}\mathbf{R}_2^I)^{-1}(\mathbf{F}_I\boldsymbol{\lambda} - \mathbf{B}_2\mathbf{K}_2^{+}\mathbf{f}_2 + \mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{f}_1).$$

As in the case of the conjugate gradient method, the conjugate projected gradient algorithm is most effective when applied to the preconditioned system of equations. It should be noted that even in the presence of floating subdomains, only $\mathbf{F}_I$ needs to be preconditioned and not the global Lagrangian matrix $\mathbf{L}$. In the case of two subdomains, $\mathbf{F}_I$ can be written in matrix form as

$$\mathbf{F}_I \;=\; [\mathbf{B}_1 \quad \mathbf{B}_2]\begin{bmatrix}\mathbf{K}_1^{-1} & \mathbf{O}\\ \mathbf{O} & \mathbf{K}_2^{-1}\end{bmatrix}\begin{bmatrix}\mathbf{B}_1^T\\ \mathbf{B}_2^T\end{bmatrix} \tag{4.5}$$

where $\mathbf{K}_j^{-1}$, $j = 1, 2$, is replaced by $\mathbf{K}_j^{+}$ if Ω_j is a floating subdomain. The objective is to find an approximate inverse $\mathbf{P}_I^{-1}$ of $\mathbf{F}_I$ that (a) does not need to be explicitly assembled (especially since $\mathbf{F}_I$ is not explicitly assembled), and (b) is amenable to parallel computations. The matrix $\mathbf{P}$ is then the preconditioner. Equation (4.5) above suggests the following choice for $\mathbf{P}_I^{-1}$:

$$\mathbf{P}_I^{-1} = [\mathbf{B}_1 \quad \mathbf{B}_2] \begin{bmatrix} \mathbf{K}_1 & \mathbf{O} \\ \mathbf{O} & \mathbf{K}_2 \end{bmatrix} \begin{bmatrix} \mathbf{B}_1^T \\ \mathbf{B}_2^T \end{bmatrix}. \tag{4.6}$$

At each iteration k, the preconditioned conjugate projected gradient algorithm involves the solution of an auxiliary system of the form

$$\mathbf{P}_I \mathbf{z}^{(k)} = \mathbf{r}^{(k)} \tag{4.7}$$

where $\mathbf{r}^{(k)}$ is the residual at the kth iteration. The particular choice of $\mathbf{P}_I^{-1}$ given in (4.6) offers the advantage of solving (4.7) explicitly without the need for any intermediate factorization. For computational efficiency, $\mathbf{P}_I^{-1}$ is implemented as

$$\mathbf{P}_I^{-1} = \mathbf{K}_1^I + \mathbf{K}_2^I \tag{4.8}$$

where $\mathbf{K}_1^I$ and $\mathbf{K}_2^I$ are the traces of $\mathbf{K}_1$ and $\mathbf{K}_2$ on Γ_I. Clearly, with this choice for the preconditioner, the auxiliary system (4.7) is "cheap," easy to solve, and perfectly parallelizable on both local and shared-memory parallel architectures.

5. Parallel characteristics of the proposed method. Like most domain decomposition-based algorithms, the proposed method is perfectly suitable for parallel processing. If every subdomain Ω_j is assigned to an individual processor p_j, all local finite element computations can be performed in parallel. These include forming and assembling the matrix $\mathbf{K}_j$ and the vector $\mathbf{f}_j$, factoring $\mathbf{K}_j$ and eventually computing the null space basis $\mathbf{R}_j$, as well as backsolving for $\mathbf{u}_j$ after $\boldsymbol{\lambda}$ and $\boldsymbol{\alpha}$ have been determined. The conjugate projected gradient algorithm described in §4 is also amenable to parallel processing. For example, the matrix-vector product $\mathbf{F}_I\mathbf{s}^{(k)}$ can be computed in parallel by assigning to each processor p_j the task of evaluating

$$\mathbf{y}_j^{(k)} = \mathbf{B}_j^k \mathbf{K}_j^{-1} \mathbf{B}_j^{k^T} \mathbf{s}_j^{(k)},$$

and exchanging $\mathbf{y}_j^{(k)}$ with the processors assigned to neighboring subdomains in order to assemble the global result. Interprocessor communication is required only during the solution of the interface problem (4.1) and takes place exclusively among neighboring processors during the assembly of the subdomain results.

At this point, we stress that the parallel solution method developed herein requires inherently less interprocessor communication than other domain decomposition-based parallel algorithms. As mentioned earlier, interprocessor communication within the proposed method occurs only during the solution of the interface problem (4.1). The reader should trace back this problem as well as the presence of the Lagrange multipliers to the integral quantity

$$(v_i - v_j, \lambda)_{\Gamma_{I_{ij}}} = \int_{\Gamma_{I_{ij}}} \lambda(v_i - v_j)\, d\Gamma \tag{5.1}$$

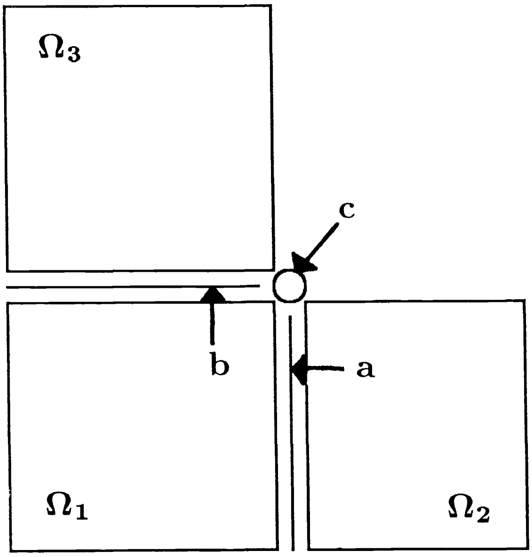


FIG. 4. *An L-shaped three-subdomain problem with a crosspoint.*

where $\Gamma_{I_{ij}}$ is the interface between subdomains Ω_i and Ω_j. If $\Gamma_{I_{ij}}$ has a zero measure, then $(v_i - v_j, \lambda)_{\Gamma_{I_{ij}}} = 0$ and no exchange of information is needed between Ω_i and Ω_j. Therefore the subdomains that interconnect along one edge in three-dimensional problems and those that interconnect along one vertex in both two- and three-dimensional problems do not require any interprocessor communication. This is unlike conventional domain decomposition methods (Schur-related methods), where assembly is required at any intersection point between two subdomains. This result can also be derived from a mechanical interpretation of relation (5.1), as demonstrated by the following example. Consider the case of an L-shaped domain uniformly partitioned into three subdomains Ω_1, Ω_2, and Ω_3 meeting at a "crosspoint" (Fig. 4). For the sake of clarity, in the following analysis the crosspoint c of interfaces a and b is treated as a separate entity. In the practical implementation of the method, it is integrated with every interface to which it belongs. Three discrete forces λ_i corresponding to the three subdomains meeting at the crosspoint c are introduced at each of its degrees of freedom. For simplicity, it is assumed that the elements across the interfaces are compatible so that the forces from both sides of an interface are equal in magnitude and opposite in direction. Moreover, for this problem, if $\mathbf{B}^k_{ij}$ denotes the boolean matrix that identifies the nodes which are shared by Ω_i and Ω_j along the interface k, the decomposition shown in Fig. 4 is such that $\mathbf{B}^k_{ij} = \mathbf{B}^k_{ji}$, and $\mathbf{B}^c_{12} = \mathbf{B}^c_{23} = \mathbf{B}^c_{13}$. Therefore if every subdomain is "glued" to every one of its neighbors, the subdomain equations of equilibrium for this problem can be written as

$$\begin{aligned}
\mathbf{K}_1\mathbf{u}_1 &= \mathbf{f}_1 + {\mathbf{B}^a_{12}}^T\boldsymbol{\lambda}^a_{12} + {\mathbf{B}^c_{12}}^T\boldsymbol{\lambda}^c_{12} + {\mathbf{B}^b_{13}}^T\boldsymbol{\lambda}^b_{12} + {\mathbf{B}^c_{13}}^T\boldsymbol{\lambda}^c_{13} \\
\mathbf{K}_2\mathbf{u}_2 &= \mathbf{f}_2 - {\mathbf{B}^a_{12}}^T\boldsymbol{\lambda}^a_{12} - {\mathbf{B}^c_{12}}^T\boldsymbol{\lambda}^c_{12} + {\mathbf{B}^c_{23}}^T\boldsymbol{\lambda}^c_{23} \qquad (5.2) \\
\mathbf{K}_3\mathbf{u}_3 &= \mathbf{f}_3 - {\mathbf{B}^b_{13}}^T\boldsymbol{\lambda}^b_{13} - {\mathbf{B}^c_{13}}^T\boldsymbol{\lambda}^c_{13} - {\mathbf{B}^c_{23}}^T\boldsymbol{\lambda}^c_{23}.
\end{aligned}$$

However, it can be argued that since Ω_2 and Ω_3 intersect at a single point, whenever Ω_1 is glued to Ω_2 and Ω_3, then Ω_2 is automatically glued to Ω_3. This holds only if there exist a set of discrete interface forces $\boldsymbol{\mu}_{ij}^k$ such that the subdomain equations of equilibrium can be simplified to

$$\begin{aligned}\mathbf{K}_1\mathbf{u}_1 &= \mathbf{f}_1 + \mathbf{B}_{12}^{a\ T}\boldsymbol{\mu}_{12}^a + \mathbf{B}_{12}^{c\ T}\boldsymbol{\mu}_{12}^c + \mathbf{B}_{13}^{b\ T}\boldsymbol{\mu}_{12}^b + \mathbf{B}_{13}^{c\ T}\boldsymbol{\mu}_{13}^c \\ \mathbf{K}_2\mathbf{u}_2 &= \mathbf{f}_2 - \mathbf{B}_{12}^{a\ T}\boldsymbol{\mu}_{12}^a - \mathbf{B}_{12}^{c\ T}\boldsymbol{\mu}_{12}^c \\ \mathbf{K}_3\mathbf{u}_3 &= \mathbf{f}_3 - \mathbf{B}_{13}^{b\ T}\boldsymbol{\mu}_{13}^b - \mathbf{B}_{13}^{c\ T}\boldsymbol{\mu}_{13}^c.\end{aligned} \tag{5.3}$$

Comparing equations (5.2) and (5.3) reveals that:

$$\begin{aligned}\boldsymbol{\mu}_{12}^a &= \boldsymbol{\lambda}_{12}^a \\ \boldsymbol{\mu}_{12}^c &= \boldsymbol{\lambda}_{12}^c \\ \boldsymbol{\mu}_{13}^c &= \boldsymbol{\lambda}_{13}^c \\ \boldsymbol{\mu}_{13}^c &= \boldsymbol{\lambda}_{13}^c + \boldsymbol{\lambda}_{23}^c \\ \boldsymbol{\mu}_{12}^c &= \boldsymbol{\lambda}_{12}^c - \boldsymbol{\lambda}_{23}^c\end{aligned} \tag{5.4}$$

which clearly demonstrates that any two subdomains that intersect at a single node do not need to be explicitly assembled.

Therefore, for a three-dimensional regular mesh that is partitioned into subcubes, the proposed method requires that each subdomain communicate with at most six neighboring subdomains (since a cube has only six faces), while the parallel Schur methods necessitate that each subdomain communicate with up to 26 neighbors (Fig. 5). For the latter method, one could of course group vertex- and edge-associated messages with the face-associated ones and perform some type of fast communication sweeps as described in [11]. However, that option creates an additional burden for the computer implementation and may not be feasible for arbitrary subdomain interconnections. Clearly, this communication characteristic makes the proposed parallel solution method very attractive for a multiprocessor with a distributed memory such as a hypercube. Indeed, the advantages of the method for this family of parallel processors are twofold: (a) the number of message-passing is dramatically reduced, which reduces the overhead due to communication start-up, and (b) the complexity of the communication requirements is improved so that an optimal mapping of the processors onto the subdomains can be reached (Bokhari [1], Farhat [4]); therefore the elapsed time for a given message is improved. Both enhancements (a) and (b) reduce the communication overhead of the parallel solution algorithm in a synergistic manner. This algorithmic feature of the proposed method is still desirable for shared-memory multiprocessors because it eases the assembly process during the interface solution and makes the latter more manageable. It is not, however, as critical for the performance as it is for local memory multiprocessors.

Finally, it should be noted that domain decomposition methods in general exhibit a larger degree of parallelism than parallel direct solvers. The efficiency of the latter is governed by the bandwidth of the given finite element system of equations. If the bandwidth is not large enough, interprocessor communication and/or process synchronization can dominate the work done in parallel by each processor. This is true not only for multiprocessors with a message-passing system, but also for supervector multiprocessors with a shared memory such as the CRAY systems, where synchronization

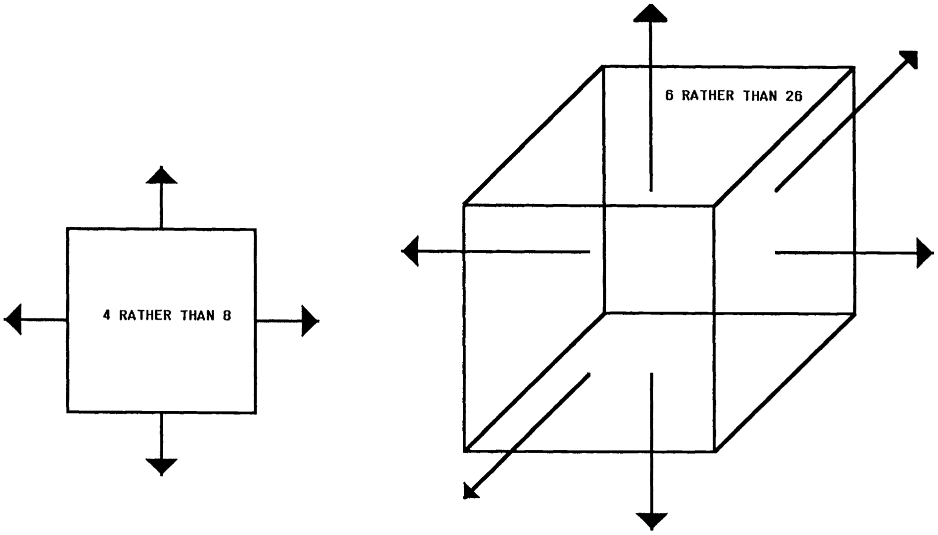


FIG. 5. *Reduced interprocessor communication patterns for two- and three-dimensional regular mesh partitions.*

primitives are rather expensive. Therefore, the computational method described in this paper should be seriously considered for large-scale problems with a relatively small or medium bandwidth. These problems are typically encountered in the finite element analysis of large space structures that are often elongated and include only a few elements along one or two directions (Farhat [5]). The method is also recommended for problems where the storage requirements of direct solvers cannot be met.

6. Applications and performance assessment. For conventional domain decomposition methods, the interface problem can be written as

$$[(\mathbf{K}_{II}^{(1)} - \mathbf{K}_{1I}^T\mathbf{K}_{11}^{-1}\mathbf{K}_{1I}) + (\mathbf{K}_{II}^{(2)} - \mathbf{K}_{2I}^T\mathbf{K}_{22}^{-1}\mathbf{K}_{2I})]\mathbf{u}_I = \mathbf{f}_{II} - \mathbf{K}_{1I}^T\mathbf{K}_{11}^{-1}\mathbf{f}_{11} - \mathbf{K}_{2I}^T\mathbf{K}_{22}^{-1}\mathbf{f}_{22} \tag{6.1}$$

where $\mathbf{K}_{II}^{(1)}$, $\mathbf{K}_{II}^{(2)}$, $\mathbf{K}_{11}$, and $\mathbf{K}_{22}$ are the stiffness matrices associated, respectively, with the interface nodes and the interior nodes of subdomains Ω_1 and Ω_2, and $\mathbf{K}_{1I}$ and $\mathbf{K}_{2I}$ are the coupling stiffnesses between, respectively, Ω_1 and Γ_I, and Ω_2 and Γ_I (see, for example, [6] for further details). Because the left-hand side of (6.1) is known as Schur's complement, we have been referring to these subdomain-based algorithms as Schur methods. A standard conjugate gradient algorithm may be used for solving (6.1). Here we compare for performance the above subdomain-based Schur method with the hybrid algorithm developed in this paper. While the preconditioner (4.8) is selected for the hybrid version, two different preconditioners are considered for Schur's method:

(a) diagonal scaling, and (b) subdomain scaling, that is,

$$\mathbf{P}^{-1} = (\mathbf{K}_{II}^{(1)} - \mathbf{K}_{1I}^T\mathbf{K}_{11}^{-1}\mathbf{K}_{1I})^{-1} + (\mathbf{K}_{II}^{(2)} - \mathbf{K}_{2I}^T\mathbf{K}_{22}^{-1}\mathbf{K}_{2I})^{-1}.$$

We focus on the static analysis on a 32-processor iPSC hypercube of a three-dimensional mechanical joint subjected to internal pressure loading. The finite element discretization of this mechanical joint using eight node brick elements is shown in Fig. 6. The mesh has 9912 elements and 29,654 degrees of freedom. Figure 7 displays its decomposition in 32 subdomains. For the hybrid formulation, most of these subdomains are floating and therefore induce each a singular local operator.

The obtained numerical results are reported in Figs. 8–10. For each algorithm, three curves are reported, which correspond to monitoring convergence with three different measures: (A) the global force relative residual, (B) the displacement relative variation, and (C) the interface relative residual.

It is interesting to note that for both algorithms, convergence using the relative global residual is harder to achieve than convergence using the relative variation of the solution. This is because the problem is ill conditioned. The interface residual is closer to the global solution variation in the hybrid method, while it is closer to the global residual in the Schur method. This is because in the former case, the interface problem is formulated in the functional space of the problem's solution derivative, while in the latter case, it is formulated in the same functional space as the problem's solution. Clearly, a stopping criterion should be based on the global relative residual. For a tolerance of 10^{-3} (typical for nonlinear problems), the preconditioned hybrid method requires fewer iterations for convergence than the diagonally preconditioned Schur method. On the other hand, the subdomain preconditioned Schur method converges faster than the other algorithms. However, it is still slower than the preconditioned hybrid method, as reported in Table 1, where $T_{msg}/itr.$, T_p, and SP denote, respectively, the time elapsed in message-passing per iteration, the total parallel time, and the overall parallel speedup. Here, the parallel speedup SP is defined as the ratio between the total time using one processor and the total time using p processors, for the same number of subdomains. It is computed as

$$SP = \frac{p}{1 + p\frac{T_{cmn}}{T_{cmp}}}$$

where p denotes the number of processors, and T_{cmn} and T_{cmp} denote, respectively, the total communication time and total computation time associated with the algorithm. Basically, SP measures how well the algorithm parallelizes on a message passing multiprocessor. It does not however reflect the intrinsic computational performance of the algorithm; the total parallel time does. Note also that the above expression of SP overcomes the practical difficulties associated with running a large problem on a single node of a hypercube.

TABLE 1

Performance results on iPSC/2

Mechanical joint, brick elements, 32 subdomains,
9912 elements, 29654 degrees of freedom
Convergence criterion: || relative global residual || < 10^{-3}

$T_{msg}/itr.$ D.S.*	$T_{msg}/itr.$ S.S.†	$T_{msg}/itr.$ Hyb.	T_p D.S.	T_p S.S.	T_p Hyb	SP D.S.	SP S.S.	SP Hyb.
17.9 m.s.	17.9 m.s.	5.4 m.s.	1103 s.	1322 s.	917 s.	24.0	23.9	28.8

* D.S. = Dia. Schur

† S.S. = Sub. Schur

All methods achieve excellent speedup. This is generally true for all balanced algorithms that require message-passing only between neighboring processors. However, for this problem, the hybrid algorithm is faster and exhibits a 20 percent higher

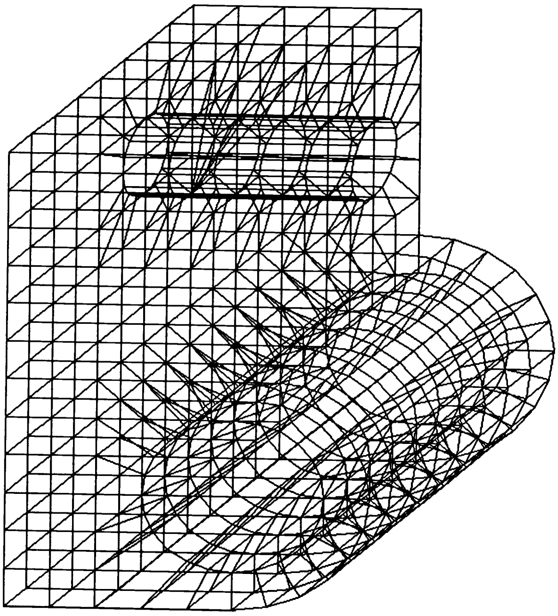


FIG. 6. *Finite element discretization of a mechanical joint.*

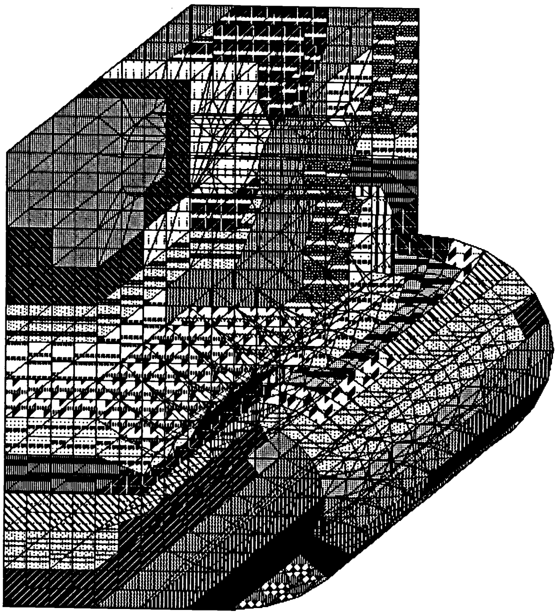


FIG. 7. *Decomposition in* 32 *subdomains.*

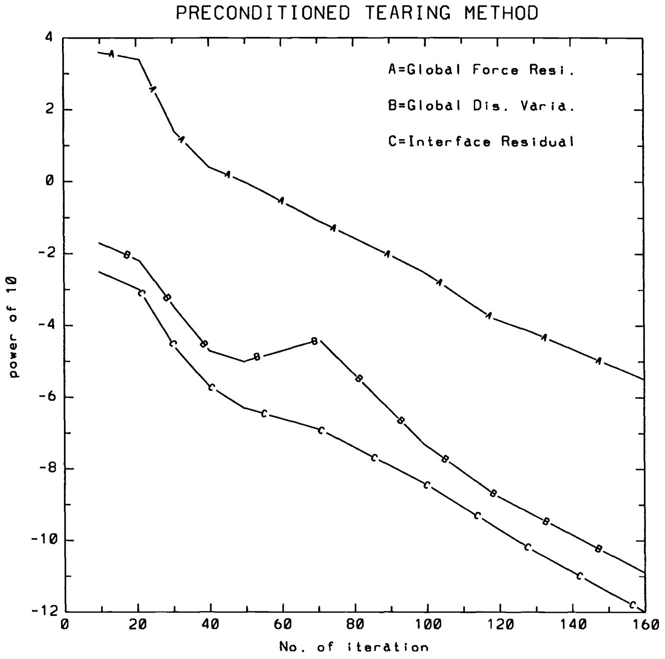


FIG. 8. *Preconditioned hybrid method.*

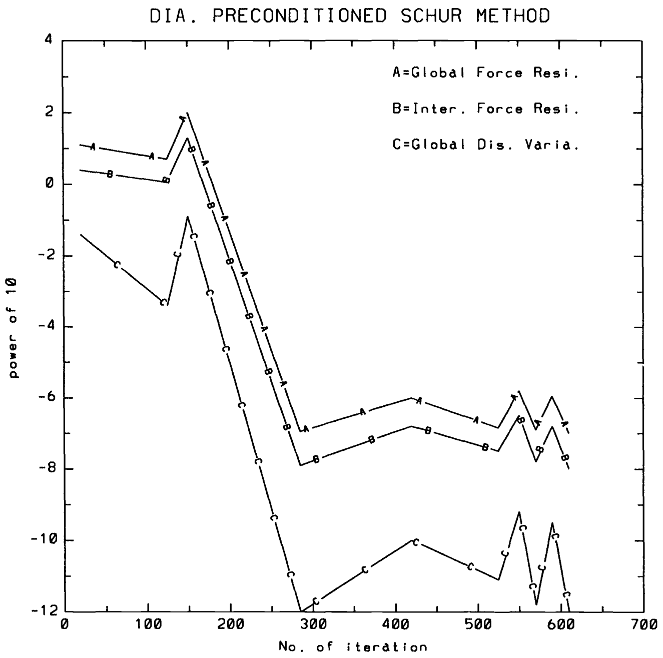


FIG. 9. *Diagonal-preconditioned Schur method.*

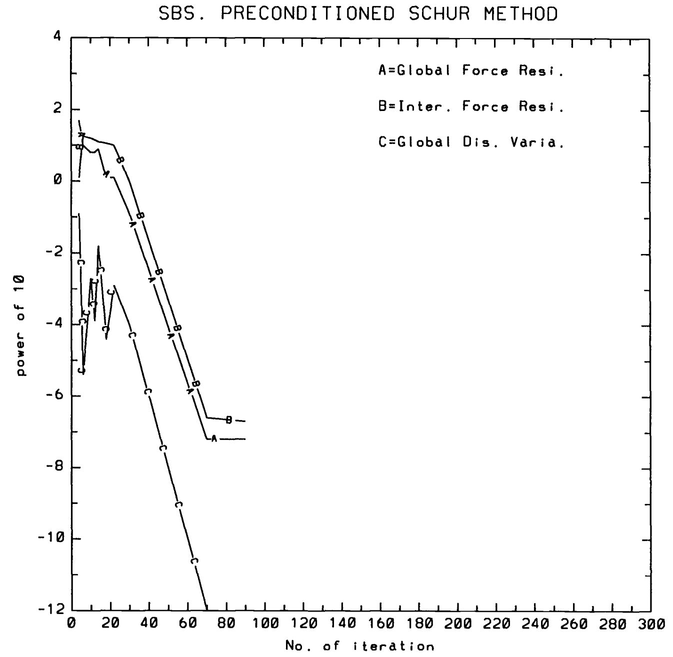


FIG. 10. *Subdomain-preconditioned Schur method.*

speedup than the conventional Schur algorithms, for which the time elapsed in interprocessor communication is 3.31 times higher. Again, because it avoids interprocessor communication along the edges and corners of the subdomains, the hybrid algorithm requires fewer message-passing startups which, in the case of short messages, are known to account for the largest portion of the time elapsed in interprocessor communication on the iPSC.

7. Conclusion. A novel subdomain-based algorithm for the parallel finite element solution of selfadjoint elliptic partial differential equations is presented. The spatial domain is partitioned into a set of totally disconnected subdomains, each assigned to an individual processor. Lagrange multipliers are introduced to enforce compatibility at the interface nodes. In the static case, each floating subdomain induces a local singularity that is resolved in two phases. First, the null space components are eliminated in parallel from each local problem and a direct scheme is applied concurrently to all subdomains in order to recover each partial local solution. Next, the contributions of these components are related to the Lagrange multipliers through an orthogonality condition. A parallel conjugate *projected* gradient algorithm is developed for the solution of the coupled system of local null space components and Lagrange multipliers, which completes the solution of the problem. When implemented on local memory multiprocessors, this proposed method requires less interprocessor communication than other conventional domain decomposition methods. It is also suitable for parallel/vector computers with shared memory. Example applications from structural mechanics are reported on the iPSC hypercube. Measured performance results illustrate the advantages of the proposed method.

Appendix A. Solving a consistent singular system $\mathbf{K}_j\mathbf{u}_j = \mathbf{f}_j$. For completeness, we include in this appendix a derivation of the solution of a consistent singular system of equations. In this work, such a system arises in every floating subdomain Ω_j and takes the form

$$\mathbf{K}_j \mathbf{u}_j = \mathbf{f}_j \tag{A.1}$$

where $\mathbf{K}_j$ is the $(n_j^s + n_j^I) \times (n_j^s + n_j^I)$ stiffness matrix associated with Ω_j, and $\mathbf{u}_j$ and $\mathbf{f}_j$ are the corresponding displacement and forcing vectors. If Ω_j has n_j^r rigid body modes, $\mathbf{K}_j$ is rank n_j^r deficient. Provided that $\mathbf{f}_j$ is orthogonal to the null space of $\mathbf{K}_j$, the singular system (A.1) is consistent and admits a general solution of the form

$$\mathbf{u}_j = \mathbf{K}_j^+ \mathbf{f}_j + \mathbf{R}_j \boldsymbol{\alpha} \tag{A.2}$$

where $\mathbf{K}_j^+$ is a generalized inverse of $\mathbf{K}_j$ — that is, $\mathbf{K}_j^+$ verifies $\mathbf{K}_j \mathbf{K}_j^+ \mathbf{K}_j = \mathbf{K}_j$, $\mathbf{R}_j$ is a basis of the null space of $\mathbf{K}_j$ — that is, $\mathbf{R}_j$ stores the n_j^r rigid body modes of Ω_j, and $\boldsymbol{\alpha}$ is a vector of length n_j^r containing arbitrary real coefficients.

Computing the rigid body modes. Let the superscripts p and r denote, respectively, a principal and a redundant quantity. The singular stiffness matrix $\mathbf{K}_j$ is partitioned as

$$\mathbf{K}_j = \begin{bmatrix} \mathbf{K}_j^{pp} & \mathbf{K}_j^{pr} \\ \mathbf{K}_j^{pr^T} & \mathbf{K}_j^{rr} \end{bmatrix} \tag{A.3}$$

where $\mathbf{K}_j^{pp}$ has full rank equal to $n_j^s + n_j^I - n_j^r$. If $\mathbf{R}_j$ is defined as

$$\mathbf{R}_j = \begin{bmatrix} -[\mathbf{K}_j^{pp}]^{-1} \mathbf{K}_j^{pr} \\ \mathbf{I}_{n_j^r} \end{bmatrix} \tag{A.4}$$

where $\mathbf{I}_{n_j^r}$ is the $n_j^r \times n_j^r$ identity matrix, then $\mathbf{R}_j$ satisfies

$$\mathbf{K}_j \mathbf{R}_j = 0.$$

Moreover, $\mathbf{I}_{n_j^r}$ has full column rank and so does $\mathbf{R}_j$. Therefore, the n_j^r columns of $\mathbf{R}_j$ as defined in (A.4) form a basis of the null space of $\mathbf{K}_j$.

Computing $\mathbf{K}_j^+ \mathbf{f}_j$. The partitioning of the singular matrix $\mathbf{K}_j$ defined in (A.3) implies that

$$\mathbf{K}_j^{rr} = \mathbf{K}_j^{pr^T} [\mathbf{K}_j^{pp}]^{-1} \mathbf{K}_j^{pr}. \tag{A.5}$$

Using the above identity, it can be easily checked that the matrix $\mathbf{K}_j^+$ defined as

$$\mathbf{K}_j^+ = \begin{bmatrix} [\mathbf{K}_j^{pp}]^{-1} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} \end{bmatrix}$$

is a generalized inverse of $\mathbf{K}_j$. Therefore, a solution of the form $\mathbf{K}_j^+ \mathbf{f}_j$ can be also written as

$$\mathbf{u}_j = \mathbf{K}_j^+ \mathbf{f}_j = \begin{bmatrix} [\mathbf{K}_j^{pp}]^{-1} \mathbf{f}_j^p \\ \mathbf{O} \end{bmatrix}.$$

In practice, $\mathbf{K}_j$ cannot be explicitly rearranged as in (A.3). Rather, the following should be implemented when $\mathbf{K}_j$ is stored in skyline form. A zero pivot that is encountered during the factorization process of $\mathbf{K}_j$ corresponds to a redundant equation which needs to be labeled and removed from the system. The zero pivot is set to one, the reduced column above it is copied into an extra right-hand side — this corresponds to a forward reduction with $\mathbf{K}_j^{pr}\mathbf{u}_j^r$ as right-hand side, and the coefficients in the skyline corresponding to that pivotal equation are set to zero. At the end of the factorization process, the nonlabeled equations define the full rank matrix $\mathbf{K}_j^{pp}$. The backward substitution is modified to operate also on the n_j^r extra right-hand sides in order to recover $\mathbf{u}_j^p = -[\mathbf{K}_j^{pp}]^{-1}\mathbf{K}_j^{pr}\mathbf{u}_j^r$.

The above procedure for solving a consistent singular system of equations has almost the same computational complexity as the solution of a nonsingular one.

REFERENCES

[1] S. H. Bokhari, *On the mapping problem*, IEEE Trans. Comput., C-30 (1981), pp. 207–214.

[2] Q. V. Dihn, R. Glowinski, and J. Periaux, *Solving elliptic problems by domain decomposition methods with applications*, in Elliptic Problem Solvers II, A. Schoenstadt, ed., Academic Press, London, 1984.

[3] M. R. Dorr, *Domain decomposition via Lagrange multipliers*, UCRL-98532, Lawrence Livermore National Laboratory, Livermore, CA, 1988.

[4] C. Farhat, *On the mapping of massively parallel processors onto finite element graphs*, Computers & Structures, 32 (1989), pp. 347–354.

[5] ———, *Which parallel finite element algorithm for which architecture and which problem*, in Computational Structural Mechanics and Multidisciplinary Optimization, R. V. Grandhi, W. J. Stroud, and V. B. Venkayya, eds., ASME, AD-Vol. 16, 1989, pp. 35–43.

[6] C. Farhat and E. Wilson, *A new finite element concurrent computer program architecture*, Int. J. Num. Meth. Eng., 24 (1987), pp. 1771–1792.

[7] P. E. Gill and W. Murray, *Numerical Methods for Constrained Optimization*, P. E. Gill and W. Murray, eds., Academic Press, London, 1974, pp. 132–135.

[8] T. H. H. Pian, *Finite element formulation by variational principles with relaxed continuity requirements*, in The Mathematical Foundation of the Finite Element Method with Applications to Partial Differential Equations, Part II, A. K. Aziz, ed., Academic Press, London, 1972, pp. 671–687.

[9] F. X. Roux, *Test on parallel machines of a domain decomposition method for a composite three dimensional structural analysis problem*, Proc. International Conference on Supercomputing, Saint Malo, France, 1988, pp. 273–283.

[10] ———, *A parallel solver for the linear elasticity equations on a composite beam*, in Proceedings of the Second International Conference on Domain Decomposition Methods, T. F. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989.

[11] Y. Saad and M. H. Schulz, *Data communication in hypercubes*, J. Parallel and Distributed Computing, 5 (1988).

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 1, pp. 397–410, January 1992

021

DOMAIN DECOMPOSITION METHODS FOR PROBLEMS WITH PARTIAL REFINEMENT*

JAMES H. BRAMBLE[†], RICHARD E. EWING[‡], ROSSEN R. PARASHKEVOV[‡], AND JOSEPH E. PASCIAK[§]

Abstract. In this paper, a flexible mesh refinement strategy for the approximation of solutions of elliptic boundary value problems is considered. The main purpose of the paper is the development of preconditioners for the resulting discrete system of algebraic equations. These techniques lead to efficient computational procedures in serial as well as parallel computing environments. The preconditioners are based on overlapping domain decomposition and involve solving (or preconditioning) subproblems on regular subregions. It is proven that the iteration schemes converge to the discrete solution at a rate which is independent of the mesh parameters in the case of two spatial dimensions. The estimates proved for the iterative convergence rate in three dimensions are somewhat weaker. The results of numerical experiments illustrating the theory are also presented.

Key words. second-order elliptic equation, domain decomposition, overlapping domain decomposition, local mesh refinement, partial refinement, overlapping Schwarz methods, preconditioners

AMS(MOS) subject classifications. 65N30, 65F10

1. Introduction.

To provide the required accuracy in many applications involving large scale scientific computation, it becomes necessary to use local mesh refinement techniques. These techniques allow the use of finer meshes in regions of the computational domain where the solution exhibits large gradients. This remains practical only if efficient techniques for the solution of the resulting discrete systems are available. It is the purpose of this paper to provide such techniques. We will give a flexible scheme for refinement as well as develop and analyze effective iterative methods for the solution of the resulting systems of discrete equations.

In this paper, we shall be interested in techniques for problems with refinements which are not quite local. As an example, one might consider a front passing through a two-dimensional domain. In this case, it might be necessary to refine in the neighborhood of the front.

There are a number of ways of developing preconditioned iterative schemes for the discrete systems resulting from local mesh refinement in the literature. Techniques based on nested multilevel spaces are given in [1], [7], [8], [12]. Techniques based on domain decomposition are given in [2], [10], [13], [14]. The analysis presented there implicitly depends on the shape of the the refinement domain, and hence the resulting algorithms may not be as effective with irregularly shaped refinement regions.

*Received by the editors April 5, 1990; accepted for publication (in revised form) October 5, 1990. This manuscript has been authored under contract number DE-AC02-76CH00016 with the U.S. Department of Energy. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes. This work was also supported in part under the National Science Foundation grant DMS84-05352 and by the U.S. Army Research Office through the Mathematical Science Institute, Cornell University. Additional supporters of this work include the Office of Naval Research under contract 0014-88-K-0370 and by the Institute for Scientific Computation at the University of Wyoming through National Science Foundation grant RII-8610680.

[†]Department of Mathematics, Cornell University, Ithaca, New York 14853.

[‡]Mathematics Department, University of Wyoming, Laramie, Wyoming 82071.

[§]Department of Applied Science, Brookhaven National Laboratory, Upton, New York 11973.

These algorithms also require the solution of a subproblem or preconditioner on the refinement regions. We shall provide alternative preconditioned iterative techniques for these problems based on overlapping domain decomposition. Our algorithms are simpler and possibly more effective when implemented since they often lead to preconditioning subproblems defined on either regular subregions or topologically "nice" meshes. The refinement region is the union of the subregions and may be irregularly shaped.

The proposed mesh refinement strategy is important in that it provides a basic approach for implementing dynamic local grid refinement. An example of a refinement strategy involves starting with a uniform coarse grid and refining in small subregions associated with a selected set of coarse-grid vertices. These subregions are allowed to overlap and there are no theoretical restrictions on the resulting refinement region (the union of the subregions). Dynamic refinement is achieved by simply dynamically changing the selected set of coarse-grid vertices.

In addition, the technique can be integrated into existing large scale simulators without a complete redesign of the code. This is because most of the computation involves tasks on either the global coarse grid or the refinement grids associated with the refinement subregions. Choosing the coarse and refinement grid structure to be that already used in the code saves considerable development costs. For example, if one uses regularly structured meshes in the coarse and refinement grids, a substantial part of the resulting algorithm only requires operations on regular grids even though the resulting final approximation space is not regular.

The outline of the remainder of the paper is as follows. In §2, we define some preliminaries and describe the second-order elliptic problems that will be considered. The overlapping domain decomposition algorithms for grids with partial refinement are given in §3. An analysis of the resulting preconditioned algorithms is given in §4. It is shown that the condition number of the preconditioned systems is bounded independently of the mesh parameters for many two-dimensional applications. The results for three dimensions are somewhat weaker and involve logarithms of the mesh parameters. Finally, the results of numerical experiments using these preconditioning techniques are given in §5.

2. The elliptic problem and preliminaries. We shall be concerned with the efficient solution of discrete equations resulting from approximation of second-order elliptic boundary value problems in a polygonal or polyhedral domain Ω contained in Euclidean space R^d, for $d = 2, 3$. We consider the problem of approximating the solution u of

$$\begin{aligned} Lu &= f \qquad \text{in } \Omega, \\ u &= 0 \qquad \text{on } \partial\Omega. \end{aligned} \tag{2.1}$$

Here L is given by

$$Lv = -\sum_{i,j=1}^{d} \frac{\partial}{\partial x_i} a_{ij} \frac{\partial v}{\partial x_j},$$

and $\{a_{ij}(x)\}$ is a uniformly positive definite, bounded, piecewise smooth coefficient matrix on Ω. The corresponding bilinear form is denoted by $A(\cdot,\cdot)$ and is given by

$$A(v,w) = \sum_{i,j=1}^{d} \int_{\Omega} a_{ij} \frac{\partial v}{\partial x_i} \frac{\partial w}{\partial x_j}\, dx, \tag{2.2}$$

and is defined for functions $v,w \in H^1(\Omega)$. Here $H^1(\Omega)$ is the Sobolev space of order one on Ω. We denote the $L^2(\Omega)$ inner product by $(\cdot,\cdot)$. The weak solution u of (2.1) is the function $u \in H_0^1(\Omega)$ satisfying

$$A(u,\varphi) = (f,\varphi) \quad \text{for all } \varphi \in H_0^1(\Omega).$$

Here, $H_0^1(\Omega)$ is the subspace of functions in $H^1(\Omega)$ whose trace vanishes on $\partial\Omega$.

We consider the above model problem for convenience. Many extensions of the techniques to be presented are possible; for example, one could consider equations with lower-order terms and different boundary conditions.

In this paper, we shall deal with various domains. These domains will always be open. The closure of a domain θ will be denoted $\bar{\theta}$. In addition, we shall use various positive constants. These will be denoted by the character C, which will take on different values in different places. However, this constant shall always be independent of the mesh parameters in the approximation schemes.

3. The overlapping algorithms. In this section, we shall define iterative methods for problems with partial refinement based on overlapping domain decomposition. Our goal is to illustrate the technique and analysis and hence, for simplicity, we shall not attempt to provide the most general theorems. Many extensions are possible and can be inferred from the analysis presented.

The analysis given in the following section requires the application of techniques from both the theory of overlapping domain decomposition [9], [11] as well as the standard domain decomposition theory [4], [5]. We first give the setup in the two-dimensional case. We start with a coarse mesh $\cup\tau_H^i$ consisting of triangles of quasi-uniform size H. The associated finite element space M_0 is defined to be the set of those continuous piecewise linear functions on the coarse mesh that vanish on $\partial\Omega$. The mesh refinement is defined in terms of a number of coarse grid subdomains $\{\Omega_i\}$, for $i = 1,\cdots,K$. By convention, Ω_i is defined to be the interior of the union of the closures of the coarse grid triangles. The refinement regions will also be referred to as "the subdomains." We assume that they have limited overlap in that any point of Ω is contained in at most a fixed number (not depending on H) of the subdomains. We define the domain of refinement Ω^r to be the union of the subdomains, $\Omega^r = \cup_{1=1}^K \Omega_i$ There are no theoretical restrictions concerning the definition of the refinement subregions except that they are defined in terms of the coarse-grid triangles and satisfy the overlap property as described above.

We provide two examples of the construction in the two-dimensional case. For both examples, the subregions are associated with coarse-grid nodes. The interior and boundary nodes of this mesh will be denoted $\{x_i\}$, for $i = 1,\cdots,N_c$. For the first example, we define the region associated with a coarse-grid node x_i as the subdomain Ω_i that contains the coarse-grid triangles having x_i as a vertex. For the second example, we consider a mesh that is topologically equivalent to a regular rectangular mesh (see Fig. 3.1). In this case, we define Ω_i to be the four quadrilaterals that share the vertex x_i. Some reasons for such a choice will be explained later. In either case, an index set $I \subseteq [1,\cdots,N_c]$ is selected and only those subdomains $\{\Omega_i\}$ with $i \in I$ are used to define the refinement region. By possibly changing the numbering of the coarse-grid nodes, we assume, without loss of generality, that $I = 1,2,\cdots,K$. There are no additional restrictions concerning this set I and hence rather complex refinement regions are possible.

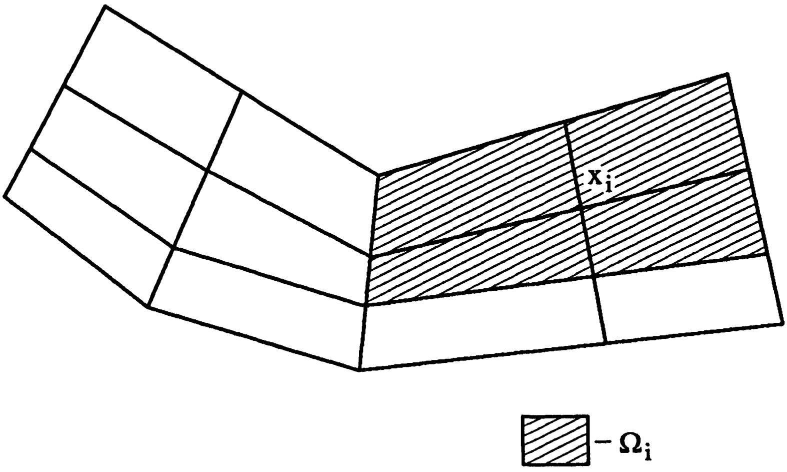


FIG. 3.1. *A distorted rectangular mesh.*

The composite space is defined in terms of a quasi-uniform mesh $\{\tau_h^i\}$ on Ω of size $h < H$ that satisfies

$$\cup_i \partial\tau_H^i \subseteq \cup_i \partial\tau_h^i.$$

The space of continuous piecewise linear functions with respect to this triangulation (which vanish on $\partial\Omega$) will be denoted by $\tilde{M}$. Note that this space is introduced for the construction and analysis of the composite grid space. It is not used in actual computation since it has too many degrees of freedom in Ω/Ω^r. The subspace M_i associated with the subdomain Ω_i is defined by

$$M_i = \{\phi \in \tilde{M} \,|\, \operatorname{supp}\phi \subseteq \Omega_i\}. \tag{3.1}$$

The composite finite element space is then defined to be

$$M = \sum_{i=0}^{K} M_i.$$

Note that the space M provides finer grid approximation in the refinement region Ω^r. An illustrative example of a mesh so generated is given in Fig. 3.2. The nodes on the boundary of the refinement region that are not coarse-grid nodes are slave nodes since, by continuity, the values of functions in M on these points are completely determined by their values on neighboring coarse-grid nodes. The operator $A_i : M_i \mapsto M_i$ is defined for $v \in M_i$ by

$$(A_i v, \phi) = A(v, \phi) \quad \text{for all } \phi \in M_i.$$

Our goal is to efficiently solve the composite grid problem: Given a function $f \in L^2(\Omega)$, find $U \in M$ satisfying

$$A(U, \phi) = (f, \phi) \quad \text{for all } \phi \in M. \tag{3.2}$$

As above, we define $A : M \mapsto M$ by

$$(Av, \phi) = A(v, \phi) \quad \text{for all } \phi \in M.$$

Problem (3.2) can then be rewritten as

$$AU = F, \tag{3.3}$$

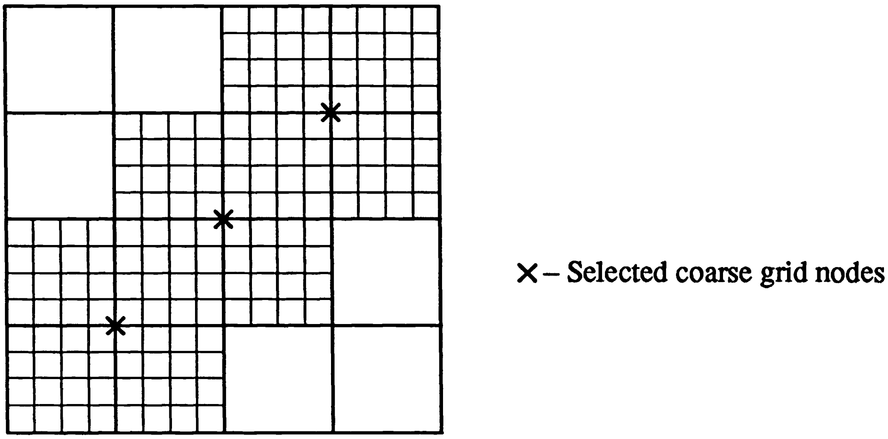


FIG. 3.2. *A composite grid.*

for appropriate $F \in M$. We will develop preconditioners for (3.3) by using overlapping domain decomposition.

There are basically two classes of these preconditioners, the additive and the multiplicative. The additive version defines the preconditioner B_a for A of (3.3) by

$$B_a = \sum_{i=0}^{K} R_i Q_i.$$

Here, Q_i denotes the $L^2(\Omega)$ projection operator onto M_i and R_i is a symmetric positive definite operator on M_i. Explicit choices for R_i will be discussed later; however, we note that it suffices to take R_i to be a preconditioner for A_i.

The multiplicative version is defined by applying the R_i consecutively. The multiplicative preconditioner B_m applied to a function $W \in M$ is defined as follows:

(1) Set $Y_0 = 0$.

(2) For $i = 1, \cdots, K+1$, define Y_i by

$$Y_i = Y_{i-1} + R_{i-1} Q_{i-1}(W - AY_{i-1}). \tag{3.4}$$

(3) For $i = K+2, \cdots, 2K+2$, define Y_i by

$$Y_i = Y_{i-1} + R_{2K+2-i} Q_{2K+2-i}(W - AY_{i-1}). \tag{3.5}$$

(4) Set $B_m W = Y_{2K+2}$.

It is not difficult to see that B_m is a symmetric linear operator on M.

The operators B_a and B_m defined above will be effective as preconditioners A if they satisfy the following:

(1) They are relatively inexpensive to evaluate.

(2) They lead to well-conditioned linear systems.

The first criterion involves implementation issues. The second criterion requires that the condition numbers $K(B_a A)$ and $K(B_m A)$ be small. In the case of the additive algorithms, this is equivalent to the existence of positive constants c_0, c_1 satisfying

$$c_0 A(v,v) \leq A(B_a Av, v) \leq c_1 A(v,v) \quad \text{for all } v \in M, \tag{3.6}$$

with c_1/c_0 small. A similar statement holds for the product algorithm. The goal of the analysis to be presented is to provide estimates for c_0 and c_1.

We note that the subdomains have limited overlap. This immediately implies that $c_1 \le CN_0$, where C depends only on the preconditioning properties of R_i, and N_0 is the maximum number of subdomains overlapping any point $x \in \Omega$. Thus, to analyze the additive algorithm, we are left to estimate the size of c_0. This will be done in the following section.

To analyze the product algorithm, we apply Theorem 2.2 of [6]. Assume that R_i is scaled so that for a fixed $\omega \in (0,2)$,

$$A(R_iA_iv, v) \le \omega A(v,v) \quad \text{for all } v \in M_i. \tag{3.7}$$

Then the product operator defined above satisfies

$$c_2A(v,v) \le A(B_mAv, v) \le A(v,v) \quad \text{for all } v \in M \tag{3.8}$$

where $c_2 \ge Cc_0$. Here c_0 is the constant in (3.6) and C is a positive constant which depends on N_0 and the preconditioning properties of R_i but not on h or H. Thus, to analyze the product algorithm, we are once again left to estimate the size of c_0 in (3.6).

Remark 3.1. It is easy to extend the above ideas to three-dimensional calculations. We consider the case where Ω is the union of rectangular parallelopipeds and the coarse-grid functions are piecewise trilinear on the rectangular parallelopipeds. The refinement subregions are defined to be the interior of the closures of coarse grid parallelopipeds. The composite mesh is defined in terms of a quasi-uniform mesh of parallelopipeds of size $h < H$. This mesh is assumed to be a refinement of the coarse-grid mesh. The space $\tilde{M}$ is defined to be the set of functions which are continuous on Ω, are trilinear with respect to the finer mesh, and vanish on $\partial\Omega$. The construction then proceeds exactly as described above for the two-dimensional case.

4. Convergence analysis. In this section, we provide an analysis for the overlapping domain decomposition preconditioners described in the previous section. As discussed earlier, we are to provide estimates for the constant c_0 appearing in (3.6). The analysis to be presented uses tools from both the theory of overlapping domain decomposition and the standard domain decomposition theory. We shall prove that under suitable hypotheses, the condition numbers $K(B_aA)$ and $K(B_mA)$ remain bounded independently of h and H in the two-dimensional case. The theorem for three dimensions guarantees that the condition numbers grow at most proportional to $(1 + \ln^2(H/h))$.

The first hypothesis for the theorems of this section provides control of the condition number $K(R_iA_i)$. Specifically, we assume that

$$C_0A(w,w) \le A(R_iA_iw, w) \le \omega A(w,w) \quad \text{for all } w \in M_i, \tag{4.1}$$

where the constants C_0 and ω remain fixed independent of h and H. For the product algorithm, we also assume that $0 < \omega < 2$. We have the following theorem.

THEOREM 4.1. *Let $d = 2$ and assume that there are no isolated points on the boundary of Ω^r. Then the condition numbers $K(B_aA)$ and $K(B_mA)$ remain bounded independently of h, H and the choice of subdomains $\{\Omega_i\}$.*

Before proving Theorem 4.1, we review some results from the theory of overlapping and nonoverlapping domain decomposition. These results will play a major role in the subsequent proof.

Let P_i denote the elliptic projection into the subspace M_i, i.e., $P_i v = w$ where w is the unique function in M_i satisfying

$$A(w, \phi) = A(v, \phi) \quad \text{for all } \phi \in M_i.$$

It immediately follows from the definitions that $Q_i A = A_i P_i$ and hence (4.1) implies that, for $v \in M$,

$$\begin{aligned} A(BAv, v) &= \sum_{i=0}^{K} A(R_i Q_i A v, v) \\ &= \sum_{i=0}^{K} A(R_i A_i P_i v, P_i v) \geq C_0 \sum_{i=0}^{K} A(P_i v, v). \end{aligned}$$

Thus, $c_0 \geq \tilde{c}_0 C_0$ for any constant $\tilde{c}_0$ satisfying the inequality

$$\tilde{c}_0 A(v, v) \leq \sum_{i=0}^{K} A(P_i v, v) \quad \text{for all } v \in M. \tag{4.2}$$

It is known (cf. [11]) that (4.2) follows provided that $\tilde{c}_0$ is a constant such that for any $v \in M$ there is a decomposition $v = \sum_{i=0}^{K} v_i$, with $v_i \in M_i$, satisfying

$$\sum_{i=0}^{K} A(v_i, v_i) \leq \tilde{c}_0^{-1} A(v, v). \tag{4.3}$$

We remark that it is easy to prove that statements (4.2) and (4.3) are equivalent.

We shall require a known result concerning overlapping domain decomposition [9], [11]. For each coarse-grid node x_i, we let $\tilde{\Omega}_i$ denote the interior of the union of the closures of the coarse-grid triangles that have x_i as a vertex. We define $\tilde{M}_i$ in terms of $\tilde{\Omega}_i$ as in (3.1). Given $w \in \tilde{M}$, there exists a decomposition $w = \sum_{i=1}^{N_c} \tilde{w}_i$, with $\tilde{w}_i \in \tilde{M}_i$, satisfying

$$\sum_{i=1}^{N_c} A(\tilde{w}_i, \tilde{w}_i) \leq C \left(A(w, w) + H^{-2} \|w\|^2 \right). \tag{4.4}$$

Here C is a constant not depending on h or H. The functions $\{\tilde{w}_i\}$ are defined in terms of a partition of unity with respect to the subdomains $\{\tilde{\Omega}_i\}$. An explicit partition can be defined from the coarse-grid nodal basis functions. These decompositions preserve support in the sense that if w vanishes at a node then every $\tilde{w}_i$ vanishes there also.

We will also need results from the standard domain decomposition theory which we introduce as the following lemma. The proof of this lemma is essentially given in [4].

LEMMA 4.1. *Let $y \in M \cap \tilde{M}_i$ be discrete harmonic on each coarse grid triangle. By this we mean that*

$$A(y, \phi) = 0$$

for all functions $\phi \in M$ which vanish on the coarse-grid mesh. Assume that y vanishes on at least one coarse-grid edge connecting $\partial\tilde{\Omega}_i$ and x_i. Let $\{\Gamma_j\}$ denote the remaining coarse-grid edges connecting $\partial\tilde{\Omega}_i$ and x_i and define y_j to be the function which is *discrete harmonic on the coarse-grid triangles, equals y on $\tilde{\Gamma}_j$ and vanishes on the remaining coarse-grid edges. Then $y = \sum_j y_j$ and*

$$\sum_j A(y_j, y_j) \le CA(y, y). \tag{4.5}$$

Remark 4.1. If the function y in Lemma 4.1 vanishes only on the point x_i (instead of a line from $\partial\tilde{\Omega}_i$ to x_i), then the above decomposition is still defined. However, in such cases, (4.5) only holds with the constant C replaced by $c\ln^2(H/h)$.

Proof of Theorem 4.1. As discussed in the previous section, it suffices to estimate the constant c_0 in (3.6). This in turn follows from the construction of a decomposition $v = \sum_{i=0}^K v_i$ with $v_i \in M_i$ satisfying (4.3).

Let Q denote the $L^2(\Omega)$ projection operator onto the subspace M_0.

We note that

$$\|(I - Q)w\|^2 \le CH^2 A(w, w) \tag{4.6}$$

and

$$A(Qw, Qw) \le CA(w, w) \tag{4.7}$$

hold for all $w \in H_0^1(\Omega)$.

We define v_0 in terms of Q by

$$v_0(x_i) = \begin{cases} v(x_i) & \text{for coarse-grid nodes } x_i \notin \Omega^r, \\ Qv(x_i) & \text{for coarse-grid nodes } x_i \in \Omega^r. \end{cases}$$

Clearly, we have that

$$\begin{aligned} A(v - v_0, v - v_0) &= A_r(v - v_0, v - v_0) \\ &\le 2[A((I - Q)v, (I - Q)v) + A_r(Qv - v_0, Qv - v_0)]. \end{aligned} \tag{4.8}$$

Here $A_r(\cdot,\cdot)$ is given by (2.2) but with integration only over the domain Ω^r. Note that $Qv - v_0$ is a function in M_0 which vanishes at all coarse-grid nodes in Ω^r and is equal to $Qv - v$ on the remaining coarse-grid nodes. Consequently,

$$A_r(Qv - v_0, Qv - v_0) \le C \sum_{x_i \in \partial\Omega^r} |(Qv - v)(x_i)|^2 . \tag{4.9}$$

Here, the sum is taken over coarse grid nodes $x_i \in \partial\Omega^r$. There are no isolated points on $\partial\Omega^r$ and hence for each $x_i \in \partial\Omega^r$, there is a coarse grid edge Γ_i contained in $\partial\Omega^r$ ending at x_i. Both functions Qv and v are linear on Γ_i and hence

$$\begin{aligned} |(Qv - v)(x_i)|^2 &\le cH^{-1} \|(Qv - v)\|_{\Gamma_i}^2 \\ &\le C[H^{-2} \|Qv - v\|_{\tau_H^i}^2 + A_{\tau_H^i}(Qv - v, Qv - v)]. \end{aligned} \tag{4.10}$$

Here τ_H^i denotes a coarse-grid triangle containing the edge Γ_i and $A_{\tau_H^i}$ denotes the form defined by (2.2) but with integration only over the region τ_H^i. The last inequality in (4.10) is a simple consequence of the divergence theorem and is well known. Combining (4.6)–(4.10) proves that

$$A(v - v_0, v - v_0) \le CA(v, v). \tag{4.11}$$

A similar argument gives that

$$\|v - v_0\|^2 \le CH^2 A(v, v). \tag{4.12}$$

We next apply the overlapping domain decomposition result to $w = v - v_0$. Specifically, we decompose $w = \sum_{i=1}^{N_c} \tilde{w}_i$, with $\tilde{w}_i \in \tilde{M}_i$ and satisfying (4.4). Note that this is clearly not the desired decomposition into the refinement subspaces $\{M_i\}$. We will distribute the functions $\tilde{w}_j$, $j = 1, \cdots, N_c$, into these subspaces. We start assigning each coarse grid-node $x_j \in \Omega^r$ to a unique subdomain $\Omega_{J(j)}$ which contains x_j. We then define

$$w_i = \sum_{J(j)=i} \tilde{w}_j.$$

We need to decompose the remaining functions $\tilde{w}_i$ for $x_i \notin \Omega^r$. Note that by the support property, $\tilde{w}_i$ vanishes unless $x_i \in \partial\Omega^r$. Consider a fixed function $\tilde{w}_i$ with $x_i \in \partial\Omega^r$. We write $\tilde{w}_i = y + y_i^0$ where $y = \tilde{w}_i$ on the boundaries of the coarse-grid triangulation and is discrete harmonic on the coarse-grid triangles (as in Lemma 4.1). The function y_i^0 vanishes on the boundaries of the coarse-grid triangulation and is orthogonal (in $A(\cdot,\cdot)$) to y. Note that the function y_i^0 is nonzero only on triangles contained in the refinement region. Thus, we can assign each of these triangles uniquely to a subdomain Ω_j and add the restriction of y_i^0 to the corresponding function w_j. The result of these modifications will still be denoted $\{w_j\}$.

We finally distribute the function y. There are no isolated points in $\partial\Omega^r$ and hence there must be a coarse-grid edge ending at x_i contained in $\partial\Omega^r$. Note, in addition, that both $\tilde{w}_i$ and y vanish on this edge. Thus, by Lemma 4.1,

$$\sum_j A(y_j, y_j) \leq CA(y, y). \tag{4.13}$$

The functions y_j are defined in Lemma 4.1 and the sum over j is taken over the coarse-grid indices corresponding to coarse-grid neighbors of x_i in Ω^r. The functions y_j are assigned to subdomains Ω_k which contain the corresponding edge (where y_j is nonzero) and the functions y_j are added into the corresponding w_k, producing a result which is still denoted w_k. It follows immediately that $w = \sum_{i=1}^K w_i$ and

$$\sum_{i=1}^K A(w_i, w_i) \leq C(A(w, w) + H^{-2} \|w\|^2). \tag{4.14}$$

Combining (4.11), (4.12), and (4.14) shows that the decomposition $v = \sum_{i=0}^K v_i$ with $v_i = w_i$ for $i = 1, \cdots, K$ satisfies (4.3). This completes the proof of the theorem. □

Remark 4.2. The hypothesis concerning isolated points on the boundary of Ω^r is included to provide a uniform bound for c_0. It is possible to show (using of [4], Thm. 1) that the constant c_0 only deteriorates like $C/\ln^2(H/h)$ if the isolated point hypothesis is not satisfied. This sort of decay is actually seen in the last numerical example in §5 where this assumption is violated.

Remark 4.3. There is very little restriction concerning the way that the domains Ω_i are defined. Note that if only one refinement domain is used, then Theorem 4.1 provides a result for the imbedded space case proposed in [2]. Alternatively, one can consider the case where Ω^r is all of Ω and hence $M = \tilde{M}$. In this case, Theorem 4.1 guarantees uniform bounds for the condition numbers without putting restrictions on the shapes of the subdomains $\{\Omega_i\}$. Thus, for example, the subdomains can be taken to be strips as long as the coarse problem is included.

Remark 4.4. There are numerous ways of modifying the above algorithm. One possibility is to include additional subspaces corresponding to the coarse-grid nodes on $\partial\Omega^r$. For such a node x_k, the space M_k would be defined to be the functions in M^r

with support in Ω_k. The same result holds with a somewhat simpler analysis since the standard domain decomposition theory is avoided. However, this algorithm has some practical disadvantages. There are more subproblems and many of them correspond to grids on irregularly shaped domains.

We next provide the result for three-dimensional applications.

THEOREM 4.2. *Let Ω be a domain in R^3 and let the mesh and approximation space be as discussed in Remark* 3.1. *Assume that the hypotheses preceding Theorem* 4.1 *hold and that Ω^r is the interior of its closure. Then*

$$K(B_aA) \le C(1+\ln^2(H/h))$$

and

$$K(B_mA) \le C(1+\ln^2(H/h)).$$

The constant C above does not depend on H or h.

Proof. The major part of the proof follows the proof of Theorem 4.1. We seek a decomposition of $v \in M$ satisfying (4.3) with $\tilde{c}_0^{-1} \le C(1+\ln^2(H/h))$. The construction of v_0 is exactly the same as in Theorem 4.1 and still satisfies (4.11) and (4.12). Here we used the assumption that Ω^r was the interior of its closure.

The overlapping domain decomposition $w = \sum_{i=1}^{N_c} \tilde{w}_i$ satisfying (4.4) is also valid in three dimensions. Once again we reduce to the problem of decomposing functions $\tilde{w}_k$ corresponding to coarse-grid nodes $x_k \in \partial\Omega^r$. As in the proof of Theorem 4.1, we write $\tilde{w}_k = y + y_k^0$, where y is discrete harmonic (with respect to the refined mesh) in the interior of the coarse parallelopipeds. The y_k^0 part is added into $\{w_i\}_{i=1}^K$.

Finally, we must take care of the function y. We write

$$y = \sum \bar{y}_{ij} + \sum \tilde{y}_l, \tag{4.15}$$

where:

(1) The functions $\{\bar{y}_{ij}\}$ are discrete harmonic (with respect to the refined mesh) in the interior of the coarse parallelopipeds.

(2) $\bar{y}_{ij} = y$ on the interior nodes on the face between coarse regions τ_H^i and τ_H^j and vanishes on the remaining nodes of $\cup\partial\tau_H^l$.

(3) $\tilde{y}_l$ equals y on the nodes of an edge of $\{\tau_H^l\}$ which is in $\tilde{\Omega}_k \cap \Omega_l$ and vanishes on all of the remaining nodes of the composite grid.

(4) The sums in (4.15) are taken over the appropriate faces and edges.

Applying Lemma 4.3 of [5] gives that

$$A(\bar{y}_{ij}, \bar{y}_{ij}) \le C(1+\ln^2(H/h))\{|\tilde{w}_k|^2_{1/2,\partial\tau_H^i} + H^{-1}|\tilde{w}_k|^2_{\tau_H^i}\}.$$

Here $|\cdot|_{1/2,\partial\tau_H^i}$ denotes the Sobolev seminorm of order 1/2 on $\partial\tau_H^i$. We clearly have

$$A(\bar{y}_{ij}, \bar{y}_{ij}) \le C(1+\ln^2(H/h))(A(w_k,w_k) + H^{-2}\|w_k\|^2).$$

Similar arguments using Lemmas 4.1–4.2 of [5] give

$$A(\tilde{y}_l, \tilde{y}_l) \le C(1+\ln^2(H/h))(A(w_k,w_k) + H^{-2}\|w_k\|^2).$$

The desired bound for $\tilde{c}_0^{-1}$ follows as in the proof of Theorem 4.1.

5. Numerical results. In this section, we provide the results of numerical examples illustrating the theory developed earlier. We shall consider the model problem

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{5.1}$$

where Δ denotes the Laplacian and Ω is the unit square $[0,1] \times [0,1]$. To define the coarse mesh, the domain Ω is first partitioned into $m \times m$ square subdomains of side length $H = 1/m$. Each smaller square is then divided into two triangles by one of the diagonals (e.g., the diagonal which goes from the bottom left to the upper right-hand corner of the square). The coarse-grid approximation space M_0 is defined to be the set of functions which are continuous on Ω, are piecewise linear with respect to the triangulation, and vanish on $\partial\Omega$. The space $\tilde{M}$ is defined from a similar finer mesh of size $h = H/l$ for some integer $l > 1$.

For our first two examples, we consider an application where it is required to refine along the diagonal connecting the origin with the point $(1,1)$. Such a refinement might be necessary if the function f has large gradients near this diagonal but is well behaved in the remainder of Ω. Accordingly, we select the coarse-grid nodes on the diagonal for refinement. We define the refinement region associated with a refinement node to be the four coarse mesh squares which have that node as a corner.

Note that the refinement region is highly irregular even though the coarse problem and the refinement subproblems involve regular rectangular meshes.

We will illustrate the rate of convergence of preconditioned algorithms for solving (3.2) where $A(\cdot,\cdot)$ is given by the Dirichlet form. To do this, we shall numerically compute the largest and smallest eigenvalue (λ_1 and λ_0, respectively) of the preconditioned operator $B_a A$. As is well known, the rate of convergence of the resulting preconditioned algorithms can be bounded in terms of the condition number $K(B_a A) = \lambda_1/\lambda_0$. We shall not report results for preconditioning with the product operator B_m, although our previous experience [6] suggests that the product version will converge somewhat faster than the additive.

Table 5.1 gives the largest and smallest eigenvalue and the condition number of the system $B_a A$ as a function of h. In this example, we took $R_i = A_i^{-1}$; i.e., we solved exactly on the subspaces $\{M_i\}$. For Table 5.1, $m = 4$, and there are three refinement subdomains $(0,1/2) \times (0,1/2)$, $(1/4,3/4) \times (1/4,3/4)$, and $(1/2,1) \times (1/2,1)$. Note that both the upper and lower eigenvalues appear to be tending to a limit as the ratio $h/H \mapsto 0$. Similar behavior is seen in Table 5.2, which corresponds to $m = 8$ and uses seven smaller refinement subregions.

TABLE 5.1
Condition numbers for 3 overlapping subregions.

h	λ_1	λ_0	$K(B_a A)$
1/8	2.44	0.50	4.9
1/16	2.50	0.41	6.1
1/32	2.51	0.38	6.6
1/64	2.52	0.36	6.9
1/128	2.52	0.35	7.1

In almost all realistic applications, the direct solution of subproblems is much more expensive than the evaluation of a suitable preconditioner. To illustrate the effect on the convergence rate of the preconditioned iteration, we next consider the previous example but with the direct solves on the subspaces replaced by multigrid preconditioners. Specifically, we employ the V-cycle multigrid algorithm (cf. [3]) using one pre- and post-smoothing Jacobi iteration on each grid level. This leads to

TABLE 5.2
Condition numbers for 7 overlapping subregions.

h	λ_1	λ_0	$K(B_aA)$
1/16	2.46	0.47	5.2
1/32	2.52	0.39	6.5
1/64	2.54	0.35	7.2
1/128	2.54	0.34	7.5

a preconditioning operator $R_i : M_i \mapsto M_i$, which satisfies

$$0.4A(v, v) \leq A(R_iA_iv, v) \leq A(v, v) \quad \text{for all } v \in M_i. \tag{5.2}$$

The constant 0.4 above was computed numerically and holds for all of the subspace problems that are required for this application, including M_0.

Tables 5.3 and 5.4 provide the eigenvalues and condition numbers for the above examples when direct solves were replaced by multigrid preconditioners. Note that in all of the reported runs, the condition number with multigrid preconditioners was at most 5/4 times as large as that corresponding to exact solves. Such an increase in condition number is negligible in a preconditioned iteration. In contrast, the computational time required for the multigrid sweep is considerably less than that needed for a direct solve (especially in more general problems with variable coefficients).

TABLE 5.3
Preconditioned subproblems, 3 overlapping subregions.

h	λ_1	λ_0	$K(B_aA)$
1/8	2.37	0.53	4.5
1/16	2.12	0.33	6.4
1/32	2.07	0.27	7.6
1/64	2.04	0.25	8.2
1/128	2.02	0.24	8.4

TABLE 5.4
Preconditioned subproblems, 7 overlapping subregions.

h	λ_1	λ_0	$K(B_aA)$
1/16	2.36	0.40	5.9
1/32	2.11	0.28	7.5
1/64	2.06	0.24	8.8
1/128	2.03	0.22	9.4

As a final example, we consider a case where the isolated point hypothesis of Theorem 4.1 is not satisfied. Specifically, we consider a coarse mesh of size $H = 1/4$ and select the four nodes with (x, y) values (1/4,1/2), (3/4,1/2), (1/2,1/4), and

(1/2,3/4). The refinement region is everything but the subsquares $[0,1/4] \times [0,1/4]$, $[0,1/4] \times [3/4,1]$, $[3/4,1] \times [0,1/4]$, and $[3/4,1] \times [3/4,1]$. Note that, to satisfy the hypotheses of the theorem, it would be necessary to include a refinement region centered at the coarse-grid node $(1/2,1/2)$. Table 5.5 gives the smallest eigenvalue for the operator $B_a A$ as a function of h. The function $(.32+.36\log_2(h^{-1}))^{-2}$ is also provided for comparison. These results suggest that smallest eigenvalue λ_0 decays as predicted by the theoretical bound $C/\ln(H/h)^2$ (see Remark 4.2).

TABLE 5.5
A "bad" example in two dimensions.

h	λ_0	$(.32+.36\log_2(h^{-1}))^{-2}$
1/8	.50	.51
1/16	.32	.32
1/32	.22	.22
1/64	.16	.16
1/128	.12	.12

REFERENCES

[1] R. E. Bank, T. Dupont, and H. Yserentant, *The hierarchical basis multigrid method*, Numer. Math., 52 (1988), pp. 427–458.

[2] J. H. Bramble, R. E. Ewing, J. E. Pasciak, and A. H. Schatz, *A preconditioning technique for the efficient solution of problems with local grid refinement*, Comput. Methods Appl. Mech. Engrg., 67 (1988), pp. 149–159.

[3] J. H. Bramble and J. E. Pasciak, *New convergence estimates for multigrid algorithms*, Math. Comp., 49 (1987), pp. 311–329.

[4] J. H. Bramble, J. E. Pasciak, and A. H. Schatz, *The construction of preconditioners for elliptic problems by substructuring*, I, Math. Comp., 47 (1986), pp. 103–134.

[5] ———, *The construction of preconditioners for elliptic problems by substructuring*, IV, Math. Comp., 53 (1989), pp. 1–24.

[6] J. H. Bramble, J. E. Pasciak, J. Wang, and J. Xu, *Convergence estimates for product iterative methods with applications to domain decomposition*, Math. Comp., 57 (1991), pp. 1–21.

[7] ———, *Convergence estimates for multigrid algorithms without regularity assumptions*, Math. Comp., 57 (1991), pp. 23–45.

[8] J. H. Bramble, J. E. Pasciak, and J. Xu, *Parallel multilevel preconditioners*, Math. Comp., 55 (1990), pp. 1–22.

[9] M. Dryja and O. Widlund, *Some domain decomposition algorithms for elliptic problems*, Iterative Methods for Large Linear Systems, L. Hayes and D. Kincaid, eds., Academic Press, New York, 1989.

[10] R.E. Ewing, R.D. Lazarov, and P.S. Vassilevski, *Local refinement techniques for elliptic problems on cell-centered grids,* II: *Two-grid iterative methods*, Math. Comp., submitted.

[11] P.L. Lions, *On the Schwarz alternating method*, in Proceedings of the First International Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., Society for Industrial Applied Mathematics, Philadelphia, PA, 1988.

[12] S.F. McCormick, *Multilevel Adaptive Methods for Partial Differential Equations*, Society for Industrial Applied Mathematics, Philadelphia, PA, 1989.

[13] S. McCormick and J. Thomas, *The fast adaptive composite grid* (FAC) *method for elliptic equations*, Math. Comp., 46 (1986), pp. 439–456.

[14] O. Widlund, *Optimal iterative refinement methods*, Proceedings of the Second International Symposium on Domain Decomposition Methods, T. F. Chan, R. Glowinski, J. Périaux and O.B. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989, pp. 114–125.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 1, pp. 411–424, January 1992

A LARGE, SPARSE, AND INDEFINITE GENERALIZED EIGENVALUE PROBLEM FROM FLUID MECHANICS*

HANS D. MITTELMANN†, CINDY C. LAW‡, DANIEL F. JANKOWSKI‡, AND G. PAUL NEITZEL‡

Abstract. A numerical method for calculating the minimum positive eigenvalue of a sparse, indefinite, Hermitian algebraic problem has been developed. The method is based on inverse iteration and is a generalization of a procedure previously employed for the simpler problem of finding the smallest eigenvalue of a positive-definite matrix. Motivation was provided by a three-dimensional research problem from hydrodynamic stability. Stability limits obtained from the application of the method to a previously studied problem are compared to independently determined results.

Key words. eigenvalue problem, thermocapillary convection, energy stability, inverse iteration

AMS(MOS) subject classifications. 76E15, 76D99, 65F15

1. Background and motivation.

Hydrodynamic stability theory is concerned with determining the conditions under which a certain flow, called the *basic state*, will remain stable or become unstable due to the inevitable presence of unknown perturbations (see Joseph (1976)). In general, these perturbations are governed by nonlinear partial differential equations. Linear-stability theory assumes the perturbations to be infinitesimally small and neglects the nonlinear terms in comparison with their linear counterparts. This theory is local in nature and results in a criterion that guarantees growth of these small disturbances. Typically, an externally controllable dimensionless parameter, say, R, is selected and linear theory yields a value R_L such that $R > R_L$ is a sufficient condition for instability.

Energy stability theory, on the other hand, adopts a global approach by examining the behavior of a generalized integral disturbance energy. Unlike linear stability theory, energy-stability theory provides a value R_E such that $R < R_E$ is a sufficient condition for stability of a given basic state to disturbances of *arbitrary amplitude.* This technique is equivalent to a stability analysis utilizing a Lyapunov function (see Sinha and Carmi (1976) and the literature cited there). The application of either theory gives rise, in general, to an eigenvalue problem.

If R_E and R_L should coincide, a rigorous stability bound is obtained. However, this is usually not the case and the proximity of R_E to R_L is a function of the physical mechanism which gives rise to the instability (see Davis (1971)). Two such mechanisms for which R_E and R_L may be expected to be relatively close to each other are buoyancy and thermocapillarity. The former is responsible for the onset of convection in an initially quiescent fluid layer that is heated from below (cooled from above) under certain conditions. This is the classically termed *Bénard convection.* Thermocapillarity, which is the variation of a liquid's surface tension with temperature, can

*Received by the editors April 5, 1990; accepted for publication (in revised form) December 5, 1990. This work was partially supported by the U.S. Air Force Office of Scientific Research under grant AFOSR-90-0080 and by the Microgravity Science and Applications Division of the National Aeronautics and Space Administration (NASA) under grant NAG-3-1054.

†Department of Mathematics, Arizona State University, Tempe, Arizona 85287–1804.

‡Department of Mechanical and Aerospace Engineering, Arizona State University, Tempe, Arizona 85287–6106.

also result in the instability of a quiescent liquid layer with a free surface (liquid-gas interface) when heated from below. This is known as *Marangoni convection.*

Buoyancy and thermocapillarity are also mechanisms that can drive *dynamic* basic states, which themselves can become unstable; for instance, the buoyancy boundary layer adjacent to a vertical heated surface and *thermocapillary convection*, which results when a temperature gradient exists *along* a free surface. Problems of technological interest in which both of these mechanisms play an important role may be found in the field of materials processing. One problem in particular that has received a considerable amount of attention is the stability of thermocapillary convection associated with the *float-zone crystal-growth process.* This process has been identified as one that may benefit from a microgravity environment. Although buoyancy forces will be significantly reduced in such an environment, thermocapillary convection will persist. Since it has been conjectured that the instability of thermocapillary convection may be responsible for undesirable striations observed in material grown by this process, the stability properties of this basic state are of interest for both terrestrial and microgravity conditions.

Recent work on the stability of thermocapillary convection has been done by Shen, Neitzel, Jankowski, and Mittelmann (1990). They employed energy stability theory to obtain sufficient conditions for stability against *axisymmetric* disturbances. Comparisons with the experimental results of Preisser, Schwabe, and Scharmann (1983) have identified a need for corresponding results for nonaxisymmetric disturbances. The present work is motivated by the need for obtaining such stability information.

The earlier work of Shen et al. (1990) required the calculation of the smallest positive eigenvalue of a generalized eigenvalue problem of the form $\mathbf{A}x = \rho\mathbf{B}x$. Energy theory, since it has its basis in a variational problem, results in a differential eigenvalue problem that is selfadjoint. The direct discretization of the variational problem employed by Shen et al. preserved this character and thus resulted in *real symmetric* matrices $\mathbf{A}$ and $\mathbf{B}$. There are several complicating features: (i) both matrices are indefinite; (ii) the basic-state quantities in $\mathbf{B}$ depend on the stability parameter; and (iii) both $\mathbf{A}$ and $\mathbf{B}$ depend on a free coupling parameter λ. The latter two complications are easily dealt with by iteration processes, once a reliable procedure for calculating the eigenvalue of interest has been developed.

The standard algorithm for the solution of eigenvalue problems of the form $\mathbf{A}x = \rho\mathbf{B}x$, for general real or complex matrices $\mathbf{A}$ and $\mathbf{B}$, is the QZ algorithm. This method computes *all* of the eigenvalues of the system by means of orthogonal transformations. However, since it does not exploit either the sparsity or symmetry of $\mathbf{A}$ and $\mathbf{B}$, and only a single eigenvalue is desired, its use in the application of interest, where the system order is typically quite large, is extremely inefficient in terms of computer resources. In addition, it is possible that the calculation of the *single* eigenvalue of interest, which does not occur at any particular stage of the overall solution process, can be contaminated by earlier calculations. These factors motivated the development of the method that was successfully used by Shen et al. (1990). The QZ algorithm was used solely to obtain some insight into the distribution of the complete family of eigenvalues for coarse discretizations (i.e., low system order) of the variational problem.

Theoretically, the inclusion of nonaxisymmetric disturbances into the discretized functional is straightforward, involving only a normal-mode decomposition in the azimuthal coordinate (cf. (2.9)). However, the corresponding computational problem is more difficult because the matrices in the algebraic eigenvalue problem become *complex* and *Hermitian.* The numerical procedure of Shen, Neitzel, Jankowski, and

Mittelmann (1990) is unable to handle this case and some modification is necessary. Basically, their procedure is a form of inverse iteration. The modification to handle complex Hermitian matrices uses a residual-based inverse iteration, which, in each step, computes an increment of the last iterate and employs additional orthogonal projections. Both of these modifications were made, since standard inverse iteration computes nearly parallel vectors and the loss of significance in the correction to the previous vector leads to slow or no convergence. The complex case is reduced to a real case of twice the dimension, which is also symmetric. This permits a relatively efficient solution of this indefinite system using preconditioned conjugate-gradient methods. So far our emphasis has been on developing a working algorithm and obtaining solutions to these yet unsolved stability problems. An analysis of the method employed may be given in future work. It must be stressed, however, that the indefiniteness of both A and B makes this eigenvalue problem more general than is usually considered in the literature for large sparse eigenproblems from the applications. The recent survey article by Kerner (1989), for example, addresses only the case in which B is definite. Nevertheless, it may be that suitable generalizations of existing algorithms, for example, of generalized Davidson's method (see Morgan and Scott (1986)) to the present case may provide alternative approaches. This method, if applicable, would allow indefinite preconditioners.

In order to test this new numerical approach, we choose to examine in this paper a related stability problem for which results are available. As in Shen et al. (1990), we consider the onset of buoyant convection in a fluid-filled cylinder heated from below with either a conducting or insulating sidewall boundary. Linear-stability limits have been computed by Charlson and Sani (1970), (1971) for both axisymmetric and nonaxisymmetric disturbances. These linear-theory results can be compared directly with those of energy theory since it can be shown for this problem that both limits coincide (see Joseph (1976)). Although the heated-cylinder problem is somewhat simpler than the float-zone problem for which results will ultimately be computed, it nevertheless serves as an excellent test problem for the numerical procedure. For the first results on the float-zone problem, see Mittelmann, Law, Jankowski, and Neitzel (1991).

In the following section the application of energy-stability theory to the heated-cylinder problem will be described. Subsequently, the new version of the inverse iteration procedure for the computation of the stability bounds will be presented. Results for the heated-cylinder problem will be compared to those of Charlson and Sani (1970) and (1971), followed by a discussion on the generalization of the method to the problem of thermocapillary convection in a model of the float-zone crystal-growth process.

2. Energy-stability analysis of the heated-cylinder problem.

Consider a right circular cylinder of height H and radius R that is filled with a Boussinesq fluid of kinematic viscosity ν and thermal diffusivity κ, and oriented with its axis of symmetry in the direction of the gravitational acceleration of magnitude g. The top and bottom surfaces of the cylinder are maintained at temperatures T_C and $T_H > T_C$, respectively, and the sidewall may be either perfectly conducting or perfectly insulating. The dimensionless Boussinesq equations that govern the temperature $T(r,\theta,z,t)$, pressure $p(r,\theta,z,t)$, and velocity $\mathbf{v}(r,\theta,z,t)$ fields are

$$\mathbf{v}_t + Ra^{1/2}\mathbf{v}\cdot\nabla\mathbf{v} = -Pr\nabla p + Pr\nabla^2\mathbf{v} + Ra^{1/2}PrT\mathbf{e}_z, \tag{2.1}$$

$$T_t + Ra^{1/2}\mathbf{v}\cdot\nabla T = \nabla^2 T, \tag{2.2}$$

$$\nabla \cdot \mathbf{v} = 0, \tag{2.3}$$

where H, $kRa^{1/2}/H$, $\rho\nu\kappa Ra^{1/2}/H^2$, H^2/κ have been used to scale lengths, velocity, pressure, and time, respectively. The dimensionless temperature is defined to be

$$T = (T' - T_m)/\Delta T,$$

where T' is the dimensional temperature, $T_m = (T_H + T_C)/2$ is the mean temperature, and $\Delta T = T_H - T_C$. The dimensionless parameters appearing in (2.1) are the Rayleigh and Prandtl numbers, which are defined as

$$Ra = \beta g H^3 \Delta T/\kappa\nu$$

and

$$Pr = \nu/\kappa,$$

where β is the coefficient of thermal expansion.

The boundary conditions to be applied require no penetration and no slip at solid boundaries, constant temperatures at the planar endwalls, conducting or insulating conditions at the sidewall, and boundedness of all quantities on the axis of symmetry. Equations (2.1)–(2.3), along with the appropriate boundary conditions, permit a solution that is motionless ($\mathbf{v} = 0$), with the linear (pure conduction) temperature profile

$$\Theta(z) = 1/2 - z \tag{2.4}$$

and corresponding pressure distribution

$$P(z) = 1/2Ra^{1/2}z(1 - z) \tag{2.5}$$

where the arbitrary constant of integration in (2.5) has been chosen to be zero. This is the *basic state* of interest in the present paper.

We begin the energy-theory analysis of the basic state in the usual fashion by deriving the energy identity. We assume there exists a solution $[\mathbf{u}, p, T]$ to the governing equations ((2.1)–(2.3)) that is a perturbation to the motionless conducting basic state, i.e.,

$$\begin{aligned}[\mathbf{u}, p, T] =& [0, P(z), \Theta(z)] \\ &+ [u'(r,z,\theta,t), v'(r,z,\theta,t), w'(r,z,\theta,t), p'(r,z,\theta,t), T'(r,z,\theta,t)].\end{aligned} \tag{2.6}$$

In our earlier work (see Shen et al. (1990)) on the float-zone problem, we had restricted consideration to axisymmetric disturbances, i.e., the disturbance had no θ-component of velocity and no θ-dependence.

Substitution of (2.6) into the governing equations and the appropriate boundary conditions leads to a system of equations and boundary conditions for the disturbance quantities. We then take the inner product of the disturbance momentum equation with $\mathbf{u}'$, add to this the disturbance energy equation multiplied by $\lambda Pr T'$, and integrate over the volume

$$V = \{(r, \theta, z) \mid 0 \le r \le A,\ 0 \le \theta \le 2\pi,\ 0 \le z \le 1\}$$

of the cylinder, using the disturbance boundary conditions. The parameter $A = R/H$ is the aspect ratio of the cylinder. The result is the exact disturbance-energy evolution equation, which may be written (see Davis and von Kerczek (1973)),

$$\frac{1}{E}\frac{dE}{dt} = \frac{1}{E}\left(-PrD - Ra^{1/2}PrI\right), \tag{2.7}$$

where

$$E = \frac{1}{2}\int_V (\mathbf{u}' \cdot \mathbf{u}' + \lambda Pr T'^2)\, dV, \quad D = \int_V (\nabla \mathbf{u}' \cdot \nabla \mathbf{u}' + \lambda \nabla T' \cdot \nabla T')\, dV,$$

$$I = -\int_V (1+\lambda) w' T' dV.$$

The velocity and temperature disturbances have been joined by a positive *coupling parameter* λ (see Joseph (1976)) to form a generalized disturbance energy, E. An upper bound is constructed for the resulting right-hand side of this equation,

$$\nu = \max_H \left(\frac{-PrD - PrRa^{1/2} I}{E} \right), \tag{2.8}$$

where the maximum is taken over the space of kinematically admissible functions,

$$H = \{\mathbf{u}', T' \mid \mathbf{u}' = T' = 0 \text{ at } z = 0, 1; \text{ boundedness at } r = 0;$$
$$\mathbf{u}' = 0 \text{ and appropriate, homogeneous thermal conditions at } r = A; \nabla \cdot \mathbf{u}' = 0\}.$$

We choose to formulate the problem so that the Rayleigh number is the stability parameter. For fixed values of the other parameters associated with the problem, the smallest value of Ra that corresponds to the condition $\nu = 0$ will be called Ra^*.

The cylindrical geometry allows a Fourier decomposition in the azimuthal coordinate of the form

$$u'(r, \theta, z, t) \rightarrow \mathbf{u}(r, z, t) e^{in\theta} + \bar{\mathbf{u}}(r, z, t) e^{-in\theta}, \tag{2.9}$$

where n is the azimuthal (integer) wave number and "$^-$" denotes complex conjugation. Since λ is a free parameter, the maximum value of Ra^* for positive values of λ is sought (see Joseph (1976)), while a minimization has to be carried out with respect to n. The resulting value is the energy-stability limit, Ra_E, defined as

$$Ra_E = \min_n \max_{\lambda > 0} Ra^*. \tag{2.10}$$

It can be shown that $Ra < Ra_E$ is a sufficient condition for stability against disturbances of arbitrary amplitude belonging to the class H. For the problem of interest here, the search for the maximizing λ can be done analytically with the result that $\lambda = 1$ is the optimal choice.

It is convenient to consider a slightly different functional than ν in (2.8) that incorporates the divergence constraint by means of a Lagrange multiplier. Hence, the maximum problem to be solved is expressed as

$$\max_h \left[-PrD - PrRa^{1/2} I + 2\int_V \pi \nabla \cdot \mathbf{u}'\, dV + \bar{\nu}(E-1) \right] = 0, \tag{2.11}$$

where $\bar{\nu}$ is a Lagrange multiplier expressing the arbitrary normalization $E = 1$, $\pi(r, z)$ is a Lagrange multiplier, and h is the extension of H obtained by removing the divergence constraint. It is easy to show that $\nu = \bar{\nu}$. Following Shen et al. (1990), we consider the variation of a discretized version F_D of the *quadratic* functional

$$F = -PrD - PrRa^{1/2} I + 2\int_V \pi \nabla \cdot \mathbf{u}'\, dV.$$

The explicit homogeneous boundary conditions that arise following normal-mode decomposition and that are applied during the discretization process are dependent upon the azimuthal wave number n (see Batchelor and Gill (1962)). The same discretization procedure employed by Shen et al. (1990) is used, but there are probably opportunities available for further study of this topic.

A stationary value of F_D is located by differentiating it with respect to each unknown and setting each of these derivatives to zero, i.e.,

$$\frac{\partial F_D}{\partial q} = 0 \qquad q = u_{i,j}, v_{i,j}, w_{i,j}, \phi_{i,j} \quad \text{or } \pi_{k,l}; \tag{2.12}$$

here the indices correspond to a nodal numbering, see Shen et al. (1990). This process yields a *generalized algebraic eigenvalue* problem. We seek its minimum positive eigenvalue as the approximate (subject to discretization error) value of Ra^*. Calling the vector consisting of the unknowns on all grid points $\overline{\mathbf{X}}$, we rewrite (2.12) in the matrix form

$$\bar{\mathbf{A}}\overline{\mathbf{X}} = \rho\overline{\mathbf{B}}\,\overline{\mathbf{X}}, \tag{2.13}$$

where $\bar{\mathbf{A}}$ and $\overline{\mathbf{B}}$ are *indefinite, Hermitian* matrices with $\bar{\mathbf{A}}$ having a banded structure. More precisely, $\bar{A}$ and $\bar{B}$ have the following block structure

$$\bar{A} = \begin{pmatrix} A_{11} & A_{12} \\ \bar{A}_{12}^T & 0 \end{pmatrix}, \qquad \bar{B} = \begin{pmatrix} B_{11} & 0 \\ 0 & 0 \end{pmatrix},$$

where the upper left submatrices arise from the above functionals, while A_{12} results from the incompressibility condition. $\bar{B}$ has null vectors associated with its lower blocks, but B_{11} in general may yield additional null vectors.

3. Numerical procedure for finding Ra_E.

Equation (2.13) represents a generalized eigenvalue problem. The matrices $\bar{\mathbf{A}}$ and $\overline{\mathbf{B}}$ are Hermitian and sparse, but, in general, indefinite. For the ultimate problem of interest, in addition to the basic-state dependence of $\overline{\mathbf{B}}$ mentioned above, $\bar{\mathbf{A}}$ and $\overline{\mathbf{B}}$ depend nonlinearly on the other parameters of the problem, namely, Pr and the coupling parameter, λ. Due to the fact that we report here on the heated cylinder problem as a test for both the discretization procedure and the eigenproblem solver, we need to consider only fixed values of these parameters, reducing (2.13) to the generalized eigenvalue problem

$$\bar{\mathbf{A}}\overline{\mathbf{X}} = \rho\overline{\mathbf{B}}\,\overline{\mathbf{X}}, \qquad \|\overline{\mathbf{X}}\| = 1, \tag{3.1}$$

where $\|\cdot\|$ denotes the Euclidean norm.

The eigenvalues ρ of (3.1) are, in general, complex since in the case of at least one of the matrices being regular the problem is equivalent to the eigenvalue problem for a non-Hermitian complex matrix. Null vectors of $\bar{\mathbf{A}}$ and $\overline{\mathbf{B}}$ correspond to zero and "infinite" eigenvalues, respectively. The method developed for the present computations makes use of the sparseness of $\bar{\mathbf{A}}$ and $\overline{\mathbf{B}}$, the fact that they are Hermitian, and computes only the eigenvalue of interest.

Since a real eigenvalue of (3.1) has to be computed, it is convenient to transform the problem so that only real arithmetic is needed. Let

$$\mathbf{A} = \begin{bmatrix} \mathrm{Re}(\bar{\mathbf{A}}) & -\mathrm{Im}(\bar{\mathbf{A}}) \\ \mathrm{Im}(\bar{\mathbf{A}}) & \mathrm{Re}(\bar{\mathbf{A}}) \end{bmatrix},$$

where $\mathrm{Re}(\cdot)$, $\mathrm{Im}(\cdot)$ denote real and imaginary parts, and let $\mathbf{B}$ be defined in a similar way. A straightforward calculation shows that the eigenvalue problem

$$\mathbf{A}X = \rho\mathbf{B}X, \qquad \mathbf{X} \neq 0 \tag{3.2}$$

has the eigenpairs $\rho, (\overline{\mathbf{X}}, -i\overline{\mathbf{X}})^T$, $\overline{\rho}, (\overline{\overline{\mathbf{X}}}, i\overline{\overline{\mathbf{X}}})^T$, where $\rho, \overline{\mathbf{X}}$ are the eigenpairs of (3.1) and $i = \sqrt{-1}$ is the imaginary unit. Thus, each eigenvalue of (3.1) corresponds to a pair of complex-conjugate eigenvalues of (3.2).

It is important to note that all eigenvalues of (3.2) have the same modulus as the corresponding eigenvalues of (3.1) and that a simple real eigenvalue of (3.1) corresponds to a double eigenvalue of (3.2), but that (3.2) has no other eigenvalue of the same modulus. Apart from the multiplicity of the eigenvalues, (3.2) is an eigenvalue problem of the form considered in Shen et al. (1990).

The stability theory requires the computation of but a single eigenvalue of the problem (3.2), namely, ρ^*, the smallest positive one. The following observations from computations of the entire spectrum using the complex QZ algorithm led to the choice of a suitable form of inverse iteration for the solution of this problem. The problem specified in (3.2) has a spectrum that is roughly symmetrically distributed in the complex plane with respect to both the real and the imaginary axis. There are a number of *infinite* eigenvalues corresponding to null vectors of $\mathbf{B}$. The smallest positive eigenvalue ρ^* was, in general, also the smallest one in modulus. There were, however, several eigenvalues both in the negative and the positive halfplane that were not much greater in modulus than ρ^*. The nullspaces of $\mathbf{A}$ and $\mathbf{B}$ did not appear to have a nontrivial intersection. Thus, inverse iteration with an appropriately chosen shift of origin was selected for the solution of (3.2).

When the matrix $\mathbf{B}$ in (3.2) is positive definite, *Rayleigh quotient* iteration, i.e., inverse iteration with a shift computed in each step from the Rayleigh quotient of the current eigenvector approximation, is known to exhibit very rapid convergence. This procedure cannot be applied here. The technique developed for the present case is a generalization of that employed by Bank and Mittelmann (1986) in the program PLTMG for the simpler problem of finding the smallest eigenvalue of a positive-definite matrix. It cannot be expected to solve for any desired eigenvalue of (3.2) without having a rather good initial approximation for this eigenvalue, but the observations mentioned above justified the application of a suitably implemented version for the computation of ρ^*. The process is started with a random normalized vector $\mathbf{X}_0$ that is such that its Rayleigh quotient

$$\rho_0 = \mathbf{X}_0\mathbf{A}X_0/\mathbf{X}_0^T\mathbf{B}X_0 \tag{3.3}$$

is well defined. Given this initial pair $\rho_0, \mathbf{X}_0$, the inverse iteration procedure is performed as follows:

1. Solve $(\mathbf{A} - s\mathbf{B})\overline{\mathbf{Y}} = (\rho_k\mathbf{B} - \mathbf{A})\mathbf{X}_k$ and define

$$\mathbf{Y} = \frac{\overline{\mathbf{Y}} - \mathbf{X}_k^T\overline{\mathbf{Y}}\mathbf{X}_k}{\|\overline{\mathbf{Y}} - \mathbf{X}_k^T\overline{\mathbf{Y}}\mathbf{X}_k\|}.$$

2. Form $\mathbf{Q} = [\mathbf{X}_k|\mathbf{Y}]$ and solve the 2×2 problem

$$\mathbf{Q}^T\mathbf{A}\mathbf{Q}\mathbf{Z} = \tau\mathbf{Q}^T\mathbf{B}\mathbf{Q}\mathbf{Z}$$

for the eigenvalues τ_1, τ_2 and associated normalized eigenvectors $\mathbf{Z}_1, \mathbf{Z}_2$. Without loss of generality, let τ_1 be the *smallest positive* eigenvalue.

3. Set $\rho_{k+1} = \tau_1$, $\mathbf{X}_{k+1} = \mathbf{Q}\mathbf{Z}_1$, and check for convergence. If not converged, increment iteration index k and repeat.

While no analysis of this algorithm will be given here, a few remarks are in order. The shift s is chosen as a positive real number. From the well-known theory of inverse iteration s has to be closer to ρ^* than to any other eigenvalue of (3.2). Since ρ^* is unknown, this may require some adjustment of s during the iteration. Experience has shown, however, that a rough knowledge of the expected ρ^* and the facts observed above on the distribution of the other eigenvalues permitted the determination of a reasonable value for the shift.

The eigenvalue problem in step 2 is basically an orthogonal projection of the original problem into the subspace spanned by the columns of $\mathbf{Q}$. Since only one eigenvalue is needed, no attempt was made to save more than one vector and solve a larger auxiliary eigenvalue problem. Simpler inverse iteration algorithms are indeed available; however, their application to the present problem did not yield satisfactory results. In general, of course, this 2×2 eigenvalue problem may have complex eigenvalues as well as real ones. While several precautions for this and other cases were put into the program, they will not be described here, being a rather technical detail. Eventually, τ_1 will be positive and approximate ρ^* while $\mathbf{QZ}_1$ approximates the associated eigenvector. The quantities $\mathbf{X}_{k+1}$ and ρ_{k+1} are related through the Rayleigh quotient (3.3).

Due to the properties of $\mathbf{A}$ and $\mathbf{B}$, the matrix on the left-hand side of the linear system in step 1 is highly indefinite but symmetric. The latest version of the program SYMMLQ (see Paige and Saunders (1975)) was used for the solution of this system. It applies a conjugate-gradient method and provides for preconditioning by a positive-definite matrix. There is no complete theory available for the preconditioning of indefinite systems. The standard diagonal preconditioning for the associated normal equations led to the choice of the diagonal matrix with its ith element equal to the Euclidean norm of the ith column of the matrix $\mathbf{A} - s\mathbf{B}$.

The convergence of the above inverse iteration procedure is, in general, linear with a factor asymptotically equal to

$$\left| \frac{s - \rho^*}{s - \rho_n} \right| < 1,$$

where ρ_n is the next nearest eigenvalue of (3.2) to s and different from ρ^*. Choosing s close to ρ^* will thus speed up convergence of the inverse iteration while generally requiring more conjugate-gradient iterations for the nearly singular system matrix. The essential computational requirement per conjugate-gradient iteration is one matrix-vector multiplication with the system matrix. Since ρ^* is, in general, a double eigenvalue of (3.2), it is also important to note that inverse iteration exhibits the same convergence behavior for eigenvalues that are equal as opposed to those that are equal in modulus but different. Again, the observations on the distribution of the spectrum and the properties of (3.2) as a transformation of (3.1) are of relevance.

Finally, as a stopping criterion, a test on the relative decrease of the residual combined with one on the convergence of subsequent eigenvalue approximations was used. The resulting code never failed to solve the above eigenvalue problem and some information on its performance will be given in the following section.

4. Results and discussion. The algorithm described in the preceding section has been implemented in a FORTRAN code and executed on an Ardent Titan Mini-supercomputer in the Advanced Research Computing Facility of the Department of Mathematics at Arizona State University. The main storage requirement was for the Hermitian matrices $\overline{\mathbf{A}}$ and $\overline{\mathbf{B}}$. Only the nonzero elements of the upper halves of the real and imaginary parts of these matrices need to be stored. Indirect addressing prevents a full vectorization of the matrix-vector multiplication executed in each step of SYMMLQ. Experiments with, for example, storage of the matrices by diagonals permitted full vectorization, but yielded no gain in performance due to the relative sparseness of the diagonals caused by a staggered-grid discretization (see Shen et al. (1990) for details). For example, for the case of aspect ratio $A = 0.2$ with $\Delta r = 0.0052$ and $\Delta z = 0.0065$, the order of the matrices is 29022; the number of nonzero elements that must be stored are

$$\begin{aligned} 157946 &\quad \text{for} \quad \text{Re}\,(\overline{\mathbf{A}}), \\ 73168 &\quad \text{for} \quad \text{Im}\,(\overline{\mathbf{A}}), \\ 50396 &\quad \text{for} \quad \text{Re}\,(\overline{\mathbf{B}}). \end{aligned}$$

The inverse iteration was started with a random vector; a shift of 80 percent of a rough guess of the expected eigenvalue was used for all calculations.

The algorithm described in the previous section was used to compute the eigenvalues Ra^* of the heated-cylinder problem described in §2. Values were calculated for various azimuthal wave numbers, aspect ratios, and discretizations. The number of iterations required for convergence ranged between 4 and 16. Future theoretical investigation is needed to explain the causes of the larger iteration counts; one possibility may be the existence of closely spaced eigenvalues. The overall cost of solving the eigenvalue problem was not very sensitive with respect to the choice of the shift s in a relatively wide range of $0 < s < \rho^*$. The above preconditioning technique resulted in a number of *cg*-iterations within SYMMLQ equivalent to a few percent of the order of the linear system *if* s was not too close to ρ^*. When s approached ρ^* the faster convergence of the inverse iteration was, in general, more than offset by the increased *cg*-iterations. For the example whose dimensions were listed above, the number of *cg*-iterations with the simple and cheap diagonal preconditioning was on the order of 1000 for an order of $\mathbf{A}$ in (3.2) of 58044.

Since the results of Charlson and Sani (1971) are presented in the form Ra^* versus aspect ratio for integer values of the azimuthal wave number n, the minimization implied in (2.10) was not carried out. These values are denoted by Ra_n in the accompanying figures. In addition to performing calculations for nonaxisymmetric disturbances, cases for axisymmetric ($n = 0$) disturbances were computed and compared with earlier results of Charlson and Sani (1970). For axisymmetric disturbances, the complex Hermitian system (2.13) simplifies to a real, symmetric one, thus providing a means for testing the algorithm's ability to reproduce the results obtained with its predecessor (see Shen et al. (1990)). As an additional check, calculations of Ra_{-n} were in complete agreement with corresponding values of Ra_n, thereby indicating the expected indifference to the sense of the azimuthal coordinate.

The convergence of the inverse-iteration procedure was addressed earlier. Also of interest is the convergence behavior of the discretization scheme. Figure 1 illustrates this for a case with $A = 2$ and $n = 1$ for a conducting sidewall. For moderate values of stepsize $h = (\Delta r^2 + \Delta z^2)^{1/2}$, convergence is $O(h^2)$, but for smaller h, this appears to accelerate.

Figures 2 through 5 show a comparison between selected energy-stability results obtained with the present algorithm and linear-stability results, which are equal to those obtained from energy theory for this problem. Since our interest was in verifying the reliability of the new inverse-iteration algorithm for indefinite, Hermitian eigenvalue problems, we did not attempt to reproduce all available results. The $n = 1$ results reported by Charlson and Sani (1971), which were obtained with a Galerkin approach, were apparently not converged (R. L. Sani, private communication); recent

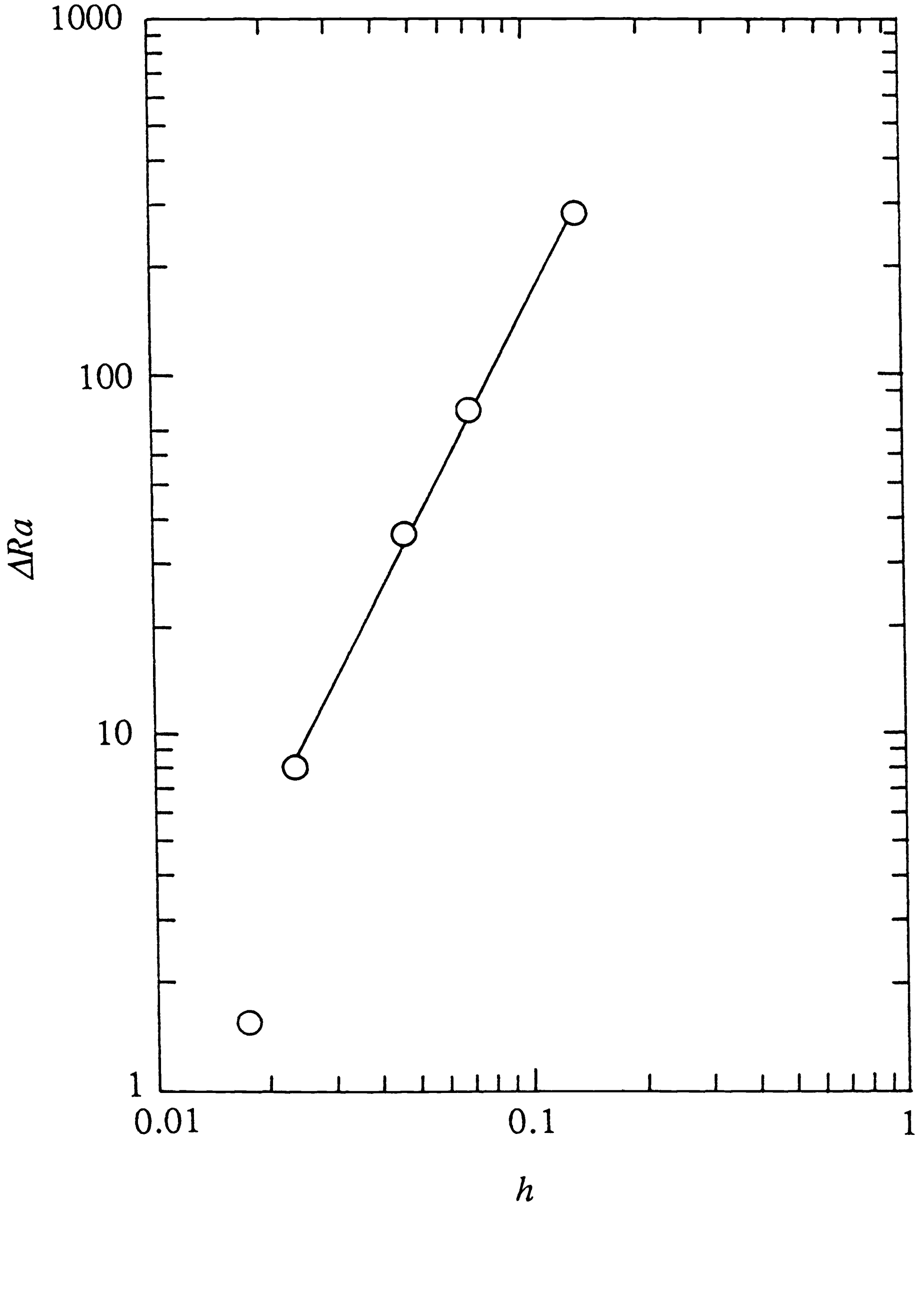


FIG. 1. *Convergence behavior for various discretizations for* $A = 0.5$, $n = 1$ *with a conducting sidewall.* ΔRa *is computed using the result from the finest discretization*; $h = (\Delta r^2 + \Delta z^2)^{1/2}$. *The straight line shows convergence to be* $O(h^2)$.

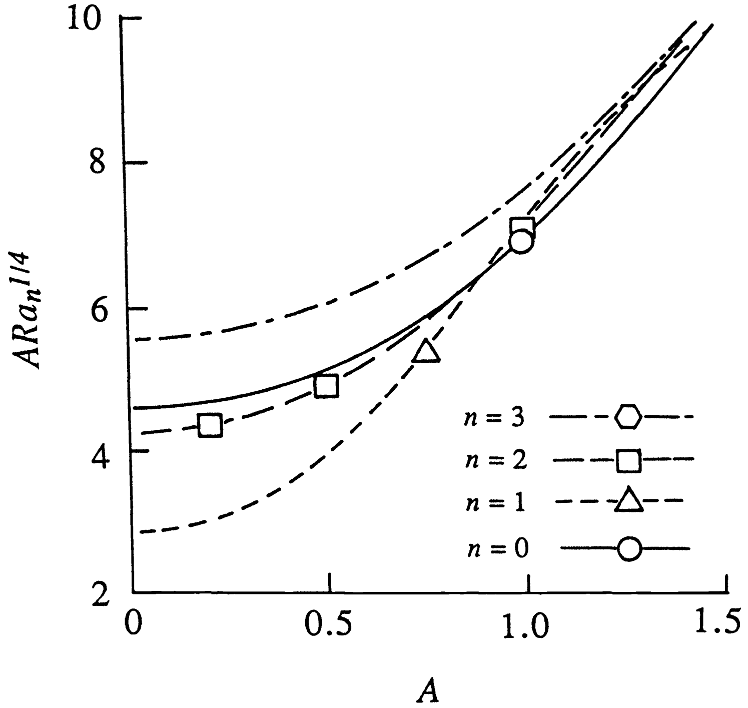


FIG. 2. *Comparison of selected results from the present computations (symbols) with those of Hardin et al.* (1990) *(curves) for* $A < 1.5$ *with insulating sidewall.*

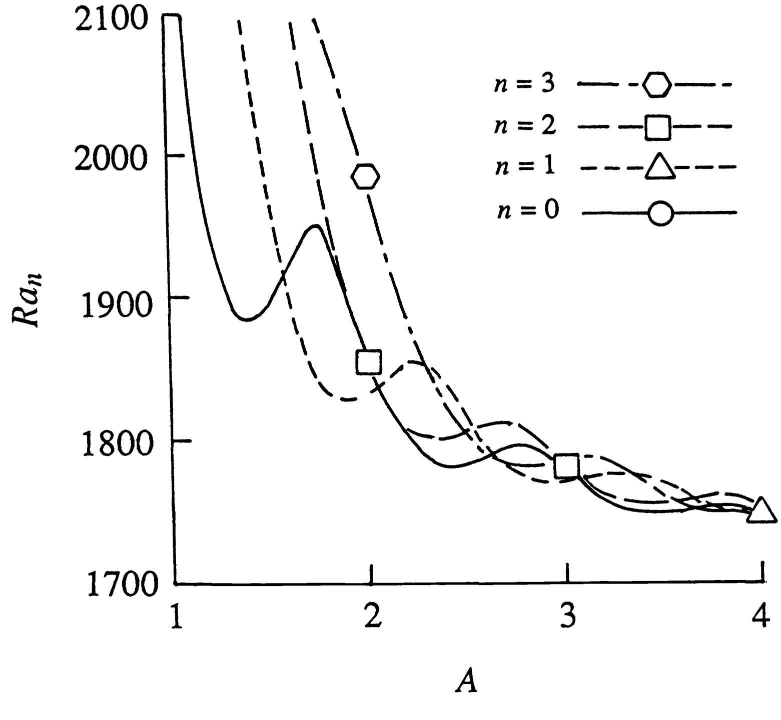


FIG. 3. *Comparison of selected results from the present computations (symbols) with those of Hardin et al.* (1990) *(curves) for* $1 < A < 4$ *with an insulating sidewall.*

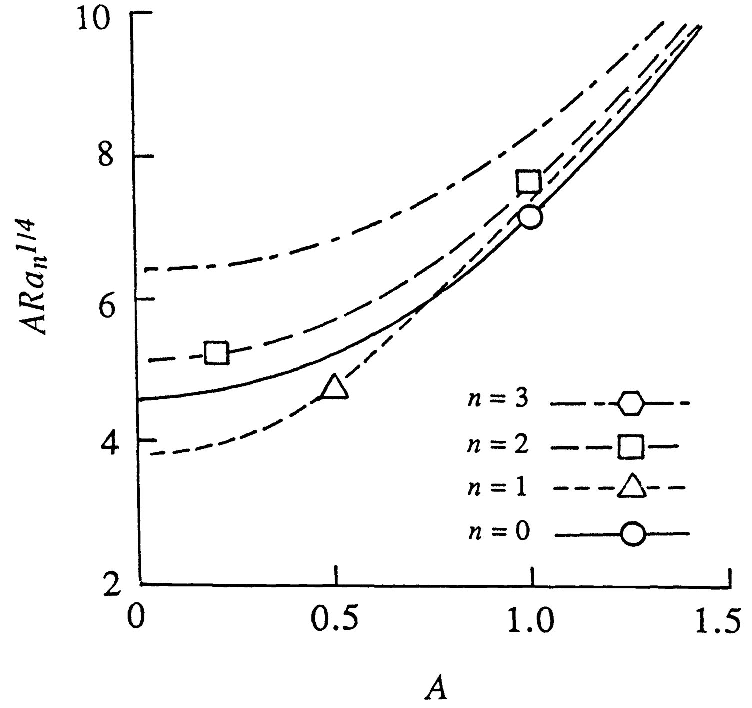


FIG. 4. *Comparison of selected results from the present computations (symbols) with those of Hardin et al.* (1990) *(curves) for $A < 1.5$ with a conducting sidewall.*

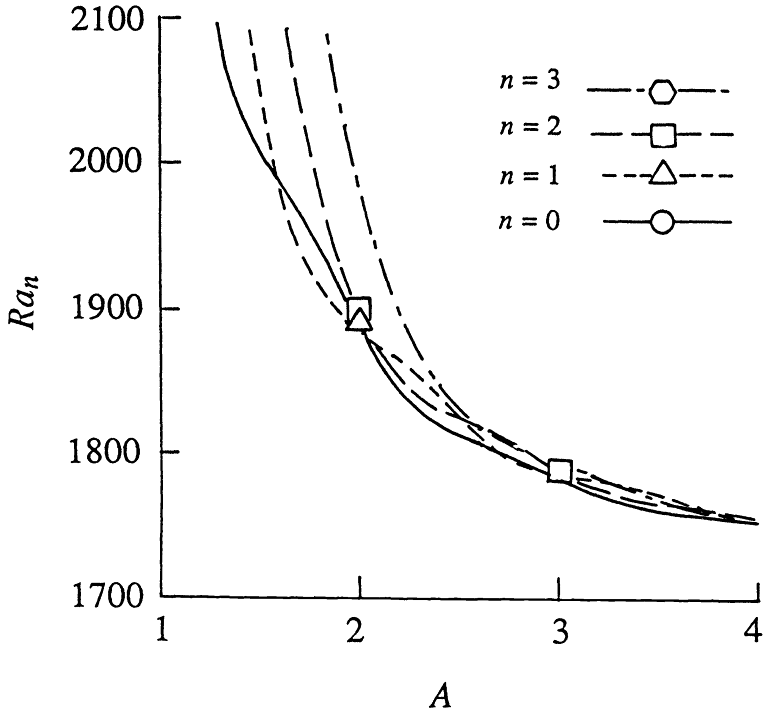


FIG. 5. *Comparison of selected results from the present computations (symbols) with those of Hardin* et al. (1990) *(curves) for $1 < A < 4$ with a conducting sidewall.*

Galerkin computations of Hardin, Sani, Henry, and Roux (1990) have obtained revised values of Ra_n; these are presented in the figures. Agreement is satisfactory in all cases, leading to the conclusion that the algorithm is implemented properly and is capable of performing such computations.

The confidence gained from the successful tests for the heated cylinder provides a basis for the extension of these ideas to the more difficult three-dimensional problem associated with the stability of thermocapillary convection in a model of the float-zone crystal-growth process. This problem will require, for fixed n, two additional outer iterations to find the energy-stability limit Ma_E (Ma denotes the dimensionless Marangoni number). First, as in (2.10), both matrices will depend on the coupling parameter $\lambda > 0$ requiring maximization with respect to this parameter. Second, the matrix $\overline{\mathbf{B}}$ will depend on the basic state and, in particular, on its Marangoni number Ma_{BS}. An accelerated fixed-point iteration method can be used to enforce the required condition $Ma^* = Ma_{BS}$. These two iterations form an intermediate level of the solution procedure with the inverse iteration being the inner, and a final minimization of the Ma_n with respect to the discrete wave number being the outer, iteration level.

A technologically significant problem associated with materials processing has driven the creation of a new numerical algorithm for finding the smallest positive eigenvalue of a sparse, indefinite, Hermitian system. Early attempts by Shen et al. (1990) to employ the QZ algorithm for the simpler problem of a real, symmetric system were only partially successful because of an inability to obtain adequate resolution due to restrictions on system order imposed by available computational resources. This difficulty was only resolved with the development of the new algorithm. The three-dimensional stability calculations that are now underway are necessary, both to obtain a complete stability picture for the problem, and also because laboratory experiments indicate that nonaxisymmetric modes are actually those responsible for causing instability in some cases. The algorithm has been successful in solving these technologically related problems; these, in turn, have provided motivation for further research in numerical analysis.

Acknowledgment. The authors thank the referees for their helpful comments.

REFERENCES

R. BANK AND H. D. MITTELMANN (1986), *Continuation and multigrid for nonlinear elliptic systems*, Lecture Notes in Mathematics 1228, Multigrid Methods II, W. Hackbusch and U. Trottenberg, eds., Springer-Verlag, Berlin, New York.

G. K. BATCHELOR AND A. E. GILL (1962), *Analysis of the stability of axisymmetric jets*, J. Fluid Mech. 14, 529–551.

G. S. CHARLSON AND R. L. SANI (1970), *Thermoconvective instability in a bounded cylindrical fluid layer*, Internat. J. Heat Mass Trans. 13, 1479–1496.

——— (1971), *On thermoconvective instability in a bounded cylindrical fluid layer*, Internat J. Heat Mass Trans. 14, 2157–2160.

S. H. DAVIS (1971), *Energy stability of unsteady flows*, in Proc. IUTAM Symposium on Unsteady Boundary Layers.

S. H. DAVIS AND C. VON KERCZEK (1973), *A reformulation of energy stability theory*, Arch. Rational Mech. Anal. 52, 112–117.

G. R. HARDIN, R. L. SANI, R. HENRY, AND B. ROUX (1990), *Buoyancy-driven instability in a vertical cylinder: Binary fluids with soret effect. Part* I: *General theory and stationary stability results*, Internat. J. Numer. Meth. Fluids 10, 79–117.

D. D. JOSEPH (1976), *Stability of Fluid Motions* I, II, Springer-Verlag, Berlin.

W. KERNER (1989), *Large-scale complex eigenvalue problems*, J. Comput. Phys. 85, 1–85.

H. D. MITTELMANN, C. LAW, D. F. JANKOWSKI, AND P. G. NEITZEL (1991), *Stability of thermocapillary convection in float-zone crystal growth*, Numerical Methods for Free Boundary Problems, ISNM-99, P. Neittaanmaki, ed., Birkhäuser-Verlag, Basel.

R. B. MORGAN AND D. S. SCOTT (1986), *Generalizations of Davidson's method for computing eigenvalues of sparse symmetric matrices*, SIAM J. Sci. Statist. Comput. 7, 817–825.

C. C. PAIGE AND M. A. SAUNDERS (1975), *Solution of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal. 12, 617–629.

F. PREISSER, P. SCHWABE, AND A. SCHARMANN (1983), *Steady and oscillatory thermocapillary convection in liquid columns with free cylindrical surface*, J. Fluid Mech. 126, 545–567.

R. L. SANI, *private communication.*

Y. SHEN, G. P. NEITZEL, D. F. JANKOWSKI, AND H. D. MITTELMANN (1990), *Energy stability of thermocapillary convection in a model of the float-zone, crystal-growth process*, J. Fluid Mech. 217, 639–660.

S. C. SINHA AND S. CARMI (1976), *On the Liapunov–Movchan and the energy theories of stability*, J. Appl. Math. Phys. 27, 607–612.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 1, pp. 425–448, January 1992

CONJUGATE GRADIENT-TYPE METHODS FOR LINEAR SYSTEMS WITH COMPLEX SYMMETRIC COEFFICIENT MATRICES*

ROLAND W. FREUND†

Abstract. Conjugate gradient-type methods for the solution of large sparse linear systems $Ax = b$ with complex symmetric coefficient matrices $A = A^T$ are considered. Such linear systems arise in important applications, such as the numerical solution of the complex Helmholtz equation. Furthermore, most complex non-Hermitian linear systems which occur in practice are actually complex symmetric. Conjugate gradient-type iterations which are based on a variant of the nonsymmetric Lanczos algorithm for complex symmetric matrices are investigated. In particular, a new approach with iterates defined by a quasi-minimal residual property is proposed. The resulting algorithm presents several advantages over the standard biconjugate gradient method. Some remarks are also included on the obvious approach to general complex linear systems by solving equivalent real linear systems for the real and imaginary parts of x. Finally, numerical experiments for linear systems arising from the complex Helmholtz equation are reported.

Key words. complex symmetric matrices, nonsymmetric Lanczos algorithm, biconjugate gradients, minimal residual property

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

1. Introduction. Conjugate gradient-type methods—used in combination with preconditioning—are among the most effective iterative procedures for solving large sparse nonsingular systems of linear equations

$$Ax = b. \tag{1.1}$$

The archetype of these schemes is the classical conjugate gradient algorithm (CG hereafter) of Hestenes and Stiefel [24] for Hermitian positive definite matrices A.

While most linear systems that arise in practice have real coefficient matrices A and real right-hand sides b, there are some important applications (see [16] and the references therein) that lead to complex linear systems. Partial differential equations that model dissipative processes usually involve complex coefficient functions and/or complex boundary conditions (see, e.g., [29]), and discretizing them yields linear systems with complex matrices A. A typical example for this category is the complex Helmholtz equation

$$-\Delta u - \sigma_1 u + i\sigma_2 u = f, \tag{1.2}$$

where σ_1, σ_2 are real coefficient functions, which describes the propagation of damped time-harmonic waves as, e.g., electromagnetic waves in conducting media. Further applications, which give rise to complex linear systems, include discretizations of the time-dependent Schrödinger equation using implicit difference schemes, inverse

*Received by the editors April 5, 1990; accepted for publication (in revised form) November 5, 1990. This work was supported in part by the Defense Advanced Research Projects Agency via Cooperative Agreement NCC 2-387 between the National Aeronautics and Space Administration (NASA) and the Universities Space Research Association

† Research Institute for Advanced Computer Science, Mail Stop Ellis Street, NASA Ames Research Center, Moffett Field, California 94035, and Institut für Angewandte Mathematik und Statistik, Universität Würzburg, D–8700 Würzburg, Federal Republic of Germany.

scattering problems, underwater acoustics, eddy current computations [2], numerical computations in quantum chromodynamics, and numerical conformal mapping.

In all these examples, the resulting coefficient matrices A are non-Hermitian. However, they still exhibit special structures. Often, A differs from a Hermitian matrix only by a shift and a rotation:

$$A = e^{i\theta}(T + i\sigma I), \quad T = T^H \quad \text{Hermitian}, \quad \sigma \in \mathbb{C},\ \theta \in \mathbb{R}, \quad i := \sqrt{-1}. \tag{1.3}$$

In almost all other cases, which lead to complex systems, the coefficient matrix is symmetric:

$$A = A^T \quad \text{is complex symmetric.} \tag{1.4}$$

Note that the two families (1.3) and (1.4) overlap. The matrix (1.3) is complex symmetric if and only if T is real.

Surprisingly, when complex linear systems (1.1) are solved in practice, usually no attempt is made to exploit the special structures (1.3) or (1.4). Indeed, there are two popular approaches. The first one (see, e.g., [1]) is to apply preconditioned CG to the Hermitian positive definite normal equations

$$A^H A x = A^H b. \tag{1.5}$$

Of course, complex numbers can always be avoided by rewriting (1.1) as a real linear system for the real and imaginary parts of x. The second popular approach is to solve this real and, in general, nonsymmetric linear system by one of the generalized CG methods such as GMRES [40]. It turns out that in both cases the resulting iterative schemes tend to converge slowly. As a consequence, complex linear systems have the bad reputation of being difficult to solve by CG-type methods. On the other hand, for the class of shifted Hermitian matrices (1.3), there are efficient CG-type algorithms [10], [11], [28], [16] for complex linear systems in their original form (1.1). We refer the reader to [16] for a detailed study and practical aspects of these schemes. In [16] it is also shown how the special structure (1.3) can be preserved by using polynomial preconditioning.

In this paper, we are mainly concerned with CG-type methods for linear systems (1.1) with coefficient matrices of the second class (1.4). In particular, we consider approaches based on a variant of the nonsymmetric Lanczos algorithm, which was successfully used for computing eigenvalues of complex symmetric matrices (see [35] and [6, Chap. 6]). This Lanczos recursion generates basis vectors for the Krylov subspace induced by A that are orthogonal with respect to a certain indefinite inner product. The standard way to obtain from this basis iterates, which approximate the exact solution of (1.1), is to enforce a biconjugate gradient (BCG hereafter) condition. Here, we propose a new approach that generates iterates via a quasi-minimal residual property. On typical examples, the resulting algorithm displays better convergence properties than the BCG approach. In particular, it produces residuals whose norms are almost monotonically decreasing, in contrast to the erratic convergence behavior that is typical for BCG. Moreover, the new technique eliminates one of the two sources of possible breakdown in the BCG approach.

The outline of the paper is as follows. In §2, we review the Lanczos recursion and related algorithms for complex symmetric matrices. Also, some theoretical properties which will be needed later on are listed. In §3, we propose the quasi-minimal residual approach for complex symmetric matrices. Section 4 contains some remarks on the problem of breakdown of the complex symmetric Lanczos recursion. In §5, we are concerned with the issue of "complex versus equivalent real linear systems."In particular, some results are presented which indicate that for Krylov subspace methods, such as CG-type algorithms, it is always preferable to solve the original complex system rather than equivalent real ones. In §6, some typical results of numerical experiments for linear systems arising from finite difference approximations to the complex Helmholtz equation (1.2) are given. Finally, in §7, we make some concluding remarks.

Throughout the paper, all vectors and matrices, unless stated otherwise, are assumed to be complex. As usual, $\overline{M} = [\overline{m_{jk}}]$, $M^T = [m_{kj}]$, and $M^H = \overline{M}^T$ denote the complex conjugate, transpose, and Hermitian matrix, respectively, corresponding to the matrix $M = [m_{jk}]$. Moreover, we set $\operatorname{Re} M = (M + \overline{M})/2$ and $\operatorname{Im} M = (M - \overline{M})/(2i)$ for its real and imaginary part, respectively. The notation

$$K_k(c, B) := \operatorname{span}\{c, Bc, \cdots, B^{k-1}c\}$$

is used for the kth Krylov subspace of $\mathbb{C}^n$ generated by $c \in \mathbb{C}^n$ and the $n \times n$ matrix B. Furthermore, $\sigma(B)$ denotes the spectrum of B. The vector norm $\|x\| = \sqrt{x^H x}$ is always the Euclidean norm. The set of all complex polynomials of degree at most k is denoted by

$$\Pi_k := \{p(\lambda) \equiv \gamma_0 + \gamma_1\lambda + \cdots + \gamma_k\lambda^k \mid \gamma_0, \gamma_1, \cdots, \gamma_k \in \mathbb{C}\}.$$

Moreover, the coefficient matrix A of (1.1) is always $n \times n$ and, unless stated otherwise, assumed to be complex symmetric. Generally, $x_k \in \mathbb{C}^n$, $k = 0, 1, \cdots$, denote iterates for (1.1) with corresponding residual vectors $r_k := b - Ax_k$. If necessary, quantities of different algorithms will be distinguished by superscripts, e.g., x_k^{BCG} and x_k^{QMR}. Finally, an iterative scheme for solving (1.1) is called a Krylov subspace method if its iterates are of the form

$$x_k \in x_0 + K_k(r_0, A) \quad \text{or, equivalently,} \quad x_k = x_0 + P(A)r_0, \; P \in \Pi_{k-1}. \tag{1.6}$$

Note that, in particular, CG-type algorithms for (1.1) fall into this category.

2. The Lanczos recursion and related algorithms for complex symmetric matrices.

The Lanczos procedure [31] for general complex $n \times n$ matrices A consists of two three-term recurrences (see, e.g., [43, pp. 388–394]). As Lanczos pointed out [33, p. 176], the general method reduces to only one recursion if A is Hermitian, respectively, complex symmetric. In particular, for these two special cases, work and storage of the general Lanczos method are halved. The resulting Hermitian Lanczos recursion has been studied extensively (see [20, Chap. 9] and the references therein). In contrast, the literature on the complex symmetric variant is scarce and restricted to the application of the algorithm to computing eigenvalues of complex symmetric matrices (see Moro and Freed [35] and Cullum and Willoughby [6, Chap. 6]). Here we hope to convince the reader that the complex symmetric Lanczos algorithm is also very useful for solving linear systems.

2.1. The Lanczos recursion.

The basic method is as follows:

ALGORITHM 2.1 (LANCZOS METHOD FOR $A = A^T$).

```
(1) Start:
    • Choose $r_0 \in \mathbb{C}^n$, $r_0 \neq 0$;
    • Set $\tilde{v}_1 = r_0$ and $v_0 = 0$.

(2) For $k = 1, 2, \cdots$ do:
    • Compute $\beta_k = (\tilde{v}_k^T \tilde{v}_k)^{1/2}$;
```

- *If* $\beta_k = 0$*: Set* $m_0 = k - 1$*, and stop;*
- *Otherwise, set* $v_k = \tilde{v}_k / \beta_k$*;*
- *Compute* $\alpha_k = v_k^T A v_k$*;*
- *Set* $\tilde{v}_{k+1} = Av_k - \alpha_k v_k - \beta_k v_{k-1}$.

In the following proposition, some elementary properties of Algorithm 2.1 are listed; proofs can be found in [5, Chap. 6]. We set

$$(2.1) \qquad V_k := [\, v_1 \quad v_2 \quad \dots \quad v_k \,] \quad \text{and} \quad T_k := \begin{bmatrix} \alpha_1 & \beta_2 & 0 & \dots & 0 \\ \beta_2 & \alpha_2 & \beta_3 & \ddots & \vdots \\ 0 & \beta_3 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \beta_k \\ 0 & \dots & 0 & \beta_k & \alpha_k \end{bmatrix}.$$

Moreover, $m_\star = m_\star(r_0, A) := \dim K_n(r_0, A)$ denotes the *grade* of r_0 with respect to A (cf. [43, p. 37]). We remark that $m_\star \geq 1$ is the smallest integer such that $K_{m_\star}$ is an A-invariant subspace of $\mathbb{C}^n$. Equivalently, if A is nonsingular and $r_0 = b - Ax_0$, $m_\star \geq 1$ is the smallest integer such that

$$(2.2) \qquad A^{-1}b \in x_0 + K_{m_\star}(r_0, A).$$

PROPOSITION 2.2. (a) *In exact arithmetic, Algorithm* 2.1 *stops after a finite number of steps* $k = m_0 + 1$ *and* $0 \leq m_0 \leq m_\star$*. Furthermore,* $\tilde{v}_{m_0+1} = 0$ *if* $m_0 = m_\star$ *("regular termination"), and* $\tilde{v}_{m_0+1} \neq 0$ *if* $m_0 < m_\star$ *("breakdown").*

(b) *For* $k = 1, 2, \cdots, m_0$*:*

$$(2.3) \qquad v_k^T v_j = \begin{cases} 0, & \text{if } k \neq j \\ 1, & \text{if } k = j \end{cases}, \quad j = 1, 2, \cdots, m_0,$$

$$(2.4) \qquad K_k(r_0, A) = \operatorname{span}\{v_1, v_2, \cdots, v_k\},$$

$$(2.5) \qquad AV_k = V_k T_k + [\, 0 \quad 0 \quad \cdots \quad 0 \quad \tilde{v}_{k+1} \,].$$

Notice that, by (2.3)–(2.4), the Lanczos vectors $v_1, \cdots, v_k$ form an orthonormal basis for $K_k(r_0, A)$ with respect to the indefinite bilinear form

$$(2.6) \qquad (x, y) := y^T x, \quad x, y \in \mathbb{C}^n.$$

In particular, if Algorithm 2.1 terminates regularly, it generates a basis of the affine space $x_0 + K_{m_\star}(r_0, A)$, which, in view of (2.2), contains the exact solution of $Ax = b$.

Note that the indefinite bilinear form (2.6) is the proper (cf. Craven [5]) "inner product" for complex symmetric matrices. Unfortunately, it has the defect that there exist vectors $v \in \mathbb{C}^n$ which are *quasi-null* [5], i.e., $(v, v) = 0$, but $v \neq 0$. Consequently, it cannot be excluded that Algorithm 2.1 actually breaks down. Indeed, in view of part (a) of Proposition 2.2, a breakdown occurs if we encounter a quasi-null vector $\tilde{v}_k$. The phenomenon of breakdown will be discussed further in §4.

Finally, we remark that there is a close connection (see Theorem 2.4 in [15]) of the complex symmetric variant 2.1 with the Lanczos algorithm for general matrices. Unlike Hermitian matrices, complex symmetric matrices do not have any special spectral properties. Indeed (see, e.g., [25, Thm. 4.4.9]), any complex $n \times n$ matrix is similar to a complex symmetric matrix. This result entails that the general nonsymmetric Lanczos method differs from the complex symmetric Algorithm 2.1 only in an additional starting vector s_0, which can be chosen independently of r_0 in 2.1.

After these preliminaries, we now turn to linear systems (1.1). In the sequel, it is always assumed that A is nonsingular.

2.2. Related algorithms. In his celebrated papers [31], [32], Lanczos also proposed a scheme, closely related to the nonsymmetric Lanczos process, for solving general non-Hermitian linear systems, namely, the biconjugate gradient algorithm (BCG). We refer the reader to [12], [39], [26] for a detailed discussion of the BCG approach.

Like the nonsymmetric Lanczos method, BCG for general linear systems is started with two vectors: the residual $r_0 = b - Ax_0$ of the initial guess x_0 and a second vector $s_0 \neq 0$. We remark that s_0 is still unspecified. Due to the lack of a criterion for the choice of s_0, we usually set $s_0 = r_0$ in practice. For the case of complex symmetric matrices A, it is straightforward to show that, in analogy to the complex symmetric variant 2.1 of the general Lanczos process, the choice $s_0 = r_0$ results in a scheme which requires only half the work and storage of general BCG. The resulting procedure is as follows:

ALGORITHM 2.3. (BCG *for* $A = A^T$)

```
(1) Start:
    • Choose $x_0 \in \mathbb{C}^n$;
    • Set $p_0 = r_0 = b - Ax_0$ and compute $r_0^T r_0$.

(2) For $k = 1, 2, \cdots$ do:
    • Compute $Ap_{k-1}$ and $p_{k-1}^T A p_{k-1}$;
    • If $p_{k-1}^T A p_{k-1} = 0$ or $r_{k-1}^T r_{k-1} = 0$: Set $m_1 = k - 1$, and stop;
    • Otherwise, set $\delta_k = r_{k-1}^T r_{k-1} / p_{k-1}^T A p_{k-1}$;
    • Compute $x_k = x_{k-1} + \delta_k p_{k-1}$ and $r_k = r_{k-1} - \delta_k A p_{k-1}$;
    • Compute $r_k^T r_k$ and set $\rho_k = r_k^T r_k / r_{k-1}^T r_{k-1}$;
    • Compute $p_k = r_k + \rho_k p_{k-1}$.
```

Note that the *transpose* in all dot products in Algorithm 2.3 is essential. In particular, Algorithm 2.3 is different from the classical CG method for positive definite matrices $A = A^H$.

In the sequel, BCG always refers to the complex symmetric Algorithm 2.3. Next, we list some basic properties of BCG that will be needed in subsequent sections. These results are immediate consequences of results (e.g., Jacobs [26]) for the general biconjugate gradient method.

PROPOSITION 2.4. (a) *In exact arithmetic, Algorithm* 2.3 *stops after a finite number of steps* $k = m_1 + 1$ *and* $0 \leq m_1 \leq m_\star$. *Furthermore,* $x_{m_1} = A^{-1}b$ *if* $m_1 = m_\star$ (*"regular termination"*), *and* $x_{m_1} \neq A^{-1}b$ *if* $m_1 < m_\star$ (*"breakdown"*).

(b) *For* $k = 1, 2, \cdots, m_1$:

$$r_{k-1}^T r_{j-1} = 0, \qquad k \neq j, \quad j = 1, 2, \cdots, m_1, \tag{2.7}$$

$$K_k(r_0, A) = \operatorname{span}\{r_0, r_1, \cdots, r_{k-1}\}. \tag{2.8}$$

(c) *Let* $k \in \{1, 2, \cdots, m_1\}$. *Then,* x_k *is uniquely determined by the Galerkin condition*

$$(b - Ax_k)^T y = 0 \quad \textit{for all} \quad y \in K_k(r_0, A), \qquad x_k \in x_0 + K_k(r_0, A), \tag{2.9}$$

with respect to the inner product (2.6).

By comparing (2.7)–(2.8) with (2.3)–(2.4), we conclude that r_{k-1} is parallel to the Lanczos vector v_k generated by Algorithm 2.1. More precisely, we easily verify that

$$r_{k-1} = (-1)^{k-1}\delta_1 \cdots \delta_{k-1}\beta_1 \cdots \beta_{k-1}\beta_k v_k, \quad k = 1, 2, \cdots, m_1. \tag{2.10}$$

Notice that there are two different causes for breakdown of Algorithm 2.3. The first one, namely, the occurrence of a quasi-null residual vector r_{k-1}, is, in view of (2.10), equivalent to the breakdown of the complex symmetric Lanczos Algorithm 2.1. In addition, Algorithm 2.3 breaks down if we encounter a search direction $p_{k-1} \neq 0$ with $p_{k-1}^T A p_{k-1} = 0$. This second cause of breakdown is more severe than the first one. As we will see in §3, it occurs if no Galerkin iterate (2.9) exists.

Closely related to the biconjugate gradient method for general linear systems (1.1) is the conjugate gradients squared algorithm (CGS hereafter) that was recently proposed by Sonneveld [41].

ALGORITHM 2.5. (CGS *for general* A)

```
(1) Start:
    • Choose $x_0 \in \mathbb{C}^n$ and $s_0 \in \mathbb{C}^n$, $s_0 \neq 0$;
    • Set $p_0 = u_0 = r_0 = b - Ax_0$ and compute $s_0^T r_0$.

(2) For $k = 1, 2, \cdots$ do:
    • Compute $Ap_{k-1}$ and $s_0{}^T Ap_{k-1}$;
    • If $s_0{}^T Ap_{k-1} = 0$ or $s_0{}^T r_{k-1} = 0$: Stop;
    • Otherwise, set $\alpha_k = s_0^T r_{k-1} / s_0^T Ap_{k-1}$;
    • Compute $q_k = u_{k-1} - \alpha_k Ap_{k-1}$;
    • Compute $x_k = x_{k-1} + \alpha_k(u_{k-1} + q_k)$ and $r_k = r_{k-1} - \alpha_k A(u_{k-1} + q_k)$;
    • Compute $s_0^T r_k$ and set $\beta_k = s_0^T r_k / s_0^T r_{k-1}$;
    • Compute $u_k = r_k + \beta_k q_k$ and $p_k = u_k + \beta_k(q_k + \beta_k p_{k-1})$.
```

Notice that, like general BCG, CGS has a second unspecified starting vector s_0. However, unlike BCG, even with the special choice $s_0 = r_0$, CGS cannot exploit the complex symmetry of A. In particular, for $A = A^T$, Algorithm 2.5 requires per iteration about twice as much work as the BCG Algorithm 2.3.

Finally, as a special case of the general connection [41] between the CGS and BCG approaches, we have the following result.

PROPOSITION 2.6. *Let* $A = A^T$, $r_0 = r_0^{\mathrm{BCG}} = r_0^{CGS}$, *and, in Algorithm* 2.5, $s_0 = r_0$. *Then, for* $k = 0, 1, \cdots, m_1$,

$$r_k^{\mathrm{BCG}} = p_k(A) r_0 \quad \text{and} \quad r_k^{CGS} = \big(p_k(A)\big)^2 r_0$$

for some $p_k \in \Pi_k$ *with* $p_k(0) = 1$.

3. A quasi-minimal residual algorithm for complex symmetric matrices.

In this section, we propose a new approach for solving complex symmetric linear systems. The method is based on the complex symmetric Lanczos Algorithm 2.1. For simplicity, we assume throughout this section that, in exact arithmetic, Algorithm 2.1 terminates regularly, i.e.,

$$\beta_k \neq 0 \quad \text{for} \quad k = 1, 2, \cdots, m_\star, \quad \beta_{m_\star + 1} = 0. \tag{3.1}$$

Moreover, always let $k \in \{1, 2, \cdots, m_\star\}$ in the following.

3.1. Basic approach. Let x_k be the kth iterate of any Krylov subspace method (1.6). Then, by (2.4) and with V_k as defined in (2.1), we have

$$x_k = x_0 + V_k z_k \quad \text{where} \quad z_k \in \mathbb{C}^k. \tag{3.2}$$

Using (2.5) and $r_0 = \beta_1 v_1$, it follows from (3.2) that

$$r_k = b - Ax_k = r_0 - AV_k z = \beta_1 v_1 - V_{k+1}\tilde{T}_k z_k = V_{k+1}\big(\beta_1 e_1 - \tilde{T}_k z_k\big). \tag{3.3}$$

Here, $e_1 := [1 \quad 0 \quad \cdots \quad 0]^T$ denotes the first unit vector,

$$\tilde{T}_k := \begin{bmatrix} T_k \\ \beta_{k+1} e_k^T \end{bmatrix} \quad \text{with} \quad e_k^T := [0 \quad \cdots \quad 0 \quad 1], \tag{3.4}$$

and, if $k = m_*$, $v_{m_*+1} := 0$. Recall that T_k was defined in (2.1).

Clearly, the aim is to choose z_k in (3.2)–(3.3) such that $r_k \approx 0$ as good as possible. In the BCG approach, this is attempted by enforcing the Galerkin condition (2.9). Using (2.3)–(2.4) and (3.3), we easily verify that (2.9) holds if and only if r_k and v_{k+1} are parallel or, equivalently, z_k is a solution of the linear system

$$T_k z = \beta_1 e_1. \tag{3.5}$$

Note that, by (3.1), (3.5) is inconsistent if T_k is singular. Thus we have the following result.

PROPOSITION 3.1. *A* BCG *iterate x_k^{BCG} satisfying the Galerkin condition* (2.9) *exists if and only if T_k is nonsingular. Moreover, if it exists, it is unique and given by*

$$x_k^{\text{BCG}} = x_0 + V_k z_k \quad \textit{and} \quad r_k = -\beta_{k+1}(z_k)_k v_{k+1} \tag{3.6}$$

where z_k is the solution of (3.5) *and $(z_k)_k$ denotes its kth component.*

Proposition 3.1 demonstrates the defects of the BCG approach. Simple examples show that singular T_k may indeed occur, and then, in view of Proposition 2.4, the BCG Algorithm 2.3 would break down in exact arithmetic. In floating-point arithmetic, such a breakdown is unlikely to happen. However, T_k may still be close to singular and then the Galerkin condition (2.9) defines a poor approximation to the true solution of (1.1). This is the reason for the typical erratic convergence behavior with wildly oscillating residual norms.

Obviously, the question arises as to whether there is a better strategy than (2.9) for choosing z_k in (3.2)–(3.3). Ideally, we would like to have the minimal residual (MR) property

$$\|b - Ax_k\| = \min_{x \in x_0 + K_k(r_0, A)} \|b - Ax\| = \min_{z \in \mathbb{C}^k} \|V_{k+1}\big(\beta_1 e_1 - \tilde{T}_k z_k\big)\|. \tag{3.7}$$

However, by (2.3), in general (see Theorem 3.4 for an exception) the columns of V_{k+1} are orthonormal only with respect to (2.6) rather than the Euclidean inner product $(x, y) = y^H x$. Consequently, solving the least-squares problem on the right-hand side of (3.7) results in an algorithm for which work and storage per iteration step k grows linearly with k. Hence, if we insist on a "true" iterative scheme with constant work and storage per iteration, this excludes the MR method.

Here, we propose the *quasi-minimal residual* (QMR) approach as a substitute for (3.7). Let

$$\Omega_{k+1} = \operatorname{diag}(\omega_1, \omega_2, \cdots, \omega_{k+1}) \quad \text{with} \quad \omega_j > 0 \quad \text{for all} \quad j$$

be a given positive diagonal weight matrix and rewrite (3.3) in the form

$$r_k = \big(V_{k+1}\Omega_{k+1}^{-1}\big)\big(\omega_1 \beta_1 e_1 - \Omega_{k+1}\tilde{T}_k z_k\big). \tag{3.8}$$

Instead of $\|r_k\|$ as in (3.7), we may at least minimize the vector of components in the representation (3.8) of r_k:

$$\min_{z\in\mathbb{C}^k} \|\omega_1\beta_1 e_1 - \Omega_{k+1}\tilde{T}_k z\|. \tag{3.9}$$

Hence, we define the iterates of the QMR method as follows:

$$x_k = x_k^{\mathrm{QMR}} = x_0 + V_k z_k \quad \text{where } z_k \in \mathbb{C}^k \text{ is the solution of (3.9).} \tag{3.10}$$

Notice that $\Omega_{k+1}\tilde{T}_k$ is a $(k+1)\times k$ matrix which, by (3.4) and (3.1), has full rank. Thus, the least squares problem (3.9) always has a unique solution z_k.

Clearly, the QMR approach still depends on the weights ω_j. A natural choice is

$$\omega_j = \|v_j\|, \quad j = 1, 2, \cdots, k+1, \tag{3.11}$$

so that all basis vectors v_j/ω_j in the representation (3.8) of r_k have Euclidean length 1. Our numerical tests (cf. §6) also confirmed (3.11) as the best strategy.

3.2. Practical implementation. Next we present an algorithm for the actual computation of the QMR iterates (3.10). The derivation is similar to that of Paige and Saunders' SYMMLQ and MINRES algorithms [36] for real symmetric matrices.

By (3.4), (2.1), and (3.1), $\Omega_{k+1}\tilde{T}_k$ is a tridiagonal $(k+1)\times k$ matrix with full column rank. Hence it admits a QR factorization of the type

$$Q_{k+1}\Omega_{k+1}\tilde{T}_k = \begin{bmatrix} R_k \\ 0 \end{bmatrix} \tag{3.12}$$

where Q_{k+1} is a unitary $(k+1)\times(k+1)$ matrix and R_k a nonsingular matrix of the form

$$R_k := \begin{bmatrix} \zeta_1 & \eta_2 & \theta_3 & 0 & \dots & 0 \\ 0 & \zeta_2 & \eta_3 & \ddots & \ddots & \vdots \\ 0 & \ddots & \zeta_3 & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \theta_k \\ \vdots & & & \ddots & \ddots & \eta_k \\ 0 & \dots & \dots & \dots & 0 & \zeta_k \end{bmatrix}. \tag{3.13}$$

The decomposition (3.12) can be generated by means of a series of k complex Givens rotations (e.g., [20, p. 47])

$$Q(c_j, s_j) = \begin{bmatrix} c_j & \overline{s_j} \\ -s_j & c_j \end{bmatrix}, \quad c_j \in \mathbb{R}, s_j \in \mathbb{C}, \quad c_j^2 + |s_j|^2 = 1, \quad j = 1, \cdots, k.$$

In particular, (3.12) is easily updated from the factorization $Q_k\Omega_k\tilde{T}_{k-1} = R_{k-1}$ of the previous step by setting

$$Q_{k+1} = \begin{bmatrix} I_{k-1} & 0 \\ 0 & Q(c_k, s_k) \end{bmatrix} \begin{bmatrix} Q_k & 0 \\ 0 & 1 \end{bmatrix} \tag{3.14}$$

and computing c_k, s_k and the new elements θ_k, η_k, ζ_k of R_k as follows:

$$\begin{aligned} &\theta_k = \overline{s_{k-2}}\omega_{k-1}\beta_k, \qquad \eta_k = c_{k-1}c_{k-2}\omega_{k-1}\beta_k + \overline{s_{k-1}}\omega_k\alpha_k, \\ &\tilde{\zeta}_k = c_{k-1}\omega_k\alpha_k - s_{k-1}c_{k-2}\omega_{k-1}\beta_k, \qquad |\zeta_k| = \left(|\tilde{\zeta}_k|^2 + \omega_{k+1}^2|\beta_{k+1}|^2\right)^{1/2}, \\ &\zeta_k = \begin{cases} |\zeta_k|\tilde{\zeta}_k/|\tilde{\zeta}_k|, & \text{if } \tilde{\zeta}_k \neq 0, \\ |\zeta_k|, & \text{if } \tilde{\zeta}_k = 0, \end{cases} \qquad c_k = \tilde{\zeta}_k/\zeta_k, \quad s_k = \omega_{k+1}\beta_{k+1}/\zeta_k. \end{aligned} \tag{3.15}$$

By (3.12) and since Q_{k+1} is unitary, (3.9) is equivalent to

$$\min_{z\in\mathbb{C}^k} \left\| \omega_1\beta_1 Q_{k+1}e_1 - \begin{bmatrix} R_k \\ 0 \end{bmatrix} z \right\|. \tag{3.16}$$

From (3.10) and (3.16) it follows that

$$x_k = x_0 + V_k z_k \quad \text{where} \quad z_k := R_k^{-1}t_k, \quad \begin{bmatrix} t_k \\ \tilde{\tau}_{k+1} \end{bmatrix} = \tilde{t}_{k+1} := \omega_1\beta_1 Q_{k+1}e_1. \tag{3.17}$$

Notice that, in view of (3.14), t_k differs from the previous vector t_{k-1} only by its kth component $\tau_k := (t_k)_k = c_k\tilde{\tau}_k$. Next, we define vectors p_j via

$$[p_1 \quad p_2 \quad \cdots \quad p_k] := V_k R_k^{-1}. \tag{3.18}$$

Finally, using (3.17)–(3.18) and (3.13), we obtain the recursion

$$x_k = x_{k-1} + \tau_k p_k, \quad \text{where} \quad p_k = \frac{1}{\zeta_k}\Big(v_k - \eta_k p_{k-1} - \theta_k p_{k-2}\Big),$$

for the QMR iterates. In combination with Algorithm 2.1, the following implementation results:

ALGORITHM 3.2 (QMR METHOD).

```
(1) Start:
    • Choose $x_0 \in \mathbb{C}^n$;
    • Set $\tilde{v}_1 = b - Ax_0$, $v_0 = p_0 = p_{-1} = 0$;
    • Set $\beta_1 = \left(\tilde{v}_1^T\tilde{v}_1\right)^{1/2}$, $\tilde{\tau}_1 = \omega_1\beta_1$, $c_0 = c_{-1} = 1$, and $s_0 = s_{-1} = 0$.

(2) For $k = 1, 2, \cdots$ do:
    • If $\beta_k = 0$, stop: $x_{k-1}$ solves $Ax = b$.
    • Otherwise, compute $v_k = \tilde{v}_k/\beta_k$ and $\alpha_k = v_k^T A v_k$;
    • Set $\tilde{v}_{k+1} = Av_k - \alpha_k v_k - \beta_k v_{k-1}$, $\beta_{k+1} = \left(\tilde{v}_{k+1}^T\tilde{v}_{k+1}\right)^{1/2}$;
    • Compute $\theta_k$, $\eta_k$, $\zeta_k$, $c_k$, and $s_k$, using formulas (3.15);
    • Set $p_k = (v_k - \eta_k p_{k-1} - \theta_k p_{k-2})/\zeta_k$;
    • Set $\tau_k = c_k\tilde{\tau}_k$, $\tilde{\tau}_{k+1} = -s_k\tilde{\tau}_k$;
    • Compute $x_k = x_{k-1} + \tau_k p_k$.
```

The assumption (3.1) guarantees that, in exact arithmetic, Algorithm 3.2 stops for $k = m_\star + 1$ and, by (2.2), x_{k-1} is indeed the solution of (1.1) then. However, in floating-point arithmetic, this finite termination property of the Lanczos recursion is no longer valid, and the stopping criterion stated in Algorithm 3.2 is not useful in practice. Instead, we should terminate the iteration as soon as $\|r_k\|$ is sufficiently reduced. We remark that r_k is not directly available in Algorithm 3.2. However, in view of (3.19), if we update one additional auxiliary vector, namely,

$$h_k = h_{k-1} + \frac{c_k\tilde{\tau}_{k+1}}{|s_1s_2\cdots s_k|^2\omega_{k+1}}v_{k+1}, \quad h_0 := r_0,$$

then $\|r_k\|$ can be computed via

$$\|r_k\| = |s_1s_2\cdots s_k|^2 \cdot \|h_k\|.$$

Finally, notice that, for the weighting strategy (3.11),

$$\|v_k\| = \frac{\sqrt{f^Tf + g^Tg}}{|\beta_k|}, \quad f := \operatorname{Re}\tilde{v}_k, \ g := \operatorname{Im}\tilde{v}_k,$$

can be obtained without extra cost during the computation of $\tilde{v}_k^T\tilde{v}_k = f^Tf - g^Tg + 2if^Tg$.

3.3. Properties. In this subsection, we list some further properties of the QMR approach.

PROPOSITION 3.3. *For* $k = 1, 2, \cdots, m_\star$:

(a)

$$r_k^{\mathrm{QMR}} = |s_k|^2 r_{k-1}^{\mathrm{QMR}} + (c_k\tilde{\tau}_{k+1}/\omega_{k+1})v_{k+1}; \tag{3.19}$$

(b) *The* BCG *iterate* x_k^{BCG} *defined by* (2.9) *exists if and only if* $c_k \neq 0$. *Moreover, if* $c_k \neq 0$, *then*

$$x_k^{\mathrm{BCG}} = x_{k-1}^{\mathrm{QMR}} + (\tilde{\tau}_k/c_k)p_k, \tag{3.20}$$

$$\|r_k^{\mathrm{BCG}}\| = |\omega_1\beta_1 s_1 s_2 \cdots s_{k-1}s_k| \cdot \|v_{k+1}\|/(\omega_{k+1}|c_k|). \tag{3.21}$$

Proof. (a) By (3.17), (3.12), and (3.8), we have

$$r_k^{\mathrm{QMR}} = \tilde{\tau}_{k+1}\tilde{w}_{k+1} \quad \text{where} \quad \tilde{w}_{k+1} := V_{k+1}\Omega_{k+1}^{-1}Q_{k+1}^H \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}. \tag{3.22}$$

With (3.14), it follows that successive vectors $\tilde{w}_{k+1}$ and $\tilde{w}_k$ are connected by

$$\tilde{w}_{k+1} = -\overline{s_k}\tilde{w}_k + (c_k/\omega_{k+1})v_{k+1}. \tag{3.23}$$

Finally, by combining (3.23) and (3.22) and using $\tilde{\tau}_{k+1} = -s_k\tilde{\tau}_k$, we obtain (3.19).

(b) First we note that (3.12), (3.4), and (3.14) imply

$$Q_k\Omega_kT_k = \begin{bmatrix} I_{k-1} & 0 \\ 0 & c_k \end{bmatrix} R_k. \tag{3.24}$$

Thus, by Proposition 3.1, x_k^{BCG} exists if and only if $c_k \neq 0$. Now assume $c_k \neq 0$. Using (3.5)–(3.6), (3.24), and (3.17), we get

$$x_k^{\mathrm{BCG}} = x_0 + V_kz_k^{\mathrm{BCG}} \quad \text{where} \quad z_k^{\mathrm{BCG}} = R_k^{-1}\begin{bmatrix} t_{k-1} \\ \tilde{\tau}_k/c_k \end{bmatrix}. \tag{3.25}$$

By comparing (3.25) with (3.17), (3.20) follows. For the proof of (3.21), notice that, by (3.25), (3.13) , and the formula for s_k in (3.15),

$$\beta_{k+1}(z_k^{\mathrm{BCG}})_k = \beta_{k+1}\tilde{\tau}_k/(\zeta_kc_k) = \tilde{\tau}_ks_k/(\omega_{k+1}c_k). \tag{3.26}$$

Furthermore, Algorithm 3.2 shows that

$$|\tilde{\tau}_k| = |\omega_1\beta_1s_1s_2\cdots s_{k-1}|. \tag{3.27}$$

Finally, by inserting (3.26)–(3.27) into the formula (3.6) for r_k^{BCG}, we arrive at (3.21). □

In view of part (b) of Proposition 3.3, the QMR method has the additional feature that it also yields *all* existing BCG iterates. This is in contrast to the BCG Algorithm 2.3, which breaks down as soon as the *first* nonexisting BCG iterate is encountered. We remark that, by (3.21), $\|r_k^{\mathrm{BCG}}\|$ can be computed without extra cost from quantities which are generated in Algorithm 3.2 anyway. In particular, we may monitor $\|r_k^{\mathrm{BCG}}\|$ during the course of the QMR algorithm, and, whenever the actual BCG iterate is desired, compute x_k^{BCG} via (3.20).

There is an important special case for which the QMR method (with weighting strategy (3.11)) is even equivalent to the MR approach (3.7). Consider the subclass of (1.3) of complex symmetric matrices of the form

$$A = T + i\sigma I, \quad T = T^T \quad \text{real symmetric}, \quad \sigma > 0. \tag{3.28}$$

Assume that $r_0 \in \mathbb{R}^n$ (this can always be achieved by a proper choice of x_0). Then it is easily verified that the Lanczos vectors v_k generated by Algorithm 2.1 are all real and therefore, by (2.3), orthonormal with respect to the usual Euclidean inner product. In particular, by (3.11), $\omega_j \equiv 1$, and the least squares problem (3.9) is equivalent to the one on the right-hand side of (3.7). Hence we have the following result.

THEOREM 3.4. *Let A be of the form* (3.28) *and $r_0 \in \mathbb{R}^n$. Then, the iterates x_k generated by Algorithm* 3.2 (*with $\omega_j \equiv 1$*) *satisfy the minimal residual property* (3.7).

4. On the breakdown of the complex symmetric Lanczos algorithm. In the general nonsymmetric Lanczos process a breakdown—more precisely, division by 0—may occur, before an invariant subspace has been found (see, e.g., [43, p. 389]). Although this happens very rarely in practice, the possibility of such breakdowns has brought the nonsymmetric Lanczos method into discredit and has certainly kept many people from actually using the algorithm. On the other hand, it is possible to devise so-called *look-ahead* [42], [38], [27], [3] modifications of the Lanczos algorithm which allow it to skip—except in the very special case of an incurable breakdown [42], [37]—over those iterations in which the standard algorithm would break down. Note that this was already observed by Gragg [21, pp. 222–223] and, in the context of the partial realization problem, by Kung [30, Chap. IV] and Gragg and Lindquist [22]. However, a complete treatment of the modified Lanczos method and its intimate connection with orthogonal polynomials and Padé approximation was presented only recently by Gutknecht [23].

Like for the general nonsymmetric Lanczos process, the complex symmetric Algorithm 2.1 may break down, i.e., stop with $v^T v = 0$ and $v \neq 0$. Recall that, throughout §3, possible breakdowns of the complex symmetric Lanczos recursion were explicitly excluded by assuming (3.1). In this section we briefly sketch the basic idea of the look-ahead procedure for the special case of the complex symmetric Lanczos recursion and derive a new theoretical result concerning so-called incurable breakdowns for complex symmetric matrices.

Assume that a breakdown occurs in Algorithm 2.1. In view of Proposition 2.2, this happens if and only if there is no complete set of $m_\star$ Lanczos vectors $v_k \in K_k(r_0, A)$, $k = 1, \cdots, m_\star$, which are orthonormal (cf. (2.3)) with respect to the indefinite inner product (2.6). Clearly, there exists a maximal subset

$$\{k_1, k_2, \cdots, k_J\} \subsetneqq \{1, 2, \cdots, m_\star\}, \quad 1 \leq k_1 < k_2 < \cdots < k_J \leq m_\star, \tag{4.1}$$

such that for each $j = 1, 2, \cdots, J$ there exists a vector $v_{k_j} \in K_{k_j}(r_0, A)$ satisfying the orthonormality relations

$$v_{k_j}^T v = 0 \quad \text{for all } v \in K_{k_j - 1}(r_0, A) \quad \text{and} \quad v_{k_j}^T v_{k_j} = 1. \tag{4.2}$$

By the definition of Krylov subspaces, $K_k(r_0, A) = \{P(A) r_0 \mid P \in \Pi_{k-1}\}$, and especially

$$v_{k_j} = P_{k_j - 1}(A) r_0 \quad \text{with} \quad P_{k_j - 1} \in \Pi_{k_j - 1}. \tag{4.3}$$

Therefore, we can rewrite (4.2) in terms of polynomials:

$$(P_{k_j - 1}, P) = 0 \quad \text{for all } P \in \Pi_{k_j - 2}, \quad (P_{k_j - 1}, P_{k_j - 1}) \neq 0, \tag{4.4}$$

with the indefinite inner product

$$(P,Q) := r_0^T P(A)Q(A)r_0. \tag{4.5}$$

A polynomial $P_{k_j-1} \in \Pi_{k_j-1}$ that fulfills (4.4) is called a regular orthogonal (with respect to (4.5)) polynomial of degree $k_j - 1$. It is well known [7], [23] that three successive regular orthogonal polynomials are connected via a three-term recurrence. By (4.3), it follows that there is a corresponding three-term recurrence relating the vectors v_{k_j-1}, v_{k_j}, and v_{k_j+1}. The look-ahead Lanczos procedure is a modification of Algorithm 2.1 which—based on this three-term relation—generates the vectors v_{k_j}, $j = 1, 2, \cdots, J$. These vectors can then be completed to a basis of K_{k_J} by setting, e.g.,

$$v_k = A^{k-k_j} v_{k_j} \quad \text{for} \quad k = k_j+1, k_j+2, \cdots, k_{j+1}-1, \quad j = 0, 1, \cdots, J-1,$$

(cf. [19]). Here, for $j = 0$, we set $k_0 := 1$. We remark that the resulting look-ahead Lanczos algorithm produces block tridiagonal matrices T_{k_j}, $j = 1, \cdots, J$, of the type (2.1) with $(k_j - k_{j-1}) \times (k_j - k_{j-1})$ matrices α_{k_j} on the block diagonal.

In exact arithmetic, the outlined algorithm terminates with the block tridiagonal T_{k_J}. Suppose that $k_J = m_\star$ in (4.1). Then T_{k_J} represents the restriction of the matrix A to the A-invariant subspace $K_{m_\star}(r_0, A)$. Obviously, in view of (2.2), the solution of (1.1) can then be computed from the quantities generated by the look-ahead Lanczos procedure. On the other hand, if $k_J < m_\star$ in (4.1), it is not possible to obtain the solution of (1.1) by means of the Lanczos process. For this reason, the case $k_J < m_\star$ is called incurable breakdown.

Next, we derive a criterion for the occurrence of incurable breakdown in the complex symmetric Lanczos algorithm. In the following, it is assumed that A is diagonalizable. Then (e.g., [25, Thm. 3.4.13]), A has a complete set of orthonormal (with respect to (2.6)) eigenvectors. In particular, r_0 can be expanded into eigenvectors of A. More precisely, by collecting components corresponding to identical eigenvalues, we get

$$r_0 = \sum_{l=1}^{m_\star} \rho_l u_l \tag{4.6}$$

$$\text{where } \rho_l \neq 0, \quad Au_l = \lambda_l u_l, \quad \text{and, if } l \neq j, \quad \lambda_l \neq \lambda_j, \quad u_l^T u_j = 0.$$

Notice that, unless all eigenvalues of A are distinct, quasi-null vectors u_l may occur in (4.6). In view of the following theorem, this is equivalent to incurable breakdown.

THEOREM 4.1. *Let $A = A^T$ be a diagonalizable $n \times n$ matrix and $r_0 \in \mathbb{C}^n$. Then in* (4.1), *$k_J = m_\star$ if and only if the eigenvectors in the expansion* (4.6) *of r_0 satisfy*

$$u_l^T u_l \neq 0 \quad \textit{for all} \quad l = 1, \cdots, m_\star. \tag{4.7}$$

Proof. We need to show that (4.7) is equivalent to the existence of a regular orthogonal polynomial of degree $m_\star - 1$ with respect to the inner product (4.5). From the general theory of orthogonal polynomials, it is well known (e.g., [4, pp. 11–12]) that regular orthogonal polynomials of degree k exist if and only if the corresponding moment matrix $M_k := (\mu_{j+l})_{j,l=0,\cdots,k}$ is nonsingular. For the case of (4.5), by (4.6), we have

$$\mu_j := r_0^T A^j r_0 = \sum_{l=1}^{m_\star} \rho_l^2 \lambda_l^j u_l^T u_l, \quad j = 0, 1, \cdots. \tag{4.8}$$

Note that moment matrices are, in particular, Hankel matrices. By applying Kronecker's theorem on the rank of infinite Hankel matrices [18, pp. 204–207] to $M_\infty := (\mu_{j+l})_{j,l=0,1,\cdots}$, it follows that

$$\operatorname{rank} M_\infty = \operatorname{rank} M_k = \operatorname{rank} M_{m-1} = m \quad \text{for all} \quad k \geq m-1, \tag{4.9}$$

where m is the number of poles of the rational function

$$f(z) := \sum_{j=0}^{\infty} \frac{\mu_j}{z^{j+1}}.$$

Using (4.8) and $\sum_{j=0}^{\infty} \lambda_l^j / z^{j+1} \equiv 1/(z-\lambda_l)$, we obtain the following expansion of f:

$$f(z) = \sum_{l=1}^{m_\star} \frac{\rho_l^2 u_l^T u_l}{z - \lambda_l} \quad \text{for all} \quad |z| > \max_{l=1,\cdots,m_\star} |\lambda_l|. \tag{4.10}$$

In particular, by (4.10), $m \leq m_\star$ with equality holding if and only if (4.7) holds true. Hence, in view of (4.9), $M_{m_\star - 1}$ is nonsingular if and only if (4.7) is fulfilled. This concludes the proof. □

As mentioned, (4.7) is guaranteed if A has only simple eigenvalues. Thus we have the following corollary.

COROLLARY 4.2. *If $A = A^T$ is an $n \times n$ matrix with n distinct eigenvalues, then incurable breakdowns cannot occur in the complex symmetric Lanczos Algorithm* 2.1.

5. Complex versus equivalent real linear systems.

In this section, we study connections between (1.1) and its equivalent real versions. Unless stated otherwise, A is now assumed to be a general complex $n \times n$ matrix.

5.1. Equivalent real linear systems.

By taking real and imaginary parts in (1.1), we can rewrite (1.1) as the real linear system

$$A_\star \begin{bmatrix} \operatorname{Re} x \\ \operatorname{Im} x \end{bmatrix} = \begin{bmatrix} \operatorname{Re} b \\ \operatorname{Im} b \end{bmatrix}, \quad A_\star := \begin{bmatrix} \operatorname{Re} A & -\operatorname{Im} A \\ \operatorname{Im} A & \operatorname{Re} A \end{bmatrix}. \tag{5.1}$$

A second real version of (1.1) is

$$A_{\star\star} \begin{bmatrix} \operatorname{Re} x \\ -\operatorname{Im} x \end{bmatrix} = \begin{bmatrix} \operatorname{Re} b \\ \operatorname{Im} b \end{bmatrix}, \quad A_{\star\star} := \begin{bmatrix} \operatorname{Re} A & \operatorname{Im} A \\ \operatorname{Im} A & -\operatorname{Re} A \end{bmatrix}. \tag{5.2}$$

Obviously, (5.1) and (5.2) are the only essentially different possibilities of rewriting (1.1) as a real $2n \times 2n$ system. Furthermore, note that $A_\star$ is nonsymmetric if and only if $A \neq A^H$ is non-Hermitian, whereas $A_{\star\star}$ is symmetric if and only if $A = A^T$. Hence, for complex symmetric linear systems the approach (5.2) appears to be especially attractive since it permits the use of simple CG-type methods such as SYMMLQ and MINRES [36] for real symmetric matrices.

In the following proposition, we collect some simple spectral properties of $A_\star$ and $A_{\star\star}$.

PROPOSITION 5.1. (a) *Let $J = X^{-1}AX$ be the Jordan normal form of A. Then $A_\star$ has the Jordan normal form*

$$\begin{bmatrix} J & 0 \\ 0 & \overline{J} \end{bmatrix} = X_\star^{-1} A_\star X_\star \quad \textit{where} \quad X_\star := \frac{1}{\sqrt{2}} \begin{bmatrix} X & -i\overline{X} \\ -iX & \overline{X} \end{bmatrix}. \tag{5.3}$$

In particular,

$$\sigma(A_\star) = \sigma(A) \cup \overline{\sigma(A)}. \tag{5.4}$$

(b) *The matrices $A_{\star\star}$ and $-A_{\star\star}$ are similar. In particular,*

$$-\lambda, \overline{\lambda}, -\overline{\lambda} \in \sigma(A_{\star\star}) \quad \textit{for all} \quad \lambda \in \sigma(A_{\star\star}). \tag{5.5}$$

Moreover,

$$\sigma(A_{\star\star}) = \{\lambda \in \mathbb{C} \mid \lambda^2 \in \sigma(\overline{A}A)\}.$$

(c) *Let $A = A^T$ be complex symmetric. Then, there exists a singular value decomposition (the so-called Takagi* SVD) *of A of the form*

$$A = U\Sigma U^T, \quad \textit{U unitary}, \quad \Sigma = \operatorname{diag}(\sigma_1, \sigma_2, \cdots, \sigma_n) \geq 0. \tag{5.6}$$

Moreover, $A_{\star\star}$ is a real symmetric matrix with spectral decomposition

$$A_{\star\star} = \begin{bmatrix} Y & -Z \\ Z & Y \end{bmatrix} \begin{bmatrix} \Sigma & 0 \\ 0 & -\Sigma \end{bmatrix} \begin{bmatrix} Y & -Z \\ Z & Y \end{bmatrix}^T \quad \textit{where} \quad Y = \operatorname{Re} U, \; Z = \operatorname{Im} U. \tag{5.7}$$

Proof. (a) First, note that

$$X_\star = S \begin{bmatrix} X & 0 \\ 0 & \overline{X} \end{bmatrix} \quad \text{where} \quad S := \frac{1}{\sqrt{2}} \begin{bmatrix} I_n & -iI_n \\ -iI_n & I_n \end{bmatrix} \quad \text{is unitary.} \tag{5.8}$$

In particular, (5.8) shows that with X also $X_\star$ is nonsingular. We readily verify that

$$S^H A_\star S = \begin{bmatrix} A & 0 \\ 0 & \overline{A} \end{bmatrix},$$

and, in view of (5.8), this implies (5.3). Equation (5.4) is an obvious consequence of (5.3).

(b) Since

$$\begin{bmatrix} 0 & I_n \\ -I_n & 0 \end{bmatrix}^{-1} A_{\star\star} \begin{bmatrix} 0 & I_n \\ -I_n & 0 \end{bmatrix} = -A_{\star\star},$$

the real matrices $A_{\star\star}$ and $-A_{\star\star}$ are similar. Hence, (5.5) holds true. The relation between $\sigma(A_{\star\star})$ and $\sigma(\overline{A}A)$ is known (see [25, p. 214] for a proof).

(c) Equation (5.6) is the well-known Takagi singular value decomposition for symmetric matrices (e.g., [25, Cor. 3.4.4]). By rewriting (5.6) in terms of the real and imaginary parts of A and U, we obtain (5.7) (cf. [25, pp. 212–213]). □

Roughly speaking, Krylov subspace methods are most effective for coefficient matrices A whose spectrum, except for possibly a few isolated eigenvalues, is contained in a half-plane which excludes the origin of the complex plane. On the other hand, if this half-plane condition is not satisfied and if a large number of eigenvalues of A straddle the origin, usually the convergence of CG-type algorithms is prohibitively slow. Typically, in these situations (see [8], [13], [14] for examples), iterations based on Krylov subspaces generated by A offer no advantage over solving the normal equations (1.5) by standard CG. See Theorem 5.4 for a theoretical result along these lines.

For complex linear systems that arise in practice the half-plane condition is usually satisfied. Indeed, mostly

$$\sigma(A) \subset \{\lambda \in \mathbb{C} \mid \operatorname{Im} \lambda \geq 0\}. \tag{5.9}$$

However, by rewriting (1.1) as real linear systems (5.1), respectively, (5.2), we deliberately create coefficient matrices whose spectra are most unfavorable for Krylov subspace methods. The case (5.2) is especially bad since, in view of (5.5), $\sigma(A_{\star\star})$ is symmetric with respect to real and imaginary axis and hence the eigenvalues always embrace the origin. Similarly, by (5.4), the coefficient matrix $A_\star$ of (5.1) in general has eigenvalues in the upper as well as in the lower half-plane. In particular, if (5.9) holds and, as in most applications, the Hermitian part $(A + A^H)/2$ of A is indefinite, the spectrum of $A_\star$ straddles the origin and the half-plane condition is not satisfied for $A_\star$. The following example illustrates this behavior.

Example 5.2. Consider the class (3.28) of complex symmetric matrices $A = T + i\sigma I$, where $T = T^T$ is real and $\sigma > 0$. Obviously,

$$\begin{aligned}\sigma(A) &= \{\lambda = \mu + i\sigma \mid \mu \in \sigma(T)\} \\ &\subset S := \big[\mu_m + i\sigma, \mu_M + i\sigma\big].\end{aligned} \tag{5.10}$$

Here μ_m and μ_M denote the smallest and largest eigenvalue of T, respectively. Note that the complex line segment S is parallel to the real axis and always contained in the upper half of the complex plane. In view of (5.4), (5.10) implies

$$\sigma(A_\star) = \{\lambda = \mu \pm i\sigma \mid \mu \in \sigma(T)\} \subset S \cup \overline{S}.$$

We remark that $S \cup \overline{S}$ is a tandem slit consisting of the two complex intervals S and $\overline{S}$, which are parallel and symmetric to each other with respect to the real axis. Moreover, the eigenvalues of $A_\star$ straddle the origin if the Hermitian part T of A is indefinite. Finally, using (3.28) and part (b) of Proposition 5.1, we obtain

$$\begin{aligned}\sigma(A_{\star\star}) &= \{\lambda = \pm\sqrt{\mu^2 + \sigma^2} \mid \mu \in \sigma(T)\} \\ &\subset \Big[-\sqrt{\mu_M^2 + \sigma^2}, -\sigma\Big] \cup \Big[\sigma, \sqrt{\mu_M^2 + \sigma^2}\Big].\end{aligned}$$

Note that the class (3.28) is closely related to shifted skewsymmetric matrices. Indeed, if, instead of $Ax = b$, we rewrite $-iAx = -ib$ as a real system (5.1), we obtain

$$(-iA)_\star = \begin{bmatrix} \sigma I_n & T \\ -T & \sigma I_n \end{bmatrix} = \sigma I_{2n} - N, \quad N := \begin{bmatrix} 0 & -T \\ T & 0 \end{bmatrix} \quad \Big(= -N^T\Big). \tag{5.11}$$

Then the eigenvalues are contained in a line segment which is parallel to the imaginary axis and symmetric with respect to the real axis:

$$\sigma\big((-iA)_\star\big) = \{\lambda = \sigma \pm i\mu \mid \mu \in \sigma(T)\} \subset [\sigma - i\rho, \sigma + i\rho], \qquad \rho = \max\{|\mu_m|, |\mu_M|\}.$$

5.2. Correspondence of Krylov subspace methods. In analogy to (1.6) for complex linear systems (1.1), a Krylov subspace method for the solution of the equivalent real systems (5.1), respectively, (5.2) generates iterates

$$\begin{bmatrix} \operatorname{Re} x_k \\ \operatorname{Im} x_k \end{bmatrix} = \begin{bmatrix} \operatorname{Re} x_0 \\ \operatorname{Im} x_0 \end{bmatrix} + P(A_\star) \begin{bmatrix} \operatorname{Re} r_0 \\ \operatorname{Im} r_0 \end{bmatrix}, \qquad P \in \Pi_{k-1}^{(r)}, \tag{5.12}$$

respectively,

$$\begin{bmatrix} \operatorname{Re} x_k \\ -\operatorname{Im} x_k \end{bmatrix} = \begin{bmatrix} \operatorname{Re} x_0 \\ -\operatorname{Im} x_0 \end{bmatrix} + P(A_{\star\star}) \begin{bmatrix} \operatorname{Re} r_0 \\ \operatorname{Im} r_0 \end{bmatrix}, \qquad P \in \Pi_{k-1}^{(r)}. \tag{5.13}$$

Here and in the sequel, $\Pi_{k-1}^{(r)}$ denotes the subset of Π_{k-1} of polynomials with real coefficients. Furthermore, the notation

$$K_k^{(r)}(c, B) := \{P(B)c \mid P \in \Pi_{k-1}^{(r)}\} \quad \Big(\subset K_k(c, B)\Big)$$

will be used.

At first glance it might appear that Krylov subspace iterations (1.6), respectively, (5.12)–(5.13) for the original complex systems, respectively, its equivalent real versions correspond to each other. However, as the following proposition shows, this is not the case in general.

PROPOSITION 5.3. *Let $k \in \mathbb{N}$.*

(a) *Let $P \in \Pi_{k-1}$. Then, $x_k = x_0 + P(A)r_0$ is equivalent to*

$$(5.14)\qquad \begin{bmatrix}\operatorname{Re} x_k\\ \operatorname{Im} x_k\end{bmatrix} = \begin{bmatrix}\operatorname{Re} x_0\\ \operatorname{Im} x_0\end{bmatrix} + P_1(A_\star)\begin{bmatrix}\operatorname{Re} r_0\\ \operatorname{Im} r_0\end{bmatrix} + P_2(A_\star)\begin{bmatrix}\operatorname{Im} r_0\\ -\operatorname{Re} r_0\end{bmatrix}$$

where $P = P_1 + iP_2$, $P_1, P_2 \in \Pi_{k-1}^{(r)}$.

(b) *Let $P \in \Pi_{k-1}^{(r)}$. Then,* (5.13) *is equivalent to*

$$(5.15)\qquad x_k = \operatorname{Re} x_k + i \operatorname{Im} x_k = x_0 + R(\overline{A}A)\overline{r_0} + S(\overline{A}A)\overline{A}r_0$$

where $R \in \Pi_{\lfloor (k-1)/2 \rfloor}^{(r)}$ and $S \in \Pi_{\lfloor (k-2)/2 \rfloor}^{(r)}$ are defined by $P(\lambda) \equiv R(\lambda^2) + \lambda S(\lambda^2)$.

Proof. First we note that, for $j = 0, 1, \cdots$,

$$(5.16)\quad (A_\star)^j = \begin{bmatrix}\operatorname{Re} A^j & -\operatorname{Im} A^j\\ \operatorname{Im} A^j & \operatorname{Re} A^j\end{bmatrix} \quad\text{and}\quad (A_{\star\star})^{2j} = \begin{bmatrix}\operatorname{Re}(\overline{A}A)^j & \operatorname{Im}(\overline{A}A)^j\\ -\operatorname{Im}(\overline{A}A)^j & \operatorname{Re}(\overline{A}A)^j\end{bmatrix},$$

as is easily verified by induction on j.

(a) Let γ_j and δ_j be the coefficients of the real polynomials P_1 and P_2, respectively. Then

$$(5.17)\qquad \begin{aligned}\operatorname{Re} P(A) &= \sum_{j=0}^{k-1}\left(\gamma_j \operatorname{Re} A^j - \delta_j \operatorname{Im} A^j\right),\\ \operatorname{Im} P(A) &= \sum_{j=0}^{k-1}\left(\gamma_j \operatorname{Im} A^j + \delta_j \operatorname{Re} A^j\right).\end{aligned}$$

By reformulating $x_k = x_0 + P(A)r_0$, by means of (5.17) and the first relation in (5.16), in terms of real and imaginary parts, we immediately obtain (5.14).

(b) A routine calculation, using the second identity in (5.16), shows that (5.13) can be rewritten as

$$\begin{bmatrix}\operatorname{Re} x_k\\ -\operatorname{Im} x_k\end{bmatrix} = \begin{bmatrix}\operatorname{Re} x_0\\ -\operatorname{Im} x_0\end{bmatrix} + \begin{bmatrix}\operatorname{Re}\{R(\overline{A}A)\overline{r}_0 + S(\overline{A}A)\overline{A}r_0\}\\ -\operatorname{Im}\{R(\overline{A}A)\overline{r}_0 + S(\overline{A}A)\overline{A}r_0\}\end{bmatrix}.$$

Hence (5.13) and (5.15) are equivalent. □

In view of part (a) of Proposition 5.3, the corresponding real equivalent of complex Krylov schemes (1.6) are iterations of the type (5.14) and not the obvious real Krylov subspace methods (5.12). Clearly, the actual choice of the polynomials in (1.6), respectively, (5.12)–(5.13) is aimed at obtaining iterates which are—in a certain sense—best possible approximations to the exact solution of the corresponding linear system. By using schemes of the type (5.12), from the first, we give up k of the $2k$ real parameters which are available for optimizing complex Krylov subspace methods (1.6). Consequently, it is always preferable to solve the complex system (1.1) rather than the real version (5.1) by Krylov subspace methods. Furthermore, numerical tests reveal that the convergence behavior of the two approaches can be drastically different (see § 6).

5.3. A connection between MR and CGNR for complex symmetric matrices. Now assume that A is a complex symmetric $n \times n$ matrix. Then, in view of part (c) of Proposition 5.1, $A_{\star\star}$ is a real symmetric indefinite matrix whose spectrum is given by

$$(5.18)\qquad \sigma(A_{\star\star}) = \{\pm\sigma_j \mid j = 1, \cdots, n\}.$$

Here $\sigma_j = \sigma_j(A) \geq 0$, $j = 1, \cdots, n$, denote the singular values of A.

Since there are simple extensions [36] of classical CG to real symmetric indefinite matrices, it is especially tempting to solve (5.2) by one of these methods. The iterates of these algorithms are defined either via a Galerkin condition or a minimal residual (MR) property. Here we consider the MR approach. Applied to (5.2) it generates a sequence of iterates z_k, $k = 1, 2, \cdots$, which are characterized by

$$\|b_{\star\star} - A_{\star\star} z_k\| = \min_{z \in z_0 + K_k^{(r)}(r_0^{\star\star}, A_{\star\star})} \|b_{\star\star} - A_{\star\star} z\|, \quad z_k \in z_0 + K_k^{(r)}(r_0^{\star\star}, A_{\star\star}). \tag{5.19}$$

Here we have set

$$b_{\star\star} := \begin{bmatrix} \operatorname{Re} b \\ \operatorname{Im} b \end{bmatrix}, \quad z_k := \begin{bmatrix} \operatorname{Re} x_k \\ -\operatorname{Im} x_k \end{bmatrix} \text{ for } k = 0, 1, \cdots, \quad r_0^{\star\star} := b_{\star\star} - A_{\star\star} z_0. \tag{5.20}$$

Roughly speaking, CG-type algorithms for real symmetric indefinite systems converge slowly if the coefficient matrix is strongly indefinite in the sense that it has many positive as well as many negative eigenvalues. Unfortunately, since by (5.18), $\sigma(A_{\star\star})$ is even symmetric to the origin, $A_{\star\star}$ exhibits this undesirable property. Indeed, numerical tests show that the convergence behavior of the MR method (5.19) is practically identical to that of the tabooed approach to (1.1) via solving the normal equations (1.5) by standard CG [24]. In the sequel, we refer to this latter method as CGNR. Notice that the iterates x_k of CGNR are defined by the minimization property

$$\|b - Ax_l\| = \min_{x \in x_0 + K_l(A^H r_0, A^H A)} \|b - Ax\|, \quad x_l \in x_0 + K_l(A^H r_0, A^H A). \tag{5.21}$$

Next we prove that MR and CGNR are even equivalent if the starting residual $r_0^{\star\star}$ satisfies a certain symmetry condition. Note that, corresponding to the spectral decomposition (5.7), $r_0^{\star\star}$ can be expanded into eigenvectors of $A_{\star\star}$ as follows:

$$r_0^{\star\star} = \begin{bmatrix} Y & -Z \\ Z & Y \end{bmatrix} c \quad \text{with} \quad c = \begin{bmatrix} c_1 \\ \vdots \\ c_{2n} \end{bmatrix} \in \mathbb{R}^{2n}. \tag{5.22}$$

THEOREM 5.4. *Let x_k^{MR}, respectively, x_l^{CGNR} denote the iterates generated by* (5.19)–(5.20), *respectively,* (5.21) *starting with the same initial guess $x_0 \in \mathbb{C}^n$. Assume that c in the expansion* (5.22) *of $r_0^{\star\star}$ satisfies*

$$|c_j| = |c_{n+j}|, \qquad j = 1, 2, \cdots, n. \tag{5.23}$$

Then,

$$x_l^{\mathrm{CGNR}} = x_{2l}^{MR} = x_{2l+1}^{\mathrm{MR}}, \qquad l = 0, 1, \cdots . \tag{5.24}$$

Proof. First note that, in view of (5.7) and (5.22), c_j and c_{n+j} are components corresponding to a pair of symmetric eigenvalues $\pm\sigma_j$ of $A_{\star\star}$. However, for any real symmetric linear system $A_{\star\star} z = b_{\star\star}$ with "symmetric" eigenvalues and "symmetric" starting residual $r_0^{\star\star}$ in the sense of (5.18) and (5.23), respectively, the MR method generates iterates with $z_k \in z_0 + K_{\lfloor k/2 \rfloor}^{(r)}(A_{\star\star} r_0^{\star\star}, A_{\star\star}^2)$ (see, e.g., [14]). Consequently the iterates defined by (5.19) satisfy

$$z_{2l} = z_{2l+1} \in z_0 + K_l^{(r)}(A_{\star\star} r_0^{\star\star}, A_{\star\star}^2). \tag{5.25}$$

In particular, by (5.20), (5.25) shows that $x_{2l}^{\mathrm{MR}} = x_{2l+1}^{\mathrm{MR}}$.

It remains to prove the first relation in (5.24). To this end, we remark that

$$(5.26)\qquad \|b_{\star\star}-A_{\star\star}z\|=\|b-Ax\| \quad\text{for all}\quad z=\begin{bmatrix}\operatorname{Re}x\\-\operatorname{Im}x\end{bmatrix},\quad x\in\mathbb{C}^n.$$

Moreover, by using (5.20) and part (b) of Proposition 5.3 (applied to polynomials $P(\lambda)\equiv\lambda S(\lambda^2)$), we deduce

$$(5.27)\qquad z_0+K_l^{(r)}(A_{\star\star}r_0^{\star\star},(A_{\star\star})^2)=\left\{\begin{bmatrix}\operatorname{Re}x\\-\operatorname{Im}x\end{bmatrix}\;\middle|\; x\in x_0+K_l^{(r)}(A^Hr_0,A^HA)\right\}$$

(notice that $\overline{A}=A^H$ in (5.15)!). In view of (5.25)–(5.27), (5.19) (for $k=2l$) can be rewritten in the form

$$(5.28)\quad \|b-Ax_{2l}^{MR}\|=\min_{x\in x_0+K_l^{(r)}(A^Hr_0,A^HA)}\|b-Ax\|,\quad x_{2l}^{\mathrm{MR}}\in x_0+K_l^{(r)}(A^Hr_0,A^HA).$$

Finally, note that the iterates of CGNR always correspond to real polynomials, i.e., $x_l^{\mathrm{CGNR}}\in x_0+K_l^{(r)}(A^Hr_0,A^HA)$. Hence, by comparing (5.21) with (5.28), we conclude that $x_l^{\mathrm{CGNR}}=x_{2l}^{\mathrm{MR}}$. □

Clearly, the special symmetry condition (5.23) will not be satisfied in general. Nevertheless, all our numerical experiments showed (cf. § 6) that (5.24) is still fulfilled approximately, i.e.,

$$(5.29)\qquad x_l^{\mathrm{CGNR}}\approx x_{2l}^{\mathrm{MR}}\approx x_{2l+1}^{\mathrm{MR}},\qquad l=0,1,\cdots.$$

6. Numerical examples. We have performed numerical experiments with all algorithms considered in this paper in numerous cases. In this section, we present a few typical results of these experiments.

Consider (1.2) on the unit square $G:=(0,1)\times(0,1)$ with $\sigma_1\in\mathbb{R}$ a constant and σ_2 a real coefficient function. First assume that u satisfies Dirichlet boundary conditions. Then, approximating (1.2) by finite differences on a uniform $m\times m$ grid with mesh size $h:=1/(m+1)$ yields a linear system (1.1) with A an $n\times n$, $n:=m^2$, matrix of the form

$$(6.1)\qquad A=T+ih^2D,\quad T:=A_0-\sigma_1h^2I,\quad D=\operatorname{diag}(d_1,d_2,\cdots,d_n).$$

Here A_0 is the symmetric positive definite matrix arising from the usual five-point discretization of $-\Delta$ and the diagonal elements of D are just the values of σ_2 at the grid points.

Similarly, if we consider the real Helmholtz equation (1.2), i.e., $\sigma_2\equiv 0$, but now with a typical complex boundary condition such as

$$\frac{\partial u}{\partial n}=i\alpha u\quad\text{on}\quad\{(1,y)\mid -1<y<1\}$$

(which is discretized using forward differences) and Dirichlet boundary conditions on the other three sides of the boundary of G, we again arrive at (6.1), where

$$(6.2)\qquad d_j=\begin{cases}\alpha/h, & \text{if } j=lm,\ l=1,\cdots,m,\\ 0, & \text{otherwise.}\end{cases}$$

The test problems presented in this section are all linear systems $Ax=b$ with complex symmetric coefficient matrices of the type (6.1). For Example 6.1, the mesh size $h=1/64$ was chosen resulting in a 3969×3969 matrix A. In Examples 6.2–6.4, $h=1/32$ and thus A is a 961×961 matrix. The right-hand side b was chosen to be a vector with random components in $[-1,1]+i[-1,1]$, with the exception of Example 6.2 where b had constant components $1+i$. As starting vector, $x_0 = 0$ was chosen.

As stopping criterion, we used

$$R_k := \frac{\|b - Ax_k\|}{\|b - Ax_0\|} \leq 10^{-6}. \tag{6.3}$$

In Figs. 6.1–6.4, the relative residual norm (6.3), R_k, is plotted versus the number N_k of matrix-vector products with A, $A_\star$, or $A_{\star\star}$. Note that $N_k = k$ is identical to the iteration number, except for CGS, respectively, CGNR, which both require two matrix-vector products $A \cdot v$, respectively, $A \cdot v$, $\overline{A} \cdot v$ per iteration and for which $N_k = 2k$. For GMRES [40], work and storage per iteration step k grows linearly with k and in practice it is necessary to use restarts. In the sequel, GMRES(k_0) and GMRES$_\star$(k_0) refer to complex and real versions—restarted after every k_0 iterations—of the GMRES method applied to (1.1) and (5.1), respectively.

In a first series of experiments, QMR (with different weighting strategies) and BCG were compared. The natural choice (3.11) turned out to be the best strategy in all cases. In the following, QMR always refers to Algorithm 3.2 with weights (3.11). Then QMR produces residual vectors whose norms are almost monotonically decreasing and generally smaller than those of the BCG residuals. However, convergence of QMR and BCG typically occurred after a comparable number of iterations. The following example is typical.

Example 6.1. Here (6.1) is a 3969 × 3969 matrix with $\sigma_1 = 200$, and the diagonal elements of D are given by (6.2) with $\alpha = 10$. In Fig. 6.1, the convergence behavior of BCG, QMR, and of the unweighted version ($\omega_j \equiv 1$) of the QMR Algorithm 3.2 is displayed.

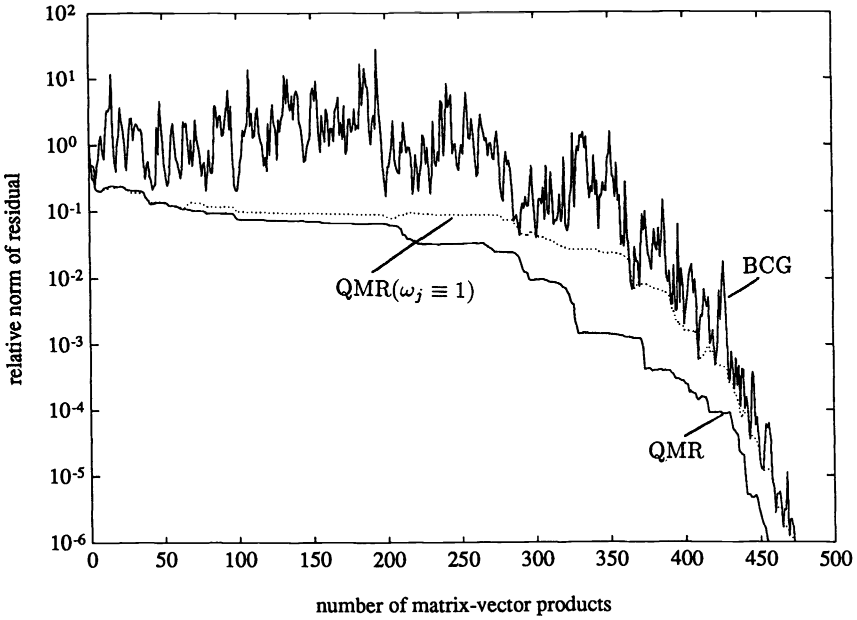


FIG. 6.1

Next we compared the CGS Algorithm 2.5 and complex GMRES with QMR and BCG. Typically, CGS needed slightly fewer iterations than QMR and BCG to reach (6.3). However, per iteration, QMR and BCG require only about half as much work and storage and thus CGS is more expensive than QMR or BCG for complex symmetric matrices. Due to the necessary restarts, GMRES was never competitive with QMR, BCG, or CGS.

Example 6.2. In (6.1) we set $n = 961$, $\sigma_1 = 100$, and d_j; $j = 1, \cdots, n$, are chosen as random numbers in $[0, 10]$. Figure 6.2 shows the convergence behavior of GMRES(20), QMR, BCG, and two runs of CGS with different starting vectors s_0, namely, $s_0 = r_0$, respectively, s_0 with random components in $[-1,1]+i[-1,1]$. Notice the extremely large residual norms in the early stage of the CGS iteration.

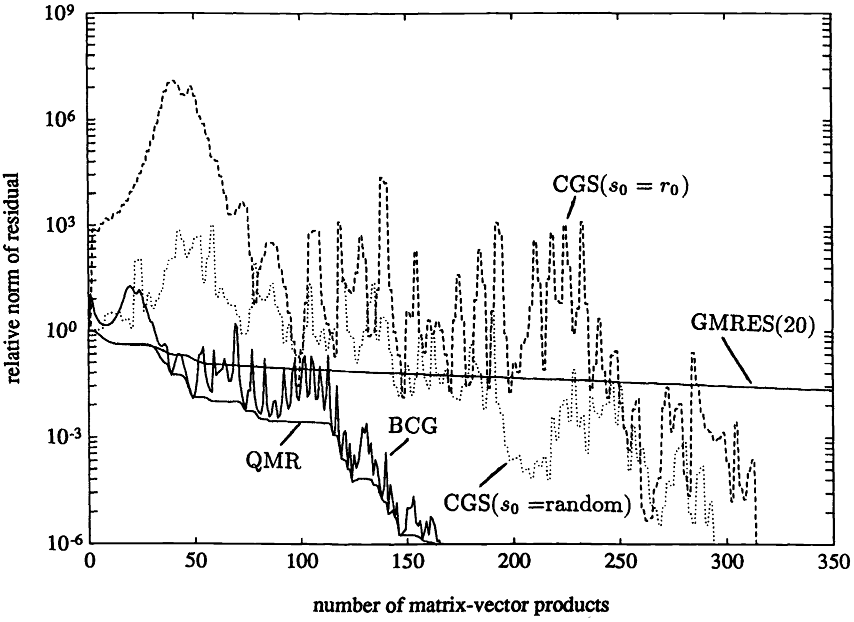


FIG. 6.2

In the following two examples we compared CG-type methods for $Ax = b$ with real schemes for the equivalent real systems (5.1), respectively, (5.2).

MR($A_{\star\star}$) denotes the minimal residual method (5.19) applied to the real symmetric system (5.2).

Example 6.3. Here, in (6.1), $n = 961$, $\sigma_1 = 100$, and d_j are given by (6.2) with $\alpha = 100$. In Fig. 6.3, the convergence behavior of QMR, MR($A_{\star\star}$), GMRES(20), GMRES(5), GMRES$_\star$(5), and CGNR is shown. Notice that, although the symmetry condition (5.23) is not fulfilled, the curves for CGNR and MR($A_{\star\star}$) are almost identical. This confirms (5.29). Finally, we tried GMRES(k_0) and GMRES$_\star$(k_0) also with other restart parameters k_0. For this example, both methods never did converge.

Example 6.4. Let A be the 961×961 matrix (6.1) with $\sigma_1 = 1000$, $D = \sigma_2 I$, $\sigma_2 = 100$, and set $\sigma := \sigma_2 h^2$. Note that A is a shifted Hermitian matrix of the form (3.28) (cf. Example 5.2). In particular, A belongs to the class of matrices (1.3) for which efficient true minimal residual algorithms for solving $Ax = b$ exist. Here we used the particular implementation, MR(A), derived in [16, Algorithm 2]. Recall that, by rewriting $-iAx = -ib$ as a real system (5.1), we obtain a shifted skewsymmetric matrix (5.11), $(-iA)_\star$. Again, for such matrices an efficient true minimal residual algorithm, denoted by MR($(-iA)_\star$), exists [9], [13]. Figure 6.4 shows the convergence behavior of MR(A), MR($A_{\star\star}$), MR($(-iA)_\star$),CGNR, and GMRES(20). Notice that MR($(-iA)_\star$) and CGNR are nearly identical. This is typical for the case in which σ is small compared to the spectral radius of T. Furthermore, if $\sigma = 0$, i.e., $(-iA)_\star$ in (5.11) is skewsymmetric, CGNR and MR($(-iA)_\star$) are even equivalent [13].

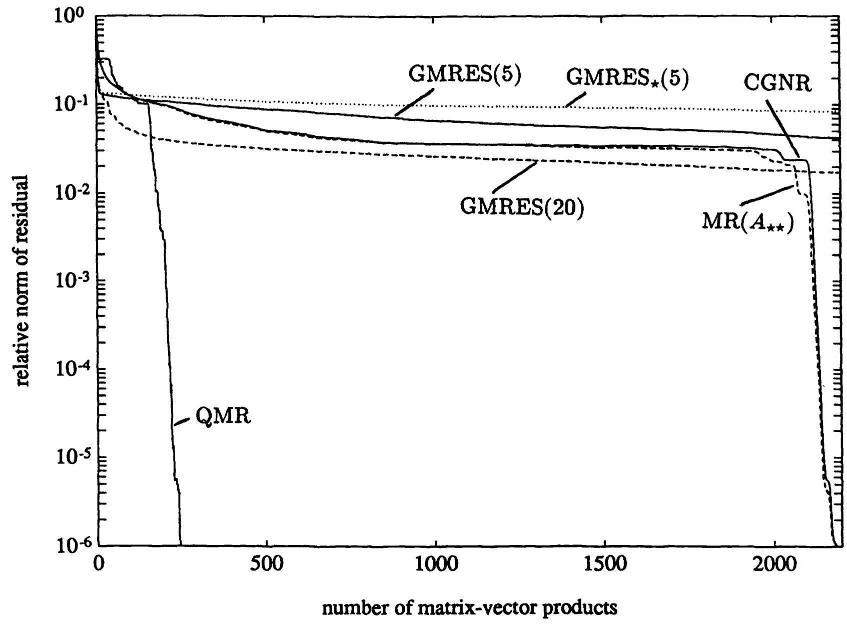


FIG. 6.3

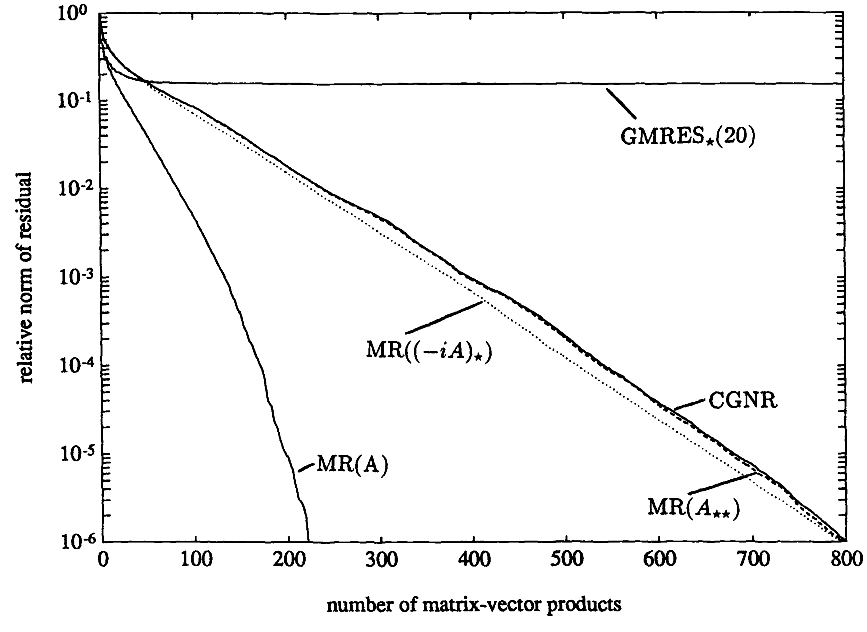


FIG. 6.4

7. Concluding remarks. Complex linear systems $Ax = b$, which arise in practice, often have complex symmetric coefficient matrices A. In this paper we have explored the use of a variant of the nonsymmetric Lanczos process for complex symmetric matrices for the solution of such linear systems. In particular, we have proposed a new method of defining approximate solutions of $Ax = b$ via a quasi-minimal residual (QMR) property. In contrast to the biconjugate gradient (BCG) approach, the QMR iterates are well defined as long as the basic Lanczos recursion does not break down. Moreover, unlike the wildly oscillating BCG residuals, the QMR residuals converge almost monotonically. Also, existing BCG iterates can be easily computed from the quantities generated during the QMR iteration. Finally, possible breakdowns—except incurable ones—of the complex symmetric Lanczos recursion can be overcome by using a look-ahead version of the Lanczos process. Incurable breakdowns occur only in very special situations. For example, they cannot occur if all eigenvalues of A are distinct.

It is very tempting (and often done in practice!) to avoid complex linear systems by solving equivalent real systems instead. We have presented some theoretical and numerical results which show that this—at least for Krylov subspace methods—is a fatal approach. Typically, the resulting real systems are unequally harder to solve by conjugate gradient-type algorithms than the original complex ones.

In this paper we have not addressed the question of how to choose preconditioners M for complex symmetric linear systems. This will be the subject of a forthcoming report. Here, we remark only that complex symmetry is preserved under preconditioning as long as M is complex symmetric. In particular, all algorithms for $A = A^T$ that we have considered can be used in conjunction with a complex symmetric preconditioner M. Note that the standard techniques, such as incomplete factorization [34], applied to $A = A^T$ generate complex symmetric preconditioners M.

Finally, we would like to mention that the quasi-minimal residual approach can also be used to stabilize the general nonsymmetric biconjugate gradient algorithm [17].

REFERENCES

[1] A. Bayliss and C.I. Goldstein, *An iterative method for the Helmholtz equation*, J. Comput. Phys., 49 (1983), pp. 443–457.

[2] C.S. Biddlecombe, E.A. Heighway, J. Simkin, and C.W. Trowbridge, *Methods for eddy current computation in three dimensions*, IEEE Trans. Magnetics, MAG-18 (1982), pp. 492–497.

[3] D. Boley and G.H. Golub, *The nonsymmetric Lanczos algorithm and controllability*, Systems Control Lett., 16 (1991), pp. 97–105.

[4] T.S. Chihara, *An Introduction to Orthogonal Polynomials*, Gordon and Breach, New York, 1978.

[5] B.D. Craven, *Complex symmetric matrices*, J. Austral. Math. Soc., 10 (1969), pp. 341–354.

[6] J.K. Cullum and R.A. Willoughby, *Lanczos Algorithms for Large Symmetric Eigenvalue Computations, Volume 1, Theory*, Birkhäuser, Basel, 1985.

[7] A. Draux, *Polynômes Orthogonaux Formels—Applications*, Lecture Notes in Mathematics 974, Springer-Verlag, Berlin, 1983.

[8] S.C. Eisenstat, *Some observations on the generalized conjugate gradient method*, Numerical Methods, Proceedings, Caracas, Venezuela, 1982, Lecture Notes in Mathematics 1005, V. Pereyra and A. Reinoza, eds., Springer-Verlag, Berlin, 1983, pp. 99–107.

[9] S.C. Eisenstat, H.C. Elman, and M.H. Schultz, *Variational iterative methods for nonsymmetric systems of linear equations*, SIAM J. Numer. Anal., 20 (1983), pp. 345–357.

[10] V. FABER AND T. MANTEUFFEL, *Necessary and sufficient conditions for the existence of a conjugate gradient method*, SIAM J. Numer. Anal., 21 (1984), pp. 352–362.

[11] ———, *Orthogonal error methods*, SIAM J. Numer. Anal., 24 (1987), pp. 170–187.

[12] R. FLETCHER, *Conjugate gradient methods for indefinite systems*, Proc. Dundee Conference on Numerical Analysis, 1975, Lecture Notes in Mathematics 506, G.A. Watson, ed., Springer-Verlag, Berlin, 1976, pp. 73–89.

[13] R. FREUND, *Über einige cg-ähnliche Verfahren zur Lösung linearer Gleichungssysteme*, Doctoral thesis, Universität Würzburg, FRG, May 1983.

[14] ———, *Pseudo Ritz values for indefinite Hermitian matrices*, Tech. Report 89.33, RIACS, NASA Ames Research Center, Moffett Field, CA, August 1989.

[15] ———, *Conjugate gradient type methods for linear systems with complex symmetric coefficient matrices*, Tech. Report 89.54, RIACS, NASA Ames Research Center, Moffett Field, CA, December 1989.

[16] ———, *On conjugate gradient type methods and polynomial preconditioners for a class of complex non-Hermitian matrices*, Numer. Math., 57 (1990), pp. 285–312.

[17] R. W. FREUND AND N.M. NACHTIGAL, QMR: *a quasi-minimal residual method for non-Hermitian linear systems*, Tech. Report 90.51, RIACS, NASA Ames Research Center, December 1990.

[18] F.R. GANTMACHER, *The Theory of Matrices*, Vol. 2, Chelsea, New York, 1959.

[19] G.H. GOLUB AND M.H. GUTKNECHT, *Modified moments for indefinite weight functions*, Numer. Math., 57 (1990), pp. 607–624.

[20] G.H. GOLUB AND C.F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[21] W.B. GRAGG, *Matrix interpretations and applications of the continued fraction algorithm*, Rocky Mountain J. Math., 4 (1974), pp. 213–225.

[22] W.B. GRAGG AND A. LINDQUIST, *On the partial realization problem*, Linear Algebra Appl., 50 (1983), pp. 277–319.

[23] M.H. GUTKNECHT, *A completed theory of the unsymmetric Lanczos process and related algorithms, Part* I, IPS Research Report No. 90–16, ETH Zürich, Switzerland, June 1990.

[24] M.R. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 409–436.

[25] R.A. HORN AND C.R. JOHNSON, *Matrix Analysis*, Cambridge University Press, Cambridge, U.K., 1985.

[26] D.A.H. JACOBS, *A generalization of the conjugate-gradient method to solve complex systems*, IMA J. Numer. Anal., 6 (1986), pp. 447–452.

[27] W.D. JOUBERT, *Generalized conjugate gradient and Lanczos methods for the solution of nonsymmetric systems of linear equations*, Ph. D. thesis, Department of Computer Sciences, University of Texas, Austin, TX, January 1990.

[28] W.D. JOUBERT AND D.M. YOUNG, *Necessary and sufficient conditions for the simplification of generalized conjugate-gradient algorithms*, Linear Algebra Appl., 88/89 (1987), pp. 449–485.

[29] J.B. KELLER AND D. GIVOLI, *Exact non-reflecting boundary conditions*, J. Comput. Phys., 82 (1989), pp. 172–192.

[30] S. KUNG, *Multivariable and multidimensional systems: Analysis and design*, Ph.D. thesis, Department of Engineering, Stanford University, Stanford, CA, June 1977.

[31] C. LANCZOS, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Res. Nat. Bur. Standards, 45 (1950), pp. 255–282.

[32] ———, *Solution of systems of linear equations by minimized iterations*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 33–53.

[33] ———, *Applied Analysis*, Prentice–Hall, Englewood Cliffs, NJ, 1956.

[34] J.A. MEIJERINK AND H.A. VAN DER VORST, *An iterative solution for linear systems of which the coefficient matrix is a symmetric $M-$matrix*, Math. Comp., 31 (1977), pp. 148–162.

[35] G. MORO AND J.H. FREED, *Calculation of* ESR *spectra and related Fokker–Planck forms by the use of the Lanczos algorithm*, J. Chem. Phys., 74 (1981), pp. 3757–3773.

[36] C.C. PAIGE AND M.A. SAUNDERS, *Solution of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal., 12 (1975), pp. 617–629.

[37] P.N. PARLETT, *Reduction to tridiagonal form and minimal realizations*, preprint, University of California, Berkeley, CA, January 1990.

[38] B.N. PARLETT, D.R. TAYLOR, AND Z.A. LIU, *A look-ahead Lanczos algorithm for unsymmetric matrices*, Math. Comp., 44 (1985), pp. 105–124.

[39] Y. SAAD, *The Lanczos biorthogonalization algorithm and other oblique projection methods for solving large unsymmetric systems*, SIAM J. Numer. Anal., 19 (1982), pp. 485–506.

[40] Y. SAAD AND M.H. SCHULTZ, GMRES: *A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.

[41] P. SONNEVELD, CGS, *a fast Lanczos-type solver for nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 36–52.

[42] D.R. TAYLOR, *Analysis of the look ahead Lanczos algorithm*, Ph.D. thesis, Department of Mathematics, University of California, Berkeley, CA, November 1982.

[43] J.H. WILKINSON, *The Algebraic Eigenvalue Problem*, Oxford University Press, Oxford, U.K., 1965.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 2, pp. 449-458, March 1992

A SCHEME FOR PARALLELIZING CERTAIN ALGORITHMS FOR THE LINEAR INHOMOGENEOUS HEAT EQUATION*

STEVEN M. SERBIN†

Abstract. This paper proposes a technique for parallelizing some algorithms for the inhomogeneous heat equation developed by Brenner, Crouzeix, and Thomée. It is known that "reduction of order" may occur unless special treatment is afforded the inhomogeneity; their schemes do so without imposing "unsatisfactory" conditions on the forcing term. Certain distinct pole rational functions with matrix argument are expanded into partial fraction form and instructions are given on how this can be used to apportion the work of solving the corresponding linear algebraic systems to processors operating concurrently; this will allow for the implementation of a high-order scheme in essentially the same time as that of a less powerful lower-order scheme such as implicit Euler. Some numerical results for the implementation are presented, which is compared to a sequential implementation for the same scheme and a similar one which is based on a rational function with a multiple pole.

Key words. parallel methods, inhomogeneous heat equation, order reduction, partial fractions

AMS(MOS) subject classifications. 65M05, 65M20

1. Introduction. The purpose of this note is to synthesize two basic ideas to produce an implementation strategy for the approximate solution of the linear inhomogeneous heat equation. It has been observed by Crouzeix [3] and also by Verwer [14] and Sanz-Serna, Verwer, and Hunsdorfer [10] (in the context of implicit Runge-Kutta methods) that the order achieved in fully discrete schemes for such problems may fail to meet that which is expected from the usual ordinary differential equation (ODE) analysis for the corresponding semidiscrete equations. Numerical examples of this "order reduction" may be found, for instance, in the report of Verwer [14], and so the issue of avoiding this reduction is quite real. The approach we wish to follow here is specific to the linear inhomogeneous heat equation and thus is based upon the analysis of Brenner, Crouzeix, and Thomée. They describe in [2] (and summarize in [13]) a class of fully discrete schemes which achieve optimal (unreduced) order of convergence while avoiding the imposition of "unsatisfactory" boundary conditions on the forcing term and certain of its derivatives.

To accomplish the time-stepping procedures which constitute these implicit algorithms, certain systems of linear algebraic equations must be solved at each time step. The second basic component of our strategy is the decomposition of certain rational functions with matrix argument into partial fraction form. Of itself, this approach is, to some extent, known. Swayne [11] used such a decomposition for a class of low-order schemes for systems of ODEs. Recently, Sweet [12] exhibited the utility of the partial fraction approach to obtain parallelism in cyclic reduction methods for solving certain block tridiagonal linear algebraic systems. Also, it was brought to our attention by one of the referees that Gallopoulos and Saad [4] had undertaken independently a similar study. Their approach, as ours, involves the partial fraction expansion of rational matrix functions, but applies essentially only to the *homogeneous* semidiscrete problem (the forcing term being constant). While illuminating and important computational studies and numerical experiments performed on a true parallel system may be found in their work and associated references cited therein, our results reported here aim specifically at the issue of the inhomogeneity and thus complement their work.

* Received by the editors October 1, 1989; accepted for publication (in revised form) October 17, 1990.

† Department of Mathematics, University of Tennessee, Knoxville, Tennessee 37996-1300.

Similar to the approach utilized by Gallopoulos and Saad for their problem, we shall note how the partial fraction approach allows, within each time step, the creation of subtasks which will be able to be accomplished concurrently; the resulting solutions will then be combined to produce the approximate solution of the heat equation at the next time level. One feature of our approach will be the ability to create the appropriate rational functions directly in partial fraction form, rather than having to use the more traditional expansions as intermediaries.

We will begin by presenting the problem and describing the Brenner-Crouzeix-Thomée algorithms. We will show the role of the partial fraction approach and elaborate on the conditions required and their fulfillment. We will show in general how these schemes can be implemented employing concurrency, and then demonstrate our approach with a particular fourth-order scheme. We will conclude with numerical examples which have been constructed to compare the partial fraction approach to a traditional computational scheme, as well as to investigate the order reduction question.

2. The inhomogeneous heat equation and some fully discrete numerical schemes. The problem described by Brenner, Crouzeix and Thomée is as follows. Let Ω be a bounded domain in R^d with smooth boundary; we wish to determine a function $u(x, t)$ satisfying

$$\begin{aligned} u_t - \Delta u &= f \quad \text{in } \Omega \times [0, \infty), \\ u &= 0 \quad \text{on } \partial\Omega \times [0, \infty), \\ u(\cdot, 0) &= 0 \quad \text{in } \Omega. \end{aligned} \tag{2.1}$$

Here, Δ is, as usual, the Laplace operator. In this section, we shall not attempt to present all of the stages leading to the fully discrete algorithms developed in [13]. To establish notation, though, we recount that (2.1) is first approximated by the semidiscrete problem

$$\begin{aligned} u_{h,t} - \Delta_h u_h &= f_h = P_0 f \quad \text{for } t \geqq 0, \\ u_h(0) &= 0, \end{aligned} \tag{2.2}$$

where Δ_h is the "semidiscrete" Laplacian defined on an appropriate finite-dimensional subspace S_h of $L_2(\Omega)$, $u_h : [0, \infty) \to S_h$ is the semidiscrete approximation to u, and P_0 denotes the L_2 projection operator from Ω to S_h. In turn, (2.2) will then be discretized in time to yield a scheme of the form

$$\begin{aligned} U^{n+1} &= r(k\Delta_h)U^n + k \sum_{i=1}^{m} q_i(k\Delta_h)P_0 f(t_n + \tau_i k), \qquad n = 0, 1, 2, \cdots \\ U^0 &= 0 \end{aligned} \tag{2.3}$$

where k is the time step, $t_n = nk$, $r(\lambda)$ and $q_i(\lambda)$, $i = 1, \cdots, m$ are rational functions bounded on the spectrum of $k\Delta_h$ uniformly in k and h and the quadrature nodes τ_i, $i = 1, \cdots, m$ are distinct real numbers (which are taken to be in $[0, 1]$). Each $U^n \in S_h$ is then an approximation to $u_h(t_n)$.

The aim of any such scheme (2.3) is to produce an efficient procedure which admits an "optimal" error estimate of the form $\|U^n - u_h(t_n)\| = \mathcal{O}(h^\beta + k^p)$, where h represents an appropriate spatial discretization parameter and the exponent β reflects the approximation property of the subspace S_h, while the exponent p is determined by the properties of the rational function r and $q_i(\lambda)$, $i = 1, \cdots, m$.

Certain conditions obtained by Brenner, Crouzeix, and Thomée are imperative. They first require the time discretization (2.3) to be "accurate of order p" (in a sense defined by applying the scheme to a scalar analogue), and establish the equivalence of the condition to the requirements

$$r(\lambda)=e^{\lambda}+\mathcal{O}(\lambda^{p+1}) \quad \text{as } \lambda\to 0 \tag{2.4}$$

and

$$\sum_{i=1}^{m} \tau_i^l q_i(\lambda)=\frac{l!}{\lambda^{l+1}}\left(e^{\lambda}-\sum_{j=0}^{l}\frac{\lambda^j}{j!}\right)+\mathcal{O}(\lambda^{p-l}) \quad \text{as } \lambda\to 0, \qquad 0\leqq l\leqq p. \tag{2.5}$$

To promote computational efficiency, an additional constraint on the schemes is that $r(\lambda)$ and all $q_i(\lambda)$ share the same denominator. With this choice, we shall discuss the usual implementation of (2.3), as well as our approach affording parallelism, in § 4.

When $m=p$, it turns out that (2.4)-(2.5) are easily satisfied (with the aid of a Vandermonde system to fix the $q_i(\lambda)$). However, a reduction of m at no loss in order of accuracy would lead to a more efficient scheme, so that possibility should be admitted.

Even when schemes are restricted to satisfy (2.4), (2.5), and, in addition, a certain stability condition imposed on $r(\lambda)$ for λ in the spectrum of $k\Delta_h$, a further objection is registered. Namely, the analysis is performed in certain Sobolev spaces $H^s(\Omega)$, and conditions requiring $f^{(l)}(t)$ to lie in certain of these spaces ($s=\max(\beta, 2p)-2l$, $l<p$) in order to obtain desired estimates force "unsatisfactory" boundary conditions on f and its derivatives, conditions that "are not necessary to ensure existence and uniqueness of the exact solution" of (2.1).

Strides are then taken to remove these objections. In doing so, it is necessary to reformulate the accuracy condition. Defining (with a slight notational deviation from [13])

$$\rho_l(\lambda)=\frac{l!}{\lambda^{l+1}}\left(r(\lambda)-\sum_{j=0}^{l}\frac{\lambda^j}{j!}\right)-\sum_{i=1}^{m}\tau_i^l q_i(\lambda), \qquad l=0,\cdots,p-1 \tag{2.6}$$

and

$$\rho_p(\lambda)=\frac{p!}{\lambda^{p+1}}\left(r(\lambda)-\sum_{j=0}^{p}\frac{\lambda^j}{j!}\right) \tag{2.7}$$

then accuracy of order p is equivalent to

$$\rho_l(\lambda)=\mathcal{O}(\lambda^{p-l}), \qquad \lambda\to 0, \qquad l=0,\cdots,p. \tag{2.8}$$

In addition, strict accuracy of order $p_0\leqq p$ is defined to be

$$\rho_l(\lambda)=0, \qquad l=0,\cdots,p_0-1. \tag{2.9}$$

After they perform further analysis, including the introduction of certain functions $h_l(\lambda)$, and in particular,

$$h_{p-1}(\lambda)=1-(p-1)\sum_{i=1}^{m}\tau_i^{p-2}q_i(\lambda)+\lambda\sum_{i=1}^{m}\tau_i^{p-1}q_i(\lambda) \tag{2.10}$$

the goal of the paper is achieved. Namely, by requiring (2.8), (2.9) with $p_0=p-1$, the stability condition $|r(\lambda)|\leqq 1$ for λ in the spectrum of $k\Delta_h$, *and* the condition that

$$\sigma(\lambda)\equiv h_{p-1}(\lambda)/(\lambda(1-r(\lambda))) \tag{2.11}$$

be bounded on the spectrum of $k\Delta_h$ *uniformly* in k and h, the optimal estimates are obtained, requiring appropriate regularity in the solution of (2.1), but not the objectionable boundary conditions on f.

Finally, for purposes of construction of the schemes for $m<p$, it is shown that the accuracy conditions needed should be (2.4), and, additionally,

$$\rho_l(\lambda)=\mathcal{O}(\lambda^{p-l}) \qquad \lambda\to 0, \qquad l=0,\cdots,m-1 \tag{2.12}$$

and a moment condition on the quadrature nodes,

$$\int_0^1 \omega(\tau)\tau^j\,d\tau=0, \qquad j=0,\cdots,p-m-1 \tag{2.13}$$

where $\omega(\tau)\equiv\prod_{i=1}^m(\tau-\tau_i)$.

3. Rational approximation via partial fractions: Selection of schemes. We suppose that

$$e^z\approx\sum_{j=0}^{\mu}\frac{a_j^{(\mu)}}{1-\gamma_j z}\equiv r(z) \tag{3.1}$$

where $\{\gamma_i\}_{i=0}^{\mu}$ is some specified collection of distinct real numbers, with $\gamma_0=0$. Thus, we are dealing here only with the case of rational approximation with real poles; the inclusion of complex poles indeed appears to be very interesting, and has been discussed for the homogeneous problem by Gallopoulos and Saad. For our problem, though, a general theory linking the optimal order of the scheme to the minimal number of quadrature nodes is unknown (to our knowledge) and so the case of complex poles will be deferred to a later study.

The approximation (3.1) will be of order at least ν (in the usual algebraic sense) if, letting $E(z)=e^z-r(z)$, it occurs that $E^{(j)}(0)=0, j=0,\cdots,\nu$. But, since differentiation of the individual terms in (3.1) and their subsequent evaluation at $z=0$ is trivial, the condition for order ν becomes simply

$$1=j!\left\{\sum_{i=0}^{\mu}\gamma_i^j a_i^{(\mu)}\right\}, \qquad j=0,\cdots,\nu. \tag{3.2}$$

In the case that $\nu=\mu$, (3.2) is then a Vandermonde system of order $\mu+1$:

$$\begin{bmatrix}1 & 1 & \cdots & 1\\ \gamma_0 & \gamma_1 & \cdots & \gamma_\mu\\ \vdots & \vdots & & \vdots\\ \gamma_0^\mu & \gamma_1^\mu & \cdots & \gamma_\mu^\mu\end{bmatrix}\begin{bmatrix}a_0^{(\mu)}\\ a_1^{(\mu)}\\ \vdots\\ a_\mu^{(\mu)}\end{bmatrix}=\begin{bmatrix}1/0!\\ 1/1!\\ \vdots\\ 1/\mu!\end{bmatrix} \tag{3.3}$$

which we shall abbreviate as $\mathrm{V}^{(\mu)}a^{(\mu)}=\Psi^{(\mu)}$.

It is well known that this system has a unique solution, which can be described as follows. Define

$$g_i(x)=\frac{\prod_{j=0,j\neq i}^{\mu}(x-\gamma_j)}{\prod_{j=0,j\neq i}^{\mu}(\gamma_i-\gamma_j)}. \tag{3.4}$$

If the coefficients of a matrix $C^{(\mu)}$, $\{c_{ij}\}$, $i,j=0,\cdots,\mu$ are defined via the expansion

$$g_i(x)=\sum_{j=0}^{\mu}c_{ij}x^j \tag{3.5}$$

then it follows easily that $[\mathrm{V}^{(\mu)}]^{-1}=C^{(\mu)}$, so that

$$a^{(\mu)}=C^{(\mu)}\Psi^{(\mu)}. \tag{3.6}$$

We can express the coefficients of $C^{(\mu)}$ in terms of elementary symmetric polynomials as follows. The elementary symmetric polynomial of degree κ on μ arguments is

$$S_\kappa(\alpha_1,\cdots,\alpha_\mu)=\sum_{j_1<j_2<\cdots<j_\kappa}\alpha_{j_1}\alpha_{j_2}\cdots\alpha_{j_\kappa}. \tag{3.7}$$

Denote

$$d_i^{(\mu)}=\prod_{j=0,j\neq i}^{\mu}(\gamma_i-\gamma_j)^{-1}. \tag{3.8}$$

It follows by expanding out (3.4) that

$$c_{ij}=(-1)^{\mu-j}S_{\mu-j}(\gamma_0,\gamma_1,\cdots,\gamma_{i-1},\gamma_{i+1},\cdots,\gamma_\mu)\cdot d_i^{(\mu)}, \tag{3.9}$$

so by (3.6),

$$a_i^{(\mu)}=\sum_{j=0}^{\mu}\frac{c_{ij}}{j!}. \tag{3.10}$$

(Alternatively, Golub and Van Loan [5] describe a recursive way to generate the solution of a Vandermonde system.)

We now undertake the task of selecting the reciprocal poles $\{\gamma_i\}$, determining the $q_i(\lambda)$, the quadrature nodes $\{\tau_i\}$, and fulfilling the conditions of § 2 under which the optimal estimates are obtained. In light of the decision to restrict our attention to real poles, we recall the result of Wanner, Hairer, and Norsett [15] that confines the maximal order of approximation in (3.1) to be $p=\mu+1$. Moreover, using the nomenclature of [8], the N-polynomial is defined to be

$$N(t)=\sum_{i=0}^{\mu}(-1)^iS_i\frac{t^{\mu-i}}{(\mu-i)!} \tag{3.11}$$

(where S_i without any arguments is taken to mean $S_i(\gamma_1,\cdots,\gamma_\mu)$), and the surfaces (sheets) on which the maximal order $p=\mu+1$ is actually achieved are characterized as

$$M=\left\{\gamma=(\gamma_1,\gamma_2,\cdots,\gamma_\mu)\,\middle|\,\int_0^1 N(t)\,dt=0\right\}. \tag{3.12}$$

This will provide us guidance in our choice of parameters.

Since the operator Δ_h will have its spectrum on the negative real axis, we wish to choose the poles $\gamma_1^{-1},\cdots,\gamma_\mu^{-1}$ so that the required order is obtained and that $r(\lambda)$ is A_0-acceptable. We have shown in [1] that it suffices that each $\gamma_i\geqq\frac{1}{2}$ for the desired stability to occur. Next, Brenner, Crouzeix, and Thomée instruct us to satisfy (2.13) if strict accuracy of order m is to be obtained; in this case, $m=p-1=\mu$ is desired. Then, it is possible to specify the rational functions $q_i(\lambda)$ from

$$\sum_{i=1}^{m}\tau_i^l q_i(\lambda)=\frac{l!}{\lambda^{l+1}}\left(r(\lambda)-\sum_{j=0}^{l}\frac{\lambda^j}{j!}\right)\equiv b_{l+1}(\lambda),\qquad l=0,\cdots,m-1. \tag{3.13}$$

Once again, we have a Vandermonde system, with solution

$$q_i(\lambda)=\frac{\sum_{j=1}^{m}(-1)^{m-j}S_{m-j}(\tau_1,\cdots,\tau_{i-1},\tau_{i+1},\cdots,\tau_m)}{\prod_{\kappa=1,\kappa\neq i}^{m}(\tau_i-\tau_\kappa)}\cdot b_j(\lambda). \tag{3.14}$$

While this is not delivered to us in partial fraction form, we will indicate such an expansion momentarily.

For, we will show that each $b_{l+1}(\lambda)$, $l=0,\cdots,\mu-1$ can be expressed as the rational function $\Phi_{l+1}(\lambda)/Q(\lambda)$, with $\deg \Phi_{l+1}<\mu$, and $\deg Q=\mu$. From (3.14), it then follows that each $q_i(\lambda)\to 0$ as $\lambda\to-\infty$, so it has the partial fraction expansion

$$q_i(\lambda)=\sum_{j=1}^{m}\frac{\delta_j^{(i)}}{1-\gamma_j\lambda}. \tag{3.15}$$

The assertion about $b_{l+1}(\lambda)$ follows easily from the results of [8]: under the assumption that $r(\lambda)\equiv P(\lambda)/Q(\lambda)$ approximates e^{λ} to order at least μ then

$$Q(\lambda)=\sum_{i=0}^{\mu}S_i(-\lambda)^i \tag{3.16}$$

and

$$P(\lambda)=\sum_{j=0}^{\mu}\lambda^j\sum_{i=0}^{j}S_i\frac{(-1)^i}{(j-i)!}. \tag{3.17}$$

Clearly, then

$$r(\lambda)-\sum_{j=0}^{l}\frac{\lambda^j}{j!}=\frac{P(\lambda)-Q(\lambda)\sum_{j=0}^{l}\lambda^j/j!}{Q(\lambda)}; \tag{3.18}$$

upon expansion of the numerator, a polynomial of degree $l+\mu$, the coefficients of λ^i, $i=0,\cdots,l$ vanish. Thence,

$$\frac{l!}{\lambda^{l+1}}\left\{P(\lambda)-Q(\lambda)\sum_{j=0}^{l}\frac{\lambda_j}{j!}\right\}$$

is indeed a polynomial of degree $(l+\mu)-(l+1)=\mu-1$, as promised.

The coefficients in (3.15) will be found, for any particular scheme, in the usual way:

$$\delta_j^{(i)}=\lim_{\lambda\to 1/\gamma_j}(1-\gamma_j\lambda)q_i(\lambda). \tag{3.19}$$

We shall show below a particular expansion of the form (3.15) for our model scheme. Finally, we must address the condition (2.11). By virtue of the choices made in (3.13), a simple computation shows that $h_{p-1}(\lambda)=-\lambda\rho_{p-1}(\lambda)$, so the condition (2.11) can be rephrased as follows: $\sigma(\lambda)=-\rho_{p-1}(\lambda)/(1-r(\lambda))$ must be bounded uniformly in k and h on the negative real axis. From (2.8), we know that $\rho_{p-1}(\lambda)=\mathcal{O}(\lambda)$, and, similarly, from (3.16) and (3.17), $P(\lambda)-Q(\lambda)=\lambda+\mathcal{O}(\lambda^2)$, $Q(\lambda)=\mathcal{O}(1)$ so $\sigma(\lambda)=\mathcal{O}(1)$ as $\lambda\to 0^-$. That $\sigma(\lambda)$ is bounded as $\lambda\to-\infty$ follows from our assertions following (3.17) with $l=\mu=p-1$, as well as the already discussed behavior of the $\{q_i(\lambda)\}$. So $\sigma(\lambda)$ is bounded for all $\lambda\leqq 0$.

4. Implementation: Parallelism. By virtue of the fact that $r(\lambda)$ and all $q_i(\lambda)$ share the same denominator $Q(\lambda)$, the "usual" implementation of (2.3) involves solution of problems of the form $Q(k\Delta_h)U^{n+1}=v^n$. Since $Q(k\Delta_h)=\prod_{j=1}^{\mu}(I-\gamma_jk\Delta_h)$, this results in a sequential algorithm requiring the solution of μ subproblems of the form $(I-\gamma_jk\Delta_h)U^{n+1,j}=U^{n+1,j-1}$ with $U^{n+1,0}=v^n$ and $U^{n+1,\mu}=U^{n+1}$. We note that in this instance, the poles need not all differ. By contrast, our computational approach is as follows. Inserting the partial fraction forms for $r(k\Delta_h)$ and $q_i(k\Delta_h)$, i.e., (3.1) and (3.15), into the fully discrete scheme (2.3), we find that

$$U^{n+1}=a_0^{(\mu)}U^n+\sum_{j=1}^{\mu}a_j^{(\mu)}(I-\gamma_jk\Delta_h)^{-1}U^n+k\sum_{i=1}^{m}\sum_{j=1}^{\mu}\delta_j^{(i)}(I-\gamma_jk\Delta_h)^{-1}P_0f(t_n+\tau_ik). \tag{4.1}$$

A simple rearrangement yields the following equation (since we have set $\mu = m$):

$$(4.2) \qquad U^{n+1} = a_0^{(\mu)} U^n + \sum_{j=1}^{\mu} (I - \gamma_j k \Delta_h)^{-1} \left\{ a_j^{(\mu)} U^n + k \sum_{i=1}^{\mu} \delta_j^{(i)} P_0 f(t_n + \tau_i k) \right\}$$

and therein lies the opportunity for parallelism. We define $W^{n,j} \in S_h$ by

$$(4.3) \qquad (I - \gamma_j k \Delta_h) W^{n,j} = a_j^{(\mu)} U^n + k \sum_{i=1}^{\mu} \delta_j^{(i)} P_0 f(t_n + \tau_i k), \qquad j = 1, \cdots, \mu$$

and then assemble

$$(4.4) \qquad U^{n+1} = a_0^{(\mu)} U^n + \sum_{j=1}^{\mu} W^{n,j}.$$

Indeed, (4.3) is normally manifested in a collection of large, sparse systems of linear algebraic equations, which can be solved concurrently on μ processors, one for each index j. Further, in a usual implementation, each processor would perform at the outset of the computation the appropriate matrix factorization, which is then available for the back substitutions performed at each time step. This approach then appears to be most suitable for a shared memory architecture with a relatively small number of processors. We mention, though, that a further opportunity for parallelism in the solution of each system corresponding to index j of (4.3) could also be explored.

5. A sample method: Numerical results. We elect to present as an example of our approach a fourth-order ($p = 4$) scheme with $\mu = m = 3$. Choosing $\tau_1 = 0$, $\tau_2 = .5$, and $\tau_3 = 1$, the moment condition (2.13), $j = 0$, is obviously met, and this selection of τ's clearly saves one evaluation of $P_0 f$ at each time step. With the specification of the γ_i's deferred, we shall proceed under the assumption that (3.9)-(3.10) will provide us with the necessary set of coefficients $\{a_i^{(\mu)}\}$. The other collection of required coefficients, the $\{\delta_j^{(i)}\}$, require elaboration for the case at hand. One can first invoke (3.13) to produce, with $Q(\lambda) = \prod_{j=1}^{3} (1 - \gamma_j \lambda)$, the formulae $q_i(\lambda) = J_i(\lambda)/Q(\lambda)$ with

$$(5.1) \qquad \begin{aligned} J_1(\lambda) &= [\tfrac{1}{6} + (\tfrac{1}{2}S_1 - 2S_2 + 4S_3)\lambda + (\tfrac{1}{6} - \tfrac{1}{2}S_1 + S_2 - S_3)\lambda^2], \\ J_2(\lambda) &= [\tfrac{2}{3} + (\tfrac{2}{3} - 2S_1 + 4S_2 - 8S_3)\lambda] \end{aligned}$$

and

$$J_3(\lambda) = [\tfrac{1}{6} + (-\tfrac{1}{6} + \tfrac{1}{2}S_1 - 2S_2 + 4S_3)\lambda + S_3\lambda^2].$$

Then, performing the operations indicated in (3.19), we determine that

$$(5.2) \qquad \delta_j^{(i)} = \gamma_j^2 J_i(\gamma_j^{-1}) \Big/ \prod_{\kappa \neq j} (\gamma_j - \gamma_\kappa), \qquad i, j = 1, \cdots, 3.$$

We present a numerical example utilizing this scheme with the following purpose in mind. We wish to explore how the partial fraction approach compares to the usual sequential implementation for the approximate solution of the semidiscrete problem. We also wish to produce evidence that order reduction does not occur when the schemes investigated here are applied to an example of a problem obtained from semidiscretization of an inhomogeneous heat equation with homogeneous Dirichlet boundary conditions. So, we have chosen a class of test problems similar to that employed by Verwer [14] in his study.

Namely, we semidiscretize the problem (2.1) in one space dimension with $\Omega = [0, 1]$ by applying a uniform grid second-order finite difference approximation in the spatial variable, so that we consider the standard system $U_t + AU = F(t)$ with A the usual symmetric $N \times N$ tridiagonal finite difference matrix with $A_{i,i+1} = -h^{-2}$ and $A_{ii} = 2h^{-2}$, with $h = 1/(N+1)$. With exact solution chosen to be $u(x, t) = t^\alpha x(1-x)$, we have $F_i(t) = t^{\alpha-1}[\alpha x_i(1-x_i) + 2t]$. One result of this choice is that as the solution to the partial differential equation (PDE) problem is quadratic in x, the spatial error vanishes (in theory), and thus the global error is totally controlled by the time step. This in no way obscures the issue of order reduction. Indeed, Verwer does report the occurrence of order reduction when this test problem is treated by the use of a standard diagonally implicit Runge-Kutta scheme. He has investigated the test problem described here using the exponent $\alpha = 2$. We cannot make use of this choice; having chosen a fourth-order scheme of the form (4.1), in fact the semidiscrete solution turns out to coincide (up to roundoff) on the spatial grid with the exact solution of the PDE. Hence, we report here results of experiments using exponents $\alpha = 3$ and 4.

It is known (cf. [9]) that among all choices of the $\{\gamma_i\}$ on a sheet of M described by (3.12), the smallest error constant results when the poles coalesce into a single pole of multiplicity μ. For the fourth-order case at hand, enforcing the A_0-stability condition restricts attention to one sheet of M on which the pertinent root of $\gamma^3 - \frac{3}{2}\gamma^2 + \frac{1}{2}\gamma - \frac{1}{24} = 0$ is $\gamma = 1.068579021301629$. However, we obviously cannot choose the multiple pole and still obtain the partial fraction decompositions desired. One objective of our numerical example, then, is to demonstrate (at least for this case) that we can move along this sheet of M to a nearby position where the poles are distinct and not effect a significant change in the quality of the scheme. This will be evidenced in two ways. We have at first the comparison between the triple pole scheme and the distinct pole algorithm. Further, we have the options of the sequential versus the partial fraction implementation of the distinct pole algorithm to consider. We note that we have not attempted here a study of the partial fraction version of the algorithm on actual parallel hardware, as in the spirit of [4]; our computations have been performed in double precision on a SUN SPARC 1+ workstation. We remark that as observed also by Gallopoulos and Saad in their context, even when implemented on a serial machine, the partial fraction version affords a significant saving in operations, in that the matrix multiplications by the numerators of $r(k\Delta_h)$ and $\{q_i(k\Delta_h)\}$ in (2.3) are eliminated, at the expense only of the linear combinations in (4.3) and (4.4).

This, in fact, defines our algorithm. Given a choice of poles, we determine $\{a_j^{(\mu)}\}$ from (3.7)-(3.10) and $\{\delta_j^{(i)}\}$ from (5.1)-(5.2), and then in each time step, we solve (4.3) for $W^{n,j}$ for $j = 1, \cdots, \mu$ and then form U^{n+1} from (4.4).

We see no obvious way to pick the distinct poles from the sheet of M. Perhaps stretching the point, we have chosen somewhat arbitrarily $\gamma_1 = 1.06$, and $\gamma_2 = 1.07$, as perturbations of the triple pole above, and then have determined $\gamma_3 = 1.07589033521764$ to satisfy (3.12). In terms of the error constant, a brief computation shows that it suffers a magnification of only 1.000095 from the minimal value at the triple pole. The resulting coefficients are $a_0 = -.6304261005497924$, $a_1 = 1{,}454.094874359267$, $a_2 = -3{,}794.673554929712$, $a_3 = 2{,}342.209106670995$, $\delta_1^{(1)} = 3{,}567.278607850113$, $\delta_2^{(1)} = -9{,}621.288547664747$, $\delta_3^{(1)} = 6{,}054.176606481301$, $\gamma_1^{(2)} = -7{,}319.518083718724$, $\delta_2^{(2)} = 19{,}811.01919385797$, $\delta_3^{(2)} = -12{,}490.83444347257$, $\delta_1^{(3)} = 5{,}293.580042689434$, $\delta_2^{(3)} = -14{,}250.03134996801$, and $\delta_3^{(3)} = 8{,}956.617973945243$. It is arguable whether or not these particular coefficients may be termed "of moderate size," but the numerical results in columns one and two of Tables 1 and 2 indicate that for this example the sequential and partial fraction versions produce nearly identical results, with agreement to about the tenth decimal place. We have recorded here the error at $t = T = 1$ with step $k = 1/K$ measured in the norm $\|\mathscr{E}\|_\infty \equiv \max_{1 \leqq i \leqq N} |u(x_i, T) - U_i^K|$; the same trends

TABLE 1
Maximum norm of error at $t = 1$; $\alpha = 3$.

$1/h$	$1/k$	Sequential algorithm; distinct poles	$\hat{p}$	Partial fraction algorithm	$\hat{p}$	Sequential algorithm; triple pole	$\hat{p}$
10	10	.44770D−04		.44770D−04		.44769D−04	
10	20	.55086D−05	3.20	.55086D−05	3.02	.55083D−05	3.02
10	40	.53259D−06	3.37	.53259D−06	3.37	.53256D−06	3.37
10	80	.42229D−07	3.66	.42228D−07	3.66	.42225D−07	3.66
10	160	.29244D−08	3.85	.29254D−08	3.85	.29242D−08	3.85
10	320	.18851D−09	3.96	.18953D−09	3.95	.18850D−09	3.96
20	20	.55182D−05		.55182D−05		.55180D−05	
40	40	.53516D−06	3.37	.53516D−06	3.37	.53512D−06	3.37
80	80	.42500D−07	3.65	.42488D−07	3.65	.42497D−07	3.65
160	160	.29400D−08	3.85	.29308D−08	3.86	.29397D−08	3.85
320	320	.18897D−09	3.96	.20733D−09	3.82	.18900D−09	3.96

TABLE 2
Maximum norm of error at $t = 1$; $\alpha = 4$.

$1/h$	$1/k$	Sequential algorithm; distinct poles	$\hat{p}$	Partial fraction algorithm	$\hat{p}$	Sequential algorithm; triple pole	$\hat{p}$
20	20	.19818D−04		.19818D−04		.19817D−04	
40	40	.19203D−05	3.37	.19203D−05	3.37	.19201D−05	3.37
80	80	.15201D−06	3.66	.15200D−06	3.66	.15200D−06	3.66
160	160	.10460D−07	3.86	.10452D−07	3.86	.10459D−07	3.86
320	320	.66866D−09	3.97	.68620D−09	3.93	.66868D−09	3.97

are present if we use an l_2 error. For our other objective, comparing columns one and three of both tables, we see that the effect of moving away from the triple pole is almost negligible in the sequential implementation, and then, of course, only very slight when the distinct pole partial fraction form is compared with the triple pole case. Finally, we have shown in Table 1 results for $\alpha = 3$ both with h fixed and k decreasing, to exemplify the usual ODE order, but also in additional runs we have simultaneously reduced both h and k (making $h = k$) to examine the order *uniform in h*. Table 2 contains results for $\alpha = 4$ and only the uniform order is investigated. Using adjacent entries in the tables, we estimate the order as $\hat{p} = \ln[\mathscr{E}(k_i)/\mathscr{E}(k_j)]/\ln[k_i/k_j]$. It is quite clear that fourth-order accuracy is being approached in both manners, and that order reduction does not occur.

We hesitate to claim that these very small differences in results between the sequential and partial fraction versions would appear all the time (i.e., larger problems, higher-order algorithms), but the results do encourage us to believe that the approach is valid and to continue similar investigations concerning different choices for the poles and to other possible situations, such as the study of second-order systems arising from the forced wave equation.

We close with the acknowledgment that there is a definite relationship between the methods here and certain implicit Runge-Kutta methods being studied, for a broader range of problems, for example, by Keeling [7] and Karakashian and Rust [6]. We believe that the approach here is particularly germane to the problem at hand, in that Brenner, Crouzeix, and Thomée have pursued the study of this particular linear problem and their results are couched in that framework. In fact, we are indebted to Professor Thomée for the insistence that we attach our interest to the PDE problem, rather than just looking at ODE systems, so that we accomplish our computational objective while satisfying the special needs of the inhomogeneous PDE system.

REFERENCES

[1] L. A. Bales, O. A. Karakashian, and S. M. Serbin, *On the A_0-acceptability of rational approximations to the exponential function with only real poles*, BIT, 28 (1988), pp. 70-79.

[2] P. Brenner, M. Crouzeix, and V. Thomée, *Single step methods for inhomogeneous linear differential equations in Banach space*, RAIRO Anal. Numér., 16 (1982), pp. 5-26.

[3] M. Crouzeix, *Sur l'approximation des équations différentielles opérationelles linéaires par des méthodes Runge-Kutta*, Ph.D. Thesis, Université de Paris VI, Paris, France, 1975.

[4] E. Gallopoulos and Y. Saad, *On the parallel solution of parabolic equations*, CSRD Report Number 854, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, Urbana, IL, 1989.

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

[6] O. A. Karakashian and W. Rust, *On the parallel implementation of implicit Runge-Kutta methods*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 1085-1090.

[7] S. L. Keeling, *On implicit Runge-Kutta methods with a stability function having distinct real poles*, BIT, 29 (1989), pp. 91-109.

[8] S. P. Norsett and G. Wanner, *The real-pole sandwich for rational approximations and oscillation equations*, BIT, 19 (1979), pp. 79-94.

[9] S. P. Norsett and A. Wolfbrandt, *Attainable order of rational approximations to the exponential function with only real poles*, BIT, 17 (1977), pp. 200-208.

[10] J. M. Sanz-Serna, J. G. Verwer, and W. H. Hunsdorfer, *Convergence and order reduction of Runge-Kutta schemes applied to evolutionary problems in partial differential equations*, Numer. Math., 50 (1986), pp. 405-418.

[11] D. A. Swayne, *Computation of rational functions with matrix argument with application to initial-value problems*, Res. Report CS-75-14, Department of Computer Science, University of Waterloo, Canada, 1975.

[12] R. A. Sweet, *A parallel and vector variant of the cyclic reduction algorithm*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 761-765.

[13] V. Thomée, *Galerkin Finite Element Methods for Parabolic Problems*, Springer-Verlag, Berlin, 1984.

[14] J. G. Verwer, *Convergence and order reduction of diagonally implicit Runge-Kutta schemes in the method of lines*, in Numerical Analysis, D. F. Griffiths and G. A. Watson, eds., Pitman Research Notes in Mathematics Series, Essex.

[15] G. Wanner, E. Hairer, and S. P. Norsett, *Order stars and stability theorems*, BIT, 18 (1978), pp. 475-489.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 2, pp. 459-483, March 1992

COMPARING ALGORITHMS FOR SOLVING SPARSE NONLINEAR SYSTEMS OF EQUATIONS*

MÁRCIA A. GOMES-RUGGIERO†, JOSÉ MARIO MARTÍNEZ†, AND ANTONIO CARLOS MORETTI†

Abstract. This paper describes implementations of eight algorithms of Newton and quasi-Newton type for solving large sparse systems of nonlinear equations. For linear algebra calculations, a symbolic manipulation is used, as well as a static data structure introduced recently by George and Ng, which allows a partial pivoting strategy for solving linear systems. A numerical comparison of the implemented methods is presented.

Key words. nonlinear systems of equations, sparse matrices, LU factorizations, Newton's method, quasi-Newton methods

AMS(MOS) subject classification. 65H10

1. Introduction. Many real-life problems require the solution of large systems of nonlinear equations:

$$F(x)=0, \qquad F=(f_1,\cdots,f_n)^T, \tag{1.1}$$

where $F:\mathbb{R}^n\rightarrow\mathbb{R}^n$ is a nonlinear C^1-function, and its Jacobian matrix $J(x)$ is sparse (see [15], [41], [46]). The best-known method for solving this type of problem is Newton's method. This is an iterative method, where the successive approximations to the solution of (1.1) are calculated according to the following formula:

$$x_{k+1}=x_k-J(x_k)^{-1}F(x_k). \tag{1.2}$$

Hence, in most cases, at each iteration of Newton's method, the derivatives $\partial f_i/\partial x_j$ must be calculated, and the linear $n\times n$ system

$$J(x_k)s=-F(x_k) \tag{1.3}$$

must be solved, in order to obtain x_{k+1}. When analytic derivatives are not available, they may be estimated using finite differences (see [7]).

Quasi-Newton methods [2]-[5], [12]-[15], [19], [22], [23], [27]-[29], [34]-[38], [40], [42], [44], [48] were also introduced to deal with situations where analytic derivatives are not available or are very expensive to calculate. They obey the formulae

$$B_k s_k=-F(x_k), \tag{1.4}$$

$$x_{k+1}=x_k+s_k. \tag{1.5}$$

At each iteration of a quasi-Newton method, only the function values $F(x_k)$ are calculated and the linear system (1.4) is solved. The new matrix B_{k+1} is obtained from B_k using recurrence relations which only involve x_k, x_{k+1}, $F(x_k)$, and $F(x_{k+1})$. Usually, B_{k+1} is chosen as one of the matrices which satisfy the "secant equation"

$$B_{k+1}s_k=y_k\equiv F(x_{k+1})-F(x_k). \tag{1.6}$$

The best-known quasi-Newton method for small dense problems is Broyden's first method [2], [14], [15]. This method uses a rank-one correction matrix to obtain B_{k+1}

* Received by the editors May 31, 1989; accepted for publication (in revised form) October 3, 1990. This work was supported by Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) and Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNP$_q$).

† Applied Mathematics Laboratory, IMECC-UNICAMP, CP 6065, 13081-Campinas, SP, Brazil.

from B_k:

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k) s_k^T}{s_k^T s_k}. \tag{1.7}$$

Using (1.7) and the Sherman-Morrison formula [21, p. 3], B_{k+1}^{-1} may be obtained from B_k^{-1} using $O(n^2)$ flops. Moreover, a QR factorization of B_{k+1} may be obtained from a QR factorization of B_k using $O(n^2)$ flops (see [40]). Hence, if (1.4), (1.5), and (1.7) are used, not only is the time for computing derivatives saved, but also the computational work for solving linear systems may be considerably reduced in relation to the computational work needed to solve (1.3) ($O(n^3)$ flops). For this reason, in many cases, Broyden's first method may be more efficient than Newton's method, even when derivatives are easily available, in spite of its lower speed of convergence.

The situation in the large sparse case is somewhat different. In fact, if B_k is a sparse matrix, and (1.7) is used, B_{k+1} generally turns out to be a dense matrix, which may have little relation to true Jacobian matrices. Broyden [3] and Schubert [37], [44] developed a variant of Broyden's first method where matrices B_k keep the same pattern of sparsity as $J(x^k)$, satisfying the secant equation (1.6), and using a minimum variation principle. However, the difference $B_{k+1} - B_k$ is no longer a rank-one matrix as in Broyden's first method, and so no easy relationship between factorizations of B_k and B_{k+1} seems to be possible. Therefore, when Schubert's method is used to generate the B_k's, the resolution of (1.4) is as expensive as the resolution of (1.3).

These observations motivated Dennis and Marwil [12] to develop the first quasi-Newton method where an LU factorization of B_{k+1} is obtained directly from an LU factorization of B_k, giving a substantially lower cost in the resolution of (1.4) in relation to the resolution of (1.3). Basically, the Dennis-Marwil method keeps the L-factor fixed and modifies the U-factor from one iteration to the following, preserving the sparsity pattern of U and using a Schubert-type formula. Unhappily, convergence properties of the Dennis-Marwil method are not quite satisfactory. In fact, local convergence is only obtained if the algorithm is restarted with $B_k = J(x_k)$ when k is a multiple of a fixed integer q. Martínez [34] introduced a method where the LDM^T factorization of B_k (see [21]) is stored and only the factor D is modified from one iteration to another, in order to obtain the factorization of B_{k+1}. This method belongs to a larger family introduced later in [35]. Unlike the Dennis-Marwil method, the methods in this family have local convergence properties without restarts, but superlinear convergence is only obtained using restart procedures. Chadee [5], generalizing a method of Johnson and Austria [28], introduced a locally superlinear method where the LU factorization of B_{k+1} is obtained simultaneously modifying the LU factors of B_k. Unfortunately, the inverses of the triangular matrices L must have a definite sparsity pattern for Chadee's method to be useful, and so its applicability seems to be limited to special structures of the Jacobian matrices. Martínez [36] introduced a very large family which includes most superlinearly convergent methods for solving systems of nonlinear equations. The Dennis-Marwil method does not belong to this family, but it may be interpreted as a limit case ($\theta \to 1$) of a parametric subfamily where the case $\theta = \frac{1}{2}$ is Chadee's method.

In this paper, we compare Newton's method, the modified Newton method, Schubert's method, the Dennis-Marwil method, three methods in the Martínez family [35], and the first method of Broyden.

As we mentioned above, sparsity of the Jacobian matrix is not preserved by Broyden approximations generated by (1.7). Therefore, as recommended in [38], we do not store B_k in our implementation of Broyden's method but instead the vectors which define the successive rank-one corrections. Both storage and computer time increase at each iteration and hence the process must be restarted when the iteration becomes excessively expensive. Storage and computer time economy is also obtained using a strategy of dropping old updates, but higher speed of convergence is achieved using Newton restarted iterations.

The implementation of methods for solving sparse nonlinear systems of equations requires a decision about the algorithm that is going to be used for linear algebra calculations. Lopes [30], [31] called our attention to some ill-conditioned banded linear systems, derived from approximation of diffusion problems using variational principles (see [30], [52]). According to [30], the resolution of these systems using the general purpose sparse linear system solver MA28 [16] with a default tolerance parameter $\alpha = 0.1$ was completely unreliable, but good results were obtained using a very strict tolerance $\alpha = 0.999$. This choice is very similar to the use of LU factorizations with partial pivoting (see [17], [18], [21]) which is one of the most stable ways of solving linear systems using LU factorizations. These very impressive experiments motivated us to privilege numerical stability over economy of storage in our implementations of nonlinear equations solvers. So we decide to use LU factorizations with partial pivoting.

Another reason of a more theoretical nature led us to the decision of using LU factorizations with partial pivoting. Namely, the local convergence theorems for the methods introduced in [12], [35] impose that the same pivoting rule which allows the LU factorization of $J(x_0)$ will also allow the LU factorization of $J(x_*)$, the Jacobian matrix at the solution. This objective is most likely attained if the partial pivoting rule is used, since it is intuitively evident that the larger the chosen pivots are for the factorization of a given matrix, the greater is the distance between that matrix and the set of matrices for which a zero pivot appears using the same permutation rule. According to these observations, we decided to use the George–Ng [20] factorization algorithm for all the linear algebra manipulations of our algorithms, an algorithm which uses partial pivoting, a static data structure, and a symbolic factorization scheme to predict fill-in in calculations.

The use of the George–Ng scheme has some additional advantages as a subroutine of a code for solving nonlinear systems. Since all Jacobian matrices $J(x_k)$ have the same structure, it is interesting to perform symbolic manipulations before the first iteration of the algorithm so that linear algebraic calculations at each iteration are exclusively numeric (not symbolic). Clearly, it is impossible to avoid nonnumeric manipulations when using a general sparse matrix package, such as MA28, unless we decide to use the same pivoting sequence at every iteration. However, this decision may be disastrous from the numerical stability point of view.

We would like to mention that Zambaldi [53] reported many experiments showing that the George–Ng method uses less CPU time (and, of course, is more stable) than MA28 (with the default parameter 0.1) for solving sparse linear systems with certain structures which appear frequently in applications. Of course, there exist structures for which the George–Ng scheme produces an excessive amount of fill-in, even after application of some minimum degree-type preprocessing scheme (see [17]), but we feel that in most of these cases iterative linear methods are more adequate than any direct linear equation algorithm, especially if we are dealing with the systems of linear equations which arise when solving nonlinear systems. In those cases, which certainly include discretizations of three-dimensional operators, inexact Newton methods are the best alternative (see [10]).

The efficiency of the George–Ng method is not restricted to systems with band structure. In fact, roughly speaking, the fill-in predicted by the George–Ng scheme is the same as the fill-in produced by the QR factorization of the matrix (see [21]) when Q is calculated using Householder transformations stored in product form. Therefore, the class of matrices for which the George–Ng scheme does not produce much fill-in is much larger than the class of band matrices, but even for band structures we found that the possibility of permuting rows to preserve stability is a positive advantage over band solvers (see [17, Chap. 8]), when numerical stability is critical.

This paper is organized as follows. In § 2 we describe the "local versions" of the algorithms used for our comparisons. That is, according to the descriptions in this section, the approximation x_{k+1} is always computed using (1.3) or (1.4) and (1.5) ignoring increases of the norm of F. These methods have local convergence properties, which are surveyed in § 3. In § 4 we describe our test functions and we present a numerical comparison of the local methods. In § 5 we discuss "global modifications" of the local methods and we present numerical experiments concerning the methods described in § 2 with a particular global modification. Finally, in § 6 we state some conclusions and we suggest some lines for future research.

2. The local algorithms. As we mentioned in § 1, we selected eight algorithms for our comparison: Newton's method, the modified Newton method, Broyden's first method, Schubert's method, the Dennis–Marwil method, and three methods of Martínez's family [34], [35]. We call the latter the diagonal-scaling method, the row-scaling method and the column-scaling method. All these methods use a small tolerance parameter $TOL > 0$, to detect and modify singularity of the matrices involved, and a step control Δ, to inhibit very large steps $x_{k+1} - x_k$. Moreover, a symbolic phase precedes the first iteration of all the algorithms, and even the first iteration is common to all of them. The symbolic phase corresponds to the symbolic manipulation of the George–Ng algorithm [20], and the first iteration corresponds to the following algorithm with $k = 0$.

ALGORITHM 2.1. NEWTON ITERATION.

Step 1. Compute $F(x_k)$, $J(x_k)$. Set $B_k = J(x_k)$.

Step 2. Compute a permutation P, a lower-triangular matrix $L = (l_{ij})$, an upper-triangular matrix $U = (u_{ij})$, and $2n$ sets I_i^L, I_i^U, $i = 1, \cdots, n$ such that

$$PB_k = LU, \tag{2.1}$$

$$l_{ii} = 1, \qquad i = 1, \cdots, n, \tag{2.2}$$

$$|l_{ij}| \leqq 1 \quad \text{for all } j = 1, \cdots, i, \qquad i = 1, \cdots, n, \tag{2.3}$$

$$\begin{gathered} I_i^L \subset \{1, \cdots, n\}, \\ I_i^U \subset \{1, \cdots, n\}, \qquad i = 1, \cdots, n, \end{gathered} \tag{2.4}$$

$$l_{ij} = 0 \quad \text{for all } j \notin I_i^L, \qquad j \neq i, \qquad i = 1, \cdots, n, \tag{2.5}$$

$$u_{ij} = 0 \quad \text{for all } j \notin I_i^U, \qquad j \neq i, \qquad i = 1, \cdots, n. \tag{2.6}$$

Step 3. If $|u_{ii}| < TOLB \equiv TOL \max_{i,j} |(B_k)_{ij}|$, replace u_{ii} by

$$sg(u_{ii})TOLB, \qquad i = 1, \cdots, n.$$

Step 4. Solve

$$Lw = -PF(x_k), \tag{2.7}$$

and

$$Us = w. \tag{2.8}$$

Step 5. If $\|s_k\|_\infty > \Delta$, replace s_k by $s_k \Delta / \|s_k\|_\infty$.

Step 6. $x_{k+1} = x_k + s_k$.

Matrices L, U, and the sets I_i^L, I_i^U which satisfy (2.1)-(2.6) are computed using the George-Ng algorithm. l_{ij} may be equal to zero for some $j<i$, $j \in I_i^L$, and u_{ij} may be equal to zero for some $j>i$, $j \in I_i^U$. In fact, the sets I_i^L, I_i^U represent the structural nonzero elements of any pair of matrices L, U when the partial pivoting algorithm is performed on a matrix with the nonzero structure of PB_k.

Algorithm 2.1 is a Newton iteration, with the safeguards against singularity and large steps given by Steps 3 and 5. Therefore, our Newton method may be described by the following algorithm.

ALGORITHM 2.2. NEWTON'S METHOD. Given an arbitrary initial point x_0, compute x_{k+1}, $k=0,1,2,\cdots$, using Algorithm 2.1.

The modified Newton method is described by the following algorithm.

ALGORITHM 2.3. MODIFIED NEWTON METHOD. Given x_0, compute x_1 using Algorithm 2.1. For $k=1,2,\cdots$, compute x_{k+1} performing Steps 4-6 of Algorithm 2.1.

Our implementation of Schubert's method requires the definition of n additional sets of indexes $I_i \subset \{1,\cdots,n\}$, $i=1,\cdots,n$. We define

$$I_i=\left\{j\in\{1,\cdots,n\} \,\middle|\, \frac{\partial f_i}{\partial x_j}(x)\neq 0 \text{ for some } x \text{ in the domain of } F\right\}. \tag{2.9}$$

With this definition, Schubert's method is described by Algorithm 2.4.

ALGORITHM 2.4. SCHUBERT'S METHOD. Given x_0, compute x_1 using Algorithm 2.1. For $k=1,2,\cdots$, compute x_{k+1} performing the following steps:

Step 1. Compute $y_{k-1}=F(x_k)-F(x_{k-1})$.
Step 2. Solve the optimization problem

$$\begin{gathered}\text{Minimize } \|B-B_{k-1}\|_F^2 \quad \text{s.t.}\\ B=(b_{ij}),\quad b_{ij}=0 \quad \text{if } j\notin I_i, \quad i=1,\cdots,n,\\ Bs_{k-1}=y_{k-1}.\end{gathered} \tag{2.10}$$

(Problem (2.10) always has a solution, the Frobenius projection of B_{k-1} on the feasible set, which is a linear manifold. Compact formulae for finding B_k, the unique solution of (2.10), are given in [37], [44].)
Step 3. Perform Steps 2-6 of Algorithm 2.1.

Algorithms 2.5-2.8 describe the Dennis-Marwil method, the diagonal-scaling method, the row-scaling method, and the column-scaling method, respectively. In the description of these methods, we denote $\{e_1,\cdots,e_n\}$, the canonical basis of $\mathbb{R}^n$.

ALGORITHM 2.5. DENNIS-MARWIL METHOD. Given x_0, compute x_1 using Algorithm 2.1. For $k=1,2,\cdots$, compute x_{k+1} performing the following steps:

Step 1. Compute $y_{k-1}=F(x_k)-F(x_{k-1})$.
Step 2. Solve $Lw=Py_{k-1}$.
Step 3. Define $v=Us_{k-1}$.
Step 4. For $i=1,\cdots,n$ perform Step 5.
Step 5. If

$$\left(\sum_{j\in I_i^U}(e_j^T s_{k-1})^2\right)^{1/2} \geqq 10^{-4}\cdot\|s_{k-1}\|_\infty$$

perform Step 5.1. Otherwise, increment i and repeat Step 5.

Step 5.1. For each j such that $u_{ij} \neq 0$, compute

$$u_{ij} \leftarrow u_{ij} + e_j^T s_{k-1} \cdot [e_i^T w - e_i^T v] \Big/ \sum_{j \in I_i^U} (e_j^T s_{k-1})^2.$$

Step 6. Perform Steps 3-6 of Algorithm 2.1.

The diagonal-scaling method [34], [35], the row-scaling method, and the column-scaling method belong to the family introduced by Martínez in [35].

As suggested by one of the referees of the first version of this paper, the diagonal-scaling method may be interpreted as follows. Given the factorization $PJ(x_0) = LU$, consider the function $\Phi(z)$ defined by

$$\Phi(z) = L^{-1} PF(U^{-1} z).$$

Define $z_k = Ux_k$ for all $k \geqq 0$. Clearly, $\Phi'(z_0) = I$. So, the diagonal-scaling method reduces to an iteration of the form

$$z_{k+1} = z_k - B_k^{-1} \Phi(z_k),$$

where the matrices B_k are chosen to be diagonal and to satisfy (if possible) the secant equation

$$B_{k+1}(z_{k+1} - z_k) = \Phi(z_{k+1}) - \Phi(z_k).$$

If some entry of B_{k+1} exceeds some a priori bound, it is left unchanged.

Similarly, the row-scaling method and the column-scaling method may be deduced by considering $\Phi(z) = U^{-1} L^{-1} PF(z)$ (with $z_k = x_k$) and $\Phi(z) = F(U^{-1} L^{-1} Pz)$ (with $z_k = P^{-1} LUx_k$), respectively.

ALGORITHM 2.6. DIAGONAL-SCALING METHOD. Given x_0, compute x_1 using Algorithm 2.1. Set $D = \text{Diag}\,(d_{ii}) = \text{Diag}\,(u_{ii})$ and replace U by $D^{-1} U$. For $k = 1, 2, \cdots$, compute x_{k+1} performing the following steps:

Step 1. Compute $y_{k-1} = F(x_k) - F(x_{k-1})$.
Step 2. Solve

$$Lw = Py_{k-1}.$$

Step 3. Define

$$v = Us_{k-1}. \tag{2.11}$$

Step 4. For $i = 1, \cdots, n$, execute Step 5.
Step 5. If

$$|e_i^T v| \geqq 10^{-4} \|s_{k-1}\|_\infty,$$

set $d_{ii} = e_i^T w / e_i^T v$; otherwise increment i and repeat this step.
Step 6. For $i = 1, \cdots, n$, if $|d_{ii}| < TOLB$, replace

$$d_{ii} \leftarrow sg(d_{ii})\, TOLB.$$

Step 7. Solve

$$LDUs_k = -PF(x_k).$$

Step 8. Perform Steps 5 and 6 of Algorithm 2.1.

ALGORITHM 2.7. ROW-SCALING METHOD. Given x_0, compute x_1 using Algorithm 2.1. Define $D = \mathrm{Diag}\,(d_{ii})$, $d_{ii} = 1$, $i = 1, \cdots, n$. Compute x_{k+1} performing the following steps:

Step 1. Compute $y_{k-1} = F(x_k) - F(x_{k-1})$.
Step 2. Define

$$v = LUs_{k-1}. \tag{2.12}$$

Step 3. Define $w = Py_{k-1}$.
Step 4. For $i = 1, \cdots, n$, execute Step 5.
Step 5. If

$$|e_i^T v| \geqq 10^{-4} \|F(x_k)\|_\infty,$$

set $d_{ii} = e_i^T w / e_i^T v$; otherwise increment i and repeat Step 5.
Step 6. For $i = 1, \cdots, n$, if $|d_{ii}| < TOLB$, replace

$$d_{ii} \leftarrow sg(d_{ii})\, TOLB.$$

Step 7. Solve

$$LDUs_k = -PF(x_k).$$

Step 8. Perform Steps 5 and 6 of Algorithm 2.1.

ALGORITHM 2.8. COLUMN-SCALING METHOD. Given x_0, compute x_1 using Algorithm 2.1. Define $D = \mathrm{Diag}\,(d_{ii})$, $d_{ii} = 1$, $i = 1, \cdots, n$. For $k = 1, 2, \cdots$, compute x_{k+1} performing the following steps:

Step 1. Define z as the solution of

$$LUz = PF(x_{k-1}). \tag{2.14}$$

Step 2. Solve

$$LUw = PF(x_k). \tag{2.15}$$

Step 3. Set

$$v = w - z.$$

Step 4. For $i = 1, \cdots, n$, execute Step 5.
Step 5. If

$$|e_i^T s_{k-1}| \geqq 10^{-4} \|s_{k-1}\|_\infty,$$

set $d_{ii} = e_i^T v / e_i^T s_{k-1}$; otherwise increment i and repeat Step 5.
Step 6. For $i = 1, \cdots, n$, if $|d_{ii}| < TOLB$, replace

$$d_{ii} \leftarrow sg(d_{ii})\, TOLB.$$

Step 7. Solve

$$\mathrm{Ds_k} = -\mathrm{w}.$$

Step 8. Execute Steps 5 and 6 of Algorithm 2.1.

Clearly, the row-scaling method and the column-scaling method may also be viewed as the methods which iterate according to (1.4) and (1.5), with $B_{k+1} = D_{k+1} J(x_0)$ or $B_{k+1} = J(x_0) D_{k+1}$, respectively, when the diagonal matrices D_{k+1} are chosen so as to satisfy the secant equation, if this is possible. Therefore, they may be interpreted as improvements (or variants) of the modified Newton method, and their implementation is hardly more expensive than the implementation of Algorithm 2.3. The expectation that the incorporation of secant information may consistently improve the performance of the modified Newton method motivated us to include Algorithms 2.7 and 2.8 in our comparative study. At Step 5 of Algorithms 2.5–2.8, we essentially test if it is possible to satisfy the secant equation for the ith component, with some relative tolerance. Later, we will comment on the relationship of this test to the overall performance of these algorithms.

Let us now describe our implementation of the first method of Broyden. As we mentioned in § 1, we used the inverse formula

$$B_{k+1}^{-1} = B_k^{-1} + \frac{(s_k - B_k^{-1}y_k)s_k^TB_k^{-1}}{s_k^TB_k^{-1}y_k}$$

storing at each iteration the vectors s_k and $s_k - B_k^{-1}y_k$. Hence we have the following product form for B_{k+1}^{-1}:

$$B_{k+1}^{-1} = \left[I + \frac{(s_k - B_k^{-1}y_k)s_k^T}{s_k^TB_k^{-1}y_k}\right] \cdots \left[I + \frac{(s_0 - B_0^{-1}y_0)s_0^T}{s_0^TB_0^{-1}y_0}\right]B_0^{-1}$$

or

$$B_{k+1}^{-1} = [I + u_ks_k^T] \cdots [I + u_0s_0^T]B_0^{-1}, \tag{2.16}$$

where

$$u_l = \frac{s_l - B_l^{-1}y_l}{s_l^TB_l^{-1}y_l}, \qquad l = 0, 1, \cdots, k. \tag{2.17}$$

Using (2.16) and (2.17), we may describe Broyden's method as follows.

ALGORITHM 2.9. BROYDEN'S METHOD. Given x_0, compute x_{k+1}, $k = 0, 1, 2, \cdots$ performing the following steps:

```
Step 1. If $k = 0$ execute Steps 1–4 of Algorithm 2.1. Go to Step 3.
Step 2. (Complete the computation of $s_k$.)
Compute
        $s_k = (I + u_{k-1}s_{k-1}^T)\tilde{s}_{k-1}$.
Step 3. (Normalize and compute the new point.)
        $\bar{s}_k \leftarrow s_k$,
        $s_k \leftarrow \lambda_k s_k$   where $\lambda_k = \min\left\{1, \frac{\Delta}{\|s_k\|_\infty}\right\}$,
        $x_{k+1} = x_k + s_k$.
Step 4. (Compute $u_k$.)
    Execute Steps 4.1–4.3.
        Step 4.1 (Compute $\tilde{s}_k = -B_k^{-1}F(x_{k+1})$.)
        Execute Steps 4.1.1–4.1.2.
            Step 4.1.1 (Compute $\tilde{s} = -B_0^{-1}F(x_{k+1})$.)
            Solve $LU\tilde{s} = -PF(x_{k+1})$.
            If $k = 0$ set $\tilde{s}_k = \tilde{s}$ and go to Step 4.2.
            Step 4.1.2
            Compute
            $\tilde{s}_k = (I + u_{k-1}s_{k-1}^T) \cdots (I + u_0s_0^T)\tilde{s}$.
    Step 4.2 (Compute $v_k = B_k^{-1}y_k \equiv B_k^{-1}F(x_{k+1}) - B_k^{-1}F(x_k)$.)
    $v_k = \bar{s}_k - \tilde{s}_k$.
    Step 4.3
    Compute $u_k = (s_k - v_k)/(s_k^T v_k)$.
Step 5. $k \leftarrow k+1$.
```

The computational cost of one iteration of Broyden's method, as described above, is $2nk+2n$ flops plus the resolution of two sparse triangular systems. Griewank [23] proposed an alternative implementation which has the same complexity as the one proposed here.

Of course, since both the memory and time of one iteration of Algorithm 2.9 tend to infinity with k, this algorithm must be restarted, say, when k is a multiple of some fixed integer q.

Remark. We used the words "solve" or "compute" to indicate that some calculation must effectively be performed, and the word "define" to indicate that the result of the calculation may be obtained from previous computations.

2.1. Singularity and step control. Step 3 of Algorithm 2.1 and Step 6 of Algorithms 2.6–2.8 correct the LU factorization when a nearly singular matrix B_k appears. However, a very ill conditioned B_k may still occur, provoking very large steps $-B_k^{-1}F(x_k)$. Therefore, in practical implementation of methods for solving nonlinear systems of equations, some control in the size of increments $x_{k+1}-x_k$ is recommended (see [20], [25]). In our codes, we adopted a very simple way of controlling the stepsize. Assuming that Δ is given by the user as an estimate of the distance between the initial guess and the solution, we simply test if $\|s_k\|_\infty = \|-B_k^{-1}F(x_k)\|_\infty$ is less than Δ or not. If it is, then the increment s_k is accepted. Otherwise, it is replaced by $s_k\Delta/\|s_k\|_\infty$. This is done at Steps 5 and 6 of Algorithm 2.1 and at Step 3 of Algorithm 2.9.

In Algorithm 2.9, the singularity of B_{k+1} is represented by the annihilation of the denominator $s_k^T v_k$ when computing u_k at Step 4. Consequently, in this algorithm, we decided to declare B_{k+1} singular if

$$|s_k^T v_k| \leqq SQMAP \|s_k\|_2 \|v_k\|_2$$

where $SQMAP$ is the square root of the machine precision. When this "quasi singularity" is detected, we reset $B_{k+1} = B_k$.

2.2. Stopping criteria. A natural stopping criterion for algorithms which solve nonlinear systems is

$$\|F(x_k)\|_\infty \leqq \varepsilon_1, \tag{2.18}$$

where ε_1 is a small positive number given by the user. When (2.18) occurs, we declare "convergence of type 0."

In production codes, (2.18) is generally replaced by a relative convergence criterion

$$\|F(x_k)\|_\infty \leqq \varepsilon_1 \|F(x_0)\|_\infty$$

(see [15]), but for comparative studies (2.18) is adequate.

However, sometimes criterion (2.18) is very difficult or impossible to achieve, maybe because of a large Jacobian at the solution. So we incorporate another convergence test:

$$\|s_k\|_\infty < \varepsilon_2 \|x^{k+1}\|_\infty + 10^{-25}. \tag{2.19}$$

When (2.19) holds, we declare "convergence of type 1."

Algorithms like Algorithms 2.2–2.9 are local in nature, and, therefore, divergence may occur for arbitrary nonlinear systems if x_0 is far from the solution. We declare "divergence" when, for some large positive number *BIG*, given by the user, the following inequality holds:

$$\|F(x_k)\|_\infty > BIG. \tag{2.20}$$

As in the case of the first convergence criterion, this "divergence criterion" is used only for comparison purposes. In production codes, we prefer to use criteria based on "lack of enough progress" as recommended, for example, in [39], [33]. We will return to this discussion in § 5.

Finally, the execution of the programs is interrupted when either a previously defined computer time or some large number of iterations is exceeded.

2.3. Restarting criteria. Sometimes, rather than executing Algorithms 2.3-2.9 in their original version, it is more efficient to restart the iterative process, performing a Newton iteration (Algorithm 2.1) instead of the original iteration, at certain steps k. The most natural restarting criterion is to execute Algorithm 2.1 for obtaining x_{k+1} when k is a multiple of a fixed integer q. For the modified Newton method, in absence of sparsity, an optimal q may be determined using Ostrowski's efficiency index (see [1], [32], [47]). However, the optimization of Ostrowski's efficiency proposed by Brent [1] is based on the theoretical asymptotic behavior of the algorithms and so it may not be very useful far from the solution of the system. So, we decided to develop a local efficiency restarting criterion (LERC) based on the following arguments.

Let us call t_k the computer time used by some algorithm at iteration k. Assume, further, that

$$\|F(x_{k+1})\| = \theta_k \|F(x_k)\|. \tag{2.21}$$

Therefore, if relation (2.21) is maintained throughout the calculation with $\theta_k < 1$ and the same type of iteration is used, the computer time to achieve (2.18) should be proportional to $-t_k/\log \theta_k$. This justifies the definition of E_k, the efficiency of iteration k, as

$$\begin{aligned} E_k &= \frac{-\log \theta_k}{t_k} \quad \text{if } \theta_k < 1, \\ E_k &= 0 \qquad\qquad \text{otherwise.} \end{aligned} \tag{2.22}$$

Assume now that l is the last index of a Newton iteration previous to iteration k. We adopt the following criterion for deciding whether to use a "normal" iteration or a Newton-type one at iteration $k+1$. If $\theta_k > 1$ or $E_l \geqq E_k$, iteration $k+1$ must be a Newton iteration. Otherwise, iteration $k+1$ must be a "normal" iteration.

A similar criterion is used in [11] for minimization problems, with good numerical results.

3. Theoretical properties. Let us assume that $F: \Omega \subset \mathbb{R}^n \to \mathbb{R}^n$, Ω an open and convex set, $F \in C^1(\Omega)$, $F(x_*) = 0$, $J(x_*)$ nonsingular, and, for all $x \in \Omega$,

$$\|J(x) - J(x_*)\| \leqq L\|x - x_*\|^p$$

for some L, $p > 0$.

For a local convergence analysis, consider the algorithms described in § 2, without correction of singularity and with no control of the stepsize. That is, eliminate Steps 3 and 5 of Algorithm 2.1, Step 4 of Algorithm 2.5, Step 6 of Algorithms 2.6-2.8, and the correction of s_k at Step 3 of Algorithm 2.9. In fact, local convergence theorems show that these steps are not necessary with the hypotheses above, if x_0 is near enough x_*. Let us survey here the convergence results related to Algorithms 2.2-2.9. The first one concerns local convergence of Newton's method and the modified Newton method.

THEOREM 3.1. *Given $r \in (0, 1)$, there exists $\varepsilon = \varepsilon(r) > 0$ such that if $\|x_0 - x_*\| \leqq \varepsilon$, the sequences (x_k) generated by Algorithm 2.2 or 2.3 converge to x_* and satisfy*

$$\|x_{k+1} - x_*\| \leqq r\|x_k - x_*\| \tag{3.1}$$

for all $k = 0, 1, 2, \cdots$. Moreover, for Algorithm 2.2 (Newton's method), there exists $c > 0$ such that

$$\|x_{k+1} - x_*\| \leqq c\|x_k - x_*\|^{p+1} \tag{3.2}$$

for all $k = 0, 1, 2, \cdots$.

Proof. See [15], [41], [46].

Like many quasi-Newton algorithms, Broyden's method and Schubert's method satisfy not only the linear convergence result (3.1), but a stronger (superlinear) convergence result which, on the other hand, is weaker than (3.2).

THEOREM 3.2. *Given $r \in (0, 1)$, there exists $\varepsilon = \varepsilon(r) > 0$ such that if $\|x_0 - x_*\| \leqq \varepsilon$, the sequence (x_k) generated by Algorithm 2.4 (Algorithm 2.9) converges to x_* and satisfies* (3.1). *Moreover, the speed of convergence is Q-superlinear, that is,*

$$\lim_{k\to\infty} \|x_{k+1} - x_*\| / \|x_k - x_*\| = 0. \tag{3.3}$$

Proof. See [15].

Algorithms 2.6–2.8 have the same type of convergence result as the modified Newton method.

THEOREM 3.3. *Given $r \in (0, 1)$, there exists $\varepsilon = \varepsilon(r) > 0$ such that if $\|x_0 - x_*\| \leqq \varepsilon$, the sequences (x_k) generated by Algorithm 2.6, 2.7, or 2.8 converge to x_* and satisfy* (3.1).

Proof. See [34] for the convergence of Algorithm 2.6. For proving the convergence of Algorithms 2.7 and 2.8, we need to interpret them as particular cases of the family introduced in [35]. This may be easily done: for Algorithm 2.7, $C(B) = I$, $D(B) = I$, $E(B) = B$, and for Algorithm 2.8, $C(B) = B$, $D(B) = I$, $E(B) = I$. Since these functions are trivially continuous, Theorem 2.1 of [35] may be applied. □

We are almost sure that the Dennis-Marwil method, without restarts, is not locally convergent. However, the members of a closely related family of methods introduced recently in [36] have local and superlinear convergence. We call the members of this family "quasi-Dennis-Marwil" methods. Given $\alpha \in (0, 1)$, the quasi-Dennis-Marwil method defined by α may be described by the following algorithm.

ALGORITHM 3.1. Given x_0, obtain x_1 using Algorithm 2.1. For computing x_k, $k = 1, 2, \cdots$, use the recurrence

$$x_{k+1} = x_k - (L_k U_k)^{-1} P F(x_k), \tag{3.4}$$

computing, at each iteration, L_{k+1}^{-1}, U_{k+1} as the solution of the following optimization problem:

$$\begin{aligned} &\text{Minimize}_{M,U}\ \alpha\|M - L_K^{-1}\|_F^2 + (1-\alpha)\|U - U_k\|_F^2 \quad \text{s.t.} \\ &U s_k = M y_k, \\ &s_k = x_{k+1} - x_k, \\ &y_k = P[F(x_{k+1}) - F(x_k)], \\ &U = (u_{ij}) | u_{ij} = 0 \quad \text{if } j \notin I_i^U. \end{aligned} \tag{3.5}$$

The relation between the Dennis-Marwil method and the method described above is given in the following theorem.

THEOREM 3.4. *Suppose x_k, L_k, U_k are given, and let us call L_{k+1}, U_{k+1} the matrices obtained using the Dennis-Marwil method. Of course, $L_{k+1}=L_k$. Let us call $L_{k+1}(\alpha)$, $U_{k+1}(\alpha)$ the ones obtained using Algorithm* 3.1. *Then,*

$$\lim_{\alpha\to 1} L_{k+1}(\alpha)=L_{k+1}, \tag{3.6}$$

$$\lim_{\alpha\to 1} U_{k+1}(\alpha)=U_{k+1}. \tag{3.7}$$

Proof. See [36].

The quasi-Dennis-Marwil methods are not practical for solving sparse nonlinear systems because sparsity of L_k is not preserved from one iteration to another. However, they have the same local convergence properties as Schubert's method (see [36]). Therefore, although the Dennis-Marwil method seems to have the poorest convergence properties among the algorithms described in § 2, the fact that "very analogous" methods in the sense of (3.6) and (3.7) have good local convergence properties makes us feel that "some part" of these properties is inherited by the Dennis-Marwil method.

Up to now, we have considered the convergence properties of the "pure" algorithms of § 2. If restarts are incorporated, the convergence results for quasi-Newton methods look very similar. In the following theorems, we consider the "restarted" versions of Algorithms 2.3 and 2.5-2.8. This means that for infinitely many iterations, x_{k+1} is computed using Algorithm 2.1, which involves a complete resetting of the LU factorization.

THEOREM 3.5. *Assume that (x_k) is obtained using Algorithm* 2.3, 2.6, 2.7, *or* 2.8, *except that for $k\in K_0$, an infinite set of indexes, the* LU *factorization of $J(x_k)$ is computed and x_{k+1} is calculated using Algorithm* 2.1. *Then there exists $\varepsilon>0$ such that if $\|x_0-x_*\|\leqq\varepsilon$, (x_k) converges Q-superlinearly to x_*.*

Proof. See [35] or use Theorems 3.1 and 3.3.

For the Dennis-Marwil method, the following slightly weaker result holds.

THEOREM 3.6. *Assume that (x_k) is obtained using Algorithm* 2.5, *except that for $k\in K_0$, an infinite set of indexes, the* LU *factorization of $J(x_k)$ is calculated and x_{k+1} is computed using Algorithm* 2.1. *Assume further that the difference between any pair of consecutive indexes of K_0 is never greater than a fixed integer q. Then the thesis of Theorem* 3.5 *holds for the sequence (x_k).*

Proof. See [12].

Remark. As a final remark in this section, let us stress the relationship between local convergence results of Algorithms 2.5 and 2.6. As we mentioned in § 1, the convergence results for these algorithms impose that the pivoting rule P which is used to factorize $J(x_0)$ will also allow the factorization of $J(x_*)$. This property will be satisfied by any pivoting rule which uses a threshold strategy like the one described by Dennis and Marwil in [12]. Now the partial pivoting strategy is just the strongest threshold strategy, since it uses the larger tolerance parameter $\mu=1$. Therefore, it is adequate for use in this type of algorithm. Moreover, if we use the partial pivoting strategy at $J(x_0)$, it is more likely that the same permutation matrix P allows the factorization of $J(x_*)$ than if we use less strict strategies, since the coefficients of the LU factorization (given P) are continuous functions of x. Therefore, the size of the neighborhoods of Theorems 3.5 and 3.6 is expected to be greater when we use partial pivoting than when we use any other threshold strategy.

4. Numerical experiments with the local algorithms. We wrote a FORTRAN code which implements the methods described in § 2. All the reported tests were run on a

VAX11/785 at the State University at Campinas, using the FORTRAN 77 compiler and the VMS Operational System. Single precision was used for real variables in all our tests. A compatible IBM-PC version of the code was also written using a Microsoft Fortran compiler. The results for this version were consistent with the results of the VAX version of the code.

We use the following notation:

N: Newton's method (Algorithm 2.2),
MN: Modified Newton method (Algorithm 2.3),
S: Schubert's method (Algorithm 2.4),
DM: Dennis-Marwil method (Algorithm 2.5),
DS: Diagonal-scaling method (Algorithm 2.6),
RS: Row-scaling method (Algorithm 2.7),
CS: Column-scaling method (Algorithm 2.8),
B: Broyden's method (Algorithm 2.9).

The numerical performance of the algorithms is described in Tables 1 and 2. In Table 1 we report the number of iterations used for an algorithm on each particular problem. The exit condition is represented by an output parameter IER. IER may assume five values: 0 for convergence of type 0, 1 for convergence of type 1, 2 for divergence, 3 for exceeded number of iterations and 4 for exceeded CPU time. In Table 2 we report the respective CPU times.

Each algorithm was run without restarts and with restarts based on the efficiency criterion described in § 3.

Let us describe the functions used in our numerical study.

Function 1. Broyden Tridiagonal. See [2], [3].

TABLE 1
Number of iterations used by the local algorithms.

Function	n	Restart	N	MN	S	DM	DS	RS	CS	B
1	5000	No	3 (3)*	9 (1)^1†	6 (1)	5 (1)	5 (1)^1	6 (1)	5 (1)^1	6 (1)
1	5000	Yes		9 (1)^1	4 (2)	5 (1)	5 (1)^1	6 (2)	5 (1)^1	6 (1)
2	5000	No	4 (4)	17 (1)^1	9 (1)	11 (1)^1	6 (1)	6 (1)	6 (1)	9 (1)
2	5000	Yes		17 (1)^1	5 (3)	8 (2)	6 (1)	6 (1)	6 (1)	9 (1)
3 $x_0=\underline{0}$	5000	No	8 (8)	100 (1)^3	*‡	46 (1)^2	100 (1)^3	13 (1)^2	100 (1)^3	11 (1)^2
3 $x_0=\underline{0}$	5000	Yes		13 (5)	10 (5)	12 (4)	12 (3)	11 (3)	12 (3)	13 (3)
3 $x_0=\underline{0.3}$	5000	No	6 (6)	100 (1)^3	11 (1)	12 (1)	19 (1)^1	100 (1)^3	13 (1)^1	6 (1)^2
3 $x_0=\underline{0.3}$	5000	Yes		11 (3)^1	7 (4)	8 (2)^1	9 (3)	8 (3)	9 (3)	8 (3)
4	225	No	3 (3)	5 (1)	4 (1)	5 (1)	7 (1)^1	6 (1)^1	6 (1)^1	4 (1)^1
4	225	Yes		5 (1)	4 (2)	5 (1)	4 (2)^1	5 (2)^1	5 (2)	4 (1)^1
4	961	No	4 (4)^1	5 (1)^1	5 (1)^1	5 (1)^1	8 (1)^1	5 (1)^1	6 (1)^1	4 (1)^1
4	961	Yes		5 (1)^1	4 (2)^1	5 (1)^1	4 (2)^1	5 (2)^1	6 (2)^1	4 (1)^1
5 $b=100$	1000	No	4 (4)	11 (1)^1	6 (1)	7 (1)	6 (1)^1	6 (1)	6 (1)^1	7 (1)
5 $b=100$	1000	Yes		11 (1)^1	4 (2)	7 (1)	6 (1)^1	6 (1)	6 (1)^1	7 (1)
6	5000	No	4 (4)	14 (1)	7 (1)	8 (1)	10 (1)	7 (1)^1	8 (1)^1	8 (1)
6	5000	Yes		14 (1)	5 (3)	6 (2)	6 (2)	6 (2)	6 (2)	8 (1)
7	5000	No	9 (9)	100 (1)^3	100 (1)^3	*	12 (1)	12 (1)	12 (1)	34 (1)^1
7	5000	Yes		16 (5)	11 (6)	11 (6)	12 (1)	12 (1)	12 (1)	13 (2)

* The number in parentheses is the number of Newton iterations.

† The superscript is the exit condition (IER). If IER = 0, the superscript is omitted.

‡ * = overflow.

TABLE 2
CPU *time (seconds) used by the local algorithms.*

Function	n	Restart	N	MN	S	DM	DS	RS	CS	B
1	5000	No	3.97	4.97	9.2	4.67	3.78	4.23	3.66	4.79
1	5000	Yes		5.1	6.07	4.95	3.70	4.94	3.70	4.72
2	5000	No	19.9	23.7	49.7	25.1	11.5	11.5	11.6	16.6
2	5000	Yes		23.0	28.2	21.4	11.6	11.5	11.3	16.7
3 $x_0=\underline{0}$	5000	No	15.0	NC*	NC	NC	NC	NC	NC	NC
3 $x_0=\underline{0}$	5000	Yes		15.3	18.2	16.2	13.0	12.5	13.2	16.9
3 $x_0=\underline{0.3}$	5000	No	11.7	NC	19.8	14.4	16.3	NC	11.8	NC
3 $x_0=\underline{0.3}$	5000	Yes		11.4	13.3	10.9	11.0	9.86	10.9	10.9
4	225	No	1.97	1.08	2.71	1.38	1.33	1.27	1.23	1.05
4	225	Yes		1.15	2.7	1.34	1.70	1.74	1.63	0.99
4	961	No	40.9	13.1	51.5	16.1	16.2	13.4	13.5	13.1
4	961	Yes		13.7	41.2	16.1	23.6	23.1	23.6	12.6
5 $b=100$	1000	No	38.8	16.4	58.1	17.1	13.6	13.0	12.9	13.8
5 $b=100$	1000	Yes		17.3	39.5	17.8	13.4	13.0	13.0	13.9
6	5000	No	10.0	10.5	23.0	12.8	9.73	7.06	7.92	9.08
6	5000	Yes		11.0	14.7	11.1	8.19	8.01	8.2	9.38
7	5000	No	12.9	NC	NC	NC	8.21	8.12	8.41	56.2
7	5000	Yes		12.5	18.1	13.2	8.18	8.21	8.35	11.3

* NC = nonconvergence. (See Table 1.)

$$f_1(x)=(3-hx_1)x_1-2x_2+1,$$
$$f_i(x)=(3-hx_i)x_i-x_{i-1}-2x_{i+1}+1, \qquad i=2,\cdots,n-1,$$
$$f_n(x)=(3-hx_n)x_n-x_{n-1}+1,$$
$$x_0=(-1,\cdots,-1)^T.$$

Algorithmic parameters: $\varepsilon_1=\varepsilon_2=10^{-4}$, $TOL=10^{-7}$, $\Delta=10$, $BIG=10^{10}$.

We ran this test with $h=0.5$ and $h=2$ and for $n=1000$, 2000, 3000, 4000, and 5000. Since no qualitative differences were detected, we report the results in Table 1 only for $h=2$ and $n=5000$. The computer CPU time of the symbolic phase of the methods (previous to the first iteration) obeys approximately the law

$$\text{time}=0.33n \text{ milliseconds.}$$

Now we give the observed relationships between n and CPU time of a typical iteration of the Algorithms 2.2-2.9:

N:	time $=0.25\ n$ milliseconds,
MN:	time $=0.09\ n$ milliseconds,
S:	time $=0.29\ n$ milliseconds,
DM:	time $=0.17\ n$ milliseconds,
DS, RS, and CS:	time $=0.11\ n$ milliseconds,
B:	time $=(0.13+0.005k)\ n$ milliseconds.

Function 2. Band Broyden [3].

$$f_i(x)=(3+5x_i^2)x_i+1-\sum_{j\in I_i}(x_j+x_j^2), \qquad i=1,\cdots,n,$$

where

$$I_i = \{i_1, \cdots, i_2\} - \{i\},$$

$$i_1 = \max\{1, i-5\}, \qquad i_2 = \min\{n, i+5\},$$

$$x_0 = (-1, \cdots, -1)^T, \quad \varepsilon_1 = \varepsilon_2 = 10^{-4}, \quad TOL = 10^{-7}, \quad \Delta = 10, \quad BIG = 10^{10}.$$

We ran this test for $n = 1000$, 2000, 3000, 4000, and 5000. For the same reasons as in Problem 1, we report in Table 1 the results only for $n = 5000$.

The CPU time of the symbolic phase of the George-Ng algorithm is approximately $1.22n$ milliseconds.

The CPU time of a typical iteration of each of the implemented methods obeys the following relationships:

N:	time $= 1.02\ n$ milliseconds,
MN:	time $= 0.22\ n$ milliseconds,
S:	time $= 1.15\ n$ milliseconds,
DM:	time $= 0.39\ n$ milliseconds,
DS, RS, and CS:	time $= 0.24\ n$ milliseconds,
B:	time $= (0.24 + 0.015\ k)\ n$ milliseconds.

Function 3. Trigexp [48].

$$f_1(x) = 3x_1^3 + 2x_2 - 5 + \sin(x_1 - x_2)\sin(x_1 + x_2),$$

$$f_i(x) = -x_{i-1}\, e^{(x_{i-1} - x_i)} + x_i(4 + 3x_i^2) + 2x_{i+1}$$
$$+ \sin(x_i - x_{i+1})\sin(x_i + x_{i+1}) - 8, \qquad i = 2, \cdots, n-1,$$

$$f_n(x) = -x_{n-1}\, e^{(x_{n-1} - x_n)} + 4x_n - 3.$$

Initial points: $(0, \cdots, 0)^T$ and $(0.3, \cdots, 0.3)^T$.

Algorithmic parameters: $\varepsilon_1 = \varepsilon_2 = 10^{-4}$, $TOL = 10^{-7}$, $\Delta = 10$, $BIG = 10^{10}$.

The Jacobian matrix of this system is tridiagonal, as in Problem 1. Therefore, the CPU time of the symbolic factorization is the same as in Problem 1. But the function is far more expensive to evaluate in this case. The CPU times of typical iterations of the algorithms, as functions of n, obey approximately the following laws:

N:	time $= 0.39\ n$ milliseconds,
MN:	time $= 0.15\ n$ milliseconds,
S:	time $= 0.34\ n$ milliseconds,
DM:	time $= 0.22\ n$ milliseconds,
DS, RS, and CS:	time $= 0.17\ n$ milliseconds,
B:	time $= (0.17 + 0.02\ k)\ n$ milliseconds.

As in the previous cases, we ran this problem for $n = 1000, 2000, \cdots, 5000$, but we report the results only for $n = 5000$.

Function 4. Poisson Problem [45]. This problem is the nonlinear system of equations arising from finite difference discretization of the Poisson boundary value problem

$$\Delta u = \frac{u^3}{1 + s^2 + t^2}, \qquad 0 \leqq s \leqq 1, \qquad 0 \leqq t \leqq 1,$$

$$u(0, t) = 1, \qquad u(1, t) = 2 - e^t, \qquad t \in [0, 1],$$

$$u(s, 0) = 1, \qquad u(s, 1) = 2 - e^s, \qquad s \in [0, 1].$$

We use an L^2 grid with $L = 15$ and $L = 31$. Therefore $n = 225$ and $n = 961$, respectively.

We ran the algorithms with $x_0 = (-1, \cdots, -1)^T$, $\varepsilon_1 = \varepsilon_2 = 10^{-4}$, $BIG = 10^{10}$, $\Delta = 5$.

The CPU times of the symbolic phase of the method of George and Ng for $L = 15$ and $L = 31$ were 1.31 seconds and 16.63 seconds, respectively. The CPU times of one ordinary iteration of the algorithms (in seconds) are shown in Table 3.

TABLE 3

Algorithm	$L=15$	$L=31$
N:	0.66	10.2
MN:	0.10	0.67
S:	0.68	10.3
DM:	0.17	1.23
DS:	0.13	0.74
RS:	0.10	0.53
CS:	0.11	0.88
$B(k=1)$:	0.15	0.89

As expected, the CPU times of Newton's and Schubert's iterations are roughly proportional to $L^4(=L^2n)$ while the CPU times of the other iterations are proportional to $L^3(=Ln)$.

Function 5.

$$f_1(x)=-2x_1^2+3x_1-2x_2+0.5x_{\alpha_1}+1.0,$$

$$f_i(x)=-2x_i^2+3x_i-x_{i-1}-2x_{i+1}+0.5x_{\alpha_i}+1.0, \qquad i=2,\cdots,n-1,$$

$$f_n(x)=-2x_n^2+3x_n-x_{n-1}+0.5x_{\alpha_n}+1.0$$

for α_i, $i=1,2,\cdots,n$, randomly chosen in the intervals: $\alpha_i\in\{\alpha_{i\,\min},\alpha_{i\,\max}\}$, where $\alpha_{i\,\min}=\max\{1,i-b\}$ and $\alpha_{i\,\max}=\min\{n,i+b\}$ for a parameter b which defines the bandwidth.

We used $(-1,\cdots,-1)^T$ as an initial point and, as in previous tests, $\varepsilon_1=\varepsilon_2=10^{-4}$, $TOL=10^{-7}$, $\Delta=10$, $BIG=10^{10}$. The structure of some typical Jacobian matrices with $n=40$, for different bandwidths and their correspondent data structures for LU factorizations, are given in Figs. 1 and 2.

The CPU time of the symbolic phase of the algorithm (with $b=100$ and $n=1000$) was 12.37 seconds.

The CPU time of typical iterations of the implemented methods for $b=100$, $n=1000$, was the following:

N: 9.7 seconds,

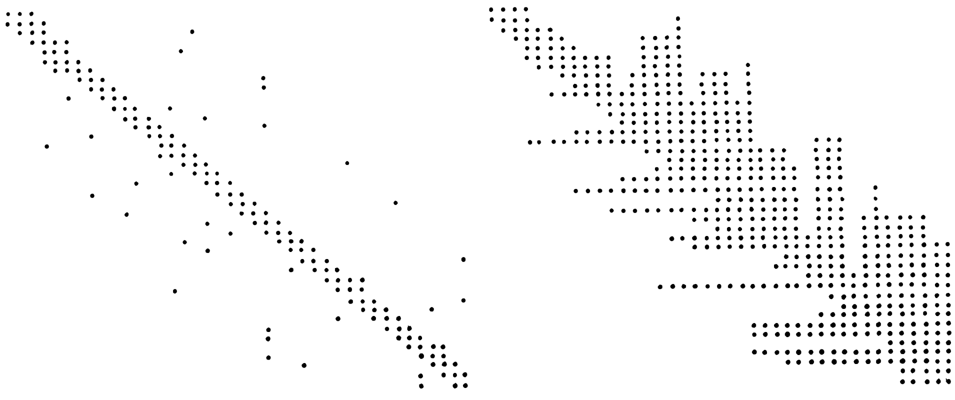


FIG. 1. *Structure of the Jacobian matrix and data structure for the* LU *factorization of Function* 5 ($b=15$).

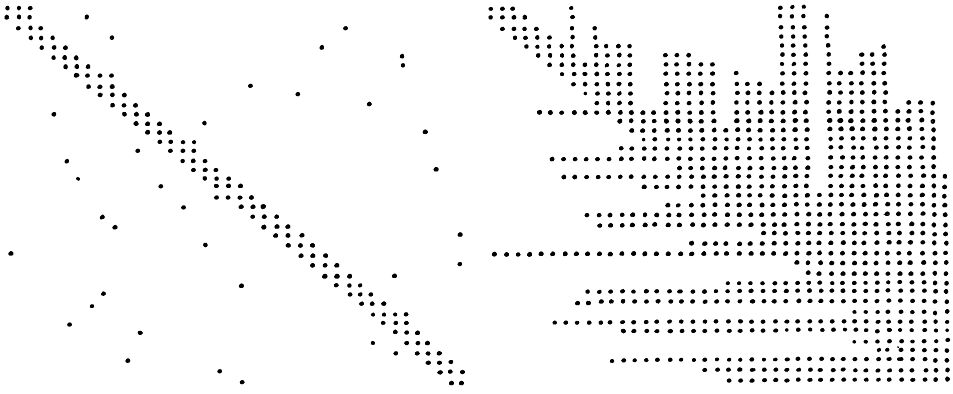


FIG. 2. *Structure of the Jacobian matrix and data structure for the* LU *factorization of Function* 5 ($b=30$).

MN: 0.67 seconds,
S: 9.68 seconds,
DM: 1.23 seconds,
DS: 0.78 seconds,
RS: 0.66 seconds,
CS: 0.64 seconds,
B: 0.68 seconds.

Function 6.

$$f_1(x)=-2x_1^2+3x_1+3x_{n-4}-x_{n-3}-x_{n-2}+0.5x_{n-1}-x_n+1,$$

$$f_i(x)=-2x_i^2+3x_i-x_{i-1}-2x_{i+1}+3x_{n-4}-x_{n-3}-x_{n-2}+0.5x_{n-1}-x_n+1, \qquad i=2,\cdots,n-1,$$

$$f_n(x)=-2x_n^2+3x_n-x_{n-1}+3x_{n-4}-x_{n-3}-x_{n-2}+0.5x_{n-1}-x_n+1.$$

We used $(-1,\cdots,-1)^T$ as initial point and the algorithmic parameters $\varepsilon_1=\varepsilon_2=10^{-4}$, $TOL=10^{-7}$, $\Delta=10$, $BIG=10^{10}$.

The structure of the Jacobian matrix and the correspondent data structure for the LU factorization produced by the symbolic phase of the George–Ng algorithm for $n=40$ is given in Fig. 3. The CPU time of the symbolic phase was $0.69n$ milliseconds. The CPU times of typical iterations of the implemented methods obey approximately the following laws:

N: time $=0.51\ n$ milliseconds,
MN: time $=0.12\ n$ milliseconds,
S: time $=0.68\ n$ milliseconds,
DM: time $=0.31\ n$ milliseconds,
DS, RS, and CS: time $=0.18\ n$ milliseconds,
B: time $=(0.16+0.008\ k)\ n$ milliseconds.

Function 7. Singular Broyden.

$$f_1(x)=((3-hx_1)x_1-2x_2+1)^2,$$

$$f_i(x)=((3-hx_i)x_i-x_{i-1}-2x_{i+1}+1)^2, \qquad i=2,\cdots,n-1,$$

$$f_n(x)=((3-hx_n)x_n-x_{n-1}+1)^2.$$

Initial point $=(-1,\cdots,-1)^T$.

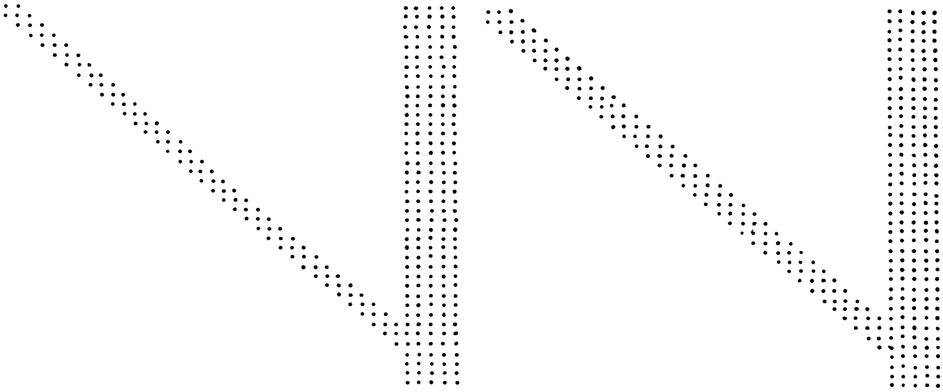


FIG. 3. *Structure of the Jacobian matrix and data structure for the* LU *factorization of Function* 6.

Algorithmic parameters: $\varepsilon_1 = \varepsilon_2 = 10^{-4}$, $TOL = 10^{-7}$, $\Delta = 10$, $BIG = 10^{10}$.

Clearly, this problem is equivalent to Problem 1, but its Jacobian matrix is singular at the solution. It was concluded in our comparative study to investigate the behavior of different algorithms in the presence of singular Jacobians. The computer CPU times of individual iterations are very similar to those of Function 1. As in Function 1, we will report the results only for $h = 2$ and $n = 5000$.

5. Global implementations. Many researchers and users of nonlinear equations codes think that too much consideration is given in the literature to local convergence properties of algorithms without regard to the possibility of developing global convergent methods derived from the local convergent ones.

However, there are a number of good reasons for that detected "deviation."

(a) Many practical problems have natural initial points which are not very far from the solution or, at least, are near enough to ensure a good behavior of purely local algorithms. This is the case of many boundary value problems and of the methods based on restoration of nonlinear feasibility in nonlinear programming.

(b) The most natural objective function which is associated to problem (1.1) is the sum of squares:

$$f(x) = \tfrac{1}{2}\|F(x)\|_2^2. \tag{5.1}$$

"Globally convergent" algorithms are ensured to converge to a stationary point of $f(x)$, possibly a local minimizer of f. Unhappily, $\nabla f(x)$ may be zero at a point where $F(x) \neq 0$ if $J(x)$ is singular. Clearly, a local minimizer of f is not a good answer to problem (1.1), if $F(x) \neq 0$. Many times, globally convergent implementations converge to undesirable local minimizers of f, while local methods (with a suitable control of steplength) converge to solutions of (1.1).

(c) Frequently, a sequence (x_k) generated by a particular local method converges to a solution x_* of (1.1) but $\|F(x_k)\|_2$ is not monotonically decreasing. In these cases a "global modification" of the sequence produces decrease of $\|F(x_k)\|$ at each iteration but at a cost of a lower speed of convergence. Sometimes, as was mentioned in [6], the new sequence converges to an undesirable local minimizer of f.

(d) Many times quasi-Newton methods exhibit a "jumping behavior," that is, while the modified Newton method usually converges in a monotone way (say, with $\|x_{k+1} - x_*\|_\infty \leqq \|x_k - x_*\|_\infty$ for all k), many quasi-Newton algorithms fail to exhibit that monotonical behavior but, finally, they converge much more quickly than the modified Newton method. For instance, when Broyden's method is applied to a linear system of equations (with $B_0 = I$), it is common to detect a number of apparently chaotic iterations before finite convergence occurs when $k = 2n$, according to Gay's theorem [19]. So, imposing norm decreasing in these cases destroys, in some sense, accumulated information of the sequence and slows down speed of convergence.

(e) When local algorithms do not converge at all, one of the most effective ways of solving problems is the "continuation strategy" (see [9], [41], [43], [49]-[51], etc.). This strategy essentially consists in replacing (1.1) by a family of problems $h(x, t) = 0$, $t \in [0, 1]$, such that $F(x) \equiv h(x, 1)$ and the solution of $h(x, 0) = 0$ is known. Sometimes the choice of h is suggested by the physical nature of the problem, but some standard mathematical choices are known to be effective (see [50]). The central idea is that one solves a sequence of problems $h(x, t_k) = 0$ so that problem $k+1$ is easy to solve using a local algorithm if the solution of problem k is known. Therefore the global continuation (or homotic) strategy essentially needs good local algorithms to be successful.

(f) Due to the numerical studies of Cosnard [8] and Moré and Cosnard [39], we know that trust region-like modifications of local algorithms (like the dogleg strategy of Powell [42]) may not be entirely satisfactory to ensure robustness of methods to solving nonlinear equations. After many numerical experiments, those authors found that the purely local Brent's method (see [1], [32]) was surprisingly effective when compared with the hybrid method of Powell [42], which is a global modification of Broyden's method. Of course, they also detected some cases where Powell's method converged while the local method did not. Due to the difficulty of incorporating global strategies based on the norm of F to Brent's method (the whole vector $F(x_k)$ is not calculated at any point in this method), Moré and Cosnard developed some step control procedures which turned to be very useful to add robustness to local methods and which, certainly, must be incorporated to production codes.

(g) Strategies for minimization of $f(x)(\equiv \frac{1}{2}\|F(x)\|_2^2)$ use $\nabla f(x)(\equiv J(x)^T F(x))$ and, thus, the Jacobian matrix of F or some difference approximation. An exception is the global strategy for Broyden's method introduced by Griewank [22], but his strategy requires additional hypotheses on F. In fact, quasi-Newton directions of the type produced by Algorithms 2.2–2.9 are not necessarily descent directions for f. Some authors (for instance, Toint in his implementation of partitioned quasi-Newton methods [48]) choose x_{k+1} as some point in the line defined by x_k and the direct prediction point such that $f(x_{k+1})$ is "sufficiently smaller" than $f(x_k)$ (see [15, p. 179]). However, it may be impossible to find that point if the line is orthogonal to $\nabla f(x_k)$, so the effectiveness of this strategy seems to be confined to practical situations where "quasi orthogonality" is unlikely to occur.

In spite of the considerations above, we decided, for completeness of our study, to implement global modifications of Algorithms 2.2–2.9.

The considerations on the nonmonotone behavior of quasi-Newton methods inspired in us a "tolerant global strategy" which essentially consists in imposing decrease of $\|F\|$ only (say) every q iterations where q is a user-supplied parameter. Clearly, this "tolerant strategy" is in the spirit of the watchdog technique for avoiding the Maratos effect in nonlinear programming (see [6]) and it is also related to the nonmonotone line search technique of Grippo, Lampariello, and Lucidi [26].

Given $x_0, x_1, \cdots, x_k$ (and $f(x)$ defined by (5.1)), we define

$$a_k = \text{Argmin}\,\{f(x_0), f(x_1), \cdots, f(x_k)\} \qquad (5.2)$$

for all $k = 0, 1, 2, \cdots$.

Assume that $q \geqq 0$ is a given integer, and $\theta \in (0, 1)$. At some stages, the global tolerant strategy will use "special iterations," which are defined by the following algorithm.

ALGORITHM 5.1. SPECIAL ITERATION. Given x_k, compute x_{k+1} performing the following steps:

Step 1. Compute the LU decomposition of $J(x_k)$ (Steps 2 and 3 of Algorithm 2.1). If, for all $i = 1, \cdots, n|u_{ii}| > TOLB$ (so that no replacement occurred at Step 3 of Algorithm 2.1), perform also Step 4 of Algorithm 2.1, defining the Newton step s_k. Else, define $s_k = -J(x_k)^T F(x_k)(\equiv \nabla f(x_k))$.
Step 2. If $\|s_k\|_\infty > \Delta$, replace $s_k \leftarrow s_k \Delta / \|s_k\|_\infty$.
Step 3. Using a safeguarded quadratic-cubic interpolation strategy (see [15, p. 126–129]) compute $\lambda_k \in (0, 1]$ such that

$$f(x_k + \lambda_k s_k) \leqq f(x_k) + 10^{-4} \lambda_k \nabla f(x_k)^T s_k.$$

Step 4. Define $x^{k+1} = x^k + \lambda_k s_k$.

Special iterations are combined with "ordinary iterations," which are the ones defined by the "local" algorithms described in § 2, according to the following "master subroutine."

ALGORITHM 5.2.
Initialization: $k \leftarrow 0$, $FLAG \leftarrow 1$.
Given x_k, execute Steps 1–4.

Step 1. If $FLAG = 1$, obtain x_{k+1} using an ordinary iteration. Else, compute x_{k+1} using a special iteration.
Step 2. If $k \not\equiv 0 (\text{mod } q)$ and $FLAG = 1$, increment $k \leftarrow k+1$. Return to Step 1.
Step 3. If $f(x_{k+1}) \leqq \theta f(a_k)$ (see definition (5.2)), set $FLAG \leftarrow 1$, $k \leftarrow k+1$ and return to Step 1.
Step 4. Redefine $x_{k+1} \leftarrow a_k$. Set $FLAG \leftarrow -1$, $k \leftarrow k+1$ and return to Step 1.

As we mentioned above, the tolerant strategy consists in iterating using a standard quasi-Newton algorithm testing every q iterations if the norm of the best point decreased enough (Algorithm 5.2, Step 3). If the sufficient decrease condition does not hold, we turn to a minimization algorithm based on Newton (or gradient) directions, which are guaranteed to be descent directions. We continue with these special iterations as long as the sufficient decrease condition of Step 3 is not satisfied. Clearly, if this condition is satisfied an infinite number of times, we will have that $\liminf_{k \to \infty} \|F(x_k)\| = 0$. Conversely, if after some finite k_0, all iterates are special, we will have, under mild conditions, that $\lim_{k \to \infty} \|\nabla f(x_k)\| = 0$ (see [15]).

There may be many other possible global strategies. Here we want to show how the algorithms presented in § 2 adapt to this global modification, and we leave the comparison between different global strategies to forthcoming works.

For comparison of Algorithms 2.2–2.9 with global strategy, we selected some problems of the set presented in § 4, with nonclassical initial points from which Newton's method had difficulties converging, or did not converge at all. We used $\theta = 0.9$. The problems were the following.

Problem 1. Function 1 with $x_0 = (10^{-3}, \cdots, 10^{-3})^T$ and $\Delta = 5000$.
Problem 2. Function 2 with $x_0 = (0, \cdots, 0)^T$ and $\Delta = 10$.
Problem 3. Function 3 with $x_0 = (-1, \cdots, -1)^T$ and $\Delta = 10$.
Problem 4. Function 4, $L = 15$, $x_0 = (-3, \cdots, -3)^T$, $\Delta = 100$.
Problem 5. Function 5, $x_0 = (-0.0008, \cdots, -0.0008)^T$, $\Delta = 10^4$, $n = 100$, $b = 100$.

Problem 6. Function 6, $x_0=(0.8,\cdots,0.8)^T$, $\Delta=10$, $n=100$.

Problem 7. Function 7, $x_0=(5,\cdots,5)^T$, $\Delta=20$, $n=100$.

The results of these experiments are reported in Table 4. For each method, we report the performance of the local version without restarts, and the performance of the "global" version, with $q=3$.

6. Conclusions. Considering the results of our "local" experiments presented in § 4, we may draw the following conclusions.

TABLE 4
Numerical results for the global algorithms.

Problem	n	Version	N	MN	S	DM	DS	RS	CS	B
1	1000	Local	17, 4.5*	NC†	NC	NC	NC	NC	NC	NC
	1000	Global	13, 3.8	24, 4.0	28, 10.0	19, 4.1	NC	31, 5.6	NC	28, 7.0
2	1000	Local	83, 81.0	NC	NC	NC	NC	NC	NC	NC
	1000	Global	19, 22.0	23, 7.2	13, 14.0	NC	NC	36, 24.0	36, 24.0	36, 26.0
3	1000	Local	14, 5.1	NC	12, 4.1	12, 2.9	29, 5.8^1‡	20, 3.8^1	29, 5.4^1	12, 2.9^1
	1000	Global	9, 3.7	8, 2.0	10, 3.6	10, 2.6	18, 3.5	18, 3.6	17, 4.3	11, 2.9
4	225	Local	NC	NC	NC	NC	NC	NC	NC	NC
	225	Global	7, 5.0	20, 4.4	16, 11.0	NC	NC	27, 6.3	19, 4.3	18, 4.0
5	100	Local	NC	NC	NC	NC	NC	NC	NC	NC
	100	Global	12, 5.9	21, 3.5	16, 7.9	20, 4.0	24, 4.6	24, 4.6	26, 4.2	29, 4.6
6	100	Local	NC	NC	NC	NC	NC	NC	NC	NC
	100	Global	20, 1.5^1	20, 1.1^1	20, 1.6^1	20, 1.3^1	20, 1.1^1	20, 1.1^1	20, 1.1^1	21, 1.3^1
7	100	Local	NC	NC	NC	NC	NC	NC	NC	NC
	100	Global	72, 3.9^1	90, 4.2^1	42, 2.6^1	21, 0.8^1	*§	51, 1.9	59, 2.1	18, 0.7^1

* The first number (an integer) is the total number of iterations. The second (real) number is CPU time (in seconds).

† NC = nonconvergence.

‡ The superscript 1 indicates "convergence of type 1." Absence of superscript indicates "convergence of type 0."

§ * = overflow.

(a) There are no important differences between the secant Algorithms 2.4–2.9 in terms of robustness or number of iterations. Clearly, Algorithm 2.3 (modified Newton) behaves worse than all the secant methods. Since superlinear convergence (without restarts) is only proved for Broyden's method and Schubert's method, while DS, CS, and RS have only linear local convergence properties and DM has an even weaker convergence theorem (see § 2), we conclude that additional theory is necessary to explain the clear superiority of Algorithms 2.5–2.8 over MN, and the very analogous behavior of these algorithms in relation to Broyden and Schubert.

The relation between the effectiveness of DM, DS, CS, and RS and the number of iterations where the "secant stability test" at Step 5 of Algorithms 2.5–2.9 is satisfied, was studied by us, and the results were very impressive. In fact, we verified that good performances of these algorithms are associated with situations where the test was always satisfied, that is, the secant equation held at all iterations. Conversely, in the cases where Algorithms 2.5–2.9 had a poor behavior, we detected many iterations (from 20–90 percent in some cases) where the test of Step 5 failed.

(b) Schubert's method is not useful for problems where derivatives are not difficult to calculate. This is not surprising since in those cases the computational work involved in one Schubert iteration is essentially the work involved in a Newton iteration. However, since in terms of robustness and number of iterations Schubert's method did not perform better than the other tested secant methods, we may conclude that Schubert's method is not a reliable alternative to other secant methods, even if derivatives were difficult to obtain. There exist secant methods which solve a linear system of equations at each iteration but incorporate information about the problem in a more clever way than Schubert's method does. These are the pointwise family of quasi-Newton methods [27], [29] for boundary value problems, and partitioned quasi-Newton methods [48] for finite element problems. The first family may be defined in the original infinite-dimensional space where the boundary value problem takes place, and thus exhibits mesh independence properties, that is, its behavior does not depend on the discretization. Partitioned quasi-Newton methods seem to exhibit mesh independence in practice (see [23], [24]) but a theoretical justification is due. Both pointwise methods and partitioned quasi-Newton methods are more efficient than Schubert's method for the particular problems to which they are applied, but we do not know if this efficiency compensates the disadvantage of solving a linear system per iteration for comparison with other secant methods.

(c) Secant methods (except Schubert) outperformed Newton's method in "not difficult" problems where the Jacobian matrix has a "bad" structure for the application of the George–Ng algorithm, notably Problems 4 and 5. However, only Newton's method did not present any failure (nonconvergence) in this set of experiments. Of course, none of these facts is surprising since it is well known that the region of convergence of local versions of Newton's method is larger than that of quasi-Newton methods.

(d) Except for Problem 4, the "diagonal methods" DS, RS, and CS were slightly better than Broyden's method, on the average, both in terms of number of iterations and CPU time. Problem 4 (Poisson) deserves more careful analysis. In fact, this was the only problem where DM, DS, RS, and CS were not better than the modified Newton method, that is, in this case, the derivative information provided by the secant equation did not add any improvement to the convergence properties of MN. On the other hand, Broyden was the best method in this case. Griewank observed that since Broyden's method is invariant for regular linear changes in the range of F, the application of this method to $F(x)=0$ (with $B_0=J(x_0)$) is equivalent to its application to $G(x)\equiv J(x_0)^{-1}F(x)=0$ using $B_0=I$. Then, using the LU factorization of $J(x_0)$, one essentially preconditions the system by the inverse of the Laplacian and thus obtains conceptually a Fredholm integral equation of the second kind, for which Q-superlinear convergence of Broyden's method on the underlying infinite-dimensional problem has been proved [25].

(e) Nonconvergence only occurred in some nonrestarted executions of quasi-Newton methods. The main contribution of restarts was to correct with opportunistic Newton iterations a "wrong trajectory" of some quasi-Newton methods. In the cases where this correction was needed, the final performance of the restarted quasi-Newton was very similar to the performance of Newton.

(f) In contrast to modified Newton, Schubert, Dennis–Marwil, and Broyden, the methods DS, RS, and CS behaved surprisingly well in the singular Problem 7. Some research is needed to detect if there exist theoretical properties behind this empirical fact.

Before commenting on the results of the "global" algorithms, some explanation is necessary. We performed many experiments besides the ones reported in Table 4. We ran many problems where Newton's method did not converge, and its global modification or the global modifications of quasi-Newton methods, even with $q=1$, did not converge either, or most frequently converged to undesirable local minimizers of $\|F(x)\|$. In the problems reported in Table 2, the global version of Newton behaved better than the local version. We selected these problems with the aim of testing the behavior of the quasi-Newton algorithms in those situations.

Our main observations concerning the global implementations are:

(a) Unlike the local cases, there are no meaningful differences between the modified Newton method and the secant Algorithms 2.4–2.9. In fact the global MN was better than the global Newton four times, while RS, CS, and Broyden outperformed Newton in three cases (the Schubert, DM, and DS did it only in two cases).

(b) Only in Problem 6 did all the global methods fail to find a solution of the system and converge to the same local minimizer of $\|F(x)\|$. In Problem 7 most methods converged to (different) local minimizers of $\|F(x)\|$ and only RS and CS found a solution of the system.

(c) The tolerant strategy was successful for improving the performance of quasi-Newton methods in the following sense: in many cases where the performance of the local quasi-Newton was very poor, the global quasi-Newton with tolerant strategy succeeded in finding the solution and it was more efficient than "global Newton" in about 50 percent of the experiments. Generally, only a few special iterations are necessary for putting the quasi-Newton method on a "right trajectory."

The implementation and testing of a global strategy led us to discover many "very difficult problems," where even Newton's method with a global strategy which imposes sufficient decrease at each iteration failed to find a solution of the system. The present challenge is to implement new global strategies which will "jump" over local minimizers to solutions of $F(x)=0$ (which are, of course, global minimizers of the norm).

A production code, which, instead of the risky divergence criteria adopted in this comparative study, uses the steplength controls and stopping criteria given by Moré and Cosnard [39], is being elaborated.

Acknowledgments. The authors acknowledge A. Griewank and two referees for very helpful suggestions.

REFERENCES

[1] R. P. Brent, *Some efficient algorithms for solving systems of nonlinear equations*, SIAM J. Numer. Anal., 10 (1973), pp. 327-344.

[2] C. G. Broyden, *A class of methods for solving nonlinear simultaneous equations*, Math. Comp., 19 (1965), pp. 577-593.

[3] ———, *The convergence of an algorithm for solving sparse nonlinear systems*, Math. Comp., 25 (1971), pp. 285-294.

[4] C. G. Broyden, J. E. Dennis, Jr., and J. J. Moré, *On the local and superlinear convergence of quasi-Newton methods*, J. Inst. Math. Appl., 12 (1973), pp. 223-245.

[5] F. F. Chadee, *Sparse quasi-Newton methods and the continuation problem*, TR SOL no. 85-8, Dept. of Operations Research, Stanford University, Stanford, CA, 1985.

[6] R. W. Chamberlain, C. Lemaréchal, H. Pedersen, and M. J. D. Powell, *The watchdog technique for forcing convergence in algorithms for constrained optimization*, Math. Programming Stud., 16 (1982), pp. 1-17.

[7] T. F. Coleman, B. S. Garbow, and J. J. Moré, *Software for the estimation of sparse Jacobian matrices*, ACM Trans. Math. Software, 10 (1984), pp. 329-345.

[8] M. Y. Cosnard, *A comparison of four methods for solving systems of nonlinear equations*, TR 75-248, Dept. of Computer Science, Cornell University, Ithaca, NY, 1975.

[9] D. F. Davidenko, *On the approximate solution of nonlinear equations*, Ukrainian Mat. Z., 5 (1953), pp. 196-206.

[10] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, *Inexact Newton methods*, SIAM J. Numer. Anal., 19 (1982), 400-408.

[11] J. E. Dennis, Jr., N. Echebest, M. T. Guardarucci, J. M. Martínez, H. D. Scolnik, and C. Vaccino, *A curvilinear search using tridiagonal secant updates for unconstrained optimization*, Report no. 9/90, IMECC-UNICAMP, Campinas, SP, Brazil, 1990.

[12] J. E. Dennis, Jr. and E. S. Marwil, *Direct secant updates of matrix factorizations*, Math. Comp., 38 (1982), pp. 459-476.

[13] J. E. DENNIS, JR. AND J. J. MORÉ, *A characterization of superlinear convergence and its application to quasi-Newton methods*, Math. Comp., 28 (1974), pp. 549-560.

[14] ———, *Quasi-Newton methods, motivation and theory*, SIAM Rev., 19 (1977), pp. 46-89.

[15] J. E. DENNIS, JR. AND R. B. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[16] I. S. DUFF, MA28—*a set of Fortran subroutines for sparse unsymmetric linear equations*, AERE R8730, HMSO, London, 1977.

[17] I. S. DUFF, A. M. ERISMAN, AND J. K. REID, *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, 1986.

[18] G. FORSYTHE AND C. B. MOLER, *Computer Solution of Linear Algebraic Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1967.

[19] D. M. GAY, *Some convergence properties of Broyden's method*, SIAM J. Numer. Anal., 16 (1979), pp. 623-630.

[20] A. GEORGE AND E. NG, *Symbolic factorization for sparse Gaussian elimination with partial pivoting*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 877-898.

[21] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[22] A. GRIEWANK, *The "global" convergence of Broyden-like methods with a suitable line search*, J. Austral. Math. Soc. Ser. B, 28 (1986), pp. 75-92.

[23] ———, *The solution of boundary value problems by Broyden based secant methods*, CTAC-85, in Proc. of the Computational Techniques and Applications Conference, University of Melbourne, J. Noye and R. May, eds., North-Holland, Amsterdam, August 1985.

[24] ———, *On the iterative solution of differential and integral equations using secant updating techniques*, in The State of Art in Numerical Analysis, A. Iserles and M. J. D. Powell, eds., Clarendon Press, Oxford, 1987, pp. 299-324.

[25] ———, *The local convergence of Broyden-like methods on Lipschitzian problems in Hilbert space*, SIAM J. Numer. Anal., 24 (1987), pp. 684-705.

[26] L. GRIPPO, F. LAMPARIELLO, AND S. LUCIDI, *A nonmonotone line search technique for Newton's method*, SIAM J. Numer. Anal., 23 (1986), pp. 707-716.

[27] W. E. HART AND S. O. W. SOUL, *Quasi-Newton methods for discretized nonlinear boundary problems*, J. Inst. Math. Appl., 11 (1973), pp. 351-359.

[28] G. W. JOHNSON AND N. H. AUSTRIA, *A quasi-Newton method employing direct secant updates of matrix factorizations*, SIAM J. Numer. Anal., 20 (1983), pp. 315-325.

[29] C. T. KELLEY AND E. W. SACHS, *A quasi-Newton method for elliptic boundary value problems*, SIAM J. Numer. Anal., 24 (1987), pp. 516-531.

[30] V. R. LOPES, *Soluções por elementos finitos de equações de difusão lineares via princípios externos duais*, Ph.D. thesis, Dept. of Mathematics, USP, S. Carlos, Brazil, 1988.

[31] ———, Private communication, 1988.

[32] J. M. MARTÍNEZ, *Generalization of the methods of Brent and Brown for solving nonlinear simultaneous equations*, SIAM J. Numer. Anal., 16 (1979), pp. 434-448.

[33] ———, *Solving nonlinear simultaneous equations with a generalization of Brent's method*, BIT, 20 (1980), pp. 501-510.

[34] ———, *A quasi-Newton method with a new updating for the LDU factorization of the approximate Jacobian*, Mat. Apl. Comput., 2 (1983), pp. 131-142.

[35] ———, *Quasi-Newton methods with factorization scaling for solving sparse nonlinear systems of equations*, Computing, 38 (1987), pp. 133-141.

[36] ———, *A family of quasi-Newton methods for nonlinear equations with direct secant updates of matrix factorizations*, SIAM J. Numer. Anal., 27 (1990), pp. 1034-1049.

[37] E. S. MARWIL, *Convergence results for Schubert's method for solving sparse nonlinear equations*, SIAM J. Numer. Anal., 16 (1979), pp. 588-604.

[38] H. MATTHIES AND G. STRANG, *The solution of nonlinear finite element equations*, Internat. J. Numer. Methods Engrg., 14 (1979), pp. 1613-1626.

[39] J. J. MORÉ AND M. Y. COSNARD, *On the numerical solution of nonlinear equations*, ACM Trans. Math. Software, 5 (1979), pp. 64-85.

[40] J. J. MORÉ AND J. A. TRANGENSTEIN, *On the global convergence of Broyden's method*, Math. Comp., 30 (1976), pp. 523-540.

[41] J. M. ORTEGA AND W. C. RHEINBOLDT, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.

[42] M. J. D. POWELL, *A hybrid method for nonlinear equations*, in Numerical Methods for Nonlinear Algebraic Equations, P. Rabinowitz, ed., Gordon and Breach, London, 1970, pp. 87-114.

[43] W. C. RHEINBOLDT, *Numerical Analysis of Parametrized Nonlinear Equations*, Wiley-Interscience, New York, 1986.

[44] L. K. SCHUBERT, *Modification of a quasi-Newton method for nonlinear equations with a sparse Jacobian*, Math. Comp., 24 (1970), pp. 27-30.

[45] H. SCHWANDT, *An interval arithmetic approach for the construction of an almost globally convergent method for the solution of the nonlinear Poisson equation on the unit square*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 427-452.

[46] H. SCHWETLICK, *Numerische Lösung nichtlinearer Gleichungen*, Deutscher Verlag der Wissenschaften, Berlin, 1978.

[47] V. E. SHAMANSKII, *A modification of Newton's method*, Ukrainian Mat. Z., 19 (1967), pp. 133-138.

[48] Ph. L. TOINT, *Numerical solution of large sets of algebraic nonlinear equations*, Math. Comp., 16 (1986), pp. 175-189.

[49] L. T. WATSON, *A globally convergent algorithm for computing fixed points of C^2 maps*, Appl. Math. Comp., 5 (1979), pp. 297-311.

[50] ———, *Engineering applications of the Chow-Yorke algorithm*, in Homotopy Methods and Global Convergence, B. C. Eaves and M. J. Todd, eds., Plenum Press, New York, 1983, pp. 287-307.

[51] L. T. WATSON, S. C. BILLUPS, AND A. P. MORGAN, *Algorithm* 652: HOMPACK: *A suite of codes for globally convergent homotopy algorithms*, ACM Trans. Math. Software, 13 (1987), pp. 281-310.

[52] J. V. ZAGO, *Approximate solution of generalized Hamiltonian equations with applications*, Ph.D. thesis, Dept. of Mathematics, University of Wisconsin at Madison, Madison, WI, 1976.

[53] M. C. ZAMBALDI, *Estruturas estáticas e dinâmicas para resolver sistemas não lineares esparsos*, Tese de Mestrado, Departamento de Matemática Aplicada, UNICAMP, Campinas, Brazil, 1990.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 2, pp. 484-498, March 1992 003

KNOT SELECTION FOR LEAST SQUARES THIN PLATE SPLINES*

JOHN R. McMAHON† AND RICHARD FRANKE‡

Abstract. An algorithm for the selection of knot point locations for approximation of functions from large sets of scattered data by least squares thin plate splines is given. The algorithm is based on the idea that each data point is equally important in defining the surface, which allows the knot selection process to be decoupled from the least squares. Properties of the algorithm are investigated, and examples demonstrating it are given. Results of some least squares approximations are given and compared with other approximation methods.

Key words. knot selection, least squares, thin plate splines, Dirichlet tessellation, scattered data

AMS(MOS) subject classification. 65D10

1. Introduction.

The problem of fitting a surface to small sets of given data has been addressed in many different ways, and several computer programs are currently available which enable one to deal with the problem effectively. Many of the methods available involve a global interpolation or approximation scheme and often involve solving a system of equations with an equivalent number of unknowns. For very large sets of data, the problem is computationally intractable. This consideration provides the motivation behind the development of a way to pare the problem down to a more manageable size.

We wish to construct a function F which approximately fits the data, since we assume the data collection is subject to measurement error. We propose to use approximation by least squares thin plate splines (TPS), where the surface function is constructed so as to minimize an error function subject to certain constraints. Solving the approximation problem will also involve as many equations as there are data points, but the number of unknowns will be significantly fewer. Part of the appeal of TPS approximation lies in the fact that it minimizes a certain linear functional and involves a linear combination of functions with no greater complexity than the natural logarithm of the distance function.

Interpolation of scattered data by the method of TPS was developed from engineering considerations by Harder and Desmarais [6]. It can be thought of as a two-dimensional generalization of the cubic spline, which models a thin beam under point loads subject to equilibrium constraints. The TPS function is derived from a differential equation which gives the deformation of an infinite, thin plate under the influence of point loads. A point load is applied at each data point so that the interpolating surface can be constructed as a sum of fundamental solutions of the TPS equation.

In using the least squares TPS approximation method to fit the surface, a fewer number of basis functions than the number of given data points is employed. These basis functions are centered at a different, smaller set of points, which, in analogy with the univariate case, we call the knots. Therefore, the problem at hand is one of selecting the knot points, and hence the basis functions. This approach differs from the use of smoothing splines, which were introduced by Wahba and Wendelberger [13] in the multidimensional case, and called Laplacian smoothing splines (LSS). LSS minimize

* Received by the editors October 26, 1987; accepted for publication (in revised form) October 29, 1990.

† Department of Mathematics, United States Military Academy, West Point, New York 10966.

‡ Department of Mathematics, Naval Postgraduate School, Monterey, California 93943. The work of this author was supported in part by the Office of Naval Research under Program Element 61153N, Project No. BR033-02-WH.

a certain functional which is a linear combination of a term measuring fidelity to the data and one measuring smoothness of the function (a generalization of the usual TPS functional). In this case there is still one basis function for each data point, but the interpolation condition is relaxed.

Given a "large" set of data points, (x_i, y_i, f_i), $i = 1, \cdots, N$, we wish to find a smaller set of knot points, $(\hat{x}_j, \hat{y}_j)$, $j = 1, \cdots, K$, which will "represent" the former reasonably well. This could be accomplished by choosing a subset of the original set or by some process which produces a representative set. The ultimate goal is to approximate the surface from which the original data arose using the representative set. Hence, a surface fit to the large set and one fit to the representative set should be essentially the same.

Approximation by least squares TPS is straightforward once the knot points are known. We construct the TPS function

$$F(x, y) = \sum_{j=1}^{K} A_j d_j^2 \log(d_j) + ax + by + c,$$

where $d_j^2 = (x - \hat{x}_j)^2 + (y - \hat{y}_j)^2$, and the coefficients A_j, a, b, and c are chosen to minimize the error function

$$E = \sum_{i=1}^{N} \{[F(x_i, y_i) - f_i]/s_i\}^2.$$

The ordinates, f_i, may be subject to random errors, say, with standard deviation s_i at the ith data point. We model the plate under the point loads at the knot points (as opposed to the data points); therefore, the constraint equations for the least squares TPS method, which may be thought of as "equilibrium conditions" on the plate, should be satisfied. Thus the error function is minimized subject to the following constraint equations:

$$\sum_{j=1}^{K} A_j = 0, \qquad \sum_{j=1}^{K} A_j \hat{x}_j = 0, \qquad \sum_{j=1}^{K} A_j \hat{y}_j = 0.$$

We use LINPACK [1] subroutines to do the actual calculations.

Previous attempts have been made to minimize the error function by considering it to be a function of the knot point locations as well as the coefficients, wherein a total of $3K$ parameters are involved. As reported on by Schmidt [11], the initial knot configuration was taken to be of tensor product form. The overall minimization process is a large nonlinear one, and is complicated by possible coalescence of knots as well as nonunique solutions (as indicated by consideration of one-dimensional cases). Also, the objective function may have many local minima so that avoiding poor local minima or searching for better local minima may be necessary. Because of these kinds of problems, our goal is to decouple the knot selection process from the least squares process.

When data are somewhat uniformly distributed, methods involving tensor product cubic splines may be desirable. Tensor product methods place knot locations on a grid, which may not reflect the actual disposition of the data points; in fact, there could be no data nearby. Even though these problems are surmountable, they could lead to nonuniqueness of the solution. The minimum norm solution that is often used may not be aesthetically appealing. We will say more about this later.

A different point of view is considered here wherein the knot point locations are predetermined based on two criteria. These criteria evolve from considerations relating the density of data to the dependent variable and mandating the importance of each individual data point. The solution of the overdetermined system of equations follows the knot point selection. A summary of the approach and results will be presented. Examples are given which illustrate rather well the ability of the scheme to select knot locations which reflect the underlying density of the data. We also report on actual surface fitting and comparison with two other methods, the LSS of Wahba and Wendelberger [13] and the tensor product bicubic Hermite method due to Foley [4]. We point out, however, that each of these approaches is geared toward a different problem than ours and thus it is difficult to compare them in a meaningful way.

We also point out three related ideas which are attempts to decrease the number of basis functions to be considered. Each is an attempt to choose a subset of points to be used to construct the approximation. Schiro and Williams [10] used an adaptive process for subset selection and overlapping regions to construct underwater topographic maps. Bozzini, diTisi, and Lenarduzzi [2] gave an algorithm for selecting a subset of points which were important to proper definition of the surface. Both of these methods made no assumptions about the density of the data points relative to the behavior of the surface, and both required consideration of the ordinate values. Another approach is an adaptive procedure suggested by LeMéhauté and Lafranche [7]. The underlying approximation is a piecewise polynomial over a triangulation; it is assumed that all derivative data necessary to construct the smooth approximation is available. Points are removed from the set as long as the approximation from the remaining set of points is accurate to within a specified tolerance.

2. The knot selection process. Given a priori flexibility in knot placement, the problem becomes the selection of knot locations, followed by solution of the system by least squares. Since the selection of knot locations is to be decoupled from the solution of the least squares problem, we establish a connection between these aspects of the problem by making two somewhat vague assumptions. First, we assume that the independent variables of the data indicate something about the behavior of the dependent variable. For example, the density of the data points may be dependent on the curvature of the surface. Hence, where relatively many data points are found, the function is assumed to be changing behavior rapidly, whereas a low density of data indicates slowly changing behavior. Although this assumption is not universally satisfied in practice, it does not seem to be an unreasonable one.

The second assumption is that each data point is equally important in defining the underlying surface. Therefore the number of data points represented by each knot should be the same or nearly the same. This leads to "equal representation" of the data points by the knot points where each data point is "close" to a knot point. A key advantage achieved in pursuing this approach is the existence of a natural heuristic for moving the knots around the plane in searching for a good knot configuration. This point will be elaborated later in the paper.

2.1. A local minimum property. We want each data point to be close to a knot point, so we try to minimize the sum of the distances squared from each data point to the nearest knot point, that is, minimize the "global" value,

$$GN^2 = \sum_{i=1}^{N} \min_j \left[(x_i - \hat{x}_j)^2 + (y_i - \hat{y}_j)^2\right].$$

This function is continuous and piecewise quadratic in $2K$ variables. The expression leads naturally to a "default" Dirichlet tessellation, a partitioning of the plane with respect to the knot points (see Fig. 1). Thus we say each data point belongs to some knot point according to the Dirichlet tile in which it lies. Data points on any of the

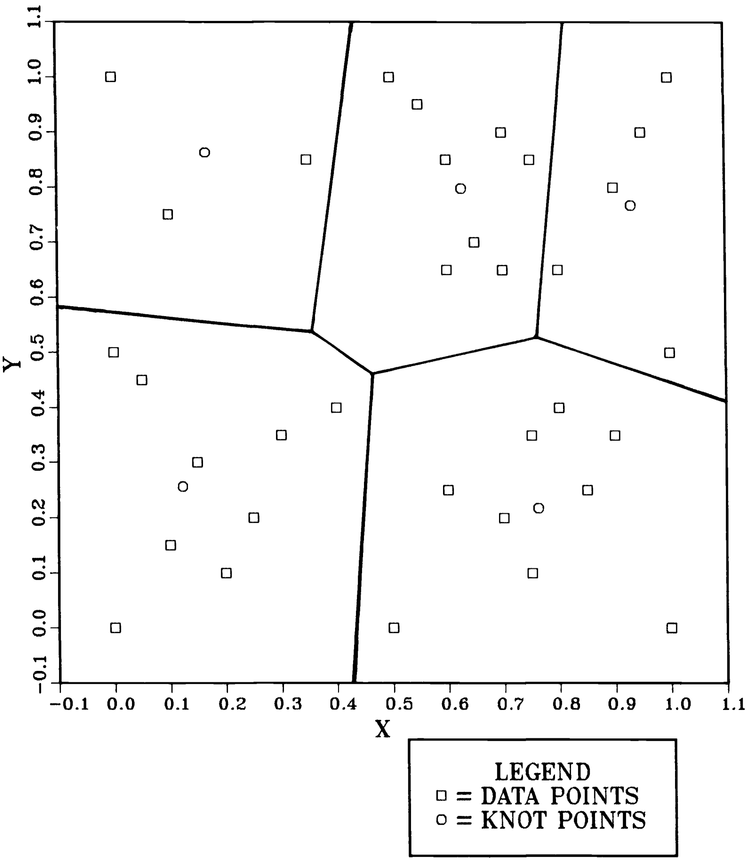


FIG. 1. *A Dirichlet tessellation with five tiles. It is constructed by connecting the perpendicular bisectors of the lines joining each of the knot points.*

tile boundaries (ties) must be resolved by a determination of to which tile they belong or some sharing mechanism. Our initial guess at the knot point configuration was taken to be quasi-gridded.

Differentiation of GN^2 with respect to the $\hat{x}_j$ and $\hat{y}_j$ show that at the minimum, each knot point will occupy the centroid with respect to the data points inside that tile. Given the initial configuration of knot points with its Dirichlet tessellation, the following algorithm for iteration to a local minimum GN^2 value is employed.

ALGORITHM 2.1. (a) Compute the centroid of each tile with respect to the data points contained within each tile.

(b) Move the knots to the corresponding centroids, which results in a new Dirichlet tessellation and a new set of knot point–data point associations; this is the configuration for the next iteration.

(c) Quit when two successive iterations yield the same knot locations, which means that a local minimum value of GN^2 has been found.

This algorithm was formulated in discussions at the Istituto per le Applicazioni della Matematica e dell'Informatica in 1983 [9], after the problem was posed by Nielson and Franke. The algorithm is known in clustering analysis as the method of k-means [12].

We note that the value of GN^2 will necessarily decrease as the iterations continue until two successive iterations yield the same configuration; this will be proven below. In the case where no data points lie in a tile for some knot point, the knot point is moved to the nearest data point. This mechanism avoids knots without data points. Furthermore, if a data point lies on a tile boundary, it is assigned to the knot with the smallest subscript (amongst the appropriate choices of knot points). Employment of a different criterion for the resolution of ties may yield different results. We note that knots cannot coalesce.

The following theorem is pertinent to this algorithm.

THEOREM 2.1. *The function GN^2 decreases with each iteration which involves movement of a knot point.*

Proof. Write GN^2 in the more convenient form

$$GN^2 = \sum_{j=1}^{K} \sum_{i \in I_j} [(x_i - \hat{x}_j)^2 + (y_i - \hat{y}_j)^2], \tag{1}$$

where $I_j = \{i\colon (x_i, y_i)$ in the tile for $(\hat{x}_j, \hat{y}_j)\}$. In (1), the interior sum is the sum of the distances squared from the data points in a tile to the knot point in that tile, and the exterior sum is over all K of the tiles. Let a prime denote the new knot points and index sets. This form leads to the expressions

$$(\hat{x}_j', \hat{y}_j') = \left(\sum_{i \in I_j} x_i / n_j, \sum_{i \in I_j} y_i / n_j \right),$$

where n_j is the number of indices in the set I_j. The new knot points will lead to a new tessellation, followed by the new index sets I_j'. Then the expression (1) is greater than or equal to

$$\sum_{j=1}^{K} \sum_{i \in I_j} [(x_i - \hat{x}_j')^2 + (y_i - \hat{y}_j')^2] \tag{2}$$

because the new knot point locations minimize the contribution of the interior sums. This expression (2), in turn, is greater than or equal to

$$\sum_{j=1}^{K} \sum_{i \in I_j'} [(x_i - \hat{x}_j')^2 + (y_i - \hat{y}_j')^2] \tag{3}$$

since an index i moves to another set only in the case wherein the corresponding data point is now closer to a different knot point, thus decreasing its contribution to the global GN^2 value. □

Finding a local minimum of GN^2 is well served by Algorithm 2.1; however, as demonstrated in a one-dimensional example [8], the function GN^2 is rife with local minima, and the local minimum value found depends on the initial configuration of knots used. We can draw similar conclusions for the multidimensional case based on the one-dimensional analogy.

The process of locating each knot occurs in two distinct steps: first, each data point is assigned to the closest knot and second, a determination is made within each tile as to the location of the centroid of its data points. The minimum distance in the expression for the GN^2 function causes it to be piecewise and it is quadratic. The minimization of GN^2 corresponds to locating each of the knots at the centroids of their respective Dirichlet tiles. As a direct result of the centroid requirement, the GN^2 function will stabilize at a local minimum value of the particular quadratic piece defined by stable knot locations.

The local minimum value of GN^2 will frequently occur out of the domain of one or more of the corresponding quadratic pieces. This leads to a "cascading" phenomenon which continues until a local minimum value occurs within the domain defined by the current set of knots. However, the global minimum value will not necessarily be attained.

2.2. The objective function. This inconsistent performance of Algorithm 2.1 in finding the global minimum value of GN^2 leads to consideration of a somewhat different criterion for locating a good configuration of knot points. We wish to exploit the second assumption specified at the end of § 2.1, while taking advantage of the minimization of the GN^2 function. Since each data point is assumed to be equally important, the Dirichlet tile for each knot should contain about the same number of data points. Thus we wish to minimize the sum of the squares of the differences between the number of knots in each tile and the average number of data points that should belong to each tile, that is, minimize the quantity

$$D=\sum_{j=1}^{K}(n_j-N/K)^2.$$

The new algorithm for determining knot locations is based on the minimization of D, subject to the constraint that each knot be located at the centroid of its tile.

This optimization leads to a natural heuristic for moving knots from a stable configuration to a possibly better configuration. We call the current configuration of knots a "base" configuration, and iterate through the algorithm as follows.

ALGORITHM 2.2. (a) Generate a new guess for the knot locations by moving the knot(s) with the smallest number of data points in their tile(s) toward the knot(s) with the largest number of knot(s) in their tile(s); the distance moved is initially a large fraction of the total distance between the knots.

(b) Iterate to a stable configuration using the first algorithm, compute the values of GN^2 and D, and compare D to the smallest value achieved to date, as represented by that of the base configuration.

(c) Repeat the process above when a smaller value of D is obtained, with the present configuration as the base configuration.

(d) When a smaller value of D is not found, take a shorter step in the movement of the knot(s) and repeat the process above.

(e) Continue with smaller and smaller steps until a smaller value of D is found (or an equal value of D with a smaller GN^2 value) or until the knot locations return to the base configuration.

(f) Perform the search in the symmetrical way when the base configuration is returned to, that is, move the knot(s) with the largest number of data points in their tiles toward the knot(s) with the smallest number of points in their tile.

(g) Quit when no smaller value of D is found.

The movement of the knots is justified by the rationale that a more equitable distribution of data points can be found by moving the tile boundaries across data points. Note that the algorithm for computing a local minimum of the GN^2 function value is embedded in Algorithm 2.2.

3. Results and examples. Using Algorithm 2.1 for the a priori selection of the knot point locations, experiments were conducted to test the scheme using different sets of test data. This was followed by verification of the scheme on two sets of real data. Results from two sets of the test data are presented here, one consisting of 200 data points called "cliff," and one consisting of 500 data points called "humps and dips."

Both sets of data were generated using known functions (see Franke [5]) in a way that forced the density of points to be approximately proportional to the curvature of the sampled function.

Figures 2 and 3 show these two test data sets graphically and illustrate the optimized knot point configurations found using Algorithm 2.1.

We also investigated how closely the constructed surface F and the "true" surface resemble one another for the cases where the data was generated from a known function. This comparison is made in the context of the root-mean-squared error (RMS) of the residuals (at the data points) and on a rectangular grid of locations in the plane. The two data sets constructed above, and one other which was generated in a similar manner, were used to compare RMS errors for the least squares Algorithm 2.1 developed here with the LSS method of Wahba and Wendelberger and the bicubic tensor product Hermite method of Foley.

The comparisons made here should be viewed in the proper perspective since the use of TPS (or other radial basis function methods) is not efficient in the construction of surfaces when compared with tensor product methods such as Foley's. This is true even apart from cost of the knot selection process, which may be significant. Thus it makes little sense to compare them on the basis of cost alone. The situation in which we envision the use of our method to be viable is one in which we expect difficulties (amounting to failure) with tensor product methods, such as those that may occur with greatly varying density of data. Such a case is mentioned in [3], and a solution within the context of least squares tensor product splines proposed. It is probable that there are situations in which the proposed solution would also fail, although we have not attempted to find any. In the case of such failures, one may be willing to make significant expenditures of computational effort to obtain a viable approximation. We believe our scheme is capable of being successful in many cases, and should be one of those available to users. No doubt the knot selection process should often be implemented as a preprocessing phase.

The dependent variable values of the experimental data sets were generated in two ways: (1) using a known function, and (2) contaminating the known function by the injection of independent, identically distributed normal random errors with a composite standard deviation of 0.05. In the first case, we would expect the RMS error on the data points and on the grid to be about the same, and to decrease as the number of knot points used to represent the data is increased.

In the contaminated case, the dependent variable at each data point is the sum of the unknown underlying function value and the error function value so that the difference between the constructed surface and the "true" surface is mainly attributable to the presence of error in the data. In the best situation, we expect the RMS error in the residuals to match the composite standard deviation of the random error injected to obtain the contaminated data. At the grid points, we expect the RMS error to be smaller than the composite standard deviation, since the grid sample is larger (33×33) and the errors are distributed more evenly throughout the entire region of interest.

Some observations can be made regarding Tables 1–3. The general trend of the RMS error on both the data points and the grid is to diminish as the number of knot points is increased. As expected with the exact data, the RMS error of the residuals and the RMS error on the grid are roughly equivalent. For the contaminated data, the RMS error of the residuals roughly matches the composite standard deviation of the data, and the RMS error on the grid is smaller than the RMS error of the residuals, as expected. In Table 1, the errors over the grid increase as the number of knot points is increased, and that of the residuals' RMS is less than the RMS of the injected errors.

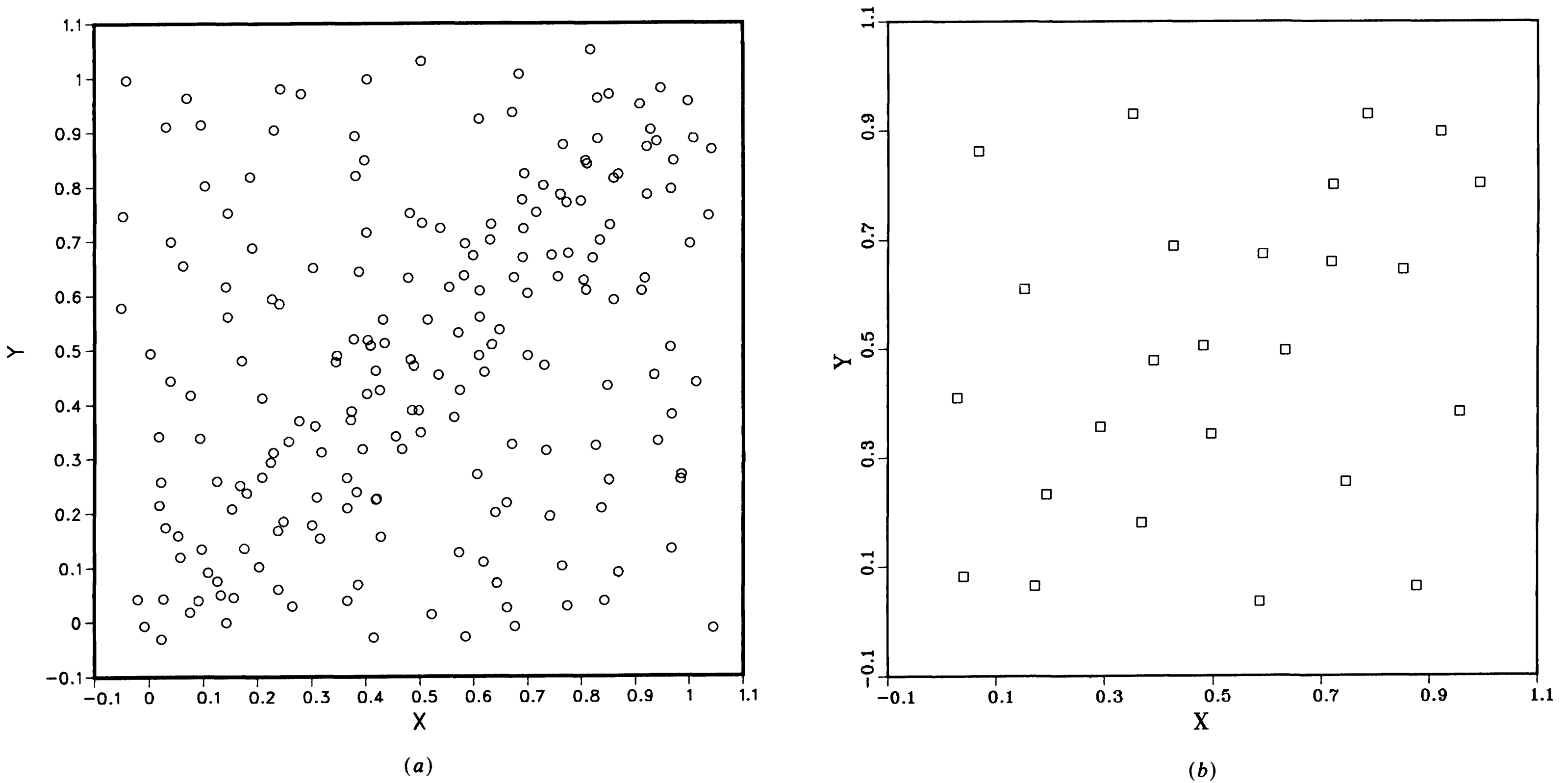


FIG. 2. *The "cliff" data set.* (a) *Note the relatively dense disposition of data points across the diagonal where the underlying surface drops off.* (b) *The 25 knot points used clearly reflect the behavior of the data set, as expected.*

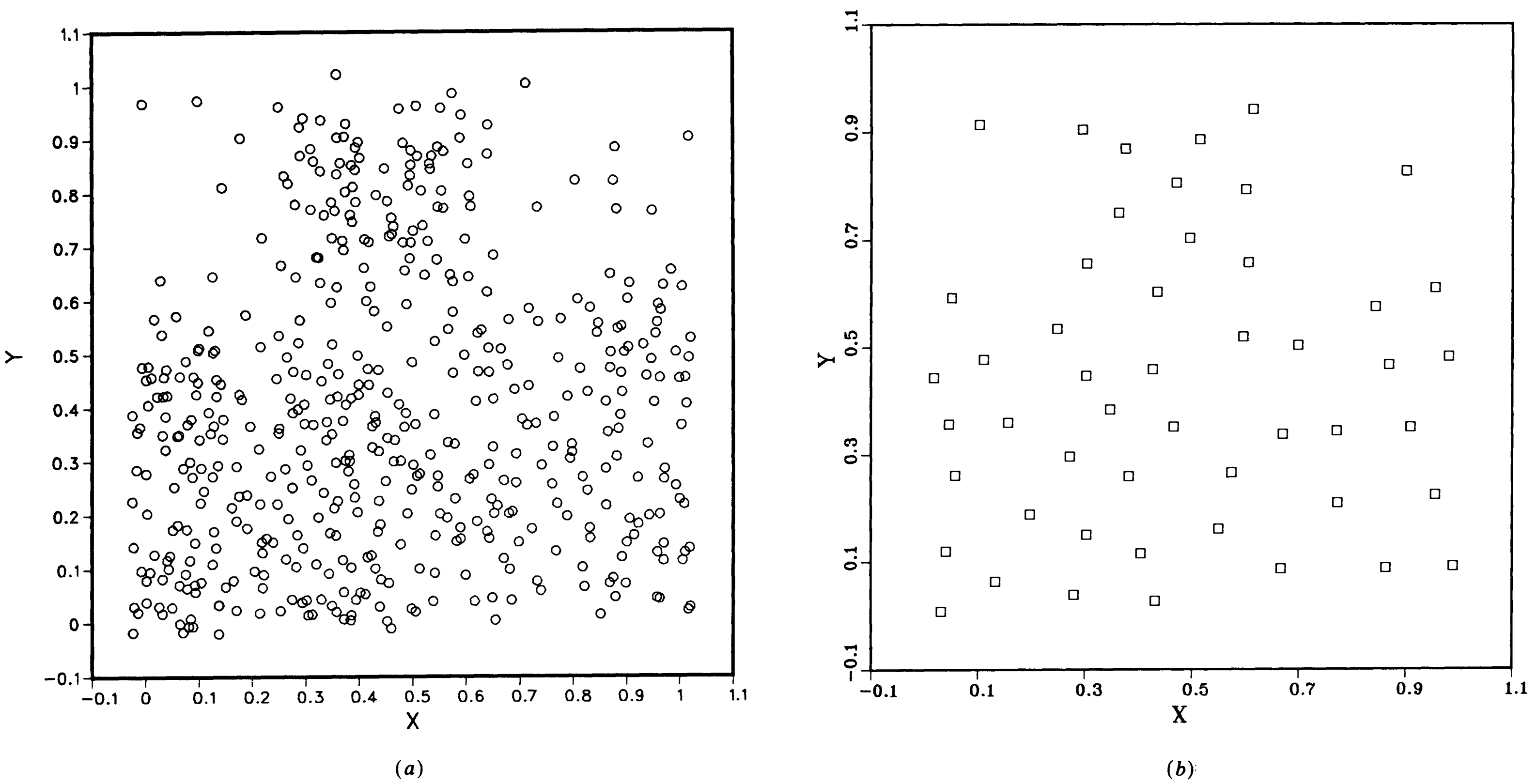


FIG. 3. *The "humps and dips" data set.* (a) *Note how clumps of data appear in three portions of the plane, indicating that the underlying surface is undergoing change.* (b) *A set of* 50 *knot points was used to represent the data.*

TABLE 1
Comparison of RMS *errors on "cliff,"* 200 *points.*

Method	Number of Data Points/ Knot Points	No Errors in Data		Contaminated Data	
		Residual	Grid	Residual	Grid
Least Squares TPS	200/20	0.01562	0.01474	0.05214	0.01795
Least Squares TPS	200/25	0.01179	0.01154	0.04805	0.02040
Foley	200/5×5	0.00777	0.00613	0.05996	0.04819
Least Squares TPS	200/35	0.00626	0.00616	0.04590	0.02146
Foley	200/6×6	0.00512	0.00417	0.05113	0.03745
Smoothing	200	0.0	0.00096	0.04272	0.01806

TABLE 2
Comparison of RMS *errors on "humps and dips,"* 200 *points.*

Method	Number of Data Points/ Knot Points	No Errors in Data		Contaminated Data	
		Residual	Grid	Residual	Grid
Least Squares TPS	200/20	0.05525	0.05465	0.07571	0.05866
Least Squares TPS	200/25	0.02520	0.02646	0.05603	0.03385
Foley	200/5×5	0.01206	0.01332	0.04819	0.04965
Least Squares TPS	200/35	0.01662	0.01843	0.05274	0.02853
Foley	200/6×6	0.00968	0.01144	0.05028	0.03962
Smoothing	200	0.0	0.00254	0.03900	0.02789

TABLE 3
Comparison of RMS *errors on "humps and dips,"* 500 *points.*

Method	Number of Data Points/ Knot Points	No Errors in Data		Contaminated Data	
		Residual	Grid	Residual	Grid
Least Squares TPS	500/20	0.02402	0.02517	0.05256	0.02738
Least Squares TPS	500/25	0.01664	0.01766	0.04818	0.02283
Foley	500/5×5	0.01346	0.01230	0.05844	0.03767
Least Squares TPS	500/50	0.00645	0.00845	0.04544	0.01961
Foley	500/7×7	0.00645	0.00552	0.05696	0.04864

In this case, undersmoothing has occurred, and the surface is "drawing" toward the error.

In comparing least squares TPS to the smoothing spline method in the exact data case, we note that the smoothing spline method yields a residual RMS error of zero. This could be expected, since there is no error in the data and the spline of interpolation is chosen. On the grid, the RMS error is small. When the data is not contaminated, the RMS error of least squares algorithm only begins to become as small as that of the smoothing splines method when the number of knots used becomes large. We also note that in the 500 data point set, no comparison is made since a potential limit for computing smoothing splines is 200–300 data points.

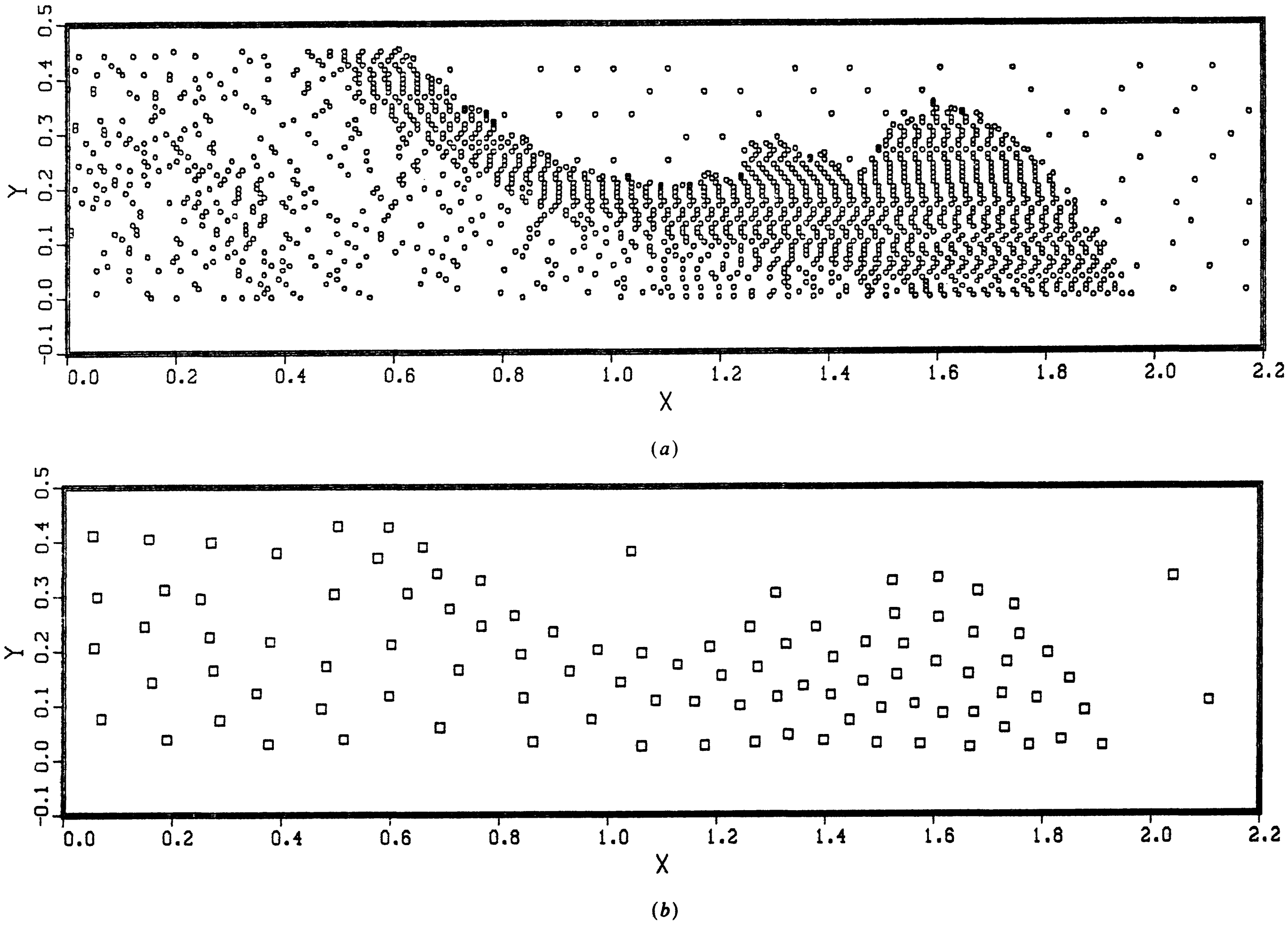
Foley's method for the contaminated case gives errors nearly equal to the composite standard deviation injected into the data. However, on the grid, the least squares method does better, an indication that smoothing is occurring, as expected. We also note that an increase in the number of grid points does not significantly improve the RMS error in Foley's method, even though an increase in the number of knots in the least squares method usually yields improved results. We used the default local approximations in Foley's method, and we note that performance of the method may be improved by using lower degree local approximations to estimate the grid values to be used.

Figure 4 depicts hydrographic data collected in Monterey Bay, consisting of 1669 points with greatly varying density. Figure 4(b) shows the results of applying Algorithm 2.1 with 100 knots, and is particularly interesting because the density of the data is faithfully replicated by the knots. We note, however, that the assumption regarding the density of the data points being indicative of the dependent variable is not necessarily a viable one in this particular application of the algorithm. These results demonstrate the ability of Algorithm 2.1 to produce reasonably representative sets of knots.

We discuss one further example arising from a practical problem. The general problem occurs with some frequency in practice, and it is not clear what the best method is for handling such data. The data consists of digitized height contours of a glacier. The contours were very densely sampled, with a total of 8345 points available. The contours were thinned by taking every third point, since this resulted in almost uniform density over part of the region while still leaving evidence of the contours. Those within a certain square were then retained, giving the total of 873 points shown in Fig. 5(a). In order to obtain an example which is not unreasonable, and under which our procedure can be expected to give reasonable results while other methods may encounter difficulties, a number of contours were stripped out of the data to leave the set of 678 points shown in Fig. 5(b). Admittedly this is a contrived example and one made to illustrate difficulties that tensor product methods can have when there are voids in the data. It is a situation which may occur in practical problems, however, and the missing data gives an opportunity to validate behaviors. The knot selection process applied to the data of Fig. 5(b) with 100 knots then gave the results shown in Fig. 5(c).

Figures 6(a) and (b) show the results when Foley's method (a 10×10 grid was used) is applied to the data of Figs. 5(a) and (b), and Fig. 6(c) shows the least squares TPS using the knots in Fig. 5(c). Figure 6(b) shows the potential misbehavior of tensor product methods in voids, while the TPS in Fig. 6(c) is well behaved. Outside of these regions the behavior of the methods is quite similar, although both Figs. 6(b) and (c) show that the details at the rear edge of the surface are incorrect because of the missing data. The RMS value of the residuals is comparable in the three cases: 5.2, 5.1, and 5.7, respectively (the contour interval is 25).

4. Conclusions. Again, we emphasize that we envision this scheme to be a useful one when other less costly schemes perform poorly or cannot be used because of the large systems of equations that must be solved. In particular, our scheme does not place knots in data deserts, avoiding the necessity of providing special handling for such cases. We caution that the search for a best knot configuration can be rather



(*a*)

(*b*)

FIG. 4. *Hydrographic data from Monterey Bay, California.* (a) *There are* 1669 *data points.* (b) *The* 100 *knot points for the data clearly mirror the underlying density.*

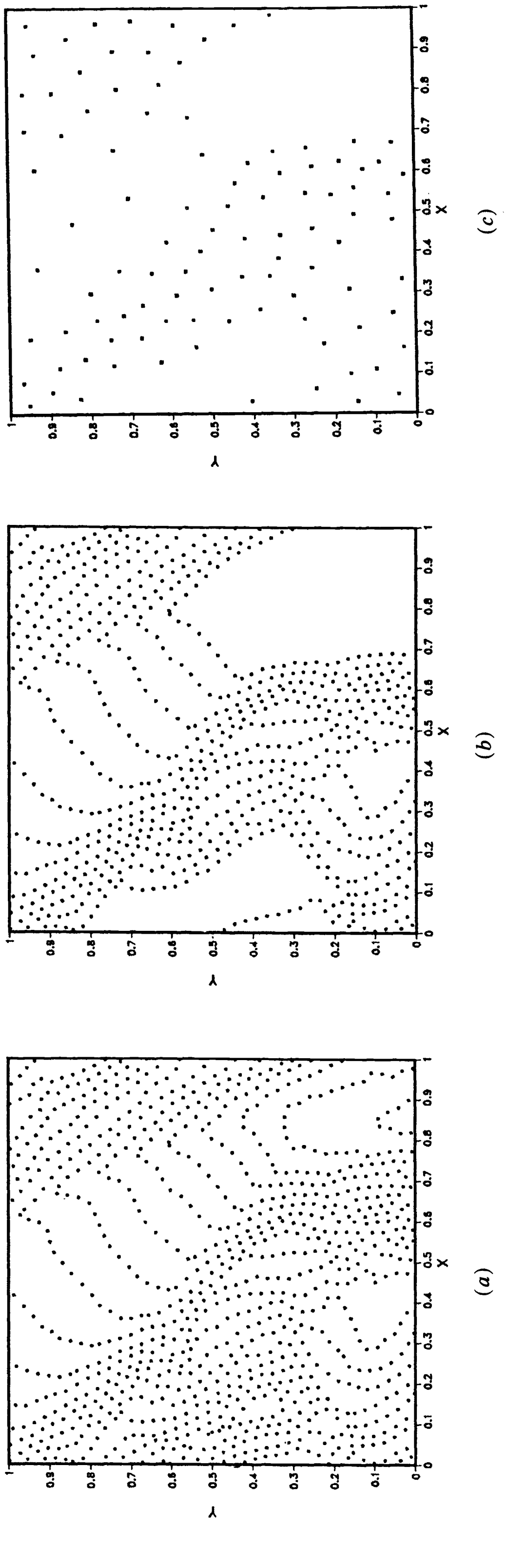
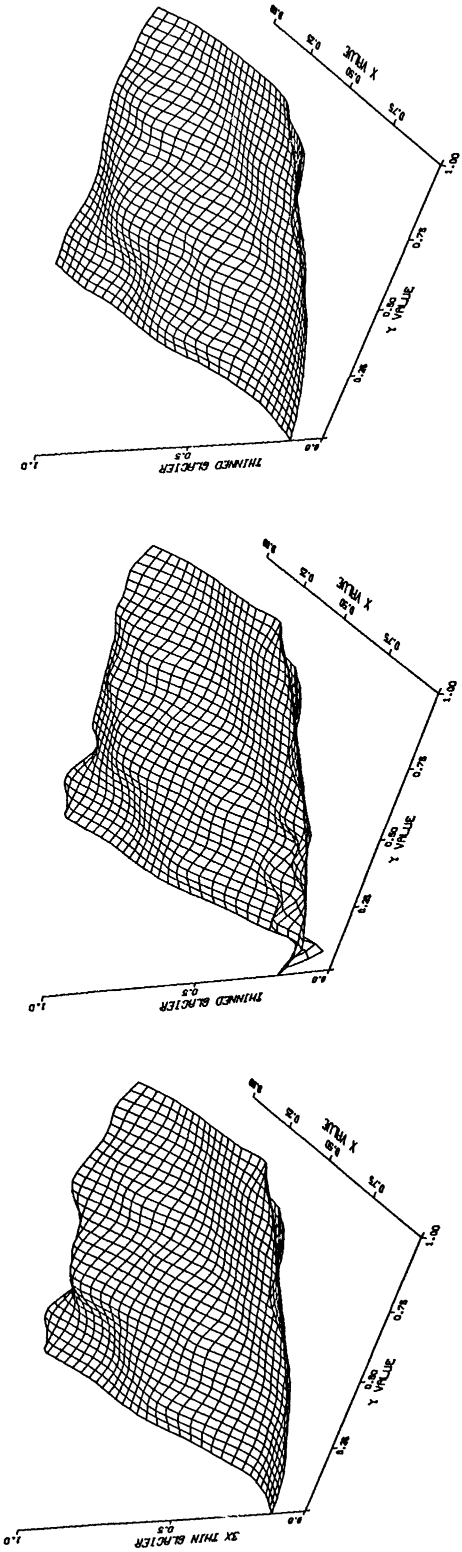


FIG. 5. (a) *Glacier contour data consisting of 873 points.* (b) *678 point subset of data in* (a). (c) 100 *knot points.*



(a) (b) (c)

FIG. 6. (a) *Foley's bicubic Hermite fit to data of Fig.* 5(a) *using a* 10 × 10 *grid.* (b) *Foley's bicubic Hermite fit to the data of Fig.* 5(b) *using a* 10 × 10 *grid.* (c) *Least squares* TPS *for the data of Fig.* 5(b), *using the* 100 *knots shown in Fig.* 5(c).

expensive. For a large number of data points with a moderately large number of knot points, such as our last two examples, the computations took up to an hour on an IBM 3033 computer. We are investigating ways of speeding up Algorithm 2.1. One idea we are pursuing is to use a method of simulated annealing along with a simpler search process. Again, the results are dependent on the initial guess, although they generally look quite good for any reasonable initial guess. It may be viable to incorporate a divide and conquer scheme such as that suggested in [14] for clustering problems using k-means. In our situation, only a good set of knots is needed, and it is unclear whether great efforts will be worthwhile since the criteria are only based on reasonable, and not rigidly correct, assumptions.

REFERENCES

[1] J. Dongarra, C. Moler, J. Bunch, and G. Stewart, *LINPACK User's Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979.

[2] M. Bozzini, F. DiTisi, and L. Lenarduzzi, *A new method in order to determine the most significant members within a large sample*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 98-104.

[3] M. G. Cox, *Data approximation by splines in one and two independent variables*, in The State of the Art in Numerical Analysis, A. Iserles and M. J. D. Powell, eds., Clarendon Press, Oxford, 1987, pp. 111-138.

[4] T. Foley, *Interpolation and approximation of* 3-*D and* 4-*D scattered data*, Comput. Math. Appl., 13 (1987), pp. 711-740.

[5] R. Franke, *Scattered data interpolation*: *Tests of some methods*, Math. Comp., 38 (1982), pp. 181-200.

[6] R. Harder and R. Desmarais, *Interpolation using surface splines*, J. Aircraft, 9 (1972), pp. 189-191.

[7] A. LeMéhauté and Y. Lafranche, *A knot removal strategy for scattered data in* R^2, in Mathematical Methods in CAGD, T. Lyche and L. L. Schumaker, eds., Academic Press, New York, 1989, pp. 419-426.

[8] J. McMahon, *Knot selection for least squares approximation using thin plate splines*, M.S. thesis, Naval Postgraduate School, Monterey, CA, June 1986.

[9] G. Nielson, R. Franke, L. Lenarduzzi, and F. Utreras, Personal communication on the Venezia criterion.

[10] R. Schiro and G. Williams, *An adaptive application of multiquadratic interpolants for numerically modelling large numbers of irregularly spaced hydrographic data*, Surveying Mapping, 44 (1984), pp. 365-381.

[11] R. Schmidt, *Ein Beitrag zur Flächenapproximation über Unregelmässig Verteilten Daten*, in Multivariate Approximation Theory III, W. Schempp and K. Zeller, eds., Birkhäuser-Verlag, Basel, 1985, pp. 363-369.

[12] S. Z. Selem and M. A. Ismail, *K-means-type algorithms*: *A generalized convergence theorem and characterization of local optimality*, IEEE Trans. Pattern Anal. Mach. Intell., PAMI-6 (1984), pp. 81-87.

[13] G. Wahba and J. Wendelberger, *Some new mathematical methods for variational objective analysis using splines and cross validation*, Monthly Weather Rev., 108 (1980), pp. 1122-1143.

[14] S. J. Wan, S. K. M. Wong, and P. Prusinkiewicz, *An algorithm for multidimensional data clustering*, ACM Trans. Math. Software, 14 (1988), pp. 153-162.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 2, pp. 499-511, March 1992

004

NUMERICAL APPROXIMATION OF PARAMETRIC ORIENTED AREA-MINIMIZING HYPERSURFACES*

HAROLD R. PARKS†

Abstract. A numerical method for finding a function nearly minimizing the gradient integral among functions having given boundary data is described. Such functions can be used to approximate parametric oriented area-minimizing hypersurfaces. Results of using this method are presented.

Key words. least gradient method, area-minimizing surfaces, area-minimizing currents, functions of least gradient

AMS(MOS) subject classifications. 65N99, 49Q05, 49-04

1. Introduction. This paper presents the results of what we believe to be the first successful implementation of a numerical method for finding the solution to the least area problem without restricting the topological type of the surfaces considered.

The problem of finding a surface of least area with given boundary was first investigated by Lagrange [L] in the mid-eighteenth century. Since that time, the least area problem and the related study of minimal surfaces and their generalizations have captured the interest of many mathematicians. Often the attention given to the subject is motivated solely by its beauty; however, there are connections to physical problems, and practical applications have arisen. Soap films provide the most well known physical realization of the least area problem, but other related physical problems are Tschaplygin gas flows and nonlinear elasticity theory. The reader should consult the introduction of Nitsche's book [NJ] and the references mentioned there for a more complete discussion.

Only in very special circumstances can one analytically compute the area-minimizing surface which spans a given boundary. Thus for many years, going back at least to 1928 when Jesse Douglas published a paper on the subject [DJ], there has been interest in numerical methods for approximating area-minimzing surfaces. In case the area-minimizing surface being sought is not the graph of a function, i.e., is parametric, there is no analytical method to determine in advance even such a gross characteristic of the surface to be approximated as its topological type. This can be a significant difficulty, because it seems to be more natural for a numerical method to lessen the topological complexity of a surface as the method proceeds than it is to increase the complexity. In [PH1] and [PH2], the author has developed the theoretical basis for a method for numerically approximating parametric oriented area-minimizing hypersurfaces which can be applied without prior knowledge of the topological type (or, for higher dimensions, the singularity structure) of the minimizing surface. Indeed, the applications described in this paper show that the method will, in practice as well as in theory, produce surfaces with the needed topological complexity even if the starting data consists of only topologically trivial surfaces.

The method from [PH1] and [PH2], which we will refer to as the *least gradient method*, is applicable in Euclidean space of any dimension, and it allows the approximation of a parametric oriented area-minimizing hypersurface spanning a given boundary,

* Received by the editors December 26, 1989; accepted for publication (in revised form) November 20, 1990. This work was partially supported by Office of Naval Research U.R.I. grant #N00014-86-K-0687. Equipment was provided, in part, by National Science Foundation grant #DMS-8703928 and by Hewlett-Packard Corporation.

† Department of Mathematics, Oregon State University, Corvallis, Oregon 97331.

provided that the given boundary is extreme in the sense that it lies on the bounding surface of a convex domain. The least gradient method is based on the fact that any level set of a function which minimizes the integral of the norm of its gradient is an area-minimizing hypersurface. This is a nontrivial theorem of Bombieri, De Giorgi, and Giusti [BDG, Thm. 1]. Underlying this result is the coarea formula [FH, 3.2.11], which, as a special case, states that the three-dimensional gradient integral is equal to the integral of the areas of the level sets,

$$\int_\Omega |Du| \, d\mathscr{L}^3 = \int \mathscr{H}^2[\Omega \cap u^{-1}(r)] \, dr,$$

where Ω is a domain in $\mathbb{R}^3$, $|Du|$ denotes the Euclidean norm of the gradient of u, $\mathscr{L}^3$ denotes Lebesgue measure, and $\mathscr{H}^2$ denotes two-dimensional Hausdorff measure. In particular, the least gradient integral is equal to the integral of the areas of the minimal surfaces with the appropriate boundary values. It turns out, as was shown in [PH1] and [PH2], that if a function only nearly minimizes the integral of the norm of its gradient, then its level sets can be used to approximate area-minimizing hypersurfaces.

This paper reports on the implementation of the least gradient method in the approximation of two-dimensional oriented parametric area-minimizing hypersurfaces in $\mathbb{R}^3$. To implement the least gradient method, the problem which must be solved numerically is a discrete approximation to the following.

$$\text{(1a)} \qquad \text{Minimize} \int_\Omega |Du| \, d\mathscr{L}^3$$

subject to the Dirichlet boundary condition

$$\text{(1b)} \qquad u = \varphi \text{ on } \partial\Omega,$$

where Ω is a bounded convex domain in $\mathbb{R}^3$. The boundary data φ in (1b) is chosen so that the one-dimensional system of curves we wish to span with an area-minimizing surface occurs as a level set of φ. The appropriate level set of the discrete approximation to the solution to (1a) and (1b) is then used as the approximation to the desired area-minimizing surface. We will apply the least gradient method to three examples.

(i) The surface to be approximated can be found analytically and is actually a graph, namely, a portion of Enneper's surface.

(ii) The surface to be approximated can be found analytically and is either two discs or a catenoid depending on the exact choice of boundary conditions.

(iii) The surface to be approximated cannot be obtained analytically (at least as far as we know) and the topological type is either that of two discs or an annulus depending on the exact choice of boundary conditions.

We remark here that in [PH1] and [PH2] certain strict assumptions were made about Ω in order to prove theorems of general applicability. One such theorem concerned the existence of a Lipschitzian function of least gradient satisfying the Dirichlet boundary condition. Here we will be solving specific problems. In each case we can easily give a proof of the existence of a Lipschitzian function of least gradient, so it is not significant that we have not worked precisely in the settings of the theorems in [PH1] and [PH2].

Various methods can be used to solve the numerical problem described above; one of the more promising methods has been investigated by Overton in [OM1] and [OM2]. In [OM1], Overton has proved the quadratic convergence of a method for minimizing a sum of Euclidean norms which, at each iteration, computes a direction of search by solving the Newton system of equations, projected, if necessary, into a linear submanifold. In [OM2], Overton has applied the method from [OM1] to the problem consisting of (1a), (1b), and an additional constraint, but with Ω a domain in $\mathbb{R}^2$; in that case, the minimization problem arose from collapse load analysis (see [SG]). While it was noted in [OM2] that the hypotheses required for the theorems in [OM1] were not satisfied, the numerical results obtained were nonetheless in close agreement with known exact solutions. We have set up our discretization so that Overton's method can be applied; however, our experience in applying Overton's method to our problem has been less sanguine than in [OM2]. Indeed it has proved essential to use search directions other than those arising from the Newton system. The difficulties here are still not fully understood.

To apply Overton's method in [OM2] without making prohibitive demands on the machine memory certain technical details were essential. (Specifically, in the notation of [OM2], the problem is storing and factoring $\tilde{A}$.) The same difficulty arises here. A major portion of the exposition in this paper deals with this. It is most efficient to assume the reader is familiar with [OM2] and limit ourselves to discussing the modifications needed.

2. Discretization. The domain Ω will be an open subset of the unit cube in $\mathbb{R}^3$. The unit cube, denoted by C, will be tessellated by subdividing the cube into $(k-1)^3$ congruent subcubes and then dividing each such subcube into six quadrirectangular tetrahedra. (In Fig. 1, the six tetrahedra are illustrated.) The fact that each tetrahedron has three mutually orthogonal edges turns out to be crucial. In that sense this subdivision of the cube, which was well known to many (see for example [TG]), is the generalization of the subdivision of the square into two right triangles; once this three-dimensional subdivision has been done, it is fairly easy to see that this construction can be continued to any dimension.

The total number of mesh points will be

$$n = k^3. \tag{2}$$

Note that k is the number of mesh points which lie on the x axis, and

$$h = 1/(k-1) \tag{3}$$

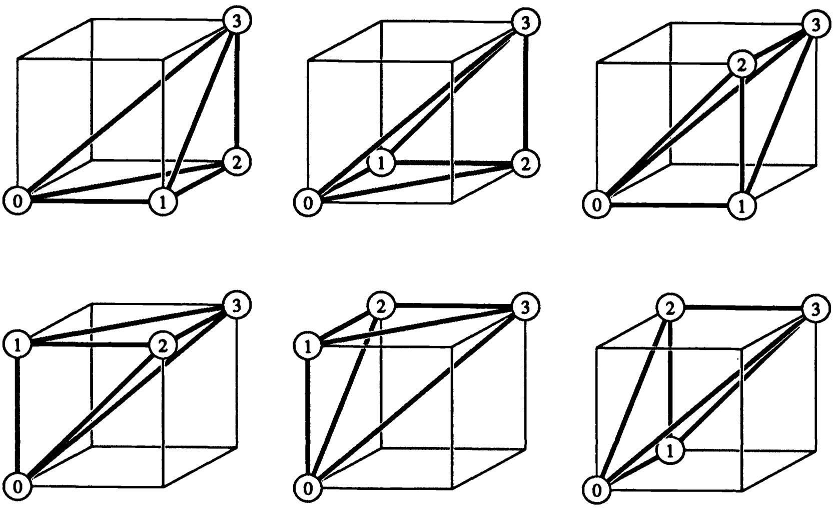


FIG. 1. *Decomposition of the cube.*

is the edge length of the subcubes, or mesh size. We will denote the mesh points by $X_1, X_2, \cdots, X_n$. The function φ will be assumed extended to $C \sim \Omega$. The function u in (1) will be replaced by a piecewise linear finite element approximation $\mathfrak{v}$, which is to agree with φ at mesh points in $C \sim \Omega$. Thus we obtain a finite-dimensional optimization problem for which the variables are the function values $\mathfrak{v}(X_i)$ at the given mesh points lying in Ω.

There are

$$m = 6(k-1)^3$$

tetrahedra in this tessellation of the unit cube. We will denote the tetrahedra by $T_1, T_2, \cdots, T_m$. The vertices of T_i will be denoted by

$$X_{\gamma(0,i)}, X_{\gamma(1,i)}, X_{\gamma(2,i)}, X_{\gamma(3,i)}.$$

We will assume that (as illustrated in Fig. 1), for each i, the vectors from $X_{\gamma(0,i)}$ to $X_{\gamma(1,i)}$, from $X_{\gamma(1,i)}$ to $X_{\gamma(2,i)}$, and from $X_{\gamma(2,i)}$ to $X_{\gamma(3,i)}$ are pairwise orthogonal and are, in fact, simply nonzero multiples of the standard basis vectors. Since $\mathfrak{v}$ is piecewise linear over this tessellation, the components of its gradient on T_i multiplied by the volume of T_i (i.e., $h^3/6$) are, possibly in some other order,

$$(\mathfrak{v}(X_{\gamma(1,i)}) - \mathfrak{v}(X_{\gamma(0,i)}))(h^2/6),$$
$$(\mathfrak{v}(X_{\gamma(2,i)}) - \mathfrak{v}(X_{\gamma(1,i)}))(h^2/6),$$
$$(\mathfrak{v}(X_{\gamma(3,i)}) - \mathfrak{v}(X_{\gamma(2,i)}))(h^2/6).$$

We now define a vector $v \in \mathbb{R}^n$ by setting $v_i = \mathfrak{v}(X_i)$, and we form a matrix A_i of dimension $n \times 3$ with

$$A_{i,\gamma(1,i),1} = A_{i,\gamma(2,i),2} = A_{i,\gamma(3,i),3} = h^2/6,$$
$$A_{i,\gamma(0,i),1} = A_{i,\gamma(1,i),2} = A_{i,\gamma(2,i),3} = -h^2/6,$$

and all other entries equal to zero. Note that the matrix A_i has only two nonzero entries in each column and those entries add to zero; this is a consequence of the fact that each tetrahedron in the tessellation has a set of three mutually perpendicular edges. This special structure of A_i turns out, as will be seen later, to be crucial to solving the problem economically. (It is also an economy that A_i need not be saved separately for each i, since it is readily constructed from γ.) Using the matrix A_i, we see that

$$\int_{T_i} |D\mathfrak{v}| \, d\mathscr{L}^3 = |A_i^T v|,$$

so that minimizing the integral of the gradient of $\mathfrak{v}$ over Ω is equivalent to the following problem:

$$\text{(4a)} \qquad \text{Minimize } F(v) = \textstyle\sum_i |A_i^T v|$$

over vectors $v \in \mathbb{R}^n$ which satisfy the boundary conditions

$$\text{(4b)} \qquad v_i = \varphi(X_i) \text{ if } X_i \text{ is not an interior point of } \Omega.$$

3. Adapting the algorithm. In solving the problem (4a), (4b), we will be following the procedure described in [OM2] which was, in turn, an adaptation of the algorithm proposed in [OM1]. As in [OM2] we set

$r_i(v) = A_i^T v$, the *residuals*,

$J(v) = \{i : |r_i(v)| = 0\}$, the *active set*,

$\hat{A}(v) = [A_{i_1}, A_{i_2}, \cdots]$, where $J(v) = \{i_1, i_2, \cdots\}$.

The problem (4a), (4b) is difficult because the gradient of the objective function, F, becomes discontinuous and the Hessian becomes unbounded whenever a residual vanishes. The idea of the algorithm is to project the objective function into the linear submanifold where zero residuals remain unchanged and where the boundary conditions (4b) are satisfied. In that space F is locally continuously differentiable. To accomplish this projection, we will construct a matrix Z consisting of a maximal set of independent n-dimensional columns such that

$$r_i(v+p)=r_i(v)=0 \quad \text{if } i\in J(v) \quad \text{and} \quad p\in \mathcal{R}(Z), \tag{5}$$

where $\mathcal{R}(Z)$ is the span of the columns of Z (i.e., the range space of Z). The requirement (5) of maintaining zero residuals is expressed by

$$\hat{A}^TZ=0,$$

but to incorporate the boundary conditions we must add columns to $\hat{A}$. For each boundary vertex X_r we add a column which has a one in the rth row and has zeros for all its other entries; this matrix is denoted by $\tilde{A}$. To maintain the zero residuals and satisfy the boundary conditions we require

$$\tilde{A}^TZ=0. \tag{6}$$

In [OM1], the matrix $\tilde{A}$ was assumed to be of full rank and the required Z was to be found by QR factorization. In our situation, as in [OM2], $\tilde{A}$ may quite well be rank deficient, and, in any case, n will surely be too large for storing and factoring $\tilde{A}$ to be practical. It is in getting around this difficulty that the special form of $\tilde{A}$ comes into play.

Definition. (i) Denote by $\mathfrak{A}=\mathfrak{A}_{n,l}$ the collection of $n\times l$ matrices such that each column contains at most two nonzero entries, and if a column does contain two nonzero entries, then they add to zero.

(ii) For $A\in\mathfrak{A}$, we define an equivalence relation, $\stackrel{A}{\sim}$, on $\{1,2,\cdots,n\}$ by setting

$i\stackrel{A}{\sim}j$ if and only if there exists a sequence i_1,
$i_2,\cdots,i_r$, with $i=i_1, j=i_r$, such that the i_lth row and the i_{l+1}st row each
have a nonzero entry in the same column, for $l=1,2,\cdots,r-1$.

The equivalence class of i will be denoted $\{i\}$.

(iii) An equivalence class $\{i\}$ will be called *null* if there is $j\in\{i\}$ such that the jth row of A contains a nonzero entry which is the only nonzero entry in its column.

(iv) Given $A\in\mathfrak{A}$ we construct $Z=Z_A$ as follows:

For each equivalence class $\{i\}$ which is not null include in Z a column which has a one in the jth row if and only if $j\in\{i\}$ and for which all other entries of the column are zeros.

With these definitions made, the following lemma should be obvious.

Lemma. *If $A\in\mathfrak{A}$, then $Z=Z_A$ is a maximal rank matrix such that $A^TZ=0$.* The main consequence of this lemma is that if we apply it to $\tilde{A}$, then the required maximal rank Z satisfying (6) is easily constructed. In fact, $\tilde{A}$ and Z can be represented as linked lists as was done in [OM2]. The remaining discussion of the adaptation of the algorithm from [OM1] is exactly as in [OM2].

4. Degeneracy. It was indicated in [OM2] that the most difficult step in adapting the algorithm from [OM1] to the problem (1a), (1b) was the check on optimality. The difficulty is referred to as *degeneracy*. A vector v that is optimal on the restricting submanifold may not be optimal in the whole space. This seems to occur in our applications in which a nonunique solution to the area-minimization problem leads to a so-called *lens*: the region in between two area-minimizing surfaces on which the function of least gradient is constant.

Since we must obtain a global optimal solution we have employed a *relaxation* method to reduce the objective function further once the optimality condition holds on the restricted manifold. We do a simple minimization varying each component of v separately. Since there are many components this is obviously slow. One efficiency is that any specific mesh point is a vertex of only 24 tetrahedra, so it is only necessary to compute a small part of the objective function. We have used the Golden Section Search [NR, 10.1] to accomplish the one-dimensional minimization.

In § 4.6 of [OM1], Overton comments that it is unlikely that a residual will be mistakenly set to zero. Essentially, this is because a set of codimension two or higher does not disconnect space. Apparently this intuition is misleading, because it has been necessary, in the problems involving a change in topological structure from initial data to optimal data, to apply the relaxation method to get off the restricted manifold.

5. Numerical results. There are situations in which it is possible to calculate analytically the exact solution to a least gradient problem. The simplest such situation is when the boundary data is the restriction of a linear function. This trivial case was useful for debugging.

5.1 As our first nontrivial test, we have used our method to approximate a portion of Enneper's surface. Since an exact parametrization is available (see [BC]) we can readily estimate the accuracy of our approximation. Specifically, we use the portion of the surface parametrized by

$$\begin{aligned} x &= 3(u - \tfrac{1}{3}u^3 + uv^2),\\ y &= \tfrac{1}{4} + 3(-v + \tfrac{1}{3}v^3 - u^2 v),\\ z &= .43 + 3(u^2 - v^2), \end{aligned}$$

which lies in the unit cube; the surface has purposely been positioned so as not to be symmetric. Supposing the part of the above surface which lies in the unit cube is given nonparametrically by

$$z = \psi(x, y),$$

we can easily and rapidly determine $\psi(x, y)$ numerically by solving for (u, v) through the iteration

$$\begin{aligned} u_0 &= 0, \qquad v_0 = 0\\ u_{n+1} &= \tfrac{1}{3}x + \tfrac{1}{3}(u_n)^3 - (u_n)(v_n)^2,\\ v_{n+1} &= -\tfrac{1}{3}(y - \tfrac{1}{4}) + \tfrac{1}{3}(v_n)^3 - (u_n)^2(v_n). \end{aligned}$$

One can then verify that the height of the surface varies from about .2 to about .8.

TABLE 1
Approximation to Enneper's surface.

k	n	Initial F	Final F	Error	Ratio of error to mesh size
7	343	1.1029466511	1.0746701779	.0045223315	.0271340
9	729	1.1016168059	1.0643732799	.0034019702	.0272158
11	1331	1.1305241338	1.0633118971	.0030688406	.0306888

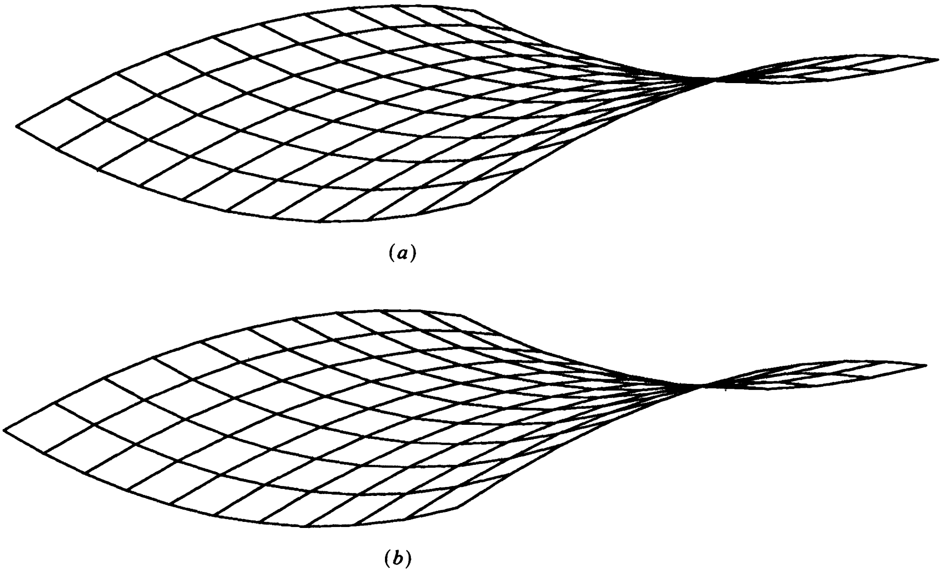


FIG. 2. (a) *Enneper's surface by the usual formulas*; (b) *Enneper's surface by the least gradient method.*

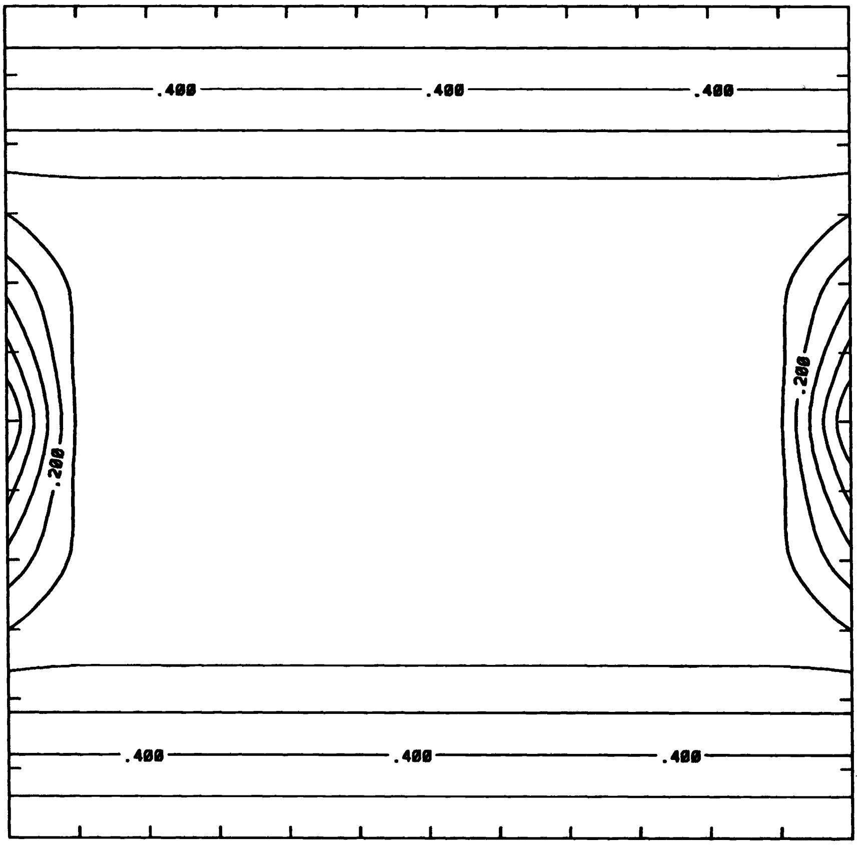


FIG. 3. *Level curves for cross section of exact solution to the catenoid problem.*

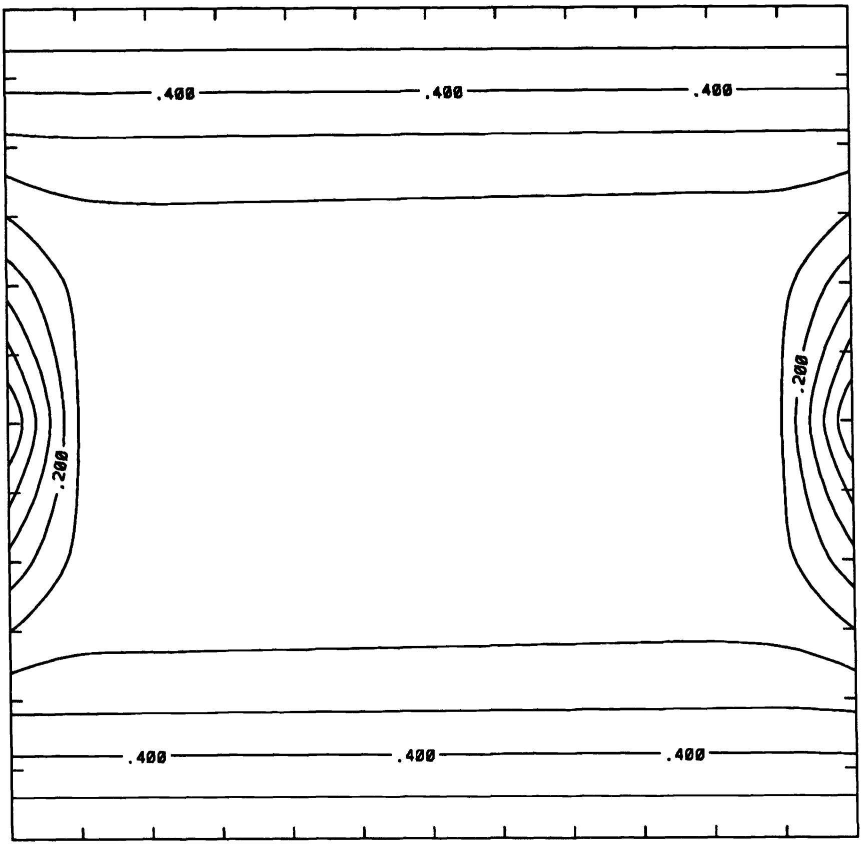


FIG. 4. *Level curves for cross section of approximate solution to the catenoid problem.*

The Dirichlet data imposed on the unit cube for the least gradient problem were chosen to be zero on the base of the unit cube, one on the top of the unit cube, and one-half on

$$\{(x, y, \psi(x, y)): xy(1-x)(1-y) = 0\}.$$

The values at the remaining boundary points were smoothly interpolated in such a way as to be increasing in the z-direction. The solution to the least gradient problem with such boundary values would necessarily be increasing in the z-direction, but we know of no method for obtaining the solution analytically. The starting data was the extension of the boundary data construction, but with a random term added.

The results of this test are summarized in Table 1. In that table, k is the number of grid points on each axis, n is the total number of grid points as in (2), and F is as in (4a). In each case the computed approximation to u was increasing in the z-direction. Linearly interpolating between grid points along lines parallel to the z-axis, we located the level where the function had the value $\frac{1}{2}$. That defines our computed approximation to ψ. The error is the maximum of the absolute value of the difference between ψ and the computed approximation to ψ, the maximum being taken over all grid points in the unit square in the x,y-plane. As a measure of the accuracy, we have compared the error with the mesh size, h.

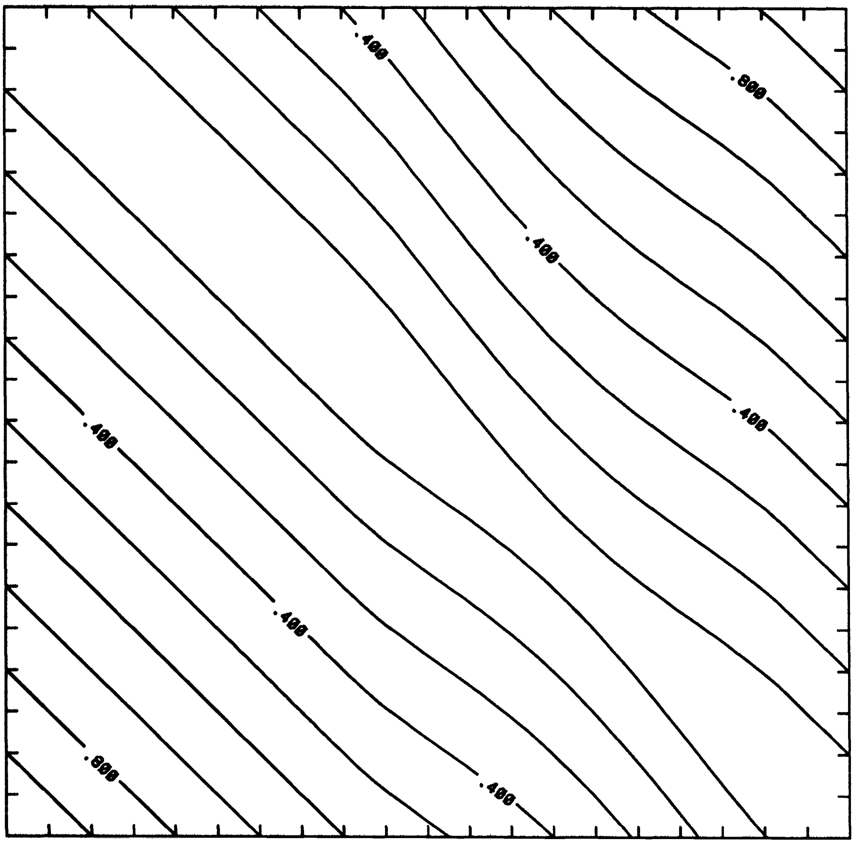


FIG. 5. *Level curves for cross section of initial guess for the problem which is not analytically solvable.*

In Fig. 2, we have used the NCAR Graphics routine EZSRFC to illustrate the portion of Enneper's surface and the approximation obtained by our method; they are visually indistinguishable.

5.2. Of course, for a nonparametric problem such as that illustrated in § 5.1, a numerical method was developed over sixty years ago by Douglas [DJ] which might well be just as fast and accurate and is certainly much faster and easier to program, but the most important attribute of our method is that it does not require a priori knowledge of the topological type of the solution. To illustrate and test this our second nontrivial problem involved the transition from two discs to the catenoid. In this case, all mesh points (x, y, z) at distance greater than or equal to one-half from the line $\{(\frac{1}{2}, \frac{1}{2}, t) : t \in \mathbb{R}\}$ will be assigned a fixed value, namely, $|z-\frac{1}{2}|$. The transition from catenoid to plane solution should occur at $|z-\frac{1}{2}|$ approximately equal to .264 (a better estimate of this is .2638987).

The function, u, of least gradient with the values $|z-\frac{1}{2}|$ on the boundary of the cylinder

$$\Omega = \{(x, y, z) : (x-\tfrac{1}{2})^2 + (y-\tfrac{1}{2})^2 < \tfrac{1}{4}, 0 < z < 1\}$$

can be directly evaluated at each point by solving a few analytic equations. To each

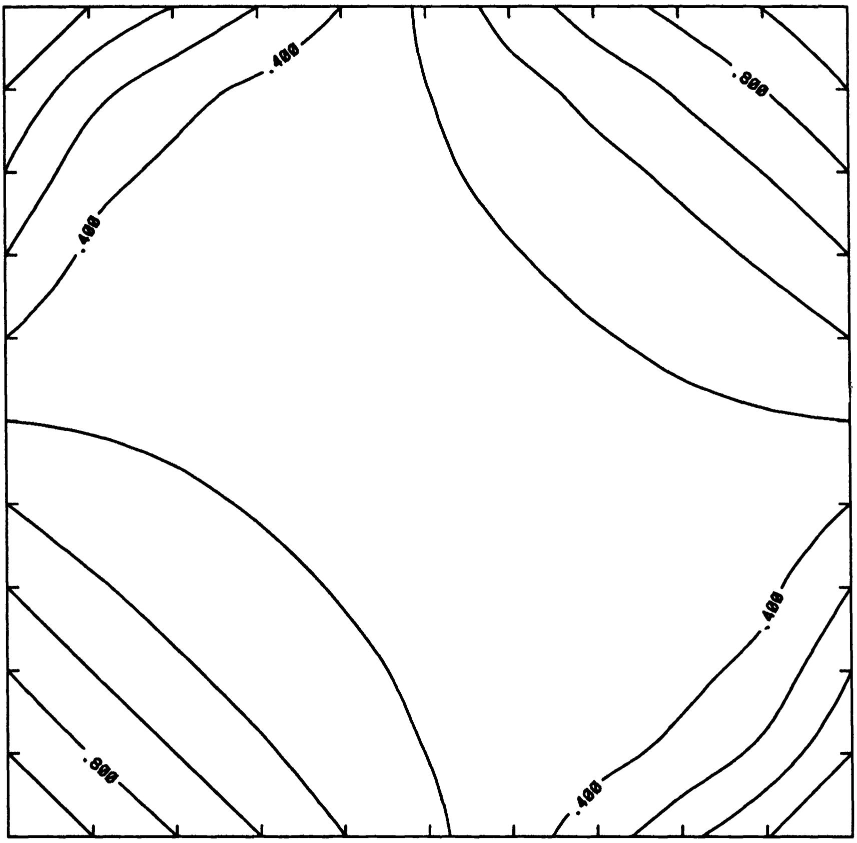


FIG. 6. *Level curves for cross section of approximate solution to the problem which is not analytically solvable.*

point (x, y, z) we associate

$$r=[(x-\tfrac{1}{2})^2+(y-\tfrac{1}{2})^2]^{1/2}$$

and

$$h=z-\tfrac{1}{2}.$$

Recall that the equation of a catenoid is

$$\lambda r=\cosh(\lambda h),$$

where λ is a parameter that we can choose. We find the transition from discs to catenoid by finding λ^* and $h^*>0$ such that

$$\lambda^*\tfrac{1}{2}=\cosh(\lambda^* h^*)$$

and the area of the catenoid, with parameter λ^*, between $h=-h^*$ and $h=h^*$ is equal to the area of two discs of radius $\frac{1}{2}$. Given any $(x, y, z)\in\Omega$, we evaluate u as follows. (i) If $|h|\geqq h^*$, then $u(x, y, z)=|z-\frac{1}{2}|$, (ii) if $|h|<h^*$, then we let r^* be the solution of

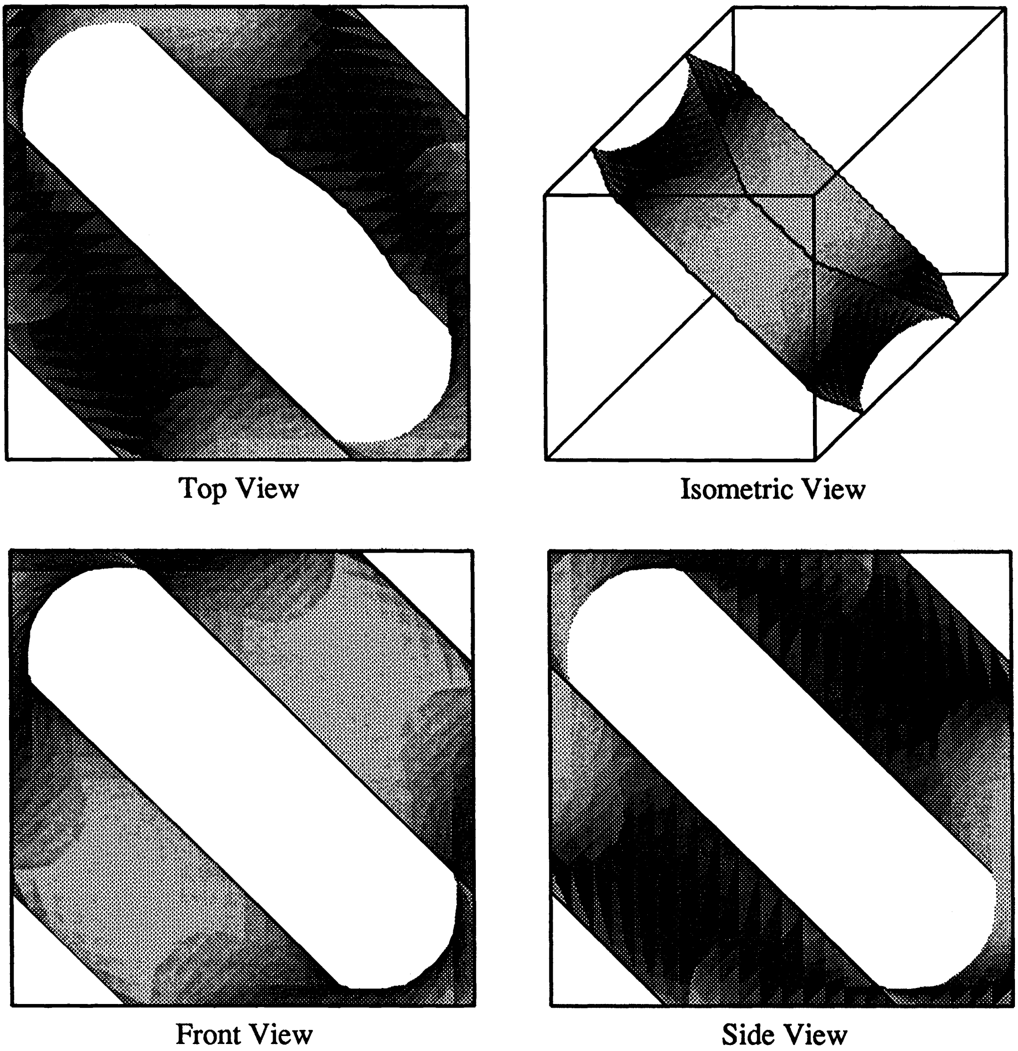
$$\lambda^* r^*=\cosh(\lambda^* h),$$

and in case $r\leqq r^*$ we have $u(x, y, z)=h^*$, while in the case $r>r^*$ we solve for λ in $\lambda r=\cosh(\lambda h)$ and set $u(x, y, z)$ equal to the solution of $\lambda\frac{1}{2}=\cosh(\lambda u)$.

In Fig. 3, we have sketched, using the NCAR Graphics routine CONREC, the level sets of $u(x, \frac{1}{2}, z)$ in the plane $\{(x, \frac{1}{2}, z) : x, z \in \mathbb{R}\}$. The presence of the lens does tend to confuse the level set routine, so we have chosen parameters for the routine which give a nice looking diagram for the known analytic solution; we have continued to use the same parameters in diagramming the approximation. The starting data we have used is equal to $|z - \frac{1}{2}|$ at (x, y, z); with that data the level sets are all discs. In this test the use of the relaxation method to get off the restricting manifold was essential. In Fig. 4, we show the contour lines from the approximate solution. Note that the topological type has become more complex, as it should.

5.3. In this case, we have taken Ω to be the entire unit cube and the boundary data is given by

$$\varphi(x, y, z) = |x + y + z - \tfrac{3}{2}| + \varepsilon(x, y),$$



Top View

Isometric View

Front View

Side View

FIG. 7. *Approximate minimal surface for the problem which is not analytically solvable (level .25).*

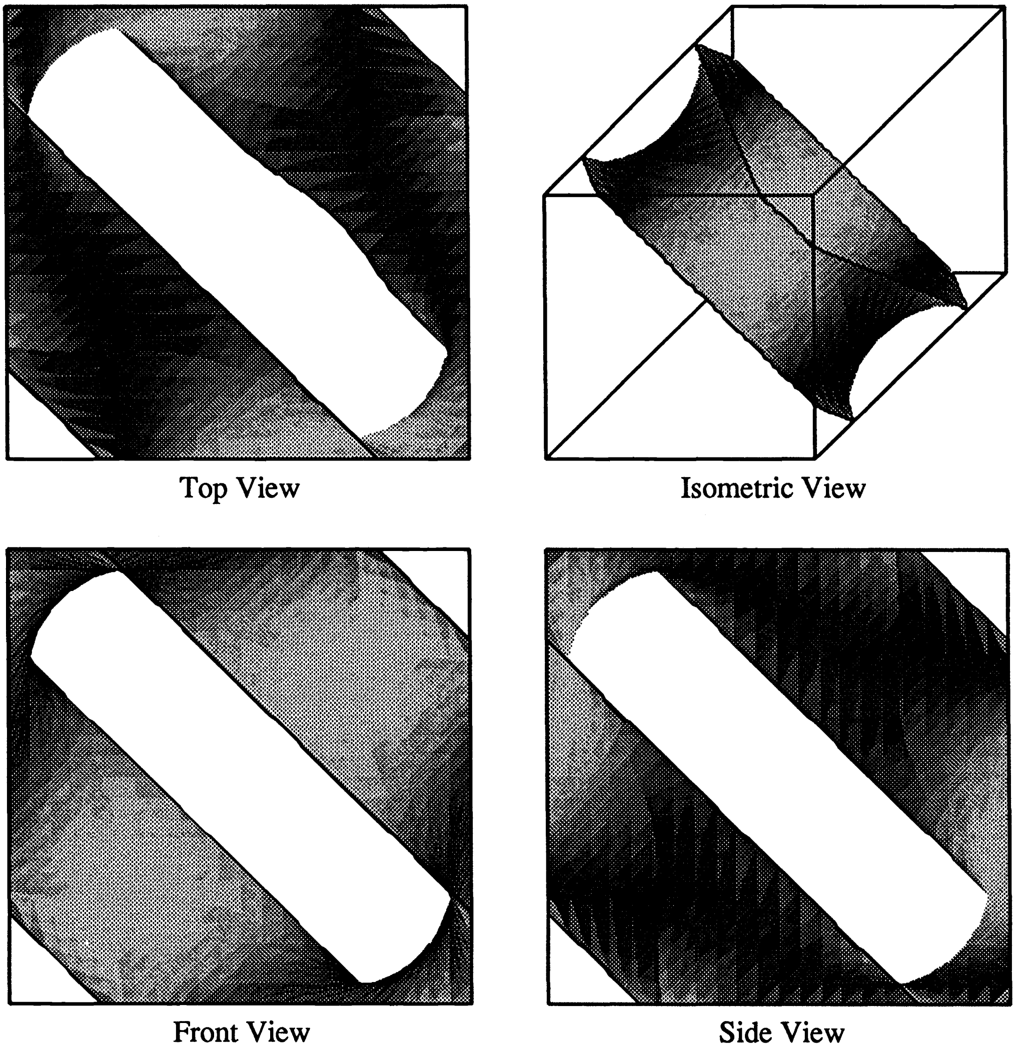


FIG. 8. *Approximate minimal surface for the problem which is not analytically solvable (level .30).*

where

$$\varepsilon(x, y) = \exp\left[-1/\left(\tfrac{1}{3} - 3(x-\tfrac{2}{3})^2 - 4(y-\tfrac{1}{2})^2\right)\right], \qquad \text{if } \tfrac{1}{3} - 3(x-\tfrac{2}{3})^2 - 4(y-\tfrac{1}{2})^2 > 0,$$

$$\varepsilon(x, y) = 0, \text{ otherwise.}$$

We would suspect that there must be a transition from the topological type of the disc to the topological type of the catenoid; however, there does not seem to be any analytic method to determine exactly where it occurs or to find the catenoid type solutions. (We would have used simply $|x+y+z-\frac{3}{2}|$ for the Dirichlet data, but it is conceivable that the function of least gradient could have been obtained analytically, since the minimal surfaces involved would all have polygonal boundaries. Of course, such a computation would be very difficult.) We have used our approximation method to approximate $u(x, y, z)$. The contours of the cross section $u(x, \frac{1}{2}, z)$ are sketched for the initial guess in Fig. 5 and for the function approximately of least gradient in Fig.

6. Figures 7 and 8 illustrate the level sets with value .25 and .30, respectively, from the same point of view. We emphasize that the topological type was not provided as part of the initial information, but is rather a result obtained from the computation.

Since in this final case, the exact solution is not known, the accuracy, as approximations of minimal surfaces, of the final surfaces pictured is also not known. In [PH1] and [PH2] it was proved that between two appropriate level sets there is a minimal surface, but the estimates from those papers are for the worst case and would require too large a value of k (recall k is the number of mesh points on each axis). Computations were terminated when the decrease in the gradient integral, especially as compared to the change in v, became sufficiently small. (In calling the subroutine that implements Overton's method for minimizing the sum of norms, we used the tolerance $\varepsilon_{PG} = 10^{-8}$ for the norm of the projected gradient.) The value of k was started at 7 and increased by interpolating from the results of the previous run. Final termination occurred when an adequate value of k was reached. Typically, $k = 11$ was enough to produce a smooth appearing contour plot for a cross section, while $k = 15$ was enough to produce a smooth appearing level set. For all practical purposes, our system limited us to $k \leqq 25$.

Acknowledgments. The author would like to thank Michael Overton for his generous assistance, including giving the author access to the FORTRAN code used to produce the numerical results presented in [OM1] and [OM2]. Further, the author would like to thank Jon Pitts and David Hendricks for their assistance with the computational work. The referee also deserves thanks for his or her careful work and helpful comments. Finally, the author would like to thank Frederick Almgren for applying gentle, but continual, pressure.

REFERENCES

[BC] J. L. M. Barbosa and A. G. Colares, *Minimal surfaces in* $\mathbb{R}^3$, Lecture Notes in Mathematics 1195, Springer-Verlag, New York, 1986.

[BDG] E. Bombieri, E. De Giorgi, and E. Giusti, *Minimal cones and the Bernstein problem*, Invent. Math., 7 (1969), pp. 243-268.

[DJ] J. Douglas, *A method of numerical solution of the problem of Plateau*, Ann. of Math. (2), 29 (1928), pp. 180-188.

[FH] H. Federer, *Geometric Measure Theory*, Springer-Verlag, New York, 1969.

[L] J.-L. Lagrange, *Essai d'une nouvelle méthode pour déterminer les maxima et les minima des formules intégrales indéfinies*, Œuvres de Lagrange I, pp. 335-362.

[NJ] J. C. C. Nitsche, *Lectures on Minimal Surfaces*, Vol. 1, Cambridge University Press, Cambridge, New York, 1989.

[NR] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes*, Cambridge University Press, Cambridge, New York, 1986.

[OM1] M. L. Overton, *A quadratically convergent method for minimizing a sum of Euclidean norms*, Math. Programming, 27 (1983), pp. 34-63.

[OM2] ———, *Numerical solution of a model problem from collapse load analysis*, Computational Methods in Applied Science and Engineering, VI, R. Glowinski and J. Lions, eds., North-Holland, Amsterdam, 1984.

[PH1] H. R. Parks, *Explicit determination of area minimizing hypersurfaces*, Duke Math. J., 44 (1977), pp. 519-534.

[PH2] ———, *Explicit determination of area minimizing hypersurfaces*, II, Mem. Amer. Math. Soc. 60, 1986.

[SG] G. Strang, *A minimax problem in plasticity theory*, Functional Analysis Methods in Numerical Analysis, M. Z. Nashed, ed., Lecture Notes in Mathematics 701, Springer-Verlag, New York, 1979.

[TG] G. L. Tindle, *Tetrahedral triangulation*, The Mathematics of Surfaces II, R. Martin, ed., Clarendon Press, Oxford, 1987.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 2, pp. 512-530, March 1992

005

NONLINEAR MULTIGRID APPLIED TO A ONE-DIMENSIONAL STATIONARY SEMICONDUCTOR MODEL*

P. M. DE ZEEUW†

Abstract. The nonlinear multigrid method is applied to a transistor problem in one dimension. A weak spot in the linearization of the well-known Scharfetter-Gummel discretization scheme is reported. Further, it is shown that both the residual transfer and the solution transfer from a fine to a coarse grid need special requirements due to the rapidly varying problem coefficients. Some modifications are proposed which make the multigrid algorithm perform well for the hard example problem.

Key words. semiconductor equations, multigrid methods

AMS(MOS) subject classifications. 65N20, 65H10

1. Introduction. There is a great demand for a proper numerical simulation of semiconductors in order to reduce the costs of constructing expensive prototypes. The search for a fast and robust algorithm has proven to be a challenge. So far only a few papers have considered the multigrid solution of the discrete semiconductor equations (e.g., see [1], [2], [6], [9], [13]) and therefore extensive further research is required.

In this paper we restrict ourselves on purpose to one space dimension as a preparatory study for the case of more space dimensions. We study a particular example problem which has been put forward by Schilders (Philips, the Netherlands). This problem models a transistor and turns out to be much harder to solve than the forward or reversed biased diode problem. We apply the nonlinear multigrid method and encounter a serious difficulty due to the nonlinearity of the problem. Some modifications are proposed which significantly increase the robustness of the nonlinear multigrid method and which look promising also for the higher-dimensional case.

2. The problem. The behavior of a steady semiconductor device can be described by the following set of equations (cf. [10]):

$$\nabla(-\varepsilon\nabla\psi)=q(p-n+D), \tag{2.1a}$$

$$\nabla J_n=+qR, \tag{2.1b}$$

$$\nabla J_p=-qR, \tag{2.1c}$$

where J_n and J_p are defined by

$$J_n=q\mu_n\left(\frac{1}{\alpha}\nabla n-n\nabla\psi\right), \tag{2.2a}$$

$$J_p=-q\mu_p\left(\frac{1}{\alpha}\nabla p+p\nabla\psi\right). \tag{2.2b}$$

Substitution of (2.2) into (2.1) results in a system of three nonlinear partial differential equations for ψ, n, and p. In (2.1) ψ represents the electrostatic potential, p and n describe the concentration of holes and electrons, respectively. Equations (2.1b) and (2.1c) are called the continuity equations, J_n is the electron current density, J_p is the hole current density, and R is the recombination-generation rate, a function of n and p. The doping profile D is a function of the space variable x. The quantities ε, q, α,

* Received by the editors June 21, 1989; accepted for publication (in revised form) November 14, 1990.

† Centre for Mathematics and Computer Science, P.O. Box 4079, 1009 AB Amsterdam, the Netherlands.

μ_n, μ_p represent the permittivity, the elementary charge, the inverse of the thermal voltage, and the electron and hole mobility, respectively.

In this paper we consider the case of only one space dimension and assume ε, α, μ_n and μ_p to be constant. It is common practice to replace the variables n and p by the hole and electron quasi-Fermi potentials ϕ_n and ϕ_p defined by the relations

$$\text{(2.3a)} \qquad n = n_i\, e^{\alpha(\psi-\phi_n)},$$

$$\text{(2.3b)} \qquad p = n_i\, e^{\alpha(\phi_p-\psi)}.$$

On the one hand, by this change of variables, the nonlinearity of the problem is strongly increased, on the other hand the values assumed by (ψ, ϕ_n, ϕ_p) are in a much more moderate range. For extensive discussions on the choice of variables see [9], [10]. Using (2.3) the equations (2.1) are transformed into

$$\text{(2.4a)} \qquad -\nabla J_\psi = n_i q(e^{\alpha(\phi_p-\psi)} - e^{\alpha(\psi-\phi_n)}) + qD,$$

$$\text{(2.4b)} \qquad -\nabla J_n = +qR,$$

$$\text{(2.4c)} \qquad -\nabla J_p = -qR,$$

where J_ψ is defined by

$$\text{(2.5a)} \qquad J_\psi = \varepsilon \nabla \psi,$$

and J_n, J_p are now defined by

$$\text{(2.5b)} \qquad J_n = \bar{\mu}_n\, e^{\alpha(\psi-\phi_n)} \nabla(\alpha\phi_n),$$

$$\text{(2.5c)} \qquad J_p = \bar{\mu}_p\, e^{\alpha(\phi_p-\psi)} \nabla(\alpha\phi_p),$$

with

$$\text{(2.5d)} \qquad \bar{\mu}_n = \frac{n_i q \mu_n}{\alpha}, \qquad \bar{\mu}_p = \frac{n_i q \mu_p}{\alpha}.$$

In this paper we adhere to the formulation (2.4)-(2.5).

2.1. A particular one-dimensional model problem. We will focus our attention to a particular (hard) one-dimensional model problem which has been supplied by Schilders [14]. Here the problem constants are

$$\text{(2.6)} \qquad \begin{array}{lll} \varepsilon = 1.035918_{10}^{-12}\ \text{As V}^{-1}\,\text{cm}^{-1}, & q = 1.6021_{10}^{-19}\ \text{As}, & \\ \mu_n = \mu_p = 500\ \text{V}^{-1}\,\text{s}^{-1}\,\text{cm}^2, & n_i = 1.22_{10}{}^{10}\ \text{cm}^{-3}, & \\ k = 1.38054_{10}^{-23}\ \text{V As K}^{-1}, & T = 300\ \text{K}, & \alpha = q/kT. \end{array}$$

The function R is given by

$$R = \frac{pn - n_i^2}{\tau(p+n+2n_i)}, \qquad \tau = 10^{-6}\ s.$$

The doping function D (in cm^{-3}) is given by

$$D(x) = 6_{10}{}^{15} + 6_{10}{}^{19} \exp(-(x/7.1_{10}^{-5})^2) - 2.15_{10}{}^{18} \exp(-(x/1.15_{10}^{-4})^2) + 1.1_{10}{}^{19} \exp(-((x-8_{10}^{-4})/1.3_{10}^{-4})^2).$$

The equations (2.4) are defined on the domain $\Omega = [0, 8_{10}^{-4}]$(cm). We have three contacts to our semiconductor device (the one-dimensional model of a transistor): the emitter (E), the basis (B) and the collector (C) (see Fig. 1).

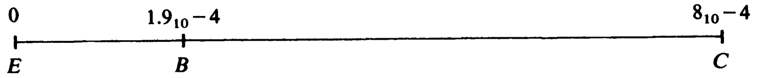


FIG. 1. *The contacts in the one-dimensional transistor problem.*

In Fig. 2 the doping function $D(x)$ is shown after the transformation $D \to sign(D)^{10}\log(1+|D|)$. Boundary conditions at the emitter E are:

(2.7a) $p-n+D=0$ (i.e., vanishing space charge),

(2.7b) $\phi_n = V_E$,

(2.7c) $J_p = 0$.

Boundary conditions at the basis B:

(2.8) $\phi_p = V_B = 0$.

Boundary conditions at the collector C:

(2.9a) $p-n+D=0$,

(2.9b) $\phi_n = V_C$,

(2.9c) $\phi_p = V_C$.

For fifteen different cases, each characterized by a pair of voltages (V_E, V_C), the solution is required (see Table 4.1). Figure 8 shows the solution-component ψ for the subsequent cases.

3. Discretization. At the outset of this section we give a short preview of its contents.

In order to abide by the law of conservation we use a finite volume technique based on the piecewise constant approximation of J_ψ, J_n, and J_p. As a consequence we arrive at a cell-centered version of the well-known Scharfetter-Gummel scheme [4], [9], [11]. We examine how the nonlinear discrete operator depends on the discrete solution.

3.1. Box integration. The interval $\Omega=(x_0, x_N)$ is split up into disjoint *boxes* $B_j=(x_{j-1}, x_j)$, $j=1\,(1)N$. A point x_j is called a *wall*, a point $x_{j-1/2}=(x_{j-1}+x_j)/2$ is called a *center*. The basis B is at the partition-wall between two boxes. Another set of

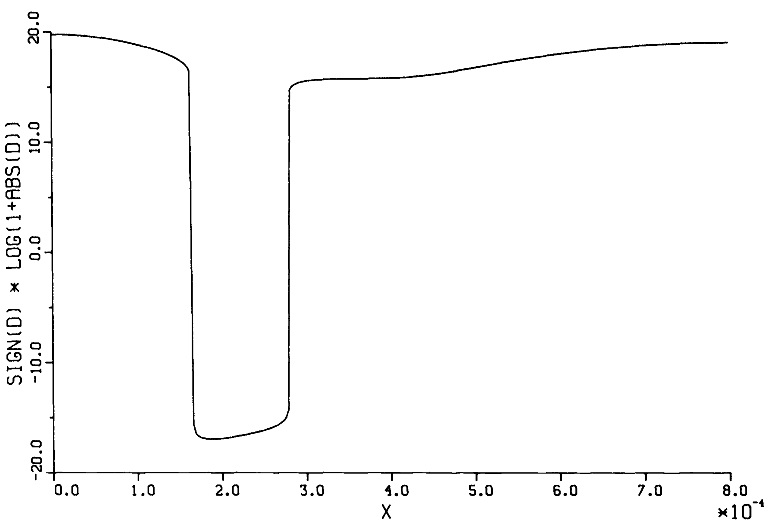


FIG. 2. *The doping profile.*

subintervals $\{D_j\}$ is defined by

$$\begin{aligned} D_0 &= (x_0, x_{1/2}), \\ D_j &= (x_{j-1/2}, x_{j+1/2}), \quad j = 1\,(1)N-1, \\ D_N &= (x_{N-1/2}, x_N). \end{aligned}$$

This set is called the set of *dual boxes.*

Now, by applying the Gauss divergence theorem in one dimension to (2.4) on the domains B_j we find

$$\begin{aligned} &-J_\psi\Big|_{x_{j-1}}^{x_j} - n_i q \int_{B_j} (e^{\alpha(\phi_p-\psi)} - e^{\alpha(\psi-\phi_n)})\, d\Omega = q \int_{B_j} D\, d\Omega, \\ (3.1)\qquad &-J_n\Big|_{x_{j-1}}^{x_j} - q \int_{B_j} R\, d\Omega = 0, \\ &-J_p\Big|_{x_{j-1}}^{x_j} + q \int_{B_j} R\, d\Omega = 0, \quad j = 1\,(1)N. \end{aligned}$$

We can write (3.1) in symbolic form as

$$(3.2)\qquad \mathcal{M}(\boldsymbol{q}) = f$$

where $\boldsymbol{q}$ denotes the vector $(\psi, \phi_n, \phi_p)^T$, $\mathcal{M}$ the nonlinear operator in the left-hand side of (3.1) and f the right-hand side of (3.1).

3.2. Box discretization. We introduce the variables $(\psi_j, \phi_{n,j}, \phi_{p,j})^T$, $j = 1\,(1)N$, which are associated with the centers $x_{j-1/2}$ of the boxes B_j. Let $\approx$ denote approximation by midpoint quadrature. We then define

$$\begin{aligned} &S_j \approx n_i q \int_{B_j} (\exp(\alpha(\phi_p - \psi)) - \exp(\alpha(\psi - \phi_n)))\, d\Omega, \\ &F_j \approx q \int_{B_j} D\, d\Omega, \\ (3.3)\qquad &R_j \approx q \int_{B_j} R\, d\Omega, \\ &j = 1\,(1)N. \end{aligned}$$

We make the assumption that J_ψ, J_n and J_p are piecewise constant on the dual set $\{D_j\}$ (see [4], [9], [11]) and correspondingly we use the notation $J_{\psi,j}$, $J_{n,j}$, $J_{p,j}$. By this assumption and applying (3.3) we arrive at the following discrete equations:

$$(3.4a)\qquad -J_{\psi,j} + J_{\psi,j-1} - S_j = F_j,$$

$$(3.4b)\qquad -J_{n,j} + J_{n,j-1} - R_j = 0,$$

$$(3.4c)\qquad -J_{p,j} + J_{p,j-1} + R_j = 0,$$

with

$$(3.5a)\qquad J_{\psi,j} = \varepsilon \frac{\psi_{j+1} - \psi_j}{x_{j+1/2} - x_{j-1/2}},$$

$$(3.5b)\qquad J_{n,j} = \bar{\mu}_n \frac{\exp(-\alpha\phi_{n,j+1}) - \exp(-\alpha\phi_{n,j})}{\exp(-\alpha\psi_{j+1}) - \exp(-\alpha\psi_j)} \cdot \frac{\alpha\psi_{j+1} - \alpha\psi_j}{x_{j+1/2} - x_{j-1/2}},$$

$$(3.5c)\qquad J_{p,j} = \bar{\mu}_p \frac{\exp(\alpha\phi_{p,j+1}) - \exp(\alpha\phi_{p,j})}{\exp(\alpha\psi_{j+1}) - \exp(\alpha\psi_j)} \cdot \frac{\alpha\psi_{j+1} - \alpha\psi_j}{x_{j+1/2} - x_{j-1/2}}.$$

At the emitter E, basis B, and collector C similar equations are obtained; for full details see [16]. Thus we have obtained the cell-centered version of the well-known Scharfetter-Gummel scheme (see [4], [9], [11]). Summarizing, we have discretized (2.4), together with the boundary conditions (2.7)-(2.9), into a set of $3N$ nonlinear equations (3.4) with the $3N$ variables ψ_j, $\phi_{n,j}$, $\phi_{p,j}$; $j=1\,(1)N$. We can write (3.4) in symbolic form as

$$\mathcal{M}_h(q_h)=f_h \tag{3.6}$$

where $\mathcal{M}_h$ denotes the nonlinear difference operator and f_h the right-hand side.

3.3. Properties of the discretized operator. In this subsection we study how the Jacobian of the nonlinear discrete operator $\mathcal{M}_h$ depends on the discrete solution. We assume the recombination term to be zero and confine ourselves to the dependency on ϕ_p. Results for ϕ_n can be derived analogously. We freeze the solution components ψ and ϕ_n and consider the ϕ_p-stencil, at box B_j, defined by the triplet

$$[stp(j,-1), stp(j,0), stp(j,+1)] \tag{3.7a}$$

with

$$stp(j,k)=\frac{\partial(-J_{p,j}+J_{p,j-1})}{\partial\phi_{p,j+k}}, \qquad k=-1,0,1. \tag{3.7b}$$

We introduce the notation

$$\Delta_j x \equiv x_{j+1/2}-x_{j-1/2},$$
$$\Delta_j\psi \equiv \psi_{j+1}-\psi_j$$

and the function $s(z):\mathbb{R}\to\mathbb{R}$ by

$$s(z)\equiv\frac{z}{e^z-1}. \tag{3.8}$$

By straightforward computation it can be verified that the following equalities hold:

$$stp(j,-1)=-\alpha\bar{\mu}_p \exp(\alpha(\phi_{p,j-1}-\psi_j))\frac{s(-\alpha\Delta_{j-1}\psi)}{\Delta_{j-1}x}, \tag{3.9a}$$

$$stp(j,0)=\alpha\bar{\mu}_p \exp(\alpha(\phi_{p,j}-\psi_j))\left\{\frac{s(-\alpha\Delta_{j-1}\psi)}{\Delta_{j-1}x}+\frac{s(\alpha\Delta_j\psi)}{\Delta_j x}\right\}, \tag{3.9b}$$

$$stp(j,+1)=-\alpha\bar{\mu}_p \exp(\alpha(\phi_{p,j+1}-\psi_j))\frac{s(\alpha\Delta_j\psi)}{\Delta_j x}, \tag{3.9c}$$

and

$$stp(j,0)=-(stp(j,-1)+stp(j,+1))+\alpha(-J_{p,j}+J_{p,j-1}). \tag{3.9d}$$

Because $s(z)>0$ for all z, it follows that

$$stp(j,-1)<0, \qquad stp(j,0)>0, \qquad stp(j,+1)<0, \tag{3.10}$$

so the ϕ_p-stencils correspond with an $\mathcal{L}$-matrix. Further, at the exact discrete solution, i.e., when $-J_{p,j}+J_{p,j-1}=0$ is satisfied, it follows from (3.9d) that

$$stp(j,0)=-(stp(j,-1)+stp(j,+1)),$$

so then the $\mathscr{L}$-matrix also possesses weak diagonal dominance (provided there is at least one stencil corresponding with a Dirichlet boundary condition, see [15]). However, in the middle of some iterative process to determine the solution, we may well have negative residuals so that (3.9d) implies the loss of diagonal dominance. Therefore ill-conditioning and numerical difficulties can be expected.

4. The Newton method and expedients. An obvious way of solving the set of nonlinear equations (3.6) is application of the Newton method. Because the Newton method is not globally convergent and the operator $\mathcal{M}_h$ is strongly nonlinear in the variables (ψ, ϕ_n, ϕ_p) we use two additional tools which are considered subsequently in this section:

(1) Correction transformation.

(2) Smoothing of the Newton-iterates.

It turns out that these expedients make the Newton method well applicable. Other modifications of the Newton method including inexact line searches and related techniques have been found to be reliable elsewhere (see [4]).

In two or more space dimensions direct application of the Newton method to (3.6) would involve large storage requirements and the solution of large linear systems. If well designed, a nonlinear multigrid algorithm holds out a prospect of both a computational complexity which is linear in the number of gridpoints and low storage requirements even for the case of two or more space dimensions. Therefore we want to apply the Newton method only for very coarse grids and we restrict the use of the Newton method as a *coarsest grid solver* for multigrid methods (§ 5).

4.1. Correction transformation. The correction transformation introduced by Schilders [10] is a device to transform the Newton-correction $(d\psi, d\phi_n, d\phi_p)$, computed by linearisation with respect to (ψ, ϕ_n, ϕ_p), into the correction for these very variables that would be obtained if linearisation were applied with respect to (ψ, n, p). Because the system in terms of (ψ, n, p) is much less nonlinear, a much better convergence behavior of the Newton method can be expected. By performing the calculations in terms of (ψ, ϕ_n, ϕ_p) and applying a transformation afterwards, we avoid complications due to the extremely wide range of values of n and p. In this way we take advantage of the benefits of both variable sets [9], [10], [14].

4.2. Smoothing. In § 4.1 we pointed out a technique to improve the global convergence behavior of the Newton method. Even yet difficulties are encountered when we apply the improved Newton method. As an example consider Fig. 3 which shows subsequent Newton iterates for case 12 starting from the solution for case 11.

The dips in the iterates are attended with very small pivot numbers while solving the linear systems. Section 3.3 explains the ill-conditioning whenever there is a large residual somewhere. Artificially increasing the main diagonal of the Jacobian turned out to be not efficient. Simply cutting off the correction at certain points is hardly justifiable because of lack of a more or less general criterion to do so. A more appropriate way of handling the phenomenon sketched above is to apply relaxation or smoothing sweeps at the beginning of the Newton process [9]. As a smoother the collective symmetric Gauss–Seidel relaxation (CSGS) can be used. It is called collective because at each box we solve collectively the three nonlinear equations which arise (employing Newton's method).

We will present here some numerical results to show the effect of smoothing. The grid is more or less uniform and satisfies $x_{N/4} = B$. The set of voltages $\{(V_E, V_C)\}$ for which a solution is required is defined in Table 4.1. For each case > 0 the solution of the previous case serves as a starting solution; in case 0 we start with $\phi_{n,j} = \phi_{p,j} = 0$,

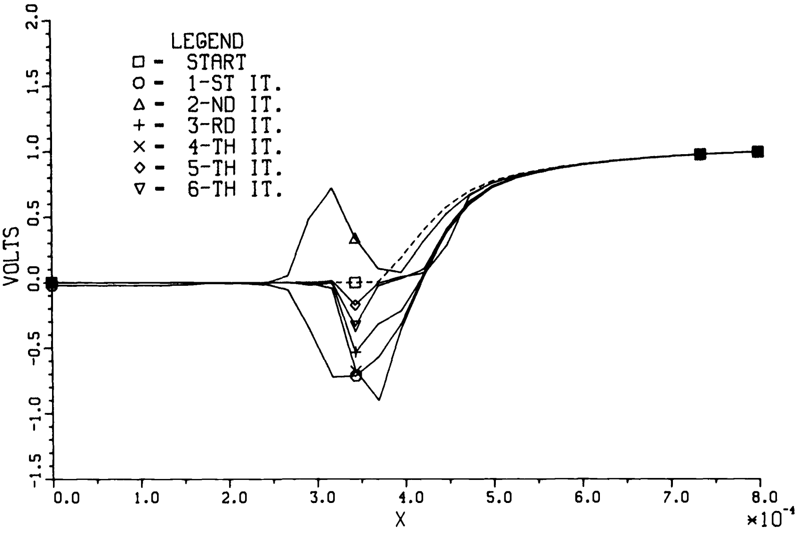


FIG. 3. *Subsequent Newton iterates of* ϕ_p.

TABLE 4.1

Subsequent voltages at the emitter and collector for which a solution is required.

Case	V_E	V_C
0	0	0.0
1	0	0.2
2	0	0.4
3	0	0.6
4	0	0.8
5	0	1
6	−0.2	1
7	−0.4	1
8	−0.6	1
9	−0.7	1
10	−0.8	1
11	−0.85	1
12	−0.9	1
13	−0.95	1
14	−1	1

for all j, and ψ is determined by assuming space charge neutrality. We use the correction transformation. For the solution of the linear systems we apply rowscaling followed by rowpivotting. Table 4.2 shows the number of Newton sweeps required to reach a correction with absnorm$<10^{-12}$, and the smallest pivot number encountered during the solution process. Table 4.2 also contains the results for the case when in addition a CSGS sweep is applied each time after a Newton-correction for which the infinity norm of the correction was larger than 0.1. This method will henceforth be referred to as Newton-CSGS. We observe that in the difficult cases 11–14 the application of smoothing sweeps has a positive effect on the efficiency and robustness of the Newton method. When smoothing is applied the smallest pivot numbers encountered keep a substantial distance from zero which shows that then the Jacobians generated within

TABLE 4.2
Number of Newton and CSGS *sweeps used, and smallest pivot numbers*; $N=32$.

	No smoothing applied		Smoothing applied		
Case	Newton sweeps	Smallest pivot number	Newton sweeps	CSGS sweeps	Smallest pivot number
0	6	$3_{10}-3$	4	1	$5_{10}-1$
1	5	$2_{10}-4$	5	2	$5_{10}-1$
2	6	$1_{10}-9$	5	2	$5_{10}-1$
3	5	$2_{10}-7$	5	2	$5_{10}-1$
4	5	$1_{10}-7$	5	2	$3_{10}-1$
5	5	$4_{10}-5$	5	2	$3_{10}-1$
6	5	$3_{10}-3$	5	2	$2_{10}-1$
7	5	$3_{10}-3$	5	2	$2_{10}-1$
8	5	$2_{10}-4$	5	2	$2_{10}-1$
9	5	$4_{10}-7$	5	2	$3_{10}-1$
10	6	$1_{10}-7$	6	3	$3_{10}-1$
11	9	$4_{10}-9$	7	3	$2_{10}-1$
12	15	$5_{10}-13$	8	4	$4_{10}-2$
13	13	$2_{10}-11$	7	3	$1_{10}-1$
14	10	$5_{10}-10$	7	3	$1_{10}-1$

the Newton method are far from being singular (e.g., compare to case 12 in Table 4.2) and therefore no large dips in the Newton-corrections do occur. Experiments for $N=16$, 64, 128 show results similar to Table 4.2.

5. The multigrid method. More advanced ways of solving a set of nonlinear equations are the full approximation scheme (FAS) [5], and the nonlinear multigrid method (NMGM) [7]. Both *multigrid methods* are very similar although the NMGM is more general. The multigrid method has already found many specific applications in the fields of elliptic, parabolic, and hyperbolic equations and integral equations as well. Recently, also in the field of semiconductor equations research on multigrid methods has been initiated ([1], [2], [6], [9], [13]). If well applied, a multigrid method can be optimal in the sense that the rate of convergence is independent of the meshsize. An important advantage of the FAS/NMGM method is that no large linear systems need to be stored and solved. The subsequent stages of a usual FAS method, applied to (3.6), are

(1) Apply p nonlinear relaxation sweeps; thus we get an approximation q_h of the solution which has a *smooth* residual $d_h \equiv f_h - \mathcal{M}_h(q_h)$.

(2) Transfer q_h and d_h from Ω_h to a coarser grid Ω_H by means of the respective restriction operators R_H and $\bar{R}_H$.

(3) Solve (approximately) on Ω_H the equation $\mathcal{M}_H(q_H) = \mathcal{M}_H(R_H q_h) + \bar{R}_H d_h$.

(4) Interpolate the correction, computed on Ω_H, onto Ω_h and add the correction to q_h.

(5) Apply q nonlinear relaxation sweeps.

The combination of stages 2, 3, and 4 is called the coarse grid correction (CGC). Stage 3 may be obtained by applying a number of σ FAS cycles on the coarser grid. In this way a recursive procedure is obtained in which a sequence of increasingly coarser grids is used. In this paper we use $p=q=\sigma=1$ throughout. In the subsections to come we will define precisely the coarse grid correction and the grid transfer operators involved. In § 6 a significant improvement of the CGC will be introduced. It consists of a solution-dependent adjustment of the restriction of the residual d_h.

5.1. Nested boxes. Let a coarse grid Ω_H, a discretization of Ω, be given by the set of boxes $\{B_{H,j}\}_{j=1(1)N}$. From Ω_H we construct the next finer grid $\Omega_h=\{B_{h,j}\}_{j=1(1)2N}$ by division of each $B_{H,j}$ into two disjoint boxes $B_{h,2j-1}$ and $B_{h,2j}$. By repetition we obtain thus a sequence of increasingly finer grids. By definition all boxes are nested. Of course, the corresponding dual boxes are not nested. For all our numerical experiments in this paper we assume in addition that $B_{h,2j-1}$ and $B_{h,2j}$ have equal size.

5.2. Restriction operators. For the problem (3.2) on Ω, let S denote the domain and V the range of nonlinear operator $\mathcal{M}$. For each discretization on Ω_h, we have the spaces S_h and V_h, the discrete analogues of S and V.

Let the restriction operator for right-hand side functions

$$\bar{R}_h : V \to V_h \tag{5.1a}$$

be defined by

$$\bar{R}_h f = f_h, \tag{5.1b}$$

$$f_{h,j}=\int_{B_{h,j}} f\,d\Omega, \qquad \forall j \text{ at } \Omega_h. \tag{5.1c}$$

It follows for the next coarser grid that

$$(\bar{R}_H f)_j=(\bar{R}_h f)_{2j-1}+(\bar{R}_h f)_{2j}, \qquad \forall j \text{ at } \Omega_H. \tag{5.2}$$

By this equality, $\bar{R}_H$ can be defined also on V_h:

$$\bar{R}_H : V_h \to V_H, \tag{5.3a}$$

$$(\bar{R}_H f_h)_j = f_{h,2j-1}+f_{h,2j}, \qquad \forall j \text{ at } \Omega_H. \tag{5.3b}$$

The restriction operator for solutions

$$R_h : S \to S_h \tag{5.4}$$

may be defined by the well-known full weighting operator [5].

5.3. Prolongation/interpolation. A prolongation transfers a solution from a coarse grid to a finer one:

$$P_h : S_H \to S_h. \tag{5.5}$$

A common and simple choice for the prolongation should be linear interpolation. However, two objections against this choice do arise. Firstly, by the use of linear interpolation it is implicitly assumed that the solution behaves like a smooth function on Ω_H. Because of the exponential behavior of the solution in some areas, this is only true on an unfeasibly fine grid. Secondly, linear interpolation does not satisfy here the so-called Galerkin condition

$$\bar{R}_H \mathcal{M}_h(P_h s_H)=\mathcal{M}_H(s_H), \tag{5.6}$$

which is a condition that ascertains the reduction of low frequency components in the residual after a CGC. Hemker [9] has introduced a prolongation which is based on the assumption of smoothness of fluxes, and which satisfies (5.6) for the simplified case that all S_j and R_j are zero, see (3.4). Here, we use the same assumption but we choose a short and convenient formulation in order to handle also the situation near the inner boundary point B. Figure 4 depicts how the dual box $[L, R]$ is divided into the boxes $[L, M]$ and $[M, R]$.

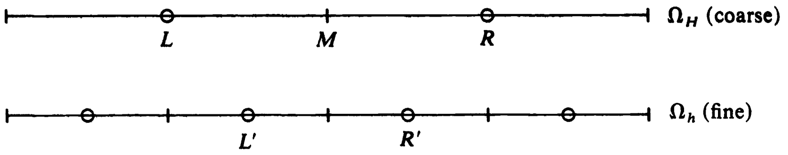


FIG. 4. *Staggering of a coarse and fine grid.*

The assumption reads that J_ψ, J_n, J_p are constant on $[L, R]$. Given the values of the variables (ψ, ϕ_n, ϕ_p) at L and R we wish to compute the values at L' and R'. From (3.5a) it follows that $\psi|_{L'}$ and $\psi|_{R'}$ can be computed by linear interpolation. For ϕ_p we first determine the value at the wall M. If we write

$$\Delta\psi = \psi|_R - \psi|_L, \qquad \Delta\phi_p = \phi_p|_R - \phi_p|_L \tag{5.7}$$

then we derive that

```
$\phi_p|_M =$
    if $\Delta\psi < 0$
    then if $\frac{\Delta\psi}{2} - \Delta\phi_p < 0$
            then $\phi_p|_R + z\left(\frac{\Delta\psi}{2}, \frac{\Delta\psi}{2} - \Delta\phi_p\right)$
            else $\phi_p|_L + \frac{\Delta\psi}{2} + z\left(\frac{\Delta\psi}{2}, -\frac{\Delta\psi}{2} + \Delta\phi_p\right)$
(5.8)       end if
    else if $\frac{\Delta\psi}{2} - \Delta\phi_p < 0$
            then $\phi_p|_R - \frac{\Delta\psi}{2} + z\left(-\frac{\Delta\psi}{2}, \frac{\Delta\psi}{2} - \Delta\phi_p\right)$
            else $\phi_p|_L + z\left(-\frac{\Delta\psi}{2}, -\frac{\Delta\psi}{2} + \Delta\phi_p\right)$
            end if
    end if
```

where the function $z: \mathbb{R}^2 \to \mathbb{R}$ is defined by

$$z(u, v) = \frac{1}{\alpha} \log\left(\frac{\exp(\alpha v) + 1}{\exp(\alpha u) + 1}\right). \tag{5.9}$$

Note that in (5.8) the function z is used with only nonpositive arguments and

$$|z(u, v)| \leqq \frac{\log(2)}{\alpha} \quad \text{for } u \leqq 0, v \leqq 0. \tag{5.10}$$

By repeating the interpolation procedure, we can compute $\phi_p|_{L'}$ from $\phi_p|_L$ and $\phi_p|_M$, and $\phi_p|_{R'}$ from $\phi_p|_M$ and $\phi_p|_R$. In the particular case that the wall M is the basis B, we do not first determine $\phi_p|_M$ by interpolation, but simply state that

$$\phi_p|_M \equiv \phi_p|_B = V_B. \tag{5.11}$$

Analogously, we can derive a formula for $\phi_n|_M$.

5.4. Coarse grid correction. Let q_h^{old} and q_H^{old} be given approximations to the solution on Ω_h and Ω_H, respectively. The CGC is defined by

$$\text{(5.12a)} \qquad \textit{compute } d_H = \bar{R}_H(f_h - \mathcal{M}_h(q_h^{old})),$$

$$\text{(5.12b)} \qquad \textit{solve } \mathcal{M}_H(q_H^{new}) = \mathcal{M}_H(q_H^{old}) + d_H,$$

$$\text{(5.12c)} \qquad \textit{compute } q_h^{new} = q_h^{old} + (P_h q_H^{new} - P_h q_H^{old}),$$

where $\bar{R}_H$ and P_h are the grid transfer operators defined in the previous subsections. Note that q_H^{new} in (5.12b) may be approximated by applying a number of σ FAS cycles on the grid Ω_H with q_H^{old} as an initial approximation. The approximation q_H^{old} may be given by means of full weighting:

$$\text{(5.13a)} \qquad q_H^{old} = R_H q_h^{old},$$

$$\text{(5.13b)} \qquad q_{H,j}^{old} = \tfrac{1}{2} q_{h,2j-1}^{old} + \tfrac{1}{2} q_{h,2j}^{old}, \qquad \forall j \text{ at } \Omega_H.$$

Another possibility is to take q_H^{old} equal to q_H^{new} obtained from the last of previous CGCs. The solution efficiency of many nonlinear problems is not influenced by either choice of q_H^{old}. In our case however it is (see § 7).

5.5. Full multigrid. The full multigrid (FMG) algorithm provides the efficient construction of an initial approximation to the solution on a fine grid, once a solution on a coarse grid has been computed [5], [7]. Let Ω_{coarse} be the coarsest grid and Ω_{fine} be the finest one. Intermediate grids are denoted by $l \in \mathbb{N}$. Operators and grid functions now have l as a subscript instead of h or H. Here we introduce an improvement of the usual FMG in a quasi-Algol description.

```
        procedure BOX-FMG ($'\mathcal{M}_{fine}(q_{fine}) = f_{fine}'$, input: $f_{fine}$, output: $q_{fine}$)
        begin
        (1)  for $l$ from fine − 1 by −1 to coarse
        (2)  do $f_l = \bar{R}_l f_{l+1}$
        (3)  end do
        (4)  SOLVE ($'\mathcal{M}_{coarse}(q_{coarse}) = f_{coarse}'$, input: $f_{coarse}$, output: $q_{coarse}$)
(5.14)  (5)  for $l$ from coarse + 1 to fine
        (6)  do $q_l = P_l q_{-1}$
        (7)     to $\gamma$
        (8)     do FAS ($'\mathcal{M}_l(q_l) = f_l'$, input: $f_l$, in/output: $q_l$)
        (9)     end do
        (10) end do
        end procedure
```

where $\bar{R}_l$ is defined by (5.3). The improvement is in the lines (1)-(3) of the procedure. The grid function f_l is independent of q_l; the components represent

$$\text{(5.15)} \qquad f_{l,j} = \int_{B_j} D \, d\Omega,$$

i.e., the dope function integrated over box B_j. By means of (1)-(3) we compute the integral as a Riemann sum over a larger number of subintervals. This is more accurate because D is a rapidly varying function. In the numerical experiments to come we use $\gamma = 1$ throughout. For SOLVE ( ) we use the techniques of § 4.

6. Adaptation of the coarse grid correction. Hemker [9] successfully applied box centered multigrid FAS iteration to the forward and the reverse biased diode problem. A key feature in his application is the prolongation based on locally constant fluxes.

This prolongation has been reformulated and made suitable for the transistor problem in § 5. Application of the same MG algorithm to the transistor problem gives rise to a complication in the CGC due to drastically varying problem coefficients. This complication and possible remedies are the topics of this section.

6.1. Improper solution transfer. The first attempt of applying multigrid to our specific problem was done by employing BOX-FMG with only two grids. The coarse grid problem (5.12b), within the CGC of FAS, was to be solved up to machine accuracy by means of Newton-CSGS. For several cases of our test problem it turned out that the two-grid algorithm gets stuck precisely at stage (5.12b) of the CGC. This is remarkable because Newton-CSGS was shown in § 4.2 to be successful for $\mathcal{M}_H(q_H)=f_H$ even for rather coarse grids. Apparently f_H is within an appropriate range of $\mathcal{M}_H$ while the right-hand side of (5.12b) may be outside such a proper range of $\mathcal{M}_H$. The computational difficulty occurs in CSGS on the coarse grid exactly where one or more of the three solution components depicts a steep gradient. Consider two adjacent boxes B_L^h and B_R^h on the fine grid which together constitute a box B^H on the coarse grid. Because of the steep gradient it may well occur that the problem coefficients, i.e., the entries of the Jacobian of $\mathcal{M}_h$, show a quite different order of magnitude on B_L^h and B_R^h, respectively. Grid function d_H, the restriction of the residual, is dominated by the fine grid box with the large coefficients. On the other hand, the operator $\mathcal{M}_H$ is generated by the particular choice of q_H^{old}. This particular choice may be full weighting applied to q_h^{old}, or the last q_H available, etc. Because of the steep gradient in q_h^{old} there is a large range of possible values for q_H^{old} at B^H. Depending on the choice of q_H^{old} the operator $\mathcal{M}_H$ may have either large or small coefficients at box B^H due to the exponential behavior of the entries in the Jacobian as a function of the solution. In the case of small coefficients, the right-hand side of (5.12b) may become out of the appropriate range for $\mathcal{M}_H$ (d_H does not depend on the particular choice of q_H^{old}) and the two-grid algorithm gets stuck.

We will now confirm the foregoing by considering our discretized problem in more detail. Consider the center of the ϕ_p-stencil given by (3.9b) and let us suppose that ψ is monotonous on $[x_{j-3/2}, x_{j-1/2}]$; then either $s(-\alpha\Delta_{j-1}\psi)\geqq 1$ or $s(\alpha\Delta_j\psi)\geqq 1$. If both $|\Delta_{j-1}\psi|$ and $|\Delta_j\psi|$ are sufficiently small then $stp(j,0)$ is approximated by

$$stp(j,0)\approx \alpha\bar{\mu}_p \exp\left(\alpha(\phi_{p,j}-\psi_j)\right)\cdot\left(\frac{1}{\Delta_{j-1}x}+\frac{1}{\Delta_j x}\right).$$

If both $|\Delta_{j-1}\psi|$ and $|\Delta_j\psi|$ are sufficiently large then $stp(j,0)$ is approximated by

$$stp(j,0)\approx \alpha\bar{\mu}_p \exp\left(\alpha(\phi_{p,j}-\psi_j)\right)\cdot\begin{cases}\alpha\dfrac{\Delta_{j-1}\psi}{\Delta_{j-1}x} & \text{if} \quad \Delta_{j-1}\psi\geqq 0,\\ -\alpha\dfrac{\Delta_j\psi}{\Delta_j x} & \text{if} \quad \Delta_j\psi\leqq 0.\end{cases}$$

These approximations show that indeed the ϕ_p-stencil is extremely sensitive to the difference $(\phi_{p,j}-\psi_j)$. Hence the ϕ_p-stencil on the coarse grid is sensitive to how $\phi_{p,j}$ and ψ_j on the coarse grid are determined from their counterparts on the fine grid. If q_H^{old} is determined by applying full weighting (linear interpolation) to q_h^{old} then

$$stp(j/2,0)\approx \exp\left(-\frac{\alpha}{2}|\Delta_j(\phi_p-\psi)|\right)\cdot\max\{stp(j-1,0), stp(j,0)\}$$

where $stp(j-1,0)$, $stp(j,0)$ (j even) are defined at the fine grid Ω_h and $stp(j/2)$ at the coarse grid Ω_H. If $(\phi_p-\psi)$ shows a steep gradient then indeed

$$stp(j/2)\ll \max\{stp(j-1,0), stp(j,0)\}.$$

Note. The possible occurrence of the above sketched phenomenon has already been noted (for general nonlinear problems) by Brandt [5, p. 279], where he discusses how the transferred solution (i.e., q_H^{old}) implicitly determines the problem coefficients on the coarse grid.

6.2. Possible remedies. Let L and R be the centers of the two adjacent boxes B_L^h and B_R^h on the fine grid Ω_h which together constitute a coarse grid box B_M^H with center M on the coarse grid Ω_H (see Fig. 5).

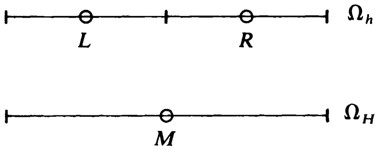


FIG. 5. *Nested boxes.*

Let us assume that $\partial\psi/\partial x = c$ is constant on $B_L^h \cup B_R^h$. The centers of the ϕ_p-stencils at L, R are then determined by the coefficients $a_L^h \equiv \exp(\alpha(\phi_p - \psi)|_L)\bar{c}$, $a_R^h \equiv \exp(\alpha(\phi_p - \psi)|_R)\bar{c}$, respectively, and the center of the ϕ_p-stencil at M by $a_M^H \equiv \exp(\alpha(\phi_p - \psi)|_M)\bar{c}$ with $\bar{c} = \alpha^2\bar{\mu}_p|c|$ (see § 6.1). Let $\Delta_M(\phi_p - \psi)$ denote the variation $\Delta_M(\phi_p - \psi) = (\phi_p - \psi)|_R - (\phi_p - \psi)|_L$. The solution at M on the coarse grid somehow relates to the solution at L and R on the fine grid (for instance by means of the full weighting restriction). If $\Delta_M(\phi_p - \psi)$ is small (a smooth solution) then obviously a_M^H does not differ much from either a_L^h or a_R^h. If $\Delta_M(\phi_p - \psi)$ is large (a steep gradient in the solution) then a_M^H may differ orders of magnitude from both a_L^h and a_R^h, and therefore the MG algorithm may get stuck as was pointed out in the previous subsection. A radical remedy to meet this situation is to prevent $\Delta_M(\phi_p - \psi)$ from getting large, i.e., to introduce local refinement of the mesh just where the solution has a large variation $\Delta_M(\phi_p - \psi)$, e.g., by means of equidistributing the variation. However, we want to be able to find solutions without much refinement, in order to apply coarse grids in our MG algorithm. Besides, a solution without much resolution can serve as a guide for where a local mesh refinement should take place. For these reasons we resort to another remedy. Let us consider the CGC (5.12). Let $d_h(L)$, $d_h(R)$ be the residuals at L, R (e.g., for the third equation (3.4c) only). At M the difference between q_H^{new} and q_H^{old} may have the order of magnitude $d_H(M)/a_M^H$ with $d_H(M) = d_h(L) + d_h(R)$. Because of (5.12c) at either L, R, or both L and R a correction with order of magnitude $d_H(M)/a_M^H$ is added to the solution q_h^{old}. Assume that (because of a steep gradient in the solution) the inequality

$$a_M^H \ll \max\{a_L^h, a_R^h\}$$

holds. Therefore

$$d_H(M)/a_M^H \gg (d_h(L) + d_h(R))/\max\{a_L^h, a_R^h\},$$

which implies that the correction that will be transferred to the fine grid becomes far too large and the solution q_h^{old} gets spoiled. A way to prevent this situation is to multiply the restricted residual d_H with

$$\theta_M^H \equiv \frac{a_M^H}{\max\{a_L^h, a_R^h\}}, \qquad 0 < \theta_M^H \leqq 1, \tag{6.1}$$

at each center M. For a smooth part of the solution this fraction will be near one, for a rapidly varying part of the solution it will be near zero, so that the solution q_h^{old} will be preserved. The foregoing is the motivation for the following modification of the FAS algorithm (MFAS) using the notation of § 5.5:

```
       Procedure MFAS ($'\mathcal{M}_l(q_l)=f_l'$, input: $f_l$, in/output: $q_l$)
       begin
       (1)  If $l=$coarse
       (2)  then SOLVE ($'\mathcal{M}_{coarse}(q_{coarse})=f_{coarse}'$, input : $f_{coarse}$, in/output: $q_{coarse}$)
       (3)  else RELAX ($'\mathcal{M}_l(q_l)=f_l'$, input: $f_l$, in/output: $q_l$)
       (4)       $d_{l-1}:=\bar{R}_{l-1}(f_l-\mathcal{M}_l(q_l))$
       (5)       $q_{l-1}:=R_{l-1}q_l$   (optional !)
       (6)       $d_{l-1}:=\Theta_{l-1}(\mathcal{M}_{l-1},\mathcal{M}_l)d_{l-1}$
(6.2)  (7)       $d_{l-1}:=d_{l-1}+\mathcal{M}_{l-1}(q_{l-1})$
       (8)       $s_{l-1}:=q_{l-1}$
       (9)       to $\sigma$
       (10)      do MFAS ($'\mathcal{M}_{l-1}(q_{l-1})=d_{l-1}'$, input: $d_{l-1}$, in/output: $q_{l-1}$)
       (11)      end do
       (12)      $q_l:=q_l+P_lq_{l-1}-P_ls_{l-1}$
       (13)      RELAX ($'\mathcal{M}_l(q_l)=f_l'$, input: $f_l$, in/output: $q_l$)
       (14) end if
       end procedure
```

The modification is in line (6). Here Θ_{l-1} represents a diagonal matrix $\mathbb{R}^{3N(\Omega_{l-1})}\rightarrow\mathbb{R}^{3N(\Omega_{l-1})}$ ($N(\Omega_{l-1})$ denotes the number of boxes at Ω_{l-1}). It is defined by

$$\Theta_{l-1}d_{l-1}=(\boldsymbol{\theta}_{l-1,1}d_{l-1,1},\cdots,\boldsymbol{\theta}_{l-1,j}d_{l-1,j},\cdots,\boldsymbol{\theta}_{l-1,N(\Omega_{l-1})}d_{l-1,N(\Omega_{l-1})})^T$$

with $d_{l-1,j}\in\mathbb{R}^3$, $\theta_{l-1,j}:\mathbb{R}^3\rightarrow\mathbb{R}^3$ and

$$\boldsymbol{\theta}_{l-1,j}\equiv\begin{pmatrix}\theta_{l-1,j,1} & 0 & 0\\ 0 & \theta_{l-1,j,2} & 0\\ 0 & 0 & \theta_{l-1,j,3}\end{pmatrix}$$

$$(\theta_{l-1,j,k}\in\mathbb{R},\ k=1,2,3).$$

For our particular semiconductor problem the $\theta_{l-1,j,k}$ are defined by:

(6.3a1) $$\theta_{l-1,j,1}=1,$$

(6.3a2) $$\theta_{l-1,j,2}=\min\{2\eta_{l-1,j},1\},\qquad \eta_{l-1,j}\in\mathbb{R},$$

(6.3a3) $$\theta_{l-1,j,3}=\min\{2\xi_{l-1,j},1\},\qquad \xi_{l-1,j}\in\mathbb{R},$$

where

(6.3b) $$\begin{aligned}\eta_{l-1,j}&\equiv\exp(\alpha(\psi_j^{l-1}-\phi_{n,j}^{l-1}))\Big/\max_{i=0,-1}\exp(\alpha(\psi_{2j+i}^{l}-\phi_{n,2j+i}^{l})),\\ \xi_{l-1,j}&\equiv\exp(\alpha(\phi_{p,j}^{l-1}-\psi_j^{l-1}))\Big/\max_{i=0,-1}\exp(\alpha(\phi_{p,2j+i}^{l}-\psi_{2j+i}^{l})).\end{aligned}$$

The superscripts $l-1, l$ refer to Ω_{l-1}, Ω_l, respectively.

The first component of the restricted residual does not need to be adjusted. The definition originates from the evaluation of expression (6.1). By means of (6.3a2)-(6.3a3) the numbers $\theta_{l-1,j,2}$ and $\theta_{l-1,j,3}$ are rounded off upwards to 1 when $\eta_{l-1,j}$, $\xi_{l-1,j}$ are $\geqq\frac{1}{2}$. Summarizing, we observe the following from (6.2)-(6.3):

(i) Where q_l is smooth, d_{l-1} will not be suppressed.

(ii) Where q_l depicts a steep gradient, d_{l-1} may be strongly suppressed.

(iii) Let Ω_0 be some fixed grid, then, for $l\rightarrow\infty$, the matrix Θ_{l-1} becomes asymptotically the identity matrix.

(iv) By a proper local mesh refinement the suppression of d_{l-1} will decrease. The performance of the modified FAS algorithm will be shown and discussed in § 7.

In the nonlinear multigrid algorithm as proposed by Hackbusch [7, p. 187], the restricted residual d_{l-1} is divided by a global parameter $s \geqq 1$ and the resulting coarse grid correction is multiplied by s. The division by an appropiate s ensures that the right-hand side of the coarse grid equation is within an appropriate range of the coarse grid operator $\mathcal{M}_{l-1}$. There are two main differences with our approach. First, the same number s is used at each different box. Second, within our class of problems we have to omit the multiplication of the correction by s. Such a multiplication would result in a far too large correction and thereby a dip or peak in the fine grid solution. In recent work of Hackbusch and Reusken [8] a global parameter ψ is proposed by which the coarse grid correction should be damped. For a limited class of problems an appropriate ψ can be computed. Important differences with our approach are the following:

(i) ψ is a damping parameter for the correction, instead of the residual.

(ii) ψ is a global parameter, i.e., the same ψ is used at each different box.

(iii) After sufficient FAS sweeps the damping parameter ψ converges to 1, the parameters $\theta_{l-1,j,2}$, $\theta_{l-1,j,3}$ do not and should not converge to 1.

(iv) The ψ parameter is meant to enlarge the domain of guaranteed convergence on the analogy of the damping parameter in the Newton method; the Θ_{l-1} operator is meant to deal with discrepancies between the operators $\mathcal{M}_{l-1}$ and $\mathcal{M}_l$ due to rapidly varying problem coefficients.

7. Numerical results. In this section we investigate the performance of our nonlinear multigrid algorithm. We focus our attention on the effects of local suppressing of the restricted residual and the choice of the coarse grid solution. The residual norm ($\|\cdot\|_{res}$) that we use is the maximum norm of the scaled residual. At level l the said scaling is done by multiplying the residual at each box with the inverse of the 3 × 3-matrix

$$\left(\frac{\partial \mathcal{M}_l|_{x_{j-1/2}}}{\partial(\psi_j^l, \phi_{n,j}^l, \phi_{p,j}^l)^T}\right).$$

The performance of the MFAS algorithm is shown in Table 7.1. In the heading of the table we use the following abbreviations:

case: see Table 4.1.

q_H^{old}: defined by . . . : see § 5.4.

without Θ: No local suppression of the restricted residual is applied.

with Θ: Local suppression of the restricted residual is applied on all coarser grids.

#MFAS, 10^{-1} red: The average number of MFAS sweeps necessary to obtain an additional reduction factor 10^{-1} of the residual norm after the application of BOX-FMG.

after FMG: The last column shows the scaled norm of the residual, after application of BOX-FMG (see (5.14), $\gamma = 1$). In each case > 0 we obtain a starting approximation of the solution on the coarsest grid by means of continuation and application of Newton-CSGS (see § 4.2).

For Table 7.1 the multigrid procedures are applied with 3 grids, with $N = 16$, 32, 64, respectively. In the event of no convergence the symbol * is written.

We observe that the use of the Θ operator, combined with a proper choice of the coarse grid solution, gives convergence for all cases. In the cases 3-6 the use of the Θ operator is essential for convergence. The use of the Θ operator does not slow down

TABLE 7.1
Performance of MFAS; *use of* 3 *grids*: $N = 16, 32, 64$, *respectively.*

	q_H^{old}: Defined by full weighting		q_H^{old}: Defined by q_H^{new}		
Case	Without Θ #MFAS, 10^{-1} red.	With Θ #MFAS, 10^{-1} red.	Without Θ #MFAS, 10^{-1} red.	With Θ #MFAS, 10^{-1} red.	With Θ After FMG
0	0.80	0.80	0.66	0.66	$4.1_{10}-4$
1	1.07	1.07	0.88	0.73	$8.8_{10}-4$
2	*	1.15	*	0.85	$1.2_{10}-3$
3	*	*	*	1.03	$1.2_{10}-3$
4	*	*	*	1.06	$7.1_{10}-4$
5	*	*	*	0.89	$5.7_{10}-4$
6	*	*	*	0.89	$5.7_{10}-4$
7	*	1.38	*	0.89	$5.7_{10}-4$
8	1.40	1.37	0.89	0.89	$5.6_{10}-4$
9	1.54	1.37	0.87	0.90	$5.7_{10}-4$
10	2.59	2.52	0.88	0.82	$1.1_{10}-3$
11	1.22	1.22	1.25	1.25	$1.3_{10}-3$
12	1.75	1.74	2.01	2.01	$9.1_{10}-4$
13	4.66	4.66	2.19	2.19	$1.9_{10}-3$
14	2.44	2.43	1.76	1.77	$2.5_{10}-3$

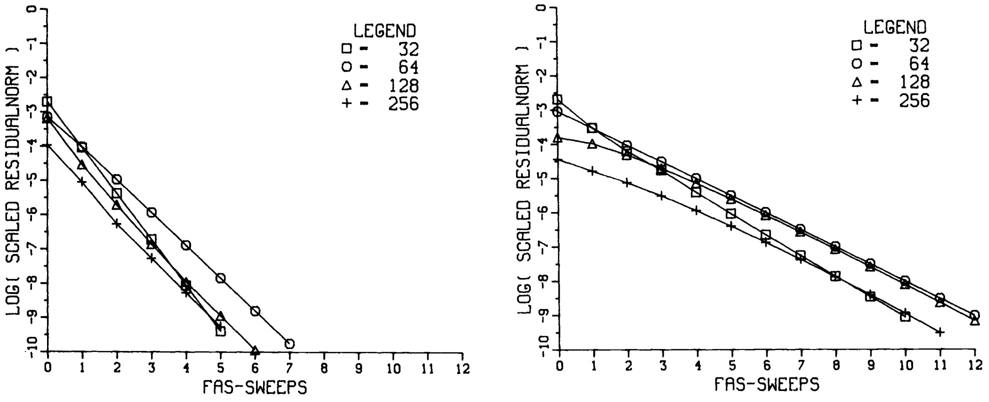
convergence in the cases where it is not needed (cases 0–2 and 7–14). We observe further that apparently the full weighting approximation of the fine grid solution on the coarse grid may be a poor one. Experiments for more and finer grids showed almost identical results for Table 7.1. Further we observe that mere application of BOX-FMG, without further MFAS sweeps, already gives fairly accurate results which may be good enough for practical purposes.

For two typical cases, case 4 and case 12, we investigate the grid-dependence of the multigrid convergence. In Fig. 6 we show the 10-logarithm of the scaled residual norms after subsequent FAS sweeps, starting from the result obtained by BOX-FMG. The coarsest grid contains 16 boxes; for the finest grid we take 32, 64, 128, and 256 boxes, respectively; Θ is applied (without application of Θ case 4 persistently depicts divergence). We observe that the multigrid convergence becomes grid-independent when the meshsize of the finest grid decreases. This indicates that the semiconductor problem has been treated correctly at each multigrid stage. Hereby it is shown that even for the strongly nonlinear (and particularly hard) problem it is possible to compose a multigrid method with optimal multigrid efficiency. Of course, considered in one space dimension only, competitive methods are available. However, the multigrid method as developed in this paper offers several clues for the foundation of an MG algorithm which solves the semiconductor problem also in more space dimensions with a computational complexity that is linear in the number of gridpoints.

In order to give some insight into the behavior of the Θ operator, we show in Fig. 7 the solution components ψ and ϕ_n for case 4 on a 64-grid and a graph of $\theta_{2,j,2}$ (see (6.3a2)).

We observe the typical behavior that $\theta_{2,j,2}$ equals 1 almost everywhere, except for some isolated points.

Fig. 8 shows the electrostatic potential ψ as computed on a grid of 128 boxes.



(a) (b)

FIG. 6. *Multigrid convergence histories*; *the coarsest grid numbers* 16 *boxes*: (a) *case* 4; (b) *case* 12.

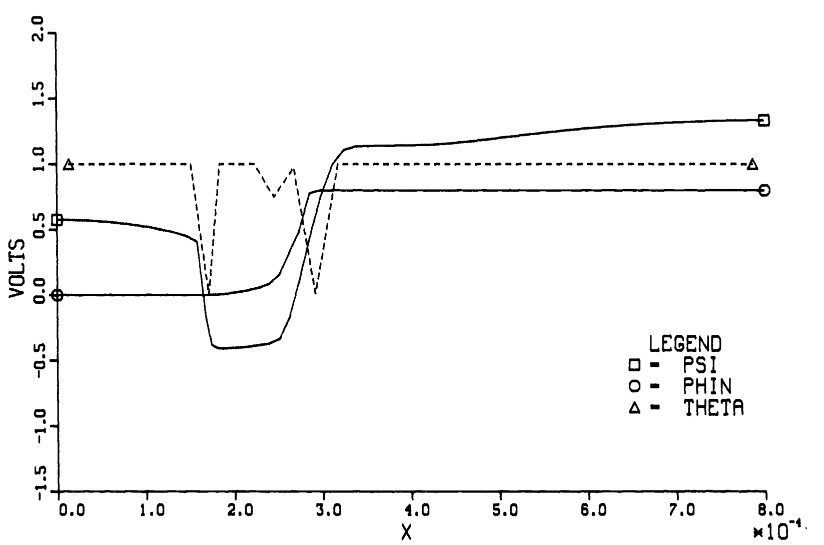


FIG. 7. *The components ψ and ϕ_n of the solution for case* 4 *and the corresponding θ.*

8. Conclusions. We find, by deriving explicit expressions for the entries of the Jacobian, that the linearization of the Scharfetter-Gummel discretization scheme contains a weak spot. When applying full multigrid followed by FAS/NMGM-iterations to our one-dimensional transistor problem, we find a serious lack of robustness which is explained by the strong nonlinearity of the discretized problem. This difficulty is met by adaptation of the coarse grid correction, which looks to be equally applicable for the higher-dimensional case. A proper choice of the coarse grid solutions is of importance too, e.g., the full weighting approximation is not satisfactory. Furnished with the improvements as proposed, we obtain a robust multigrid algorithm with a convergence which is independent of the meshsize.

Acknowledgements. The author wishes to thank Prof. Dr. P. W. Hemker and Prof. Dr. P. Wesseling for their useful comments and Ms. M. Middelberg for typing the manuscript.

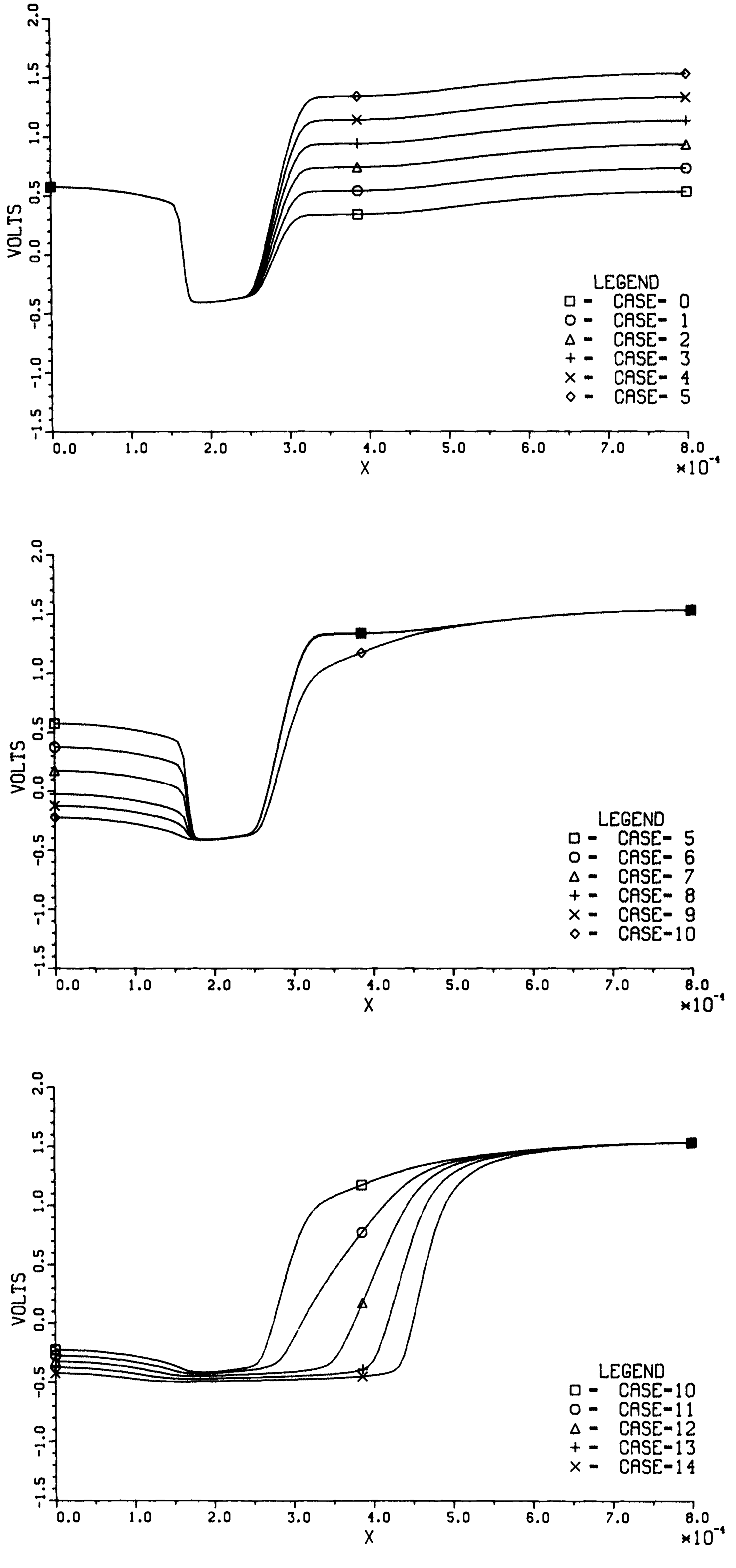


FIG. 8. *The electrostatic potential ψ on a 128-grid.*

REFERENCES

[1] R. E. Bank, J. W. Jerome, and D. J. Rose, *Analytical and numerical aspects of semiconductor device modelling*, Computing Methods in Applied Sciences and Engineering, R. Glowinski and J. L. Lions, eds., North-Holland, Amsterdam, 1982, pp. 593-597.

[2] R. E. Bank and H. D. Mittelmann, *Continuation and multi-grid for nonlinear elliptic systems*, Multigrid Methods II, W. Hackbusch and U. Trottenberg, eds., Springer-Verlag, Berlin, New York, 1986, pp. 23-37.

[3] R. E. Bank and D. J. Rose, *Analysis of a multilevel iterative method for nonlinear finite element equations*, Math. Comp., 39 (1982), pp. 453-465.

[4] R. E. Bank, D. J. Rose, and W. Fichtner, *Numerical methods for semiconductor device simulation*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 416-435.

[5] A. Brandt, *Guide to multigrid development*, Lecture Notes in Mathematics 960, W. Hackbusch and U. Trottenberg, eds., Springer-Verlag, Berlin, 1981, pp. 220-312.

[6] S. P. Gaur and A. Brandt, *Numerical solution of semiconductor transport equations in two dimensions by multi-grid method*, Advances in Computer Methods for Partial Different Equations II, R. Vichnevetsky, ed., IMACS (AICA), New Brunswick, NJ, 1977, pp. 327-329.

[7] W. Hackbusch, *Multi-grid methods and applications*, Springer Series in Computational Mathematics 4, Springer-Verlag, Berlin, 1985.

[8] W. Hackbusch and A. Reusken, *On global multigrid convergence for nonlinear problems*, Robust Multigrid Methods, Notes on Numerical Fluid Dynamics, W. Hackbusch, ed., Vieweg-Verlag, Braunschweig, 1988.

[9] P. W. Hemker, *A nonlinear multigrid method for one-dimensional semiconductor device simulation: The diode*, BAIL V Proceedings of the Fifth International Conference on Boundary and Interior Layers, Shanghai, China, 1988.

[10] S. J. Polak, C. den Heijer, W. H. A. Schilders, and P. A. Markowich, *Semiconductor device modelling from the numerical point of view*, Internat. J. Numer. Methods Engrg., 24 (1987), pp. 763-838.

[11] D. Scharfetter and H. K. Gummel, *Large-signal analysis of a silicon Read diode oscillator*, IEEE Trans. Electron Dev., ED-16, (1969), pp. 64-77.

[12] S. Selberherr, *Analysis and simulation of semiconductor devices*, Springer-Verlag, Berlin, New York, 1984.

[13] A. S. L. Shieh, *Solution of coupled systems of PDEs by the transistorized multi-grid method*, in Proceedings of a Conference on Numerical Solutions of VLSI Devices, Boston, 1984.

[14] B. P. Sommeijer, W. H. Hundsdorfer, C. T. H. Everaars, P. J. van der Houwen, and J. G. Verwer, *A numerical study of a 1D stationary semiconductor model*, Note NM-8702, Dept. of Numerical Mathematics, Centre for Mathematics and Computer Science, Amsterdam, the Netherlands, 1987.

[15] D. M. Young, *Iterative solution of large linear systems*, Academic Press, New York, London, 1971.

[16] P. M. de Zeeuw, *Nonlinear multigrid applied to a 1D stationary semiconductor model*, Report NM-R8905, Dept. of Numerical Mathematics, Centre for Mathematics and Computer Science, Amsterdam, the Netherlands, 1989.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 2, pp. 531-540, March 1992

006

AN ITERATIVE METHOD FOR MATRIX SPECTRAL FACTORIZATION*

T. J. HARRIS† AND J. H. DAVIS‡

Abstract. A computationally efficient algorithm is implemented to factorize a multivariate spectrum at a discrete number of frequencies. This method uses an iterative causal projection procedure to factorize the spectrum. The causal projection is computed using fast Fourier transforms.

Key words. multivariable spectral factorization, Ricatti equation, conjugate periodic function, discrete Hilbert transform, optimal control, attenuation factors

AMS(MOS) subject classifications. 15A09, 49D40, 62M10, 65F05

1. Introduction. The spectral factorization of para-Hermitian matrices arises in prediction theory [20], [24], [25]; linear optimal control theory [6], [13], [19]; and in the construction of approximate inverses for multivariable control [18]. Numerous computational procedures have been proposed. Generally, these may be classified as either parametric or nonparametric methods. In the parametric approach, the matrix to be factored is expressed explicitly in terms of the Laplace or z-transform operator. Constructive methods for factorization may employ (i) symmetric factor extraction [4], [22]; (ii) construction of a state space realization and solution via the Ricatti equation [1], [2], [9], [23]; (iii) an iterative Newton-Raphson solution to a set of bilinear equations [16], [24]; or (iv) decomposition of block Hankel and Toeplitz matrices [19], [21], [26]. In the nonparametric approach, the matricial spectrum at a discrete number of frequencies is factored using a Newton-Raphson iteration as in [24], [25] or an optimal gain iteration as in [6]. These techniques involve a causal projection operator in the iteration, and hence rely on the properties of functions which are analytic and harmonic in and upon the unit disk.

In this paper a computationally efficient algorithm is implemented to factorize the matricial spectrum at a discrete number of frequencies. This method uses an iterative causal projection procedure to factorize the spectrum. To calculate the causal projection in each iteration, it is necessary to numerically approximate a discrete Hilbert transform. This is accomplished with a fast Fourier transform. The use of attenuation factors [12] is investigated to reduce the computational requirements. A benchmark example and an application arising in the control of a pilot scale packed bed reactor are used to illustrate the methodology. The proposed method is also compared to the parametric method proposed in [26] with respect to computation time and solution accuracy.

2. Preliminaries. Consider an $m \times m$ matrix function $H(e^{j\theta})$ defined on the interval $[-\pi, \pi]$ which has the following properties:

(P)
(1) $H(e^{j\theta})$ is Hermitian, i.e., $H(e^{j\theta}) = H^T(e^{-j\theta})$, where ^{T} denotes transpose,
(2) $H(e^{j\theta})$ is positive definite on the interval, i.e., $H(e^{j\theta}) > 0$,
(3) $H(e^{j\theta})$ is in the ring of absolutely convergent Fourier series.

$H(e^{j\theta})$ thus admits a Fourier series expansion of the form

$$H(e^{j\theta}) = \sum_{k=-\infty}^{\infty} H_k e^{j\theta k}. \tag{1}$$

* Received by the editors May 29, 1989; accepted for publication (in revised form) December 6, 1990. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada.

† Department of Chemical Engineering, Queen's University, Kingston, Ontario, Canada, K7L 3N6.
‡ Department of Mathematics and Statistics, Queen's University, Kingston, Ontario, Canada, K7L 3N6.

Since $H(e^{j\theta})$ is Hermitian, the coefficients H_k are real and $H_{-k}=H_k^T$.

A fundamental result is that a function $H(e^{j\theta})$ having properties (P) can be factorized as [10]:

$$H(e^{j\theta})=F(e^{j\theta})WF^*(e^{j\theta}). \tag{2}$$

$F(e^{j\theta})$, known as the left factor, is an $m\times m$ matrix function having a Fourier expansion in nonnegative powers of $e^{j\theta}$,

$$F(e^{j\theta})=\sum_{k=0}^{\infty}F_k e^{j\theta k} \tag{3}$$

with $F_0=I$. Furthermore, the coefficients $\langle F_k\rangle$ are real valued. $F^*(e^{j\theta})$ denotes the conjugate transpose of $F(e^{j\theta})$. W is a positive definite matrix.

$F(e^{j\theta})$ admits an extension to the interior of the unit disk such that

$$F(z)=\sum_{k=0}^{\infty}F_k z^k \tag{4}$$

is analytic in the region $|z|<1$. The generating function $F(z)$ is invertible in the sense that $F(z)$ has all its zeros in z, outside the unit circle. In the case of the convolution algebra approach to the factorization problem, the condition is invertibility in the convolution algebra [10]. Properties (P) are sufficient for $H(e^{j\theta})$ to be characterized as a full rank spectral density function [20]. The factorization of $H(e^{j\theta})$ via equation (2) is commonly known as spectral factorization.

Wilson [24] has developed two iterative procedures for spectral factorization. Both methods are based on a second order Taylor series expansion and subsequent Newton-Raphson iteration. In the first, the coefficients $\langle F_k\rangle$ are determined directly from the coefficients $\langle H_k\rangle$. This algorithm can be applied when the Fourier expansion of $H(z)$ can be represented by truncated series of degree L. F_0 is constrained to be upper triangular, and W is taken as the identity matrix. The method requires the solution of a bilateral polynomial equation. This equation can be transformed to a system of linear equations which require inversion of a matrix of dimension $(Lm^2\times Lm^2)$. Jezek and Kucera [16] demonstrated that a Shur reduction can be used to solve the bilateral polynomial equations. This eliminates the need to invert the aforementioned matrix. Their approach requires the calculation of a determinant of an $m\times m$ polynomial matrix of degree L.

The second approach of Wilson is an iteration for $F(e^{j\theta})$, where θ is evaluated at a number of equispaced points on the unit circle. The coefficients of the generating function are then determined by taking the inverse Fourier transform of $F(e^{j\theta})$. A much more streamlined algorithm than the one proposed by Wilson is obtained, however, by insisting upon the normalized factorization described above. The following iterative procedure is arrived at [6].

Define the sequence of iterates, $F_{n+1}(e^{j\theta})$, $F_n(e^{j\theta})$ via

$$F_{n+1}(e^{j\theta})=W_n^{-1}[F_n^*(e^{j\theta})^{-1}H(e^{j\theta})F_n(e^{j\theta})^{-1}]^+(e^{j\theta})F_n(e^{j\theta}) \tag{5}$$

for $\theta\in[0,2\pi]$. W_n is the constant coefficient in the Laurent expansion of the power series for $F_n^*(e^{j\theta})^{-1}$ $H(e^{j\theta})$ $F_n(e^{j\theta})^{-1}$. The projection operator $[\ ]^+$ of a function $G(z)$ in terms of the Fourier expansion

$$G(z)=\sum_{k=-\infty}^{\infty}G_k z^k,\qquad |z|=1 \tag{6}$$

is

$$[G(z)]^+ = \sum_{k=0}^{\infty} G_k z^k, \qquad |z| \leqq 1. \tag{7}$$

The proposed factorization involves an iteration for $F_n(e^{j\theta})$ and an iteration for W_n. The iteration (5) is initialized with $F_0(e^{j\theta}) = I$. The origin and properties of the iteration (5) are discussed in [6, Appendix A].

The iteration (5) requires calculation of W_n, which is the mean value of the function $G(e^{j\theta}) = F_n^*(e^{j\theta})^{-1} H(e^{j\theta}) F_n(e^{j\theta})^{-1}$. W_n is given by

$$W_n = \frac{1}{2\pi} \oint_0^{2\pi} G(e^{j\theta})\, d\theta. \tag{8}$$

The projection operator must be calculated on $|z| = 1$. $[G(e^{j\theta})]^+(e^{j\theta})$ can be calculated from the formulas of Sokhotskyi [15]

$$[G(e^{j\theta})]^+(e^{j\theta}) = \tfrac{1}{2}[W_n + G(e^{j\theta}) + j\mathbb{H}[G(e^{j\theta})](e^{j\theta})], \tag{9}$$

where $\mathbb{H}[G(e^{j\theta})](e^{j\theta})$ is the Hilbert transform of $G(e^{j\theta})$ and is given by

$$\mathbb{H}[G(e^{j\theta})](e^{j\theta}) = 1/(2\pi j)\mathrm{PV} \oint G(e^{j\theta})(e^{j\tau} + e^{j\theta})/(e^{j\tau} - e^{j\theta})\, d\tau. \tag{10}$$

PV denotes that the integral is to be interpreted in the principal value sense. Equation (10) is recognized as the Cauchy principal value integral of the function $G(z)$, for $|z| = 1$.

The form of the iteration and restrictions on $H(e^{j\theta})$ guarantee that the elements of $G(e^{j\theta})$ are representable by Fourier series. Therefore the (p, q)th element, denoted by $g_{pq}(e^{j\theta})$ admits a Fourier series expansion of the form

$$g_{pq}(e^{j\theta}) = \sum_{k=-\infty}^{\infty} a_{pqk}\, e^{jk\theta}. \tag{11}$$

The representation of the Hilbert transform of $g_{pq}(e^{j\theta})$ in terms of the Fourier expansion for $g_{pq}(e^{j\theta})$ is [8]

$$\mathbb{H}[G(e^{j\theta})](e^{j\theta}) = -j \sum_{k=-\infty}^{\infty} \varepsilon_k a_{pqk}\, e^{jk\theta}, \qquad \varepsilon_k = \begin{cases} 1, & k > 0, \\ 0, & k = 0, \\ -1, & k < 0. \end{cases} \tag{12}$$

3. Numerical procedure. In the frequency domain calculation (8)-(12) are evaluated at $N = 2^s + 1$ equispaced points on the half circle between $[0, \pi]$. These points on the half circle are denoted by θ_r, $r = 0, 1, \cdots, N$. It is not necessary to evaluate $H(e^{j\theta})$ for negative θ since $H(e^{j\theta})$ is Hermitian. The values of $G(e^{j\theta})$ for negative θ are simply the conjugate of those for positive θ.

The function $g_{pq}(e^{j\theta})$ is approximated by the truncated series

$$\tilde{g}_{pq}(e^{j\theta}) = \sum_{k=-N}^{N}{}' \tilde{a}_{pqk}\, e^{jk\theta}. \tag{13}$$

The $'$ denotes that the summation is to be taken with the factor $\frac{1}{2}$ when $k = +/-N$. The circumflex $\sim$ denotes that the quantity is associated with the truncated expansion. $\tilde{a}_{pqk}$ is the approximated Fourier coefficients of $g_{pq}(e^{j\theta})$ [8]. The approximate Hilbert transform, denoted by $\mathbb{H}[\tilde{g}_{pq}(e^{j\theta})](e^{j\theta})$, is given by

$$\mathbb{H}[\tilde{g}_{pq}(e^{j\theta})](e^{j\theta}) = -j \sum_{k=-N}^{N}{}' \varepsilon_k \tilde{a}_{pqk}\, e^{jk\theta}, \qquad \varepsilon_k = \begin{cases} 1, & k > 0, \\ 0, & k = 0, \\ -1, & k < 0. \end{cases} \tag{14}$$

A fast Fourier transform (FFT) can be used to evaluate the Fourier coefficients $\langle \tilde{a}_{pqk} \rangle$, and an inverse transform can be used to evaluate $\tilde{g}_{pq}(e^{j\theta})$ at θ_r, $r = 0, 1, 2, \cdots, N$ [11]. Since $G(e^{j\theta})$ is an even function, the coefficients $\langle \tilde{a}_{pqk} \rangle$ are real valued. It is also straightforward to verify that $\mathbb{H}[\tilde{g}_{qp}(e^{j\theta})](e^{j\theta}) = -[\mathbb{H}[\tilde{g}_{pq}(e^{j\theta})](e^{j\theta})]^*$ for $p \neq q$. This latter identity reduces the computational requirements of the algorithm since it is only necessary to evaluate $m(m+1)/2$ FFTs in each iteration. The proposed FFT solution method is computationally more efficient than the method of Wilson [24] where the principal value integral in (10) is evaluated as an ordinary improper integral using Simpson's rule integration formula.

Remarks. (1) To obtain a factorization of the form $H(e^{j\theta}) = E^*(e^{j\theta}) WE(e^{j\theta})$, the algorithm is applied to $H^*(e^{j\theta})$. $E(e^{j\theta})$, known as the right factor, is calculated as $E(e^{j\theta}) = F^*(e^{j\theta})$.

(2) The algorithm is terminated whenever

$$\max_{\theta} \frac{1}{m} |I - H(e^{j\theta})^{-1}[F_n(e^{j\theta}) W_n F_n^*(e^{j\theta})]|_\infty < \varepsilon \qquad \text{for } \theta = \theta_r = r\pi/2^s, \ r = 0, 1, \cdots, 2^s.$$

(3) A bound on the achievable accuracy is

$$\max_{\theta} \frac{1}{m} |I - H(e^{j\theta})^{-1} H(e^{j\theta})|_\infty \qquad \text{for } \theta = \theta_r = r\pi/2^s, \ r = 0, 1, \cdots, 2^s.$$

(4) Frequently one wants to obtain the spectral factor of $H(z)$, where the solution is given in terms of the generating function. To employ the above procedure, the spectrum $H(e^{j\theta})$ is obtained by evaluating $H(z)$ at $N = 2^s + 1$ points on the unit circle, i.e., $z = \exp(jr\pi/N)$, $r = 0, 1, 2, \cdots, N$. The algorithm described above is implemented. The coefficients of the generating function $\langle F_k \rangle$ are obtained by taking the inverse Fourier transform of $F(e^{j\theta})$.

(5) As indicated in [6, Appendix A], (5) can be used instead of a parametric method to obtain the steady state gain for the Kalman-Bucy filtering problem in state variable form.

4. Error analysis. The primary source of errors in the algorithm arises from truncating the Fourier expansion (14) approximating the Hilbert transform with (12). An estimate of the error in evaluating the Hilbert transform arising from truncation of the Fourier series is [8]

$$(15) \qquad |\mathbb{H}[g_{pq}(e^{j\theta})](e^{j\theta}) - \mathbb{H}[\tilde{g}_{pq}(e^{j\theta})](e^{j\theta})| \leqq 2 \sum_{|k| \geqq N}' |a_{pqk}|.$$

The $'$ denotes that the term with $k = N$ is to be taken with the factor $\frac{1}{2}$. A tighter estimate has been developed in [14]. The truncation error is zero when $g_{pq}(e^{j\theta})$ has a Fourier expansion in fewer than N terms. The truncation error can be reduced by increasing the number of terms used to approximate $g_{pq}(e^{j\theta})$. This increases the computational overhead in the algorithm. An alternate approach is to use the theory of attenuation factors [3], [7], [12]. This approach is equivalent to using either a windowed or smoothed estimate for the Fourier coefficients or a periodic spline to approximate $g_{pq}(e^{j\theta})$. In either case, the result is that ε_k in (14) is a function of frequency and independent of the projected function $G(e^{j\theta})$. For an Euler spline of order σ, ε_k is [3]

$$(16) \qquad \varepsilon_k = \begin{cases} (1 - (k/N)^\sigma), & k > 0, \\ 0, & k = 0, \\ -(1 - (k/N)^\sigma), & k < 0. \end{cases}$$

Error estimates for the conjugate function have been developed in [3], [12]. The error estimates of [3] require that the norm of the derivative of $g_{pq}(e^{j\theta})$ be less than one in magnitude. Gutknecht [12] has shown that attentuation factors based on spline interpolants have certain optimality properties in the sense of minimizing a norm of $\mathbb{H}[g_{pq}(e^{j\theta})](e^{j\theta})-\mathbb{H}[\tilde{g}_{pq}(e^{j\theta})](e^{j\theta})$. However, the optimality of the attenuation factors based on spline interpolants does not ensure that the error is minimized at the interpolation points. The effectiveness of attenuation factors is examined in the next section.

Chawla and Ramakrishnan [5] developed an N point quadrature formula for approximating Hilbert transforms that is exact if the Fourier expansion for $g(e^{j\theta})$ has fewer than $2N$ terms. It is straightforward to show that their procedure is equivalent to calculating the approximated Hilbert transform using an Euler spline of order one, and correcting the expression in (12) by adding the term $-[dg(e^{j\theta})/d\theta]/N$. The derivative of $g(e^{j\theta})$ is not known. It may be approximated by a central difference formula. Alternatively, this derivative can be approximated from the Fourier expansion for $g(e^{j\theta})$. If the latter approach is taken, it can be shown that the approximate Hilbert transform is given by (13).

5. Numerical examples. The first example is taken from [24]. The dimension is $m=2$. The calculations are performed at $N=1+2^s$ points between $[0, \pi]$ with s taking on values between 3 and 7. Attenuation factors were calculated using linear interpolation and a cubic interpolation polynomial [12]. These attenuation factors are shown in Fig. 1. The quadrature procedure of [5] was also implemented with a central difference approximation for the derivative.

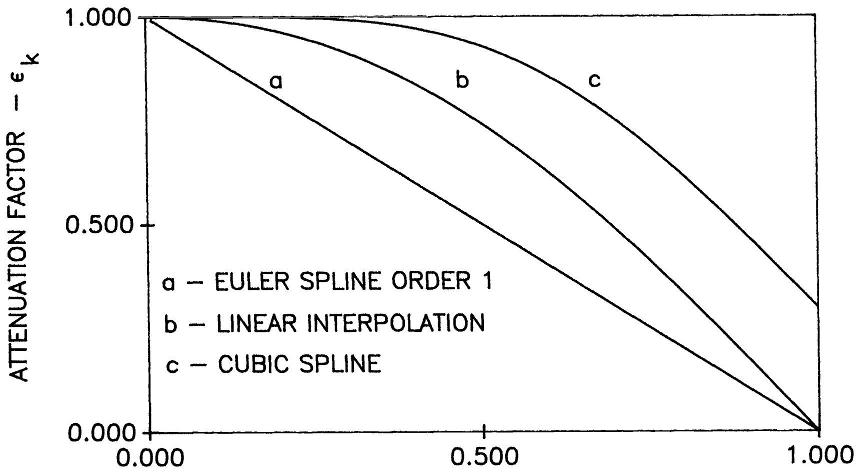


FIG. 1. *Attenuation factors. Example* 1.

To investigate the effectiveness of attenuation factors, the maximum infinity norm of $\frac{1}{2}|I-H(e^{j\theta})^{-1}[F_n(e^{j\theta})W_nF_n^*(e^{j\theta})]|$ was calculated after six iterations of the algorithm for increasing values of N. The infinity norm was calculated using 129 equispaced points on the interval $[0, \pi]$. These results are summarized in Table 1. Included in this table are the CPU times, and a bound on the achievable accuracy. The calculations were performed on an IBM 3081. From this table, we note that the use of attenuation

TABLE 1
Error norm and computation time for example 1.

	Attenuation Method				
s	Equation (14)	Linear interpolation	Cubic spline	Euler spline $\sigma=1$, derivative approximation	Relative time
3	0.46	0.12	0.78	0.12	1.00
4	0.18	0.60	0.28	0.42	1.15
5	0.41E−02	0.12	0.21E−01	0.59E−01	1.42
6	0.22E−03	0.30E−01	0.17E−02	0.22E−03	2.09
7	0.45E−04	0.45E−03	0.30E−04	0.40E−03	3.21

Notes. (1) The error norm is defined as

$$\max_\theta \tfrac{1}{2}|I-H(e^{j\theta})^{-1}[F_n(e^{j\theta})W_nF_n^*(e^{j\theta})]|_\infty, \qquad \theta=\theta_r=r\pi/128, \qquad r=0,1,\cdots,128.$$

(2) The achievable error bound

$$\max_\theta \tfrac{1}{2}|I-H(e^{j\theta})^{-1}H(e^{j\theta})|_\infty, \qquad \theta=\theta_r=r\pi/128, \qquad r=0,1,\cdots,128$$

was calculated as 0.81E−5.

(3) The calculations were performed on an IBM 3081. The computation time is relative to the time taken to solve the problem from the first line of this table, which was 0.47 seconds. The computation time was essentially independent of the method used to calculate the attenuation factors.

factors can reduce the accuracy. Of the methods examined, no single method consistently gave more accurate results than using no attenuation factors.

Based on these results, one would conclude that the use of attenuation factors may not be desirable. An effective strategy is to use as many equispaced points as possible, without using attenuation factors. A heuristic approach was adopted to select the number of points on the unit circle. The algorithm is implemented as described with 2^s+1 points, with s typically 4 or 5. s was increased by 1 whenever

$$(17) \qquad \sum_{k=N/4}^{N/2}\{|\tilde{a}_{pqk}|+|\tilde{a}_{pq-k}|\}\geqq .001\sum_{k=1}^{N/2}\{|\tilde{a}_{pqk}|+|\tilde{a}_{pq-k}|\}, \qquad p,q=1,2,\cdots,m.$$

This procedure was continued until some upper limit on s was reached. For many examples studied, $s=7$ was found to be adequate. The computation time is typically reduced by forty percent compared to using a fixed value of $s=7$.

The second example is taken from [17]. In designing a controller to regulate the exit concentrations from a pilot scale chemical reactor, it is necessary to compute the left spectral factor of the matrix polynomial $H(z)=M^T(z)M(z^{-1})$, where $M(z^{-1})$ is given by

$$M(z^{-1})=\begin{bmatrix} 1.8-6.64z^{-1}+5.158z^{-2} & 0.5384z^{-1}-0.3307z^{-2} \\ -0.7498-2.62z^{-1}-2.266z^{-2} & 0.1592z^{-2}-0.0866z^{-3} \\ -0.1069z^{-3}-0.3453z^{-4} & -0.0614z^{-4}-0.0594z^{-5} \end{bmatrix}.$$

The determinant of $M(z^{-1})$ has a zero located at infinity and zeros located at $z=-5.3514$, $z=-.5791$, $z=.4735\pm.5145j$, $z=.6701\pm.2291j$. To compute the spectral factor, the spectrum of $H(z)$ is evaluated at 129 points between 0 and π by letting $z=\exp(\pi jr/N)$, $r=0,1,\cdots,128$. $H(e^{j\theta})$ satisfies properties (P). The condition number of $H(e^{j\theta})$ is shown in Fig. 2.

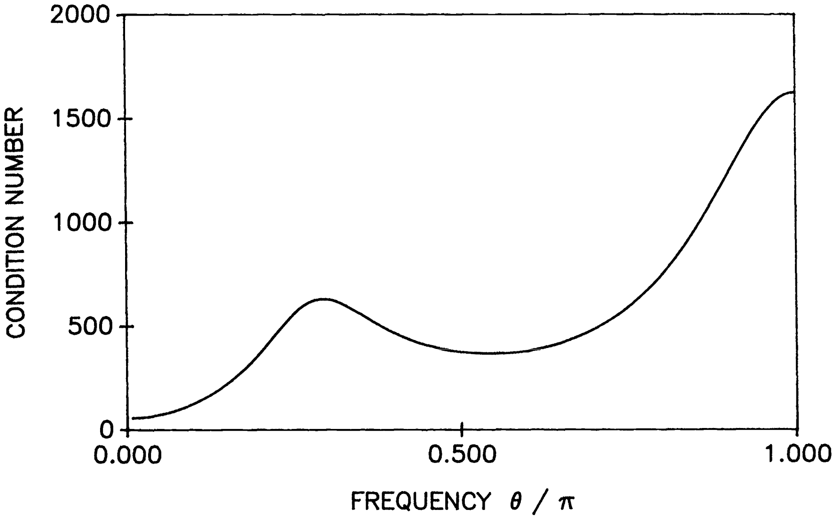


FIG. 2. *Condition number of spectrum. Example* 2.

The iteration described above was employed. The initial value for s was 4. The final value was 7. The iteration was terminated when the $\max \frac{1}{2}|I - H(e^{j\theta})^{-1}[F_n(e^{j\theta}) W_n F_n^*(e^{j\theta})]|_\infty < .001$. Six iterations and 0.69 seconds of CPU time were required. The maximum infinity norm upon termination was 0.12E − 4. The bound on the achievable performance was 0.76E − 5. The coefficients $\langle F_k \rangle$ were determined by taking the Fourier transform of $F(e^{j\theta})$. The final values for $F_{ij}(z^{-1})$ are

$$F_{11}(z^{-1}) = 1.000 - 1.192z^{-1} + 0.497z^{-2} + 0.080z^{-3} + 0.0167z^{-4},$$

$$F_{12}(z^{-1}) = 0.029z^{-1} - 0.032z^{-2} + 0.020z^{-4} + 0.001z^{-5},$$

$$F_{21}(z^{-1}) = -2.076z^{-1} + 4.115z^{-2} + 0.683z^{-4} + 0.148z^{-5},$$

$$F_{22}(z^{-1}) = 1.000 - 0.329z^{-1} - 0.333z^{-2} + 0.221z^{-3} - 0.083z^{-4} - 0.025z^{-5}.$$

6. Comparison with a parametric method. In this section, we compare the computation time and solution accuracy of the proposed algorithm to the parametric method proposed in [26] which involves a Bauer-type factorization. The iteration proposed in [26] involves the Cholesky factorization of an $m \times m$ matrix in each iteration. The computations can be carried out in place, in a vector of length $m^2 \times (L+1)(L+2)$, where L is the degree of the polynomial to be factored. That algorithm exhibits both geometric and monotone convergence [19]. Unlike the nonparametric method proposed in this paper, the algorithm in [26] is capable of factorizing a spectrum whose determinant has zeros on $|z| = 1$.

We want to find the left spectral factor of the $m \times m$ matrix polynomial $H(z) = M^T(z)M(z^{-1})$, where $M(z^{-1})$ is of the form

$$M(z^{-1}) = M_0 + M_1 z^{-1} + \cdots + M_L z^{-L}. \tag{18}$$

A comparative study is influenced by the dimension m and L, the location of the zeros of the determinant of $M(z^{-1})$, the condition number of the spectrum, and the desired accuracy. To provide a manageable basis for comparison between these two methods, the matrix M_k was chosen to be lower diagonal with common elements $k+1$. The

TABLE 2

Relative solution times and error norms for the comparative study.

	m/L	2	5	10	20	25	30	40	50
Nonparametric		1.0 0.13E−4	1.44 0.57E−5	1.64 0.64E−5	2.24 0.84E−5	2.36 0.65E−5	2.40 0.11E−4	2.56 0.58E−5	3.12 0.61E−5
Parametric	2	0.4 0.94E−4	0.44 0.21E−2	0.68 0.16E−2	1.12 0.54E−3	1.44 0.39E−3	1.76 0.29E−3	2.88 0.21E−3	4.2 0.86E−3
Norm bound		<E−7	0.48E−6	0.66E−6	0.77E−6	0.48E−6	<E−7	0.22E−5	0.13E−5
Nonparametric		7.8 0.30E−4	9.44 0.27E−4	10.64 0.35E−4	15.0 0.32E−4	15.4 0.27E−4	15.8 0.25E−4	16.68 0.25E−4	20.3 0.25E−4
Parametric	5	2.08 0.43E−4	2.6 0.87E−3	3.7 0.62E−3	7.08 0.51E−3	9.6 0.55E−3	13.4 0.30E−3	32 0.23E−2	60.76 0.13E−1
Norm bound		<E−7	0.93E−6	0.95E−6	0.16E−5	0.25E−5	<E−7	0.51E−5	0.47E−5
Nonparametric		29.2 0.13E−3	49.9 0.27E−3	63.0 0.20E−3	79.6 0.27E−3	81.4 0.14E−3	82.8 0.16E−3	101.8 0.15E−3	120.7 0.16E−3
Parametric	10	10.68 0.30E−4	12.6 0.52E−3	18.0 0.40E−3	49.3 0.65E−3	82 0.86E−3	130.1 0.12E−2	258.2 0.11E−1	497 0.46E−2
Norm bound		0.48E−5	0.34E−5	0.18E−5	0.29E−5	0.29E−5	<E−7	0.12E−4	0.93E−5

Notes. (1) The error norm and norm bound are defined in the notes attached to Table 1.

(2) The solution time is the computation time relative to the time taken to solve the factorization with $m=2$ and $L=2$ using the nonparametric method, which was 0.26 seconds.

zeros of the determinant of $M(z^{-1})$, in the z-plane, are given by the roots of the equation

$$(z^L+2z^{L-1}+\cdots+Lz+(L+1))^m=0. \tag{19}$$

All of the zeros of this polynomial are greater than one in magnitude.

We seek a solution of the form

$$H(z)=F^T(z)WF(z^{-1}), \tag{20}$$

where $F(z^{-1})$ is an $m\times m$ matrix polynomial of degree L. It is readily verified that the solution is given by

$$R^{1/2}F_k=M_{L-k}, \qquad k=0,\cdots,L \tag{21}$$

where R is the lower diagonal Cholesky factor of W.

The common feature to both algorithms is that an estimate of W is produced in each iteration. Both algorithms were terminated using the criterion suggested in [26], that is, when $1/m$ trace $(I-W_{n+1}W_n^{-1})<.0001$. The infinity norm error discussed in the previous sections was calculated for each method after termination at 129 equispaced points on the interval $[0, \pi]$. An estimate of the achievable accuracy was also calculated.

The results of this numerical simulation are summarized in Table 2 for $L=2$, 5, 10, 20, 25, 30, 40, and 50 and $m=2$, 5, and 10. The computation time is that required to perform the factorization and to calculate the infinity norm error. No more than seven iterations were required to satisfy the termination criterion using the nonparametric method. The parametric method requires that at least L iterations be performed. On average, $L+5$ iterations were required to terminate the algorithm. Several conclusions are apparent from the data in Table 2. First, the time to compute the nonparametric solution is much less sensitive to the degree of the matrix polynomial than is the solution time for the parametric method. Second, there is a crossover point at which the solution time for the parametric method increases very rapidly. Third, with the exception of the case where $m=10$ and $L=2$, the frequency-dependent error norm was less for the nonparametric method than for the parametric method, even though both methods employed the same termination criterion.

7. Summary. A computationally attractive algorithm has been developed for the spectral factorization of a multivariable spectrum using fast Fourier transforms. In the examples studied, it was found that attenuation factors do not ensure that a more accurate solution is obtained. An area of future work is to develop the algorithm to enable a factorization when the determinant of $H(z)$ has zeros on $|z|=1$, and to investigate how readily the algorithm can be vectorized.

REFERENCES

[1] B. D. O. ANDERSON, *An algebraic solution to the spectral factorization problem*, IEEE Trans. Automat. Control, 4 (1967), pp. 410-414.

[2] B. D. O. ANDERSON, K. HITZ, AND N. D. DIEM, *Recursive algorithm for spectral factorization*, IEEE Trans. Circuits and Systems, 21 (1974), pp. 742-750.

[3] H. BRASS, *Zur Numerischen Berechnung der Konjugierten Funktion*, in Numerical Methods of Approximation Theory, L. Collatz, G. Meinardus, and H. Werner, eds., 6, Birkhäuser, Basel, 1982, pp. 43-62.

[4] F. M. CALLIER, *On polynomial matrix spectral factorization by symmetric extraction*, IEEE Trans. Automat. Control, 30 (1985), pp. 453-464.

[5] M. M. CHAWLA AND T. R. RAMAKRISHNAN, *Numerical evaluation of integrals of periodic functions with Cauchy and Poisson type kernels*, Numer. Math., 22 (1974), pp. 317-323.

[6] J. DAVIS AND R. G. DICKINSON, *Spectral factorization by optimal gain iteration*, SIAM J. Appl. Math., 43 (1983), pp. 289-301.

[7] W. GAUTSCHI, *Attenuation factors in practical Fourier analysis*, Numer. Math., 18 (1972), pp. 373-400.

[8] D. GAIER, *Ableitungsfreie Abschatzungen bei Trigonometrischer Interpolation und Konjugierten-Bestimmung*, Computing, 12 (1974), pp. 145-148.

[9] T. T. GEORGIOU, *On a Schur-algorithm based approach to spectral factorization: State-space formulae*, Systems Control Lett., 10 (1983), pp. 123-129.

[10] I. C. GOHBERG AND M. G. KREIN, *Systems of integral equations on a half line with kernels depending on the difference of arguments*, Amer. Math. Soc. Transl., 14 (1960), pp. 217-288.

[11] M. H. GUTKNECHT, *Fast algorithms for the conjugate periodic function*, Computing, 22 (1979), pp. 79-91.

[12] ———, *The evaluation of the conjugate function of a periodic spline on a uniform mesh*, J. Comput. Appl. Math., 16 (1986), pp. 181-201.

[13] T. J. HARRIS AND J. F. MACGREGOR, *Design of multivariable linear-quadratic controllers using transfer functions*, AIChE J., 33 (1987), pp. 1481-1495.

[14] P. HENRICI, *Pointwise error estimates for trigonometric interpolation and conjugation*, J. Comput. Appl. Math., 8 (1982), pp. 31-32.

[15] ———, *Applied and Computational Complex Analysis*, Vol. III, John Wiley, Toronto, Canada, 1986.

[16] J. JEZEK AND V. KUCERA, *Efficient algorithm for spectral factorization*, Automatica—J. IFAC, 21 (1985), pp. 663-670.

[17] D. J. KOZUB, J. F. MACGREGOR, AND J. D. WRIGHT, *Application of LQ and IMC controllers to a packed bed reactor*, AIChE J., 33 (1987), pp. 1496-1503.

[18] D. J. KOZUB, J. F. MACGREGOR, AND T. J. HARRIS, *Optimal IMC inverses: design and robustness considerations*, Chem. Engrg. Sci., 44 (1989), pp. 2121-2136.

[19] V. KUCERA, *Discrete Linear Control: The Polynomial Approach*, John Wiley, Toronto, Canada, 1979.

[20] P. MASANI, *Recent trends in multivariate prediction theory*, in Multivariate Analysis, P. R. Krishnaiah, ed., Academic Press, New York, 1966, pp. 350-382.

[21] J. RISSANEN, *Algorithm for triangular decomposition of block Hankel and Toeplitz matrices with applications to factoring positive matrix polynomials*, Math. Comp., 27 (1973), pp. 147-153.

[22] E. ROBINSON, *Multichannel Time Series Analysis*, Holden-Day, San Francisco, 1967.

[23] W. G. TUEL, *Computer algorithm for spectral factorization of rational matrices*, IBM J. Res. Develop., (1968), pp. 163-170.

[24] G. T. WILSON, *The factorization of matricial spectral densities*, SIAM J. Appl. Math., 23 (1972), pp. 420-426.

[25] ———, *A convergence theorem for spectral factorization*, J. Multivariate Anal., 8 (1978), pp. 222-232.

[26] D. C. YOULA AND N. N. KAZANJIAN, *Bauer-type factorization of positive matrices and the theory of matrix polynomials orthogonal on the unit circle*, IEEE Trans. Circuits and Systems, 25 (1978), pp. 57-69.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 2, pp. 541-549, March 1992

ON THE STRUCTURE OF JACOBIANS FOR SPECTRAL METHODS FOR NONLINEAR PARTIAL DIFFERENTIAL EQUATIONS*

ROBERT D. RUSSELL†, DAVID M. SLOAN‡, AND MANFRED R. TRUMMER§

Abstract. When solving nonlinear partial differential equations with periodic boundary conditions, one frequently uses spectral methods. Implicit time-stepping, the computation of steady-state solutions, or the approximation of inertial manifolds lead to nonlinear equations which can be solved by Newton's method. In this note the structure of the Jacobian matrix is described for typical nonlinearities, namely, those involving products of the unknown function and its derivatives.

Key words. spectral methods, Galerkin methods, nonlinear PDEs, convolutions

AMS(MOS) subject classifications. 65P05, 42A10, 65J15, 65N30, 35K55, 42A85

1. Introduction. There is a growing interest in computing solutions of evolutionary partial differential equations (PDEs), and highly accurate discretisation in space is often achieved using spectral methods. Two of the most commonly used spectral methods are the standard spectral Galerkin method and the pseudospectral method [1], [5]. The latter, which is algebraically equivalent to a collocation method, differs from a Galerkin method in its treatment of nonlinear terms. Convolution sums are evaluated in a way that introduces aliasing errors. These errors arise due to the inability of a discrete grid to distinguish between large classes of periodic functions. There is ample evidence to suggest that for many problems—particularly nonevolutionary problems with smooth solutions—aliasing errors have little influence on accuracy once sufficient resolution has been obtained. However, in long-time integrations of nonlinear evolutionary problems, the presence of aliasing errors can influence the onset of nonlinear instabilities (see [12], [13]). The aliased pseudospectral method is less likely than the alias-free Galerkin method to yield semidiscrete equations which satisfy appropriate conservation properties. Since lack of conservation can have a deleterious effect on stability, it follows that Galerkin methods have some advantages in such circumstances. There are techniques which may be used to remove aliasing errors from pseudospectral computations (see [2]), and the de-aliased method is then algebraically equivalent to the Galerkin method. However, the removal of aliasing adds to the computational cost and reduces the efficiency relative to the Galerkin method.

Here we describe a technique which simplifies the treatment of certain nonlinearities by the alias-free Galerkin method. When solving nonlinear evolutionary equations using the Galerkin method with implicit time-stepping techniques, one can face the problem of evaluating Jacobian matrices in terms of the expansion coefficients. In this paper we expose the structure of Jacobians arising from a variety of nonlinearities in Fourier spectral methods, and we suggest efficient ways to compute the Jacobian

* Received by the editors August 1, 1990; accepted for publication (in revised form) January 6, 1991.

† Department of Mathematics and Statistics, Simon Fraser University, Burnaby, British Columbia, V5A 1S6, Canada. The research of this author was supported by Natural Sciences and Engineering Research Council of Canada grant A8781.

‡ Department of Mathematics, University of Strathclyde, Glasgow G1 1XH, Scotland. The research of this author was supported by Natural Sciences and Engineering Research Council of Canada grants A8781 and OGP0036901.

§ Department of Mathematics and Statistics, Simon Fraser University, Burnaby, British Columbia, V5A 1S6, Canada. The research of this author was supported by Natural Sciences and Engineering Research Council of Canada grant OGP0036901.

and to evaluate nonlinearities. An important application for which this structure has been exploited successfully is the computation of approximate inertial manifolds for dissipative equations (see [11]).

This note was originally motivated by the study of the Kuramoto-Sivashinsky equation [8]

$$\text{(KS)}\qquad u_t + 4u_{xxxx} + \theta[u_{xx} + uu_x] = 0, \quad u(x,t) = u(x+2\pi, t), \quad (x,t) \in \mathbb{R}\times\mathbb{R}.$$

If one discretises in space using a spectral method, the PDE is transformed into a system of ordinary differential equations (ODEs). For instance, if we try to find an odd solution of (KS) of the form

$$u(x,t) = \sum_{k=1}^{n} b_k(t) \sin kx, \tag{1}$$

then the Galerkin approach yields the following system of nonlinear ODEs for the Fourier coefficients $\mathbf{b} := (b_1, \cdots, b_n)^T$:

$$\frac{d}{dt} b_k(t) + 4k^4 b_k(t) + \theta[-k^2 b_k(t) + h_k(\mathbf{b})] = 0, \qquad k = 1, \cdots, n, \tag{2}$$

where

$$h_k(\mathbf{b}) = \frac{1}{2}\left\{ -k \sum_{j=1}^{n-k} b_j b_{j+k} + \sum_{j=1}^{k-1} j b_j b_{k-j} \right\}. \tag{3}$$

Let $\Lambda \in \mathbb{R}^{n\times n}$ denote the diagonal matrix with kth entry $\Lambda_{kk} := 4k^4 - \theta k^2$, and let $\mathbf{h}(\mathbf{b}) := (h_1(\mathbf{b}), \cdots, h_n(\mathbf{b}))^T$. We may rewrite system (2) in vector form

$$\dot{\mathbf{b}}(t) + \Lambda \mathbf{b}(t) + \theta \mathbf{h}(\mathbf{b}(t)) = 0. \tag{4}$$

Whether we integrate (4) with an implicit time-stepping scheme or compute steady-state solutions directly by setting $\dot{\mathbf{b}} = 0$ in (4), we are faced with solving a nonlinear system of algebraic equations for $\mathbf{b}$, with the essential nonlinearity $\mathbf{h}(\mathbf{b})$.

For general nonlinear PDEs, the resulting nonlinear algebraic equations are frequently solved by ordinary fixed-point iteration. However, it turns out that for many nonlinearities the Jacobian matrix $(\partial h_j(\mathbf{b})/\partial b_k)$ has a fairly simple structure and can be computed rather efficiently. Therefore, Newton's method becomes a viable alternative to ordinary iteration procedures. Higher accuracy in time can be achieved more efficiently, commensurate with spectral accuracy in space.

2. Complex and real Fourier series. In §3 we compute Jacobian matrices of special nonlinearities, using complex Fourier series for u and the convolution theorem. While one can instead use a (truncated) cosine/sine series for the unknown real-valued 2π-periodic function u, this often tends to obscure the underlying simple structure of the Jacobian. To see this consider both representations, namely truncated series of complex exponentials,

$$u(x) = \sum_{k=-n}^{+n} c_k e^{ikx}, \tag{5}$$

where $i^2 = -1$, and trigonometric polynomials

$$u(x) = \frac{a_0}{2} + \sum_{k=1}^{n} a_k \cos kx + \sum_{k=1}^{n} b_k \sin kx. \tag{6}$$

The relation between the complex Fourier coefficients c_k and the real cosine/sine coefficients a_k, b_k is given by

$$\begin{aligned} a_k &= c_k + c_{-k}, & c_k &= \tfrac{1}{2}(a_k - ib_k), \\ b_k &= i(c_k - c_{-k}), & c_{-k} &= \tfrac{1}{2}(a_k + ib_k). \end{aligned} \tag{7}$$

Let $F(u)$ be a (nonlinear) function. Restricting F to the finite-dimensional subspaces spanned by complex exponentials or trigonometric polynomials as in (5) or (6), respectively, we write

$$u \to PF(u(x)) = \sum_{k=-n}^{+n} f_k e^{ikx} = \frac{g_0}{2} + \sum_{k=1}^{n} g_k \cos kx + \sum_{k=1}^{n} h_k \sin kx, \tag{8}$$

where the g_k, h_k, and f_k are related in the same way as the a_k, b_k, and c_k in (7). P denotes the orthogonal projection onto the space containing functions of the form (5) or (6), respectively. Equation (8) gives rise to two related mappings on finite-dimensional vector spaces, namely,

$$\mathbf{c} \to \mathbf{f} \tag{9C}$$

on the complex vector space C^{2n+1}, and

$$(\mathbf{a}, \mathbf{b}) \to (\mathbf{g}, \mathbf{h}) \tag{9R}$$

on the real vector space $\mathbb{R}^{2n+1}$. (If u and $F(u)$ are not real-valued, there is no need to avoid the complex Fourier series.)

For many types of nonlinearities, it is fairly straightforward to compute the Jacobian matrix $\partial \mathbf{f}/\partial \mathbf{c}$ of (9C), as we see below. The Jacobian matrix of the mapping in (9R) can then be obtained easily using $g_k = f_k + f_{-k}$, $h_k = i(f_k - f_{-k})$. Letting $J_{j,k} := \partial f_j/\partial c_k$, the chain rule gives

$$\frac{\partial f_j}{\partial a_k} = \frac{1}{2}\{J_{j,k} + J_{j,-k}\}, \qquad \frac{\partial f_j}{\partial b_k} = \frac{i}{2}\{-J_{j,k} + J_{j,-k}\}. \tag{10}$$

The formulas relating the Jacobian J of (9C) and the Jacobian of (9R) can be written in compact form as a unitary transformation

$$\begin{bmatrix} \partial g_j/\partial a_k \\ \partial g_j/\partial b_k \\ \partial h_j/\partial a_k \\ \partial h_j/\partial b_k \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -i & i & -i & i \\ i & i & -i & -i \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} J_{j,k} \\ J_{j,-k} \\ J_{-j,k} \\ J_{-j,-k} \end{bmatrix}. \tag{11}$$

3. Jacobian of "monomials." We now return to the problem of computing the matrix $J = \partial \mathbf{f}/\partial \mathbf{c}$ and present an algorithm for the case where $F(u)$ involves products of powers of the unknown function u and its space derivatives u_x, u_{xx}, etc. Our basic tool is the convolution theorem of Fourier analysis (see, e.g., [7]). Note that we write the unknown function u as a sum of complex exponentials in (5), even though u is usually real-valued.

The convolution of two sequences $\mathbf{x} = (x_j)$ and $\mathbf{y} = (y_j)$ is the new sequence $\mathbf{x}^*\mathbf{y}$ given by

$$(\mathbf{x}^*\mathbf{y})_k := \sum_j x_j y_{k-j}. \tag{12}$$

The identity element for convolution is the sequence $\mathbf{e} = (e_j)$, with $e_0 := 1$, and $e_j := 0$ for $j \neq 0$. The convolution of two sequences is well defined if both sequences $\mathbf{x}$, $\mathbf{y}$ are square summable. For the new sequence $\mathbf{x}^*\mathbf{y}$ to be square summable, it is sufficient that, in addition, one of the sequences has compact support. In our applications, all sequences involved have compact support, and all subsequent results hold without any further technical assumptions. However, it is important to bear in mind that all these results hold for *bi-infinite* sequences and *bi-infinite* matrices only, although in practical terms one just needs to use sufficiently large vectors and/or matrices. See also the comments on bandwidth below.

We remark that sequences and convolution form a semigroup. Since convolution is commutative and since there is an identity element, this semigroup is a commutative monoid (all properties of an Abelian group except for the existence of an inverse element).

For simplicity, we start with powers of u only. If u has Fourier coefficients $\mathbf{c}$, then u^2 has Fourier coefficients $\mathbf{c}^*\mathbf{c}$. The Fourier coefficients of u^m are the elements of the sequence $\mathbf{c}^{*m} := \mathbf{c}^* \cdots {}^*\mathbf{c}$, the m-fold convolution of $\mathbf{c}$ with itself. This definition extends to the case $m = 0$, with $\mathbf{c}^{*0} := \mathbf{e}$.

Convolutions can also be expressed in terms of matrices and matrix multiplications.

DEFINITION 1. Let $\mathbf{c} = (c_j)$ be a sequence. We define the (bi-infinite) complex Toeplitz matrix $\Gamma(\mathbf{c})$ by

$$\Gamma(\mathbf{c}) = (\gamma(\mathbf{c})_{jk}), \qquad \gamma_{jk} := c_{j-k}. \tag{13}$$

Then $\mathbf{c}$ is the zeroth column of the matrix $\Gamma(\mathbf{c})$, $\mathbf{c}^*\mathbf{c}$ is the zeroth column of the matrix $\Gamma(\mathbf{c})^2$, and $\mathbf{c}^{*m}$ is the zeroth column of the matrix $\Gamma(\mathbf{c})^m$, for every nonnegative integer m.

It is straightforward to prove the following proposition by induction.

PROPOSITION 1. *Let $m \geqq 1$, and $F(u) = u^m$. The Jacobian matrix of the induced mapping $\mathbf{c} \to \mathbf{f}(\mathbf{c}) = \mathbf{c}^{*m}$ is given by*

$$\frac{\partial f_j}{\partial c_k} = m(\mathbf{c}^{*m-1})_{j-k}. \tag{14}$$

With the matrix interpretation above, this can also be simply expressed as

$$J = \frac{\partial \mathbf{c}^{*m}}{\partial \mathbf{c}} = m\Gamma(\mathbf{c})^{m-1}. \tag{15}$$

Proposition 1 is the analogue to differentiating powers in calculus. Note that if $c_j = 0$ for $|j| > n$, then $\Gamma(\mathbf{c})$ has bandwidth n, and $\Gamma(\mathbf{c})^m$ has bandwidth mn. Even if in the end we only look at Fourier modes with frequencies not larger than n, we still must allow the intermediate bandwidth to grow in order to capture all nonlinear interactions.

We proceed to state the product rule for differentiating convolutions. Notice that, unlike a scalar product, the convolution of two sequences is yet another sequence. We therefore extend the definition of the convolution operation to a convolution product of a sequence with a (bi-infinite) matrix. Doing this in a "natural" way results in a matrix as the product.

DEFINITION 2. Let $A = (a_{jk})$ be a bi-infinite matrix, and $\mathbf{x} = (x_k)$ be a sequence. We further denote by $\mathbf{e}^{(j)}$ the jth standard basis sequence, i.e., the sequence consisting of all zeros, except for the jth component which equals 1 (convolution $\mathbf{x}^*\mathbf{e}^{(j)}$ corresponds to shifting the sequence $\mathbf{x}$ by j). Then $B := \mathbf{x}^*A$ is the bi-infinite matrix whose jth column is the (ordinary) convolution of the jth column of A with the sequence $\mathbf{x}$:

$$(\mathbf{x}^*A)\mathbf{e}^{(j)} := \mathbf{x}^*(A\mathbf{e}^{(j}). \tag{16}$$

The reason for writing $\mathbf{x}^*A$ rather than $A^*\mathbf{x}$ lies in the way this convolution corresponds to ordinary matrix multiplication and the Toeplitz matrices defined in (13).

PROPOSITION 2. *Let I be the identity matrix, let S be a diagonal matrix, and let $\mathbf{x}=(x_j)$ be a sequence. Then*

$$\mathbf{x}^*I=\Gamma(\mathbf{x}), \qquad \mathbf{x}^*S=\Gamma(\mathbf{x})S, \tag{17}$$

where Γ is defined in (13).

PROPOSITION 3. *Γ as defined in* (13) *is a homomorphism between the commutative monoid (semigroup) of sequences with the convolution operation, and the commutative monoid (semigroup) of Toeplitz matrices with matrix multiplication. If $\mathbf{x}=(x_j)$ and $\mathbf{y}=(y_j)$ are sequences, then*

$$\Gamma(\mathbf{x}^*\mathbf{y})=\Gamma(\mathbf{x})\Gamma(\mathbf{y}). \tag{18}$$

Furthermore, this homomorphism is consistent with the definition of the convolution of a sequence with a matrix in the following sense:

$$\mathbf{x}^*\Gamma(\mathbf{y})=\Gamma(\mathbf{x}^*\mathbf{y})=\Gamma(\mathbf{x})\Gamma(\mathbf{y}). \tag{19}$$

Proof. We compute the (j, k)-element of the matrices in (18),

$$\Gamma(\mathbf{x}^*\mathbf{y})_{jk}=\sum_p x_p y_{j-k-p}=\sum_q x_{j-q}y_{q-k}=\sum_q \Gamma(\mathbf{x})_{jq}\Gamma(\mathbf{y})_{qk}=(\Gamma(\mathbf{x})\Gamma(\mathbf{y}))_{jk}.$$

To prove (19), we use Proposition 2 and the associative law:

$$\mathbf{x}^*\Gamma(\mathbf{y})=\mathbf{x}^*(\mathbf{y}^*I)=(\mathbf{x}^*\mathbf{y})^*I=\Gamma(\mathbf{x}^*\mathbf{y}). \qquad \square$$

PROPOSITION 4 (PRODUCT RULE). *Let $\mathbf{x}=(x_j)$ and $\mathbf{y}=(y_j)$ be sequences depending on the sequence $\boldsymbol{\alpha}=(\alpha_k)$. Denote the Jacobians of $\mathbf{x}(\boldsymbol{\alpha})$ and $\mathbf{y}(\boldsymbol{\alpha})$ by $J^{\mathbf{x}}:=\partial\mathbf{x}/\partial\boldsymbol{\alpha}$ and $J^{\mathbf{y}}:=\partial\mathbf{y}/\partial\boldsymbol{\alpha}$, respectively. Then the Jacobian of $\mathbf{z}:=\mathbf{x}^*\mathbf{y}$ is*

$$J^{\mathbf{z}}:=\frac{\partial\mathbf{z}}{\partial\boldsymbol{\alpha}}=\mathbf{y}^*J^{\mathbf{x}}+\mathbf{x}^*J^{\mathbf{y}}. \tag{20}$$

The proof is straightforward, using commutativity of (ordinary) convolution.

Proposition 1 allows us to compute Jacobians of positive integral powers of u. To find the Jacobian of a product involving u and its space derivatives u_x, u_{xx}, etc., we need to employ the Fourier spectral derivative matrix

$$D:=\operatorname{diag}(\cdots, -in, -i(n-1), \cdots, -2i, -i, 0, i, 2i, \cdots, i(n-1), in, \cdots). \tag{21}$$

The pth derivative of u has Fourier coefficients $D^p\mathbf{c}$.

There is an interesting relationship between D and the matrix Γ defined in (13), which allows us to write $\Gamma(D^p\mathbf{c})$ in terms of products of $\Gamma(\mathbf{c})$ and D.

PROPOSITION 5. *Let $\mathbf{c}=(c_j)$ be a sequence, and D the Fourier spectral derivative matrix defined in* (21). *Then*

$$\Gamma(D\mathbf{c})=[D,\Gamma\mathbf{c}]:=D\Gamma(\mathbf{c})-\Gamma(\mathbf{c})D. \tag{22}$$

$\Gamma(D\mathbf{c})$ is the commutator of D and $\Gamma(\mathbf{c})$, and (22) *can be used recursively to express $\Gamma(D^p\mathbf{c})$ in terms of $\Gamma(\mathbf{c})$ and D only.*[1]

This result can be shown by direct calculation, or, more elegantly, by comparing the Jacobians of uu_x and $\frac{1}{2}(u^2)_x$ (see Example 1 below).

[1] $\Gamma(D^p\mathbf{c})=\sum_{k=0}^{p}(-1)^{p-k}\binom{p}{k}D^k\Gamma(\mathbf{c})D^{p-k}$, as has been kindly pointed out by one of the referees.

We now give a few examples which show how products of u, u_x, u_{xx}, etc., can be handled.

Example 1. $F(u)=u^m u_x$, $m \geqq 0$. Noting that $u^m u_x = (1/(m+1))(u^{m+1})_x$, we obtain from (15) the Jacobian as

$$J = D\Gamma(\mathbf{c})^m. \tag{23}$$

Let us have a closer look at the case $m=1$, i.e., $F(u)=uu_x$. This is the Burgers' nonlinearity arising in the Kuramoto-Sivashinsky equation, or in the nonlocal Burgers' equation (e.g., [4]). The product rule gives

$$J = D\mathbf{c} * I + \mathbf{c} * D = \Gamma(D\mathbf{c}) + \Gamma(\mathbf{c})D, \tag{24}$$

which equals $D\Gamma(\mathbf{c})$ because of (22).

Example 2. $F(u)=uu_{xx}$. Using the product rule and Proposition 3, we can express the result as

$$J = D^2\mathbf{c} * I + \mathbf{c} * D^2 = \Gamma(D^2\mathbf{c}) + \Gamma(\mathbf{c})D^2. \tag{25}$$

Noting that $uu_{xx} = \frac{1}{2}(u^2)_{xx} - u_x^2$, the Jacobian can also be written as

$$J = D^2\Gamma(\mathbf{c}) - 2\Gamma(D\mathbf{c})D. \tag{26}$$

The expressions (25) and (26) are equal, because from (22)

$$\Gamma(D^2\mathbf{c}) = D\Gamma(D\mathbf{c}) - \Gamma(D\mathbf{c})D = D^2\Gamma(\mathbf{c}) - 2D\Gamma(\mathbf{c})D + \Gamma(\mathbf{c})D^2,$$

so

$$\Gamma(D^2\mathbf{c}) + \Gamma(\mathbf{c})D^2 = D^2\Gamma(\mathbf{c}) - 2(D\Gamma(\mathbf{c}) - \Gamma(\mathbf{c})D)D = D^2\Gamma(\mathbf{c}) - 2\Gamma(D\mathbf{c})D.$$

Example 3. $F(u)=u_x^2$. The product rule applied to $D\mathbf{c} * D\mathbf{c}$ gives

$$J = 2D\mathbf{c} * D = 2\Gamma(D\mathbf{c})D, \tag{27}$$

which can also be found using the chain rule for mappings between vector spaces. For example, the Kolmogorov-Sivashinsky-Spiegel equation (KSS) (see, e.g. [3]) has a term u_x^2. The same equation features the term treated in the next example.

Example 4. $F(u)=(u_x^3)_x$. First we compute the Jacobian K of u_x^3. From Example 3 and the product rule, we get (note that we compute the Jacobian of $D\mathbf{c} * D\mathbf{c} * D\mathbf{c}$)

$$K = D\mathbf{c} * 2\Gamma(D\mathbf{c})D + (D\mathbf{c} * D\mathbf{c}) * D.$$

From Propositions 2 and 3 we finally obtain

$$K = 2\Gamma(D\mathbf{c} * D\mathbf{c})D + \Gamma(D\mathbf{c} * D\mathbf{c})D = 3\Gamma(D\mathbf{c})^2 D. \tag{28}$$

By induction we can also show that the Jacobian of u_x^m is $m\Gamma(D\mathbf{c})^{m-1}D$.

The function $F(u)$ has an outer derivative, which corresponds to a diagonal scaling by D from the left to yield the Jacobian

$$J = 3D\Gamma(D\mathbf{c})^2 D. \tag{29}$$

Example 5. $F(u)=(u^m)_{xx}$. This nonlinearity occurs in the one-dimensional Cahn-Hilliard equation [4]. Applying our convolution calculus we obtain

$$J = mD^2\Gamma(\mathbf{c})^{m-1}. \tag{30}$$

For $m=2$ we can also write $(u^2)_{xx} = 2(u_x^2 + uu_{xx})$. The Jacobians of each of the two terms are in Examples 2 and 3:

$$J = 2\{2\Gamma(D\mathbf{c})D + \Gamma(D^2\mathbf{c}) + \Gamma(\mathbf{c})D^2\}. \tag{31}$$

Equivalently, if we use (26) instead of (25), we obtain

$$J=2\{2\Gamma(D\mathbf{c})D+D^2\Gamma(\mathbf{c})-2\Gamma(D\mathbf{c})D\}=2D^2\Gamma(\mathbf{c}), \tag{32}$$

which agrees with (30).

As long as we compute with sufficient bandwidth, we can use any form of the function F to compute the Jacobians. However, if we compute with a fixed bandwidth (i.e., $2n+1$ by $2n+1$ matrices), different ways of writing F may lead to different results.

4. Application to the Kuramoto–Sivashinsky equation. One important application for which the structure of the Jacobian matrix is of special interest is the computation of approximate inertial manifolds for dissipative PDEs [11], [9]. This problem motivated the study of the structure of the Jacobians presented in this paper. While skipping many details, we give a brief account of the method. We consider the evolution equation

$$u_t+Au+F(u)=0, \tag{33}$$

where A is a self-adjoint positive operator defined on a Hilbert space H, and F is the nonlinear part defined on the domain of A. Let $w_1, w_2, \cdots$ be orthogonal eigenfunctions of the operator A with corresponding eigenvalues $0<\lambda_1\leqq\lambda_2\leqq\cdots$. P denotes the usual (orthogonal) spectral projection onto the subspace spanned by the first m eigenfunctions, and Q is the spectral projection onto the subspace spanned by $w_{m+1}, \cdots, w_n$. We also set $p:=Pu$ and $q:=Qu$. An *approximate inertial manifold* (AIM) $q=\phi(p)$ for (33) can then be constructed by solving the differential algebraic equations

$$\dot{p}+Ap+PF(p+q)=0, \tag{34a}$$

$$Aq+QF(p+q)=0 \tag{34b}$$

(see [11], [14]). This AIM is referred to as the *steady* AIM, and has many nice properties. The relationship $q=\phi(p)$ can be determined by solving (34b); substituting $q=\phi(p)$ into (34a) yields an *approximate inertial form* of (33), and the resulting finite-dimensional ODE system has, under certain assumptions, many of the long-term dynamics features of the original equation (33).

To solve (34b) accurately, we use Newton's method. Here p is fixed, and we need the Jacobian matrix of $QF(p+q)$ with respect to the "high frequency modes" q. We are therefore only interested in the $(n-m)\times(n-m)$ trailing principal submatrix of the full Jacobian matrix $\partial F/\partial u$ corresponding to the partial derivatives of the high frequency modes of $F(u)$ with respect to the high frequency modes of u. It is easy to see that in the common case $n=2m$ the Jacobian only depends on the low frequency modes for *quadratic* nonlinearities like u^2, uu_x, u_x^2, uu_{xx}, etc. Therefore, the relation between the high frequency spectrum and the low frequency spectrum is a *linear* one and Newton's method for solving (34b) converges in one step (see [11]). For cubic or higher order terms this dependence is no longer linear.

We illustrate this for the Jacobian of uu_x (the essential nonlinearity of equation (KS)) in the odd case, i.e., on the subspace spanned by the first n sine functions $\sin x, \cdots, \sin nx$. From Example 1 the Jacobian matrix is

$$J=\partial\mathbf{f}/\partial\mathbf{c}=D\Gamma(\mathbf{c}). \tag{35}$$

Using (7), we express J in terms of the b_k by substituting $c_k=-\frac{1}{2}\operatorname{sign}(k)ib_{|k|}$. Finally, from (11) we obtain

$$\begin{aligned}\frac{\partial h_j}{\partial b_k}&=\tfrac{1}{2}\{ijc_{j-k}-ijc_{j+k}+ijc_{-j-k}-ijc_{-j+k}\}\\&=\tfrac{1}{2}j\{\operatorname{sign}(j-k)b_{|j-k|}-b_{j+k}\}.\end{aligned} \tag{36}$$

Any b_k with an index falling out of the range $1 \leqq k \leqq n$ is set to zero. Thus, the Jacobian $\partial \mathbf{h}/\partial \mathbf{b}$ is the sum of a skew-symmetric Toeplitz matrix and a Hankel matrix, with a diagonal scaling from the left. If $S = \operatorname{diag}(1, 2, 3, \cdots, n)$, and

$$(37) \qquad T := \begin{bmatrix} 0 & -b_1 & -b_2 & \cdots & -b_{n-1} \\ b_1 & 0 & -b_1 & \cdots & -b_{n-2} \\ b_2 & b_1 & 0 & \cdots & -b_{n-3} \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ b_{n-1} & b_{n-2} & b_{n-3} & \cdots & 0 \end{bmatrix}, \qquad H := \begin{bmatrix} b_2 & b_3 & b_4 & b_5 & \cdots & 0 \\ b_3 & b_4 & b_5 & \cdot & \cdots & 0 \\ b_4 & b_5 & b_6 & \cdot & \cdots & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdots & \cdot \\ b_n & 0 & 0 & \cdot & \cdots & 0 \\ 0 & 0 & 0 & \cdot & \cdots & 0 \end{bmatrix},$$

then

$$(38) \qquad \frac{\partial \mathbf{h}}{\partial \mathbf{b}} = \tfrac{1}{2} S(T - H).$$

In inertial manifold computations, when $n = 2m$, we are only interested in $\partial h_j / \partial b_k$ for $j, k = m+1, \cdots, 2m = n$, i.e., in the lower right-hand submatrix of (38). With $\hat{T}$ the trailing principal m by m submatrix of T ($\hat{T}$ is skew symmetric and Toeplitz, with $(0, b_1, b_2, \cdots, b_{m-1})^T$ as its first column), and $\hat{S} = \operatorname{diag}(m+1, m+2, \cdots, 2m)$, the Jacobian of the Fourier coefficients h_k of the last m modes of uu_x with respect to the Fourier coefficients $b_{m+1}, \cdots, b_{2m}$ is simply $\frac{1}{2}\hat{S}\hat{T}$. Clearly, this matrix is independent of $b_{m+1}, \cdots, b_{2m}$.

5. Conclusions. We presented an algorithm to compute Jacobians of nonlinear functions arising in the application of (Fourier) spectral methods to dissipative PDEs. We believe that this algorithm is an excellent way to derive the form of these Jacobians analytically. Note that the nonlinear function itself could also be computed via convolutions. Our method applies to the Fourier spectral method in general, not only to computing inertial manifolds (where we used it).

One could use the procedure outlined above to numerically evaluate Jacobians. In this case, one would not perform the matrix multiplications, but rather compute via convolutions, in particular, compute $\Gamma(\mathbf{c})^m$ as $\Gamma(\mathbf{c}^{*m})$. If $\mathbf{c}$ has compact support with $c_j = 0$ for $|j| > n$, one chooses the length of the sequence to be at least $2mn+1$, i.e., pad the sequence $\mathbf{c}$ with $(m-1)n$ extra zeros on either side. The convolutions can then be performed via (discrete) fast Fourier transforms (FFT) (see, e.g., [6], [7]) in $O(mn \log(mn))$ operations. Expressions like $\Gamma(\mathbf{c})^2\Gamma(D\mathbf{c})D$ can be evaluated as efficiently, with a bandwidth of n times the degree of the expression; in this case we have three Γ-matrices, hence the degree is 3 and the required bandwidth is $3n$. Again, the nonlinear function $\mathbf{c} \to \mathbf{f}$ (cf. (9C)) can also be evaluated via convolutions and FFT, provided there is sufficient padding of the sequences. Even though all these Jacobians can be expressed in terms of D and $\Gamma(\mathbf{c})$ only, for practical computations there is often no advantage in doing so. It is just as economical to perform convolutions with $\mathbf{c}$ and $D^p\mathbf{c}$ as with $\mathbf{c}$ alone. One should simply aim to keep the number of terms as small as possible. The convolution approach using FFTs will be faster only for a fairly large number of modes—experiments indicate a threshold of 64 to 128 modes. However, similar thresholds for efficiency are observed for Fourier spectral methods in general.

A wide variety of (dissipative) PDEs arising in physical applications has nonlinearities of the types discussed in this note. Among these equations are equation (KS); the Cahn-Hilliard equation (e.g., [10]),

$$(\mathrm{CH}) \qquad u_t + u_{xxxx} + (\alpha u - \beta u^2 - \gamma u^3)_{xx} = 0;$$

Burgers' equation,

$$\text{(B)} \qquad u_t - \nu u_{xx} + u u_x = 0;$$

the Kolmogorov-Sivashinsky-Spiegel equation [3],

$$\text{(KSS)} \qquad u_t + u_{xxxx} + ((2 - \theta u_x^2) u_x)_x + u_x^2 + \alpha u = 0;$$

and the Chafee-Infante reaction-diffusion equation (see, e.g., [4]),

$$\text{(CI-RD)} \qquad u_t - \nu u_{xx} + u^3 - u = 0.$$

REFERENCES

[1] J. P. Boyd, *Chebyshev and Fourier Spectral Methods*, Springer-Verlag, Berlin, 1989.

[2] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zang, *Spectral Methods in Fluid Dynamics*, Springer-Verlag, New York, 1987.

[3] P. Constantin, C. Foias, B. Nicolaenko, and R. Témam, *Spectral barriers and inertial manifolds for dissipative partial differential equations*, J. Dynamics & Differential Equations, 1 (1989), pp. 45-73.

[4] ———, *Integral Manifolds and Inertial Manifolds for Dissipative Partial Differential Equations*, Springer-Verlag, New York, 1989.

[5] D. Gottlieb and S. A. Orszag, *Numerical Analysis of Spectral Methods: Theory and Applications*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1977.

[6] P. Henrici, *Fast Fourier methods in computational complex analysis*, SIAM Rev., 21 (1979), pp. 481-527.

[7] ———, *Applied and Computational Complex Analysis, Vol. III*, John Wiley & Sons, New York, 1986.

[8] J. M. Hyman and B. Nicolaenko, *The Kuramoto-Sivashinsky equation: A bridge between PDE's and dynamical systems*, Phys. D, 18 (1986), pp. 113-126.

[9] M. S. Jolly, I. G. Kevrekidis, and E. S. Titi, *Approximate inertial manifolds for the Kuramoto-Sivashinsky equation: Analysis and computations*, Phys. D, 44 (1990), pp. 38-60.

[10] A. Novick-Cohen and L. A. Segel, *Nonlinear aspects of the Cahn-Hilliard equation*, Phys. D, 10 (1984), pp. 277-298.

[11] R. D. Russell, D. M. Sloan, and M. R. Trummer, *Some numerical aspects of computing inertial manifolds*, SIAM J. Sci. Statist. Comput., to appear.

[12] D. M. Sloan and A. R. Mitchell, *On nonlinear instabilities in leap-frog difference schemes*, J. Comput. Phys., 67 (1986), pp. 372-395.

[13] A. Stuart, *A note on high/low-wave-number interactions in spatially discrete parabolic equations*, IMA J. Appl. Math., 42 (1989), pp. 27-42.

[14] E. S. Titi, *On approximate inertial manifolds to the Navier-Stokes equations*, J. Math. Anal. Appl., 149 (1990), pp. 540-557.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 2, pp. 550-572, March 1992

IMPROVING THE ACCURACY OF INVERSE ITERATION*

ELIZABETH R. JESSUP† AND ILSE C. F. IPSEN‡

Abstract. The EISPACK routine TINVIT is an implementation of inverse iteration for computing eigenvectors of real symmetric tridiagonal matrices. Experiments have shown that the eigenvectors computed with TINVIT are numerically less accurate than those from implementations of Cuppen's divide-and-conquer method (TREEQL) and of the QL method (TQL2). The loss of accuracy can be attributed to TINVIT's choice of starting vectors and to its iteration stopping criterion.

This paper introduces a new implementation of TINVIT that computes each eigenvector from a different random starting vector and performs an additional iteration after the stopping criterion is satisfied. A statistical analysis and the results of numerical experiments with matrices of order up to 525 are presented to show that the numerical accuracy of this new implementation is competitive with that of the implementations of the divide-and-conquer and QL methods. The extension of this implementation to larger order matrices is discussed, albeit in less detail.

Key words. symmetric tridiagonal eigenvalue problem, inverse iteration, EISPACK

AMS(MOS) subject classification. 65F15

1. Introduction. Our goal is to determine an accurate method for computing all eigenvalues and eigenvectors of real symmetric tridiagonal matrices that is efficient both sequentially and in parallel. Experimental results in [16], [17] indicate that bisection with inverse iteration is generally the fastest and most efficient parallel eigensolver on a distributed-memory hypercube multiprocessor such as the Intel iPSC and that it is also the fastest sequential method for many problems. The computed eigendecompositions, however, are less accurate than those computed by existing implementations of Cuppen's divide-and-conquer method (TREEQL) [5], [12] or the QL method (TQL2) [3], [22]. The tested implementations of bisection are the EISPACK routines BISECT and TSTURM [22], both of which produce eigenvalues to high *absolute* accuracy [22]. With minor modification, each of these routines would produce eigenvalues to high *relative* accuracy when small relative changes in the matrix lead to small relative changes in the eigenvalues [7]. The loss of accuracy can thus be attributed to the tested implementation of inverse iteration (EISPACK's TINVIT). In this paper, we identify the factors influencing the accuracy of inverse iteration and present a new serial implementation of inverse iteration that computes eigenvectors to high absolute accuracy.

Suppose that T is an $n \times n$ real symmetric tridiagonal matrix with eigendecomposition

$$T = U\Lambda U^T, \qquad \Lambda = \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix}, \qquad U = (u_1 \cdots u_n),$$

where the diagonal elements Λ are the eigenvalues of T and the column u_i of the orthogonal matrix U is the eigenvector associated with eigenvalue λ_i. Given an

* Received by the editors March 12, 1990; accepted for publication (in revised form) January 2, 1991. The work presented in this paper was supported by Defense Advanced Research Projects Agency under contract N00014-88-K-0573.

† Department of Computer Science, University of Colorado, Boulder, Colorado 80309-0430 (jessup@boulder.colorado.edu).

‡ Department of Computer Science, Yale University, New Haven, Connecticut 06520 (ipsen@cs.yale.edu).

accurately computed eigenvalue $\hat{\lambda}_j$, inverse iteration computes the corresponding eigenvector u_j by performing the power method with the shifted matrix $(T-\hat{\lambda}_jI)^{-1}$.

ALGORITHM 1.1 (Inverse Iteration)

```
Select a starting vector $y^{(0)}$.
For $k = 1, 2, \cdots$ until convergence:
    Solve $(T-\hat{\lambda}_jI)y^{(k)} = y^{(k-1)}$ for $y^{(k)}$.
$\hat{u}_j = y^{(k)}/\|y^{(k)}\|_2$.
```

Representing the starting vector $y^{(0)}$ as a linear combination of the eigenvectors $y^{(0)} = \sum_{i=1}^{n} \eta_i u_i$ gives for the first iterate

$$y^{(1)} = \sum_{i=1}^{n} \frac{\eta_i}{\lambda_i - \hat{\lambda}_j} u_i.$$

If the contribution η_j of u_j in $y^{(0)}$ is not too small, and if $\hat{\lambda}_j$ is close to λ_j, the contribution $\eta_j/(\lambda_j - \hat{\lambda}_j)$ of u_j in the next iterate $y^{(1)}$ is large, and $y^{(1)}$ is a better approximation to u_j than is $y^{(0)}$ [24, p. 321]. Likewise, in the next iteration, the contribution of u_j in $y^{(2)}$ increases to $\eta_j/(\lambda_j - \hat{\lambda}_j)^2$ and so on for subsequent iterations. Thus, the iterates $y^{(k)}$ usually converge to u_j in only a few iterations.

If all eigenvalues are well separated, and if the starting vector in each eigenvector computation contains a large enough component η_i, inverse iteration using shifts $\hat{\lambda}_1, \cdots, \hat{\lambda}_n$ in turn computes an orthogonal set of eigenvectors. However, if some eigenvalues are close together, inverse iteration as outlined in Algorithm 1.1 produces eigenvectors that are not orthogonal. An additional step to orthogonalize iterates associated with close eigenvalues is necessary.

This discussion of the inverse iteration algorithm shows that if the eigenvalues $\hat{\lambda}_i$ are determined to working precision (which is true for BISECT) and if the linear system solution and the orthogonalization of iterates corresponding to close eigenvalues are carried out accurately (which is true for TINVIT), the overall accuracy of inverse iteration is determined by

(1) the choice of starting vector,
(2) the reorthogonalization criterion,
(3) the iteration stopping criterion.

We will examine the EISPACK routine TINVIT with regard to each factor in turn and gradually improve its accuracy to that of the implementations of the QL and Cuppen's divide-and-conquer methods. The numerical experiments presented involve matrix orders up to $n = 525$. We also discuss in less detail the extension of the implementation to larger order problems.

This paper is organized as follows: A simple perturbation result is presented in § 2 to define measures of high absolute accuracy for the computed eigenvalue decomposition. The EISPACK implementation TINVIT is described in § 3, and its lack of numerical accuracy explored in § 4. A new implementation of inverse iteration based on the experiments in § 4 is presented in § 5. The numerical accuracy of this improved implementation is compared to implementations of Cuppen's divide-and-conquer method and of the QL method in § 6. The use of random starting vectors in the new implementation of inverse iteration is justified by a statistical analysis in § 7. The most important points of this paper are summarized in § 8.

2. The computed eigendecomposition. This section shows that the computed eigendecomposition has high absolute accuracy if its residual and the deviation of the eigenvectors from orthogonality are small.

Suppose that the diagonal elements of Λ are the eigenvalues of $T = U\Lambda U^T$ in descending order

$$\lambda_1 \geqq \cdots \geqq \lambda_n,$$

and that the column u_i of the orthogonal matrix U is the eigenvector associated with λ_i satisfying $\|u_i\|_2 = 1$. The spectral radius of T is denoted by

$$|\lambda|_{\max} \equiv \max\{|\lambda_1|, |\lambda_n|\}.$$

It is further assumed throughout the paper that the matrix T is unreduced, that is, that none of the immediate sub- or superdiagonal elements of T is zero. Otherwise, the matrix would consist of a direct product of disjoint, lower order matrices whose eigendecompositions can be computed independently [24, p. 315]. Although an unreduced tridiagonal matrix has distinct eigenvalues in exact arithmetic, it may still have computationally coincident ones in finite precision.

Number the computed eigenvalues so that they satisfy the same order as the corresponding exact eigenvalues, i.e.,

$$\hat{\lambda}_1 \geqq \cdots \geqq \hat{\lambda}_n.$$

The accuracy of the computed eigendecomposition $\hat{U}\hat{\Lambda}\hat{U}^{-1}$ of T is then determined by the largest residual error $\mathcal{R}$ of any computed eigenpair and by the deviation from orthogonality $\mathcal{O}$ of the computed eigenvectors:

$$\mathcal{R} = \frac{1}{|\hat{\lambda}|_{\max}} \max_{1 \leqq i \leqq n} \|T\hat{u}_i - \hat{\lambda}_i \hat{u}_i\|_2, \qquad \mathcal{O} = \|\hat{U}^T\hat{U} - I\|_\infty,$$

where $|\hat{\lambda}|_{\max} = \max\{|\hat{\lambda}_1|, |\hat{\lambda}_n|\}$. The particular norms for $\mathcal{R}$ and $\mathcal{O}$ were chosen because they are convenient to analyze and to compute. The outcome of the numerical experiments in the later sections does not change if the matrix norm $\mathcal{O}$ is replaced by the vector norm $\max_{1 \leqq i \leqq n} \|(\hat{U}^T\hat{U} - I)e_i\|_\infty$. Our analysis is restricted to the above norm-based criteria; other quality measures that are applicable when T is known to very high accuracy are discussed in [2], [6]-[8].

Theorem 2.1 below shows that the computed eigendecomposition $\hat{U}\hat{\Lambda}\hat{U}^{-1}$ is the exact eigendecomposition of a matrix $T + E$ close to T if residuals and deviation from orthogonality are small. The error matrix E is in general neither symmetric nor tridiagonal; hence, $\hat{U}$ is not in general exactly orthogonal.

THEOREM 2.1. *Let $\hat{U}\hat{\Lambda}\hat{U}^{-1}$ be the computed eigendecomposition of a symmetric tridiagonal matrix T and let*

$$\mathcal{R} = \frac{1}{|\hat{\lambda}|_{\max}} \max_{1 \leqq i \leqq n} \|T\hat{u}_i - \hat{\lambda}_i \hat{u}_i\|_2, \qquad \mathcal{O} = \|\hat{U}^T\hat{U} - I\|_\infty.$$

Then there exists a matrix E such that

$$T + E = \hat{U}\hat{\Lambda}\hat{U}^{-1}, \qquad \|E\|_2 \leqq \frac{|\hat{\lambda}|_{\max}\sqrt{n}\,\mathcal{R}}{\sqrt{1 - \sqrt{n}\,\mathcal{O}}}.$$

Proof. Let

$$E_1 = \frac{1}{|\hat{\lambda}|_{\max}}(T\hat{U} - \hat{U}\hat{\Lambda}), \qquad E_2 = \hat{U}^T\hat{U} - I.$$

Because for any $n \times n$ matrix A, $\|A\|_2 \leqq \sqrt{n} \max_{1 \leqq i \leqq n} \|Ae_i\|_2$, where e_i is the ith canonical vector,

$$\|E_1\|_2 \leqq \sqrt{n} \max_{1 \leqq i \leqq n} \|E_1 e_i\|_2 = \sqrt{n}\,\mathcal{R}, \qquad \|E_2\|_2 \leqq \sqrt{n}\,\|E_2\|_\infty = \sqrt{n}\,\mathcal{O}.$$

From the definition of E_1,

$$T\hat{U}\hat{U}^{-1} - \hat{U}\hat{\Lambda}\hat{U}^{-1} = |\hat{\lambda}|_{\max} E_1 \hat{U}^{-1},$$

so that

$$\hat{U}\hat{\Lambda}\hat{U}^{-1} = T\hat{U}\hat{U}^{-1} - |\hat{\lambda}|_{\max} E_1 \hat{U}^{-1} = T + E,$$

where $E = -|\hat{\lambda}|_{\max} E_1 \hat{U}^{-1}$, and

$$\|E\|_2 = \| |\hat{\lambda}|_{\max} E_1 \hat{U}^{-1} \|_2 \leqq |\hat{\lambda}|_{\max} \|E_1\|_2 \|\hat{U}^{-1}\|_2 \leqq |\hat{\lambda}|_{\max} \sqrt{n}\, \mathcal{R} \|\hat{U}^{-1}\|_2.$$

We use the Neumann lemma [14, p. 59] to bound $\|\hat{U}^{-1}\|_2$:

$$\|\hat{U}^{-1}\|_2^2 = \|(\hat{U}^T\hat{U})^{-1}\|_2 = \|(I + (I - \hat{U}^T\hat{U}))^{-1}\|_2$$

$$\leqq \frac{1}{1 - \|I - \hat{U}^T\hat{U}\|_2} \leqq \frac{1}{1 - \sqrt{n}\, \mathcal{O}}.$$

Thus,

$$\|E\|_2 \leqq \frac{|\lambda|_{\max} \sqrt{n}\, \mathcal{R}}{\sqrt{1 - \sqrt{n}\, \mathcal{O}}}. \qquad \square$$

Under the assumptions

$$\|T\hat{U} - \hat{U}\hat{\Lambda}\|_2 \leqq \varepsilon_1, \qquad \|\hat{U}^T\hat{U} - I\|_2 \leqq \varepsilon_2,$$

the error matrix is bounded above by

$$\|E\|_2 \leqq \frac{|\hat{\lambda}|_{\max} \mathcal{R}}{\sqrt{1 - \mathcal{O}}},$$

which is independent of the matrix order. A related result is proven in [14, p. 416].

3. The EISPACK routine TINVIT. The steps for computing all eigenvectors of an unreduced symmetric tridiagonal matrix T by inverse iteration are given in Algorithm 3.1.

ALGORITHM 3.1 (Basic Implementation of Inverse Iteration)

```
For $j = n, n-1, \cdots, 1$:
    1. Choose a starting vector $y_j$.
    2. Solve $(T - \hat{\lambda}_j I) z_j = y_j$ for $z_j$.
    3. If the reorthogonalization criterion is satisfied, orthogonalize $z_j$ against
       iterates associated with computed eigenvalues close to $\hat{\lambda}_j$.
    4. If the stopping criterion is not satisfied, set $y_j = z_j / \|z_j\|_\infty$ and go to step 2.
    5. The computed eigenvector is $\hat{u}_j = z_j / \|z_j\|_2$.
```

As suggested in [23, p. 143] and [24, p. 329], the EISPACK implementation TINVIT performs the linear system solution in step 2 by Gaussian elimination with partial pivoting and the orthogonalizations in step 3 by the modified Gram–Schmidt algorithm. Because TINVIT yields less accurate eigenvectors than do existing implementations of the QL method (TQL2) [3], [22] or Cuppen's divide-and-conquer method (TREEQL) [5], [12], the loss in accuracy must be due to one or more of the following three factors: the choice of starting vector, the reorthogonalization criterion, and the stopping criterion. TINVIT deals with these issues as follows.

3.1. The starting vector. As argued in § 1, a good starting vector y_j has a large enough contribution of an eigenvector associated with the current eigenvalue λ_j to yield an iterate with dominant components in the eigenspace associated with λ_j.

Without advance knowledge of the eigenvectors, however, it is difficult to ensure a high quality starting vector. For instance, the canonical basis vectors e_1 and e_n should not be used as starting vectors because they are often nearly orthogonal to some eigenvectors of a symmetric tridiagonal matrix T [23, p. 147]. The vector of all ones is also a poor choice as it is orthogonal to half of the eigenvectors of a symmetric tridiagonal Toeplitz matrix [15].

Analytic determination of a good starting vector is complicated by the role of roundoff error in inverse iteration. As shown in [21], [23]-[25], one or two iterations in finite precision arithmetic are generally sufficient to produce a significant iterate component in the correct direction unless the starting vector is exactly orthogonal to that direction.

In agreement with [23, p. 147], TINVIT avoids explicit formation of the starting vector y_j as follows. The tridiagonal system $(T-\hat{\lambda}_j I)z_j = y_j$ is solved by using Gaussian elimination with partial pivoting to factor the matrix $T-\hat{\lambda}_j I = L_j U_j$. (Throughout this paper, we disregard the permutation matrix for simplicity.) The vector y_j is chosen so that the result of the forward substitution equals e, the vector of all ones. That is, the computation $L_j e = y_j$ need never be carried out, and the solution of the first linear system in each eigenvector computation amounts to solving only the second of the two triangular systems $U_j z_j = e$.

When computationally coincident eigenvalues (i.e., eigenvalues that are identical to working precision) are used as shifts, their iterates converge to a single eigenvector and fail to span the whole eigenspace. However, these iterates are very sensitive to the value of $\hat{\lambda}_j$ [24, p. 329]. Wilkinson suggests that the computationally coincident eigenvalues be slightly perturbed so as to make them distinct and that inverse iteration be used with the perturbed eigenvalues to produce iterates that are linearly independent. The increased distance of $\hat{\lambda}_{i+1}, \cdots, \hat{\lambda}_{i+k}$ from $\hat{\lambda}_i$ should affect only the speed of convergence and not the accuracy of inverse iteration [24, p. 329].

TINVIT replaces computationally coincident eigenvalues $\hat{\lambda}_i = \hat{\lambda}_{i-1} = \cdots = \hat{\lambda}_{i-k}$ by

$$\hat{\lambda}_i < \hat{\lambda}_i + \varepsilon_M \|T\|_R < \cdots < \hat{\lambda}_i + (k-1)\varepsilon_M \|T\|_R,$$

where ε_M is machine epsilon, and

$$\|T\|_R \equiv \max_{1 \leqq j \leqq n} \{|\alpha_j| + |\beta_j|\}$$

for a matrix T with diagonal elements $\alpha_1, \cdots, \alpha_n$ and off-diagonal elements $\beta_2, \cdots, \beta_n$, and $\beta_1 \equiv 0$. Note that $\|T\|_R$ does not satisfy the definition of a matrix norm but that an unreduced tridiagonal matrix T satisfies $\|T\|_R \leqq \|T\|_\infty$.

3.2. The reorthogonalization criterion. The above strategy for perturbing computationally coincident eigenvalues is intended to produce computed eigenvectors that are linearly independent. To assure orthogonal computed eigenvectors, the iterates associated with close eigenvalues are reorthogonalized against each other. In TINVIT, two adjacent eigenvalues $\hat{\lambda}_j$ and $\hat{\lambda}_{j+1}$ are considered close if

$$\hat{\lambda}_j - \hat{\lambda}_{j+1} < 10^{-3} \|T\|_R.$$

Note that eigenvalues *close* enough for reorthogonalization may or may not be *computationally coincident.* The process of reorthogonalizing the iterates proceeds as follows.

The eigenvectors are computed successively, according to the ascending order of the eigenvalues. That is, at the time of computation of $\hat{u}_j$, the computation of the eigenvectors $\hat{u}_{j+1}, \cdots, \hat{u}_n$ has already been completed. If a computed eigenvalue $\hat{\lambda}_j$ is close to the computed eigenvalue $\hat{\lambda}_{j+1}$, then the iterate z_j is reorthogonalized against $\hat{u}_{j+1}$ and against all eigenvectors against which $\hat{u}_{j+1}$ was orthogonalized. In the outline of TINVIT in § 3.4, the data structure CLUSTER(i) contains the indices $i+1, \cdots, i+k$ of all those vectors, against which z_i must be orthogonalized.

3.3. The stopping criterion. In [23, p. 145], Wilkinson shows that if an iterate z_j has a large norm after reorthogonalization but before normalization, the eigenpair $(\hat{\lambda}_j, z_j)$ has a small residual error. Specifically, the large iterate norm $\varepsilon_M \|z_j\|_2 = \Omega(n^{-1/2})$ leads to the small residual $\|(T-\hat{\lambda}_j I)z_j\|_2 = O(\varepsilon_M n^{1/2})$ [23, p. 145]. Furthermore, the large norm of z_j (after reorthogonalization) indicates that the iterates associated with $\hat{\lambda}_{j+1}, \cdots, \hat{\lambda}_n$ were linearly independent (before reorthogonalization) so that the computed eigenvector $\hat{u}_j = z_j/\|z_j\|_2$ is orthogonal to $\hat{u}_{j+1}, \cdots, \hat{u}_n$. (The connection between large iterate norm and successful orthogonalization by the modified Gram-Schmidt procedure is demonstrated in § 4.1.)

From the perturbation result in § 2 we can then conclude that $(\hat{\lambda}_j, z_j)$ is an eigenpair of a matrix close to T and hence that z_j is an accurate eigenvector. Because $\|z_j\|_2 \geqq \|z_j\|_\infty$, the two-norm can be replaced by the cheaper infinity norm for convergence testing. Thus, if $\|y_j\|_\infty = 1$ and $\varepsilon_M \|z_j\|_\infty > 1$, then z_j is a good eigenvector approximation. The difficulty lies in determining just how large $\|z_j\|_\infty$ should be. TINVIT stops iteration if $\varepsilon_M \|z\|_\infty \geqq 1$ (ignoring scaling factors).

3.4. Implementation of inverse iteration. A sketch of the EISPACK routine TINVIT is given as Algorithm 3.2 below. Numerical details such as scaling factors used to prevent overflow are not included. The computed eigenvalues are in descending order $\hat{\lambda}_1 \geqq \cdots \geqq \hat{\lambda}_n$, and ε_M is machine epsilon.

ALGORITHM 3.2 (Outline of TINVIT)

```
For $j = n, n-1, \cdots, 1$
    0. Perturb computationally coincident eigenvalues: if $j < n$ and $\hat{\lambda}_j - \hat{\lambda}_{j+1} \leqq 0$, then replace $\hat{\lambda}_j$ with $\hat{\lambda}_{j+1} + \varepsilon_M \|T\|_R$.
    1. Initialize the set of eigenvalues close to $\hat{\lambda}_j$: $CLUSTER(j) = \emptyset$. If $j < n$ and $\hat{\lambda}_j - \hat{\lambda}_{j+1} < 10^{-3}\|T\|_R$, then $CLUSTER(j) = CLUSTER(j+1) \cup \{j+1\}$.
    2. Initialize the iterate norm $\sigma_j \equiv 0$.
    3. Factor $T - \hat{\lambda}_j I = L_j U_j$.
    4. Loop until $\varepsilon_M \sigma_j \geqq 1$ (error exit after 5 iterations).
        4.a. If this is the first iteration, solve $U_j z_j = e$, otherwise solve $L_j U_j z_j = y_j$.
        4.b. Reorthogonalize $z_j$ against all $\hat{u}_i$ with $i \in CLUSTER(j)$.
        4.c. Set $\sigma_j = \|z_j\|_\infty$ and $y_j = z_j/\sigma_j$.
    5. The computed eigenvector is $\hat{u}_j = y_j/\|y_j\|_2$.
```

4. Experimental results. The experimental results in [17] show that TQL2 or TREEQL generally yield residuals $\mathcal{R} = (1/|\hat{\lambda}|_{\max}) \max_i \|T\hat{u}_i - \hat{\lambda}_i \hat{u}_i\|_2$ less than 10^{-14} for matrix orders $n \leqq 525$; and deviations from orthogonality $\mathcal{O} = \|\hat{U}^T\hat{U} - I\|_\infty$ less than 10^{-14} for $n \approx 32$, less than 10^{-13} for $n \approx 100$, and less than 10^{-12} for $n \approx 512$. (A similar dependence on the matrix order occurs when the deviation from orthogonality is instead measured by $\mathcal{O} = \max_i \|(\hat{U}^T\hat{U} - I)e_i\|_2$.) The EISPACK routine TSTURM (a combination of BISECT and TINVIT) yields respective residuals $\mathcal{R}$ less than 10^{-14}, 10^{-13}, and 10^{-12} and orthogonality measures $\mathcal{O}$ less than 10^{-12}, 10^{-11}, and 10^{-10} for matrix orders 32, 100, and 512, respectively.

The numerical experiments in this section were designed to determine which features of TINVIT need to be modified so that it is at least as accurate in practice as the QL routine TQL2 [22] and the divide-and-conquer routine TREEQL [12]. All experiments were performed in double precision on a single Sequent Symmetry S81 processor using the Weitek 1167 floating-point accelerator. The eigenvalues were computed with the EISPACK routine BISECT to working precision. On this machine, machine epsilon $\varepsilon_M = 2.22 \times 10^{-16}$.

This paper presents representative results selected from the ones in [17]. The first test matrix [1, 2, 1] illustrates the case of matrices without close eigenvalues. The matrix [1, 2, 1] is a symmetric tridiagonal Toeplitz matrix of order n having twos on the diagonal and ones on the first sub- and superdiagonals. Its exact eigenvalues are well separated and given by [15]

$$\lambda_j = 2\left(1 + \cos \frac{j\pi}{n+1}\right), \qquad 1 \leqq j \leqq n.$$

For matrix orders $n \leqq 525$, the computed eigenvalues $\hat{\lambda}_i$ are also well separated.

The second test matrix is the "glued Wilkinson matrix" W_g^+ and represents one of the most difficult test cases for dealing with groups of close eigenvalues. It is constructed as follows: The "Wilkinson matrix" W_{21}^+ of order $n = 21$ has diagonal elements $10, 9, \cdots, 1, 0, 1, \cdots, 9, 10$ and immediate off-diagonal elements equal to one. It possesses pairs of eigenvalues that are very close [24, p. 309]. The spacing between eigenvalues in a pair decreases with increasing magnitude of the eigenvalues, and the eigenvalues in the largest pairs are computationally coincident with regard to double precision. The glued Wilkinson matrix W_g^+ of order $21j$ is formed by placing j copies of W_{21}^+ along the diagonal of the matrix and setting off-diagonal elements equal to 10^{-14} at the positions $\beta_{21}, \beta_{42}, \cdots$ where the submatrices join. For matrix orders $n > 200$, W_g^+ has clusters of eigenvalues near the integers $1, 2, \cdots, \lfloor n/2 \rfloor$ [20].

The conclusions drawn from numerical experiments with these two matrix types are supported by tests on random matrices in [17].

4.1. Starting vectors. In this section, we examine the influence of the starting vector on the accuracy of inverse iteration and on the number of iterations performed. To this end, we use the following vectors as starting vectors for the computation of $\hat{u}_j$.

1. The "correct" eigenvector $\hat{u}_j$: This starting vector is the eigenvector $\hat{u}_j$ computed by inverse iteration with a random starting vector. Each starting vector $\hat{u}_1, \cdots, \hat{u}_n$ has residuals $\mathcal{R} < 10^{-14}$ for all orders and orthogonalities $\mathcal{O} < 10^{-14}$ for $n \leqq 42$, $\mathcal{O} < 10^{-13}$ for $n \leqq 105$, and $\mathcal{O} < 10^{-12}$ for $n \leqq 525$.

2. $\hat{u}_n + \tau \hat{u}_j$: This linear combination of $\hat{u}_n$ and $\hat{u}_j$ is used as the starting vector to compute $\hat{u}_j$ for $1 \leqq j \leqq n-1$, and a random starting vector is used to compute $\hat{u}_n$. When $\tau = 0$ and $\hat{\lambda}_{n-1} > \hat{\lambda}_n$, $\hat{u}_n + \tau \hat{u}_j$ is roughly orthogonal to the eigenvectors associated with $\lambda_1, \cdots, \lambda_{n-1}$. Increasing the value of τ amounts to increasing the contribution of the desired eigendirection in the starting vector and thus determines the minimal size of the contribution that is sufficient for convergence.

3. Random vectors: These vectors have uniformly distributed pseudorandom components between -1 and 1 generated with the linear congruential random number generator from NETLIB [11]. For each matrix order n, a single $n \times n$ random matrix is generated. In one set of experiments, we use the first column of this matrix as the starting vector for *all* eigenvectors. In the second set of experiments, we use column j of the matrix as the starting vector for the jth eigenvector.

4. The TINVIT starting vector y_j: This starting vector is not computed explicitly. Instead it is assumed to be the right-hand side of the lower triangular system $L_j e = y_j$, where e is the vector of all ones, and $T - \hat{\lambda}_j I = L_j U_j$ is the LU decomposition (disregarding the permutation matrix).

For the purposes of this section, TINVIT was modified to perform the same number of iterations for all eigenvectors (the number of iterations is determined by the required accuracy but does not exceed five). Computationally coincident eigenvalues were perturbed as in step 0 of Algorithm 3.2 except when different starting vectors were used for each shift. For different random starting vectors, the rate of convergence and accuracy of inverse iteration are preserved even if computationally coincident eigenvalues are not perturbed, that is, even if step 0 of TINVIT is omitted. The following two sections distinguish between the experimental results for the cases of well separated and close eigenvalues.

4.1.1. Starting vectors for matrices with well-separated eigenvalues. Table 1 shows the number of iterations required by inverse iteration to compute the eigenvectors to the same accuracy as TQL2 or TREEQL for each type of starting vector.

TABLE 1
Number of inverse iterations to compute eigenvector $\hat{u}_j$ for matrix $[1, 2, 1]$ *of order n. The same number of iterations is performed for each $\hat{u}_j$.*

Starting vector	$n=32$ Number of iterations for $\mathcal{R}<10^{-14}$ $\mathcal{O}<10^{-14}$	$n=100$ Number of iterations for $\mathcal{R}<10^{-14}$ $\mathcal{O}<10^{-13}$	$n=512$ Number of iterations for $\mathcal{R}<10^{-14}$ $\mathcal{O}<10^{-12}$
$\hat{u}_j$	1	1	1
$\hat{u}_n$	2	2	4
$\hat{u}_n+10^{-16}\hat{u}_j$	2	2	3
$\hat{u}_n+10^{-8}\hat{u}_j$	2	2	2
$\hat{u}_n+10^{-2}\hat{u}_j$	2	2	2
same random	2	2	2
different random	2	2	2
TINVIT	2	2	2

High accuracy is achieved in one iteration only when accurately computed eigenvectors $\hat{u}_j$ are the starting vectors. More than two iterations are needed only for larger n and only when the starting vector is orthogonal or nearly orthogonal to the computed eigenvector ($\tau \leqq 10^{-16}$). All other starting vectors require two iterations.

Thus, for matrices $[1, 2, 1]$ of order $n \leqq 512$, a starting vector component η_j of magnitude 10^{-8} in the desired direction u_j suffices for rapid convergence, i.e., two iterations. Performing more iterations than listed in Table 1 does not significantly change the accuracy. These results are supported by numerical experiments on random matrices with minimal eigenvalue spacing of 10^{-4} [17].

In summary, when all eigenvalues are well separated the performance of inverse iteration does not strongly depend on the starting vector: random starting vectors and the TINVIT starting vector provide a large enough component in the desired direction for fast convergence.

4.1.2. Starting vectors for matrices with groups of close eigenvalues. Table 2 shows the number of iterations for the glued Wilkinson matrix W_g^+ with $n=42$, 105, and 525 for the different starting vectors. As for matrix $[1, 2, 1]$, accurate eigenvectors are produced in one iteration only when the starting vector is the eigenvector. Two iterations suffice when the starting vector has a correct component of size at least 10^{-8} or when a different random starting vector is used for each eigenvector. The remaining starting

TABLE 2

Number of inverse iterations to compute eigenvector $\hat{u}_j$ for the glued Wilkinson matrix W_g^+ of order n. The same number of iterations is performed for each $\hat{u}_j$.

Starting vector	$n=42$ Number of iterations for $\mathcal{R}<10^{-14}$ $\mathcal{O}<10^{-14}$	$n=105$ Number of iterations for $\mathcal{R}<10^{-14}$ $\mathcal{O}<10^{-13}$	$n=525$ Number of iterations for $\mathcal{R}<10^{-14}$ $\mathcal{O}<10^{-12}$
$\hat{u}_j$	1	1	1
$\hat{u}_n$	3	3	>5
$\hat{u}_n+10^{-16}\hat{u}_j$	3	3	>5
$\hat{u}_n+10^{-8}\hat{u}_j$	2	2	2
$\hat{u}_n+10^{-2}\hat{u}_j$	2	2	2
same random	2	3	3
different random	2	2	2
TINVIT	2	3	>5

vectors require more than two iterations. For $n = 525$, inverse iteration does not converge in five iterations when the iterations are started with $\hat{u}_n$, $\hat{u}_n+10^{-16}\hat{u}_j$ or with the TINVIT starting vector.

Table 3 illustrates the connection between the convergence rate of the iterates and their linear dependence for different types of starting vectors and the matrix W_g^+. The numbers in Table 3 were obtained as follows: The iterates z_j, $1 \leqq j \leqq n$, before the reorthogonalization step 4.b in the *first* iteration of Algorithm 3.2 compose the columns of an $n \times n$ matrix. The smallest singular value of this matrix is listed in the first column of numbers, and the smallest norm σ_j attained by the z_j, $1 \leqq j \leqq n$, after reorthogonalizing in step 4.b is listed in the second column of numbers. The same information is given for z_j in the second iteration of Algorithm 3.2 in the last two columns. These data show that except in the case of different random starting vectors, the iterates after the first iteration are linearly dependent. Thus, the modified Gram-Schmidt procedure breaks down and produces vectors that are almost zero. Likewise, the second iteration fails to produce linearly independent iterates for all but the different random starting vectors. (The singular value for the matrix of iterates from the same random starting vector is so small, 10^{-18}, that the iterates can be considered numerically linearly dependent.)

TABLE 3

Singular values of the matrix of iterates and the smallest iterate norm for the glued Wilkinson matrix W_g^+ of order $n=525$ after one and after two iterations of TINVIT.

Starting vector	Smallest singular value of first iterates	Minimal iterate norm $\min_j \|z_j\|_\infty$ after one iteration	Smallest singular value of second iterates	Minimal iterate norm $\min_j \|z_j\|_\infty$ after two iterations
$\hat{u}_n$	0	0	0	$1.24d-13$
same random vector	0	$4.69d-12$	10^{-18}	$7.04d-04$
different random vector	0.02	$4.94d-04$	0.08	>1.00
TINVIT	0	$4.94d-12$	0	$1.06d-12$

While the TINVIT starting vectors are difficult to analyze, the other choices suggest a possible correlation between linearly dependent starting vectors and iterates: linearly dependent starting vectors lead to linearly dependent iterates in the case of computationally coincident eigenvalues. Table 4 shows that the smallest singular value for a matrix composed of n different random starting vectors is much larger than zero; the only exception is $n=512$, where the 512th column is linearly dependent on the first 479. Because none of the test matrices has a group of eigenvalues including both the 479th and the 512th eigenvalues, inverse iteration starts out with linearly independent random starting vectors for all iterates associated with the same group of close eigenvalues.

TABLE 4
The smallest singular value for a matrix of different random starting vectors.

Matrix order	Smallest singular value
42	0.0362
100	0.0341
105	0.0198
512	$1.d-229$
525	0.0128

In summary, when a different random starting vector is used to compute each eigenvector of the glued Wilkinson matrix, both the starting vectors and the iterates are highly likely to be linearly independent. This correlation between linear dependence of starting vectors and number of iterations can be observed to a lesser degree for other large matrices with groups of close eigenvalues [17].

4.2. Stopping criterion. The experimental results in this section show that TINVIT's choice of stopping criterion causes inverse iteration to stop before highest accuracy is attained. We will examine an alternative that consistently improves the accuracy. For the experiments in this section, TINVIT was modified to compute each eigenvector from a different random starting vector and to use unperturbed computed eigenvalues as shifts.

For matrices $[1,2,1]$ and W_g^+, Table 5 shows how the accuracy of the computed eigendecomposition depends on the norm of the computed iterates (after reorthogonalization in step 4.b of Algorithm 3.2). The TINVIT stopping criterion works correctly for both orders of the glued Wilkinson matrix W_g^+: unit iterate norm and full accuracy are both attained on the second iteration. It fails, however, on the matrix $[1,2,1]$, where all iterates have greater than unit norm but less than full accuracy on the first iteration. The same conclusions can be drawn from experiments with random matrices in [17]. It seems, therefore, that at least two iterations should always be performed regardless of iterate norm when different random starting vectors are used. In other words, after the iterate norm is large enough and the loop in step 4 of Algorithm 3.2 is exited, the algorithm should perform one more iteration. The additional iteration was already suggested in [24, p. 324], but was not implemented in TINVIT.

We have not found a simple correlation between size of the iterate norms and the number or size of the groups of close eigenvalues.

4.3. Reorthogonalization. For the purpose of this section, TINVIT was modified to compute each eigenvector from a different random starting vector and to perform

TABLE 5

Iterate norm, residual, and orthogonality for matrices [1, 2, 1] *and* W_g^+ *after one and after two inverse iterations. A different random starting vector is used for each eigenvector computation.*

Matrix	Iteration	Minimal iterate norm $\min_j \|z_j\|_\infty$	Maximal residual $\mathscr{R}$	Deviation from orthogonality $\mathscr{O}$
$[1, 2, 1](n = 100)$	1	>1.00	$3.18d-14$	$3.05d-12$
	2	>1.00	$1.58d-16$	$3.20d-14$
$W_g^+(n = 105)$	1	0.14	$1.47d-13$	$1.68d-11$
	2	>1.00	$8.70d-16$	$3.85-15$
$[1, 2, 1](n = 512)$	1	>1.00	$1.60d-11$	$4.62d-09$
	2	>1.00	$3.93d-16$	$1.57d-13$
$W_g^+(n = 525)$	1	$4.94d-04$	$3.42d-09$	$6.02d-07$
	2	>1.00	$5.99d-15$	$1.98d-14$

one more iteration after the iterate norm becomes large enough, i.e., one more iteration after exiting the loop in step 4 of Algorithm 3.2. In the numerical experiments below, we vary the distance at which adjacent eigenvalues are considered to be so close as to require orthogonalization of the associated iterates.

Table 6 shows the residuals $\mathscr{R}$ and deviations from orthogonality $\mathscr{O}$ for matrix [1, 2, 1] of order $n = 100$ as the reorthogonalization criterion is varied from 0 to $10^{10}\|T\|_R$ after one and after two inverse iterations. These data confirm that more orthogonalization is not a substitute for extra iterations because small residuals are not attained until the second iteration even with reorthogonalization of all eigenvectors. Table 7 shows the same situation for W_g^+ when $n = 105$ and $n = 525$, as well as the fraction of inverse iteration time spent in the modified Gram-Schmidt procedure.

Increasing the reorthogonalization criterion beyond that of TINVIT does not significantly improve the accuracy for matrices [1, 2, 1] and W_g^+. It can, however, substantially increase the computation time. With the TINVIT criterion $10^{-3}\|T\|_R$, most of the eigenvectors of W_g^+ are reorthogonalized (80 percent when $n = 105$ and 96 percent when $n = 525$), but reorthogonalization occurs in many small groups. In contrast, with the criterion $10^{10}\|T\|_R$ all eigenvectors are reorthogonalized as one group, and the cost rises markedly although the accuracy hardly changes.

TABLE 6

Accuracy for matrix [1, 2, 1] *with different reorthogonalization criteria when* $n = 100$.

Criterion	Number of vectors orthogonalized	One iteration		Two iterations	
		$\mathscr{R}$	$\mathscr{O}$	$\mathscr{R}$	$\mathscr{O}$
$10^{10}\|T\|_R$ (all)	99	$6.09d-13$	$6.41d-15$	$2.12d-16$	$5.75d-15$
$10^{-1}\|T\|_R$	97	$1.87d-12$	$7.40d-15$	$2.13d-16$	$6.12d-15$
$10^{-2}\|T\|_R$	33	$7.69d-14$	$4.97d-12$	$1.67d-16$	$1.82d-15$
$10^{-3}\|T\|_R$	2	$3.18d-13$	$3.05d-12$	$1.58d-16$	$3.20d-14$
$10^{-5}\|T\|_R$	1	$1.10d-13$	$2.75d-11$	$1.90d-16$	$4.28d-14$
0	0	$2.28d-13$	$2.68d-11$	$1.61d-16$	$4.68d-14$

TABLE 7

Accuracy and computation time for W_g^+ after two inverse iterations with different reorthogonalization criteria when $n = 100$ and $n = 525$. The last column shows the fraction of reorthogonalization (MGS) *time in inverse iteration.*

Order	Criterion	$\mathcal{R}$	$\mathcal{O}$	Number of vectors	Time for inverse iteration	Fraction MGS time
$n = 105$	$10^{10}\|T\|_R$ (all)	$1.97d-16$	$4.57d-15$	104	30.6	0.67
	$10^{-1}\|T\|_R$	$1.60d-16$	$3.14d-15$	88	27.7	0.10
	$10^{-3}\|T\|_R$	$8.70d-16$	$3.85d-15$	84	1.4	0.07
	$10^{-5}\|T\|_R$	$2.09d-16$	$2.52d-13$	78	1.4	0.05
	0	$1.85d-16$	2.05	0	0	0
$n = 525$	$10^{10}\|T\|_R$ (all)	$7.58d-15$	$1.53d-14$	524	5807.1	0.98
	$10^{-1}\|T\|_R$	$4.69d-15$	$3.51d-14$	523	5287.6	0.73
	$10^{-3}\|T\|_R$	$5.99d-15$	$1.98d-14$	504	338.1	0.15
	$10^{-5}\|T\|_R$	$3.46d-15$	$6.24d-13$	498	253.1	0.12
	0	$2.04d-16$	6.38	0	0	0

These experiments show that the best possible orthogonality can generally be attained only by the time-consuming process of reorthogonalizing all eigenvectors. The accuracy desired here, however, can usually be achieved by means of the TINVIT reorthogonalization criterion along with different random starting vectors and the improved stopping criterion of § 4.2.

5. A new implementation of inverse iteration. The improvements to inverse iteration developed in § 4 are incorporated into the following algorithm. These changes are based on experiments in § 4 with matrix orders $n \leqq 525$ and may not apply to much larger matrix orders. Some comments on solving larger order problems conclude this section.

ALGORITHM 5.1 (Improved Inverse Iteration Algorithm (III))

```
For $j = n, n-1, \cdots, 1$
    1. Initialize the set of eigenvalues close to $\hat{\lambda}_j$: $CLUSTER(j) = \emptyset$.
       If $j < n$ and $\hat{\lambda}_j - \hat{\lambda}_{j+1} < 10^{-3}\|T\|_R$, then
       $CLUSTER(j) = CLUSTER(j+1) \cup \{j+1\}$.
    2. Generate a random vector $x_j$ with uniformly distributed components in
       the interval $[-1, 1]$, and form the starting vector $y_j = x_j / \|x_j\|_2$.
    3. Initialize the iterate norm $\sigma_j \equiv 0$.
    4. Factor $T - \hat{\lambda}_j I = L_j U_j$.
    5. Loop until $\varepsilon_M \sigma_j \geqq 1$ (error exit after 5 iterations).
       5.a. Solve $L_j U_j z_j = y_j$.
       5.b. Reorthogonalize $z_j$ against all $\hat{u}_i$ with $i \in CLUSTER(j)$.
       5.c. Set $\sigma_j = \|z_j\|_\infty$ and $y_j = z_j / \sigma_j$.
    6. Repeat step 5 once.
    7. The computed eigenvector is $\hat{u}_j = y_j / \|y_j\|_2$.
```

Tables 8 and 9 compare the computation time of the EISPACK routine TSTURM with that of the EISPACK routine BISECT combined with Algorithm III (B/III). Because of the additional iterations performed, the computation time of III is substantially higher than that of TINVIT. For matrix $[1, 2, 1]$ of order $n = 512$, however, very

TABLE 8
Times, residuals, and orthogonalities for eigensystems computed by TSTURM *and by* B/III *for matrix* [1, 2, 1].

	TSTURM				B/III		
n	Time to compute eigenvalues (seconds)	Time to compute eigenvectors (seconds)	$\mathcal{R}$	$\mathcal{O}$	Time to compute eigenvectors (seconds)	$\mathcal{R}$	$\mathcal{O}$
32	1.1	0.3	4.15d − 15	4.00d − 13	0.4	1.30d − 16	4.27d − 15
100	11.3	2.0	2.46d − 14	8.48d − 12	3.2	1.56d − 16	3.15d − 14
512	276.7	72.2	1.26d − 13	4.48d − 11	125.8	4.11d − 16	1.78d − 13

TABLE 9
Times, residuals, and orthogonalities for eigensystems computed by TSTURM *and by* B/III *for matrix* W_g^+.

	TSTURM				B/III		
n	Time to compute eigenvalues (seconds)	Time to compute eigenvectors (seconds)	$\mathcal{R}$	$\mathcal{O}$	Time to compute eigenvectors (seconds)	$\mathcal{R}$	$\mathcal{O}$
42	1.6	0.4	4.25d − 15	2.3d − 13	0.6	1.61d − 16	2.61d − 15
105	4.72	3.0	5.11d − 14	2.36d − 12	4.8	6.98d − 16	4.43d − 15
525	23.3	171.1	1.14d − 13	4.08d − 11	333.4	5.55d − 15	1.69d − 14

little orthogonalization of eigenvectors takes place (see Table 6), and eigenvector computation is cheap compared to eigenvalue computation. The longer time of Algorithm III represents only a 13 percent increase in total computation time for B/III over TSTURM.

The storage requirements for Algorithm III are the same as for TINVIT. The time for generation of random starting vectors in Algorithm III is small compared to the total computation time. It constitutes less than 4 percent of the total eigenvector computation time for matrix [1, 2, 1] of order $n \leqq 512$ and for W_g^+ of order $n \leqq 525$. A 1000×1000 matrix of random elements can be generated in 14.00 seconds.

Algorithm III allows accurate solution of problems up to about order 525. It thus gives a roughly four-fold increase of the order solved accurately (or at all) by TINVIT. It also provides guidelines for solving problems of larger order. Experiments performed on an IBM RS6000/520 after the compilation of data for this paper show that with minor modification, the new implementation allows accurate solution of all tested problems up to order 2000. (No larger orders were tried.) To achieve this accuracy, it is necessary to perform two (rather than one) extra iterations after the iterate norm grows sufficiently large. It is also necessary to separate computationally coincident eigenvalues by a perturbation that is small relative to the magnitude of the eigenvalues. Because the longer vectors are likely to have smaller infinity norms, it is also possible to decrease the stopping criterion. If the maximum number of iterations is reached without sufficient norm growth, it is advisable to compute the residual of the computed vector before accepting it as a computed eigenvector. Even with a small residual, the computed vector may not be fully orthogonal to the other computed eigenvectors. A complete version of this code is available as routine DSTEIN in LAPACK [1].

6. Comparison with other methods. This section offers an experimental comparison of Cuppen's divide-and-conquer method, the QL method, and bisection with inverse iteration. The respective implementations are TREEQL [12], TQL2 [22], and B/III. TREEQL switches from divide and conquer to TQL2 for subproblems of order $n \leqq 50$. For a given problem, the relative speeds of the three methods depend on the degree of deflation in TREEQL, the amount of matrix splitting in TQL2, and the clustering of eigenvalues for B/III. Because they illustrate the range of results for all matrices from [17], we use the three test matrices $[1, 2, 1]$, W_g^+, and $[1, u, 1]$ as a basis for comparison. The matrix $[1, u, 1]$ has ones in its first subdiagonal and superdiagonal and the value $i \times 10^{-6}$ in the ith diagonal position. It undergoes little deflation when its eigendecomposition is computed by TREEQL. As none of these test matrices contains row sums of widely differing magnitudes, we exclude IMTQL2 from the comparison: the performance and accuracy of TQL2 and IMTQL2 are nearly identical for these test matrices [17].

Table 10 shows that the maximal residual $\mathcal{R} = (1/|\hat{\lambda}|_{\max}) \max_{1 \leqq i \leqq n} \|T\hat{u}_i - \hat{\lambda}_i \hat{u}_i\|_2$ and deviation from orthogonality $\mathcal{O} = \|\hat{U}^T\hat{U} - I\|_\infty$ of the eigendecompositions computed by the three methods do not differ significantly.

TABLE 10
Maximal residual and deviation from orthogonality of eigendecompositions computed by B/III, TREEQL, *and* TQL2 *for matrices* $[1, 2, 1]$, W_g^+, $[1, u, 1]$.

Matrix order	Method	Maximal residual $\mathcal{R}$	Deviation from orthogonality $\mathcal{O}$
$n = 32$ or 42			
	TREEQL	$3.26d-15$	$5.59d-15$
	TQL2	$1.52d-15$	$1.30d-14$
	B/III	$1.80d-16$	$6.20d-15$
$n = 100$ or 105			
	TREEQL	$6.07d-14$	$2.75d-15$
	TQL2	$2.39d-15$	$1.06d-14$
	B/III	$3.67d-15$	$8.52d-14$
$n = 512$ or 525			
	TREEQL	$3.96d-15$	$1.67d-13$
	TQL2	$1.66d-14$	$2.50d-13$
	B/III	$6.05d-15$	$7.92d-13$

As shown in Figs. 1-3, however, the computation times for the problems can differ significantly. The top graphs in these figures show the different computation times for matrix orders $n \leqq 60$. B/III is slowest for matrices $[1, 2, 1]$ and $[1, u, 1]$ of order $n \leqq 20$. TQL2 is fastest for $[1, 2, 1]$ and $[1, u, 1]$ of order $n \leqq 20$ and slowest for all matrices of order $50 \leqq n \leqq 525$. TREEQL is fastest for W_g^+ of all orders and for $[1, 2, 1]$ of order $20 \leqq n \leqq 60$ due to moderate ($[1, 2, 1]$) and heavy (W_g^+) deflation. The bottom graphs in Figs. 1-3 show the different computation times for matrix orders $60 \leqq n \leqq 512$. For $n = 512$, TREEQL is about 2 to 40 times faster than TQL2, and B/III is about eight times faster than TQL2.

Because the degree of deflation and the grouping of eigenvalues are rarely known in advance, it is generally not possible to select the fastest serial method for a given

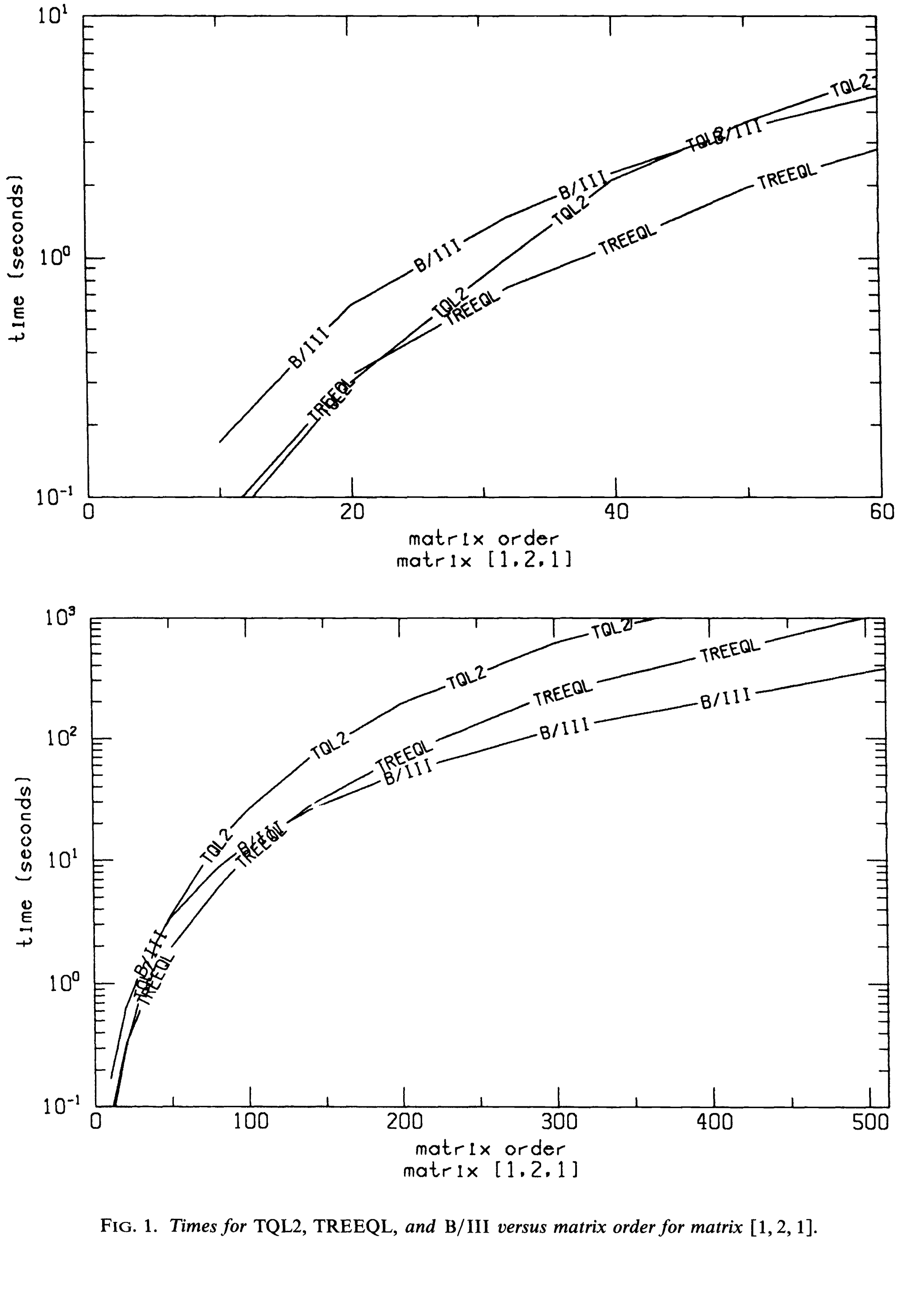


FIG. 1. *Times for* TQL2, TREEQL, *and* B/III *versus matrix order for matrix* [1, 2, 1].

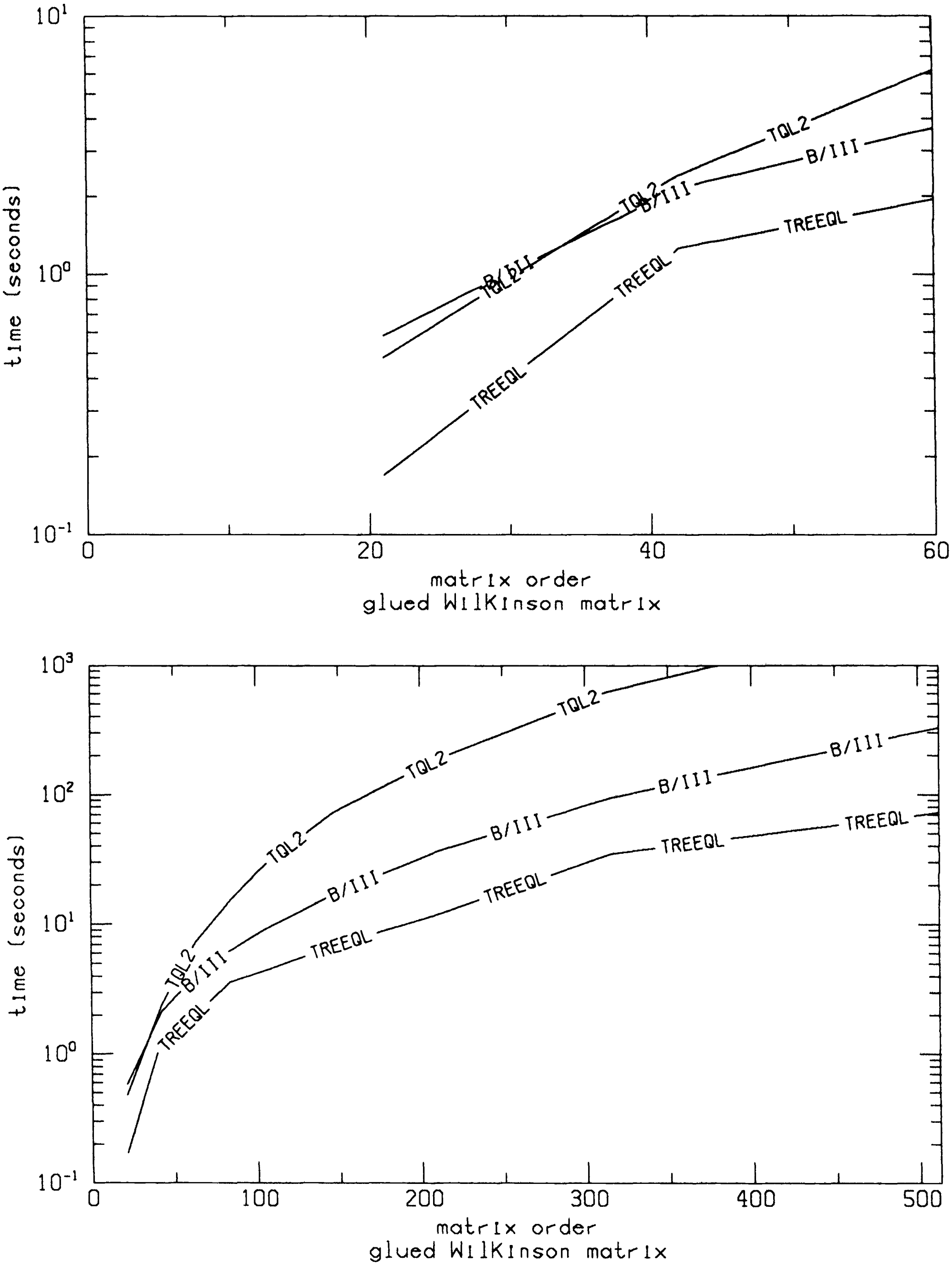


FIG. 2. *Times for* TQL2, TREEQL, *and* B/III *versus matrix order for the glued Wilkinson matrix* W_g^+.

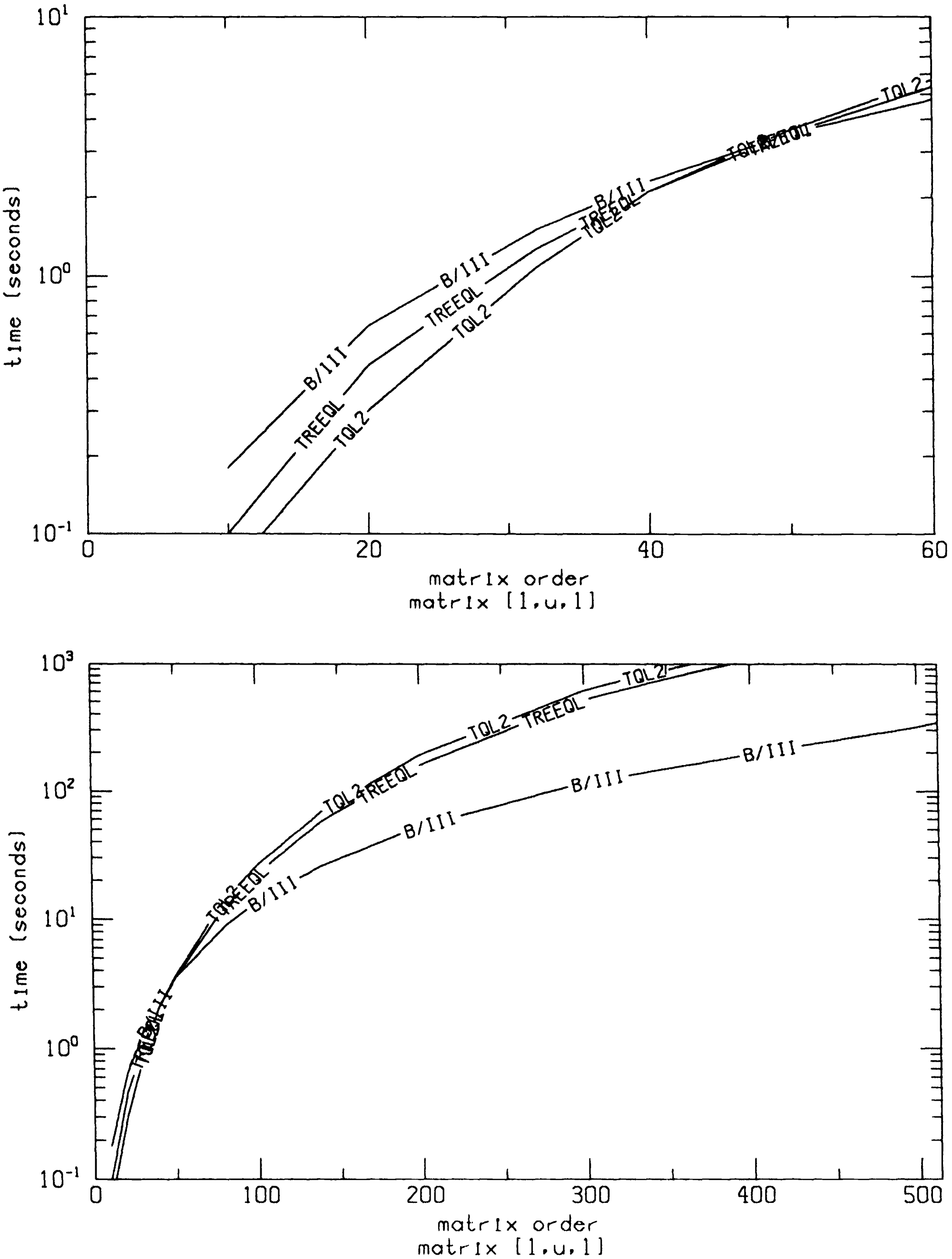


FIG. 3. *Times for* TQL2, TREEQL, *and* B/III *versus matrix order for matrix* [1, *u*, 1].

problem. In all of our experiments, however, B/III is much faster than TQL2 and equally accurate. For larger matrix orders, B/III is fastest for light to medium deflation, while TREEQL is fastest for heavy deflation.

7. A statistical analysis of inverse iteration. The preceding sections establish experimentally the design choices for an accurate implementation of inverse iteration. Because they rely on the use of starting vectors with randomly distributed components, we now give a statistical analysis to explain some of the experimental observations. Section 7.1 states our assumptions, § 7.2 defines a good eigenvector approximation, § 7.3 determines the expected quality of a random starting vector and briefly discusses the limitations of the analysis, and § 7.4 estimates the error in applying the analysis based on starting vectors with normally distributed components to starting vectors with uniformly distributed components. Statistical analyses of methods for computing *eigenvalues* can be found, for example, in [10], [19].

7.1. Assumptions. A unit-norm starting vector $y = x/\|x\|_2$ is computed from a vector x with independent random components, each of which has a normal distribution with mean 0 and variance 1 (normal (0, 1)). Such vectors y are uniformly distributed on the unit n-sphere [9]. Because the distribution of their components is invariant under orthogonal transformations, these vectors can be represented in terms of the orthonormal basis of eigenvectors $\{u_1, u_2, \cdots, u_n\}$ of the symmetric tridiagonal matrix T:

$$y = (\eta_1, \eta_2, \cdots, \eta_n)^T, \qquad \text{where } y = \sum_{i=1}^{n} \eta_i u_i, \qquad \|u_i\|_2 = 1, \qquad \sum_{i=1}^{n} \eta_i^2 = 1.$$

7.2. The quality of an approximate eigenvector. A vector y as defined above is a good approximation to u_i if η_i is much larger than any other component, i.e., if $\eta_i^2 \geqq 1 - \varepsilon^2$ for some error tolerance $0 \leqq \varepsilon \leqq 1$. Geometrically, the angle θ_i between y and u_i satisfies $\cos \theta_i \geqq \sqrt{1 - \varepsilon^2}$. Similarly, y is a good approximation to a linear combination of eigenvectors $u_1, \cdots, u_d$ if $\sum_{i=1}^{d} \eta_i^2 \geqq 1 - \varepsilon^2$. Because random vectors are uniformly distributed on the sphere, the probability that y is a good approximation to the linear combination is just the fraction of the surface area of the sphere defined by all vectors whose components $\xi_1, \cdots, \xi_d$ satisfy $\sum_{i=1}^{d} \xi_i^2 \geqq 1 - \varepsilon^2$.

The probability that $\sum_{i=1}^{d} \eta_i^2 \geqq 1 - \varepsilon^2$ is determined by integrating the probability density function of the sum $\sum_{i=1}^{d} \eta_i^2$ between $1 - \varepsilon^2$ and 1. If the component ξ_i has a normal (0, 1) distribution, then $\eta_i^2 = \xi_i^2 / \sum_{j=1}^{n} \xi_j^2$ has the beta distribution $B(\frac{1}{2}, \frac{n-1}{2})$, and the sum $\sum_{i=1}^{d} \eta_i^2$ has a $B(\frac{d}{2}, \frac{n-d}{2})$ distribution [9] with probability density function $t^{(d/2)-1/2}(1-t)^{((n-d)/2)-1}$. The probability that y is a good approximation to a linear combination of eigenvectors $u_1, \cdots, u_d$ is thus given in the following theorem.

THEOREM 7.1. *Let x have independent random components, each with a normal* (0, 1) *distribution, and let $y = x/\|x\|_2 = (\eta_1, \cdots, \eta_n)^T$. Given $0 \leqq \varepsilon \leqq 1$, the probability that $\sum_{i=1}^{d} \eta_i^2 \geqq 1 - \varepsilon^2$ is*

$$(1) \qquad P\left(\sum_{i=1}^{d} \eta_i^2 \geqq 1 - \varepsilon^2\right) = 1 - \alpha \int_0^{1-\varepsilon^2} t^{\frac{d}{2}-1}(1-t)^{\frac{n-d-2}{2}}\, dt \equiv 1 - \alpha I$$

with

$$\alpha = \frac{\gamma(\frac{n}{2})}{\gamma(\frac{d}{2})\gamma(\frac{n-d}{2})},$$

where $\gamma(\omega)$ is the gamma function with argument ω.

7.3. The quality of the starting vectors. The experiments in § 4 show that random vectors make good starting vectors for inverse iteration. It turns out that the random starting vectors used were linearly independent and not orthogonal to the eigenvectors being computed. In this section, we demonstrate the practical usefulness of Theorem 7.1 and establish the number of times a random starting vector can be reused for computing eigenvectors associated with well-separated eigenvalues.

For rapid convergence of an iterate to an eigenvector u_i, it is essential that the starting vector y have a large enough component in the u_i direction. The probability that a component η_i is of size at least $\sqrt{1-\varepsilon^2}$ is given in Theorem 7.1 with $d=1$. Table 11 gives these probabilities for matrix orders $n=100$, 1000, and 10,000 for a range of $1-\varepsilon^2$ values. The integral in Theorem 7.1 was computed by Gauss-Legendre quadrature with 100 nodes. For $n \leqq 10{,}000$, the probability that any one component of y is at least 0.0001 is no smaller than $1-10^{-16}$, while the probability that any one component is at least 0.001 is no smaller than 0.9. The unit two-norm of the starting vector guarantees that not all components are very small. Recall from Table 2 that a component of size 10^{-8} is sufficient for fast convergence. For a given tolerance $1-\varepsilon^2$, the probability bounds decrease as n increases: as the number of components in a vector increases, the probability that any one component is large decreases.

TABLE 11
Lower bounds on the probability that $\eta_i^2 \geqq 1-\varepsilon^2$. Numbers in parentheses indicate the number of zero decimal places.

$\sqrt{1-\varepsilon^2}$	For $n=100$ $P(\lvert\eta_i\rvert \geqq \sqrt{1-\varepsilon^2})$	For $n=1000$ $P(\lvert\eta_i\rvert \geqq \sqrt{1-\varepsilon^2})$	For $n=10{,}000$ $P(\lvert\eta_i\rvert \geqq \sqrt{1-\varepsilon^2})$
$<10^{-4}$	1.00 (16)	1.00 (16)	1.00 (16)
10^{-3}	0.99	0.97	0.90
10^{-2}	0.92	0.76	0.34
10^{-1}	0.34	0 (2)	0 (16)
>0.7	0 (16)	0 (16)	0 (16)

Table 4 shows that sets of n randomly generated starting n-vectors tend to be linearly independent. This observation is supported statistically by the result from [13] restated here as Lemma 7.1.

LEMMA 7.1. *Assume that the vectors x_i have independent random components with normal $(0,1)$ distributions. Let $\sigma_{\min}$ denote the smallest singular value of $X=(x_1, \cdots, x_n)$. Then, as $\sigma \rightarrow 0$, the probability that $\sigma_{\min}^2 < \sigma^2$ is*

$$P(\sigma_{\min}^2 < \sigma^2) \sim \sigma\sqrt{n}.$$

Thus, the probability of a nearly singular matrix of starting vectors X is small but grows with increasing number of columns of X. For example, the probability of $\sigma_{\min} < 10^{-16}$ is on the order of 10^{-7} when $n=512$.

For the computation of eigenvectors associated with well-separated eigenvalues, linear independence of the starting vectors seems less important, and one could try to use the same random starting vector for all of them. Given $\varepsilon > 0$, the same starting vector y can be used to compute eigenvectors $u_1, \cdots, u_d$ if $\eta_i^2 \geqq 1-\varepsilon^2$ for $1 \leqq i \leqq d$. The following theorem shows how the number d of eigenvectors for which y can be used depends on the probability ρ with which each of the d eigenvectors provides a significant contribution of y.

THEOREM 7.2. (Reuse of starting vectors.) *Assume that x has independent random components with normal $(0,1)$ distributions and that $y=x/\|x\|_2$. If $d \leqq \lfloor \frac{1-\rho}{\alpha\mathcal{I}} \rfloor$, then $\eta_i^2 \geqq 1-\varepsilon^2$ for $1 \leqq i \leqq d$ with probability at least ρ.*

Proof. Let $y = x/\|x\|_2 = (\eta_1, \cdots, \eta_n)^T$. Then

$$P(\eta_i^2 \geqq 1-\varepsilon^2, 1 \leqq i \leqq d) = 1 - P(\eta_i^2 < 1-\varepsilon^2, 1 \leqq i \leqq d)$$

$$\geqq 1 - \sum_{i=1}^{d} P(\eta_i^2 \leqq 1-\varepsilon^2) = 1 - d\alpha\mathcal{J},$$

where the last equality comes from Theorem 7.1. Setting $\rho = 1 - d\alpha\mathcal{J}$ gives $d \leqq \lfloor \frac{1-\rho}{\alpha\mathcal{J}} \rfloor$. □

Table 12 shows values of d for several choices of $\sqrt{1-\varepsilon^2}$ when $n = 100$, 1000, 10,000: the number of times y can be reused decreases with increasing matrix order n, for fixed ρ and ε. This echoes the trend observed in Table 11: a long vector of norm one is less likely to have large components and so is less acceptable for reuse. From the numerical experiments, we know that a component of size 10^{-8} suffices for rapid convergence. According to Table 12 all components are of size 10^{-4} for random vectors up to length 1000 with probability 0.99 so that the same starting vector can be used to compute all eigenvectors for matrices of order up to $n = 10{,}000$ with probability 0.99.

TABLE 12

The number of times d a starting vector can be used, when $\eta_i^2 \geqq 1-\varepsilon^2$ for $1 \leqq i \leqq d$ with probability ρ.

ρ	$\sqrt{1-\varepsilon^2}$	For $n = 100$ d	For $n = 1000$ d	For $n = 10{,}000$ d
0.5	$<10^{-4}$	100	1000	10,000
	10^{-3}	50	16	5
	10^{-2}	6	2	<1
	10^{-1}	<1	<1	<1
0.9	$<10^{-4}$	100	1000	10,000
	10^{-3}	10	3	1
	10^{-2}	1	<1	<1
	10^{-1}	<1	<1	<1
0.99	$<10^{-4}$	100	1000	10,000
	10^{-3}	1	<1	<1
	10^{-2}	<1	<1	<1
	10^{-1}	<1	<1	<1

The main significance of the statistical analysis is in justifying the choice of random starting vector. The analysis can be applied to the results of inverse iteration in only a limited way. If $z = (T - \hat{\lambda}_j I)^{-k} y = (\zeta_1, \cdots, \zeta_n)^T$ is the unnormalized kth iterate, and no reorthogonalization has taken place, the probability that any one component ζ_j is larger in magnitude than $\sqrt{1-\varepsilon^2}$ is [17]

$$P(\zeta_j^2 \geqq 1-\varepsilon^2) \geqq 1 - \alpha \int_0^{(\lambda_j - \hat{\lambda}_j)^{2k}(1-\varepsilon^2)} t^{-1/2}(1-t)^{(n-3/2)}\, dt.$$

According to the mean value theorem for definite integrals, the integral $\mathcal{J}$ is bounded above by $(\lambda_j - \hat{\lambda}_j)^{2k}(1-\varepsilon^2)$. Thus, if $\hat{\lambda}_j$ is a good approximation to λ_j, then $\mathcal{J}$ is small and the probability that z_j approximates u_j is close to one. If $\lambda_j - \hat{\lambda}_j = \lambda_{j+1} - \hat{\lambda}_j$, then z_j approximates u_j and u_{j+1} with equal probability. When λ_j and λ_{j+1} are close but not equal, one of the two eigenvectors u_j and u_{j+1} will be approximated better than the other only if $(1-\varepsilon^2)(\lambda_j-\hat{\lambda}_j)$ and $(1-\varepsilon^2)(\lambda_{j+1}-\hat{\lambda}_j)$ lie where the integrand $f(t)=t^{-1/2}(1-t)^{(n-3)/2}$ has a large derivative. Hence, although the effects of additional iterations on the accuracy can be assessed in terms of the integral $\mathcal{I}$, a qualitative interpretation seems difficult.

Just as the statistical analysis falls short in determining the preferred number of iterations, it fails regarding stopping and reorthogonalization criteria: even the simplest approximations of distributions become unwieldy [4], [17], [18], and probability density functions become dependent on the exact eigenvalues and on other assumptions that are difficult to verify [17]. Therefore, we did not extend the statistical analysis to the iterates.

7.4. Practical considerations. The preceding statistical analysis qualitatively confirms the experimental observations regarding the starting vectors in § 4. However, the analysis is based on starting vectors with independent, normally distributed components, while the experiments were performed with uniformly distributed components in $[-1, 1]$ having some degree of dependence. Thus, the experimental starting vectors of length n are not uniformly distributed on the unit n-sphere but rather on an n-cube of height 2. Although normally distributed pseudorandom numbers can be generated at a higher cost than uniform ones [9], we will now show that uniform random numbers are acceptable substitutes.

The error in applying the analysis to uniformly distributed starting vectors may be estimated by circumscribing an n-sphere of radius $\sqrt{n}$ about the hypercube. A vector $y=(\eta_1,\cdots,\eta_n)^T=\sum_{i=1}^n \eta_i u_i$ on this sphere is a good approximation to a multiple $\sqrt{n}u_i$ of the eigenvector u_i if $|\eta_i|\geqq\sqrt{n(1-\varepsilon^2)}$. Following the derivation in § 7.2, the probability of this occurrence is

$$P(\eta_i^2\geqq n(1-\varepsilon^2))=1-\alpha\int_0^{n(1-\varepsilon^2)} t^{-1/2}(1-t)^{(n-3)/2}\,dt.$$

Lower bounds for this probability computed by Gauss-Legendre quadrature are given in Table 13.

TABLE 13

Lower bounds on the probability that $\eta_i^2\geqq n(1-\varepsilon^2)$. Numbers in parentheses indicate the number of zero decimal places.

$\sqrt{1-\varepsilon^2}$	For $n=100$ $P(\eta_i^2\geqq n(1-\varepsilon^2))$	For $n=1000$ $P(\eta_i^2\geqq n(1-\varepsilon^2))$	For $n=10{,}000$ $P(\eta_i^2\geqq n(1-\varepsilon^2))$
$\leqq 10^{-6}$	1.00 (16)	1.00 (16)	1.00 (16)
10^{-5}	1.00 (16)	0.99	0.90
10^{-4}	0.99	0.36	0.34
10^{-3}	0.92	0.33	0 (16)
10^{-2}	0.34	0 (2)	0 (16)
$\geqq 10^{-1}$	0 (16)	0 (16)	0 (16)

The probability of a large component η_i in the uniform case is not as high as in the normally distributed case, but components with magnitude 10^{-6} can be expected with near certainty, and this value still suffices for fast convergence. The linear independence of the pseudorandom starting vectors is demonstrated in Table 4.

8. Conclusions. Algorithm III presented in § 5 of this paper replaces the EISPACK routine TINVIT for computing the eigenvectors of a symmetric tridiagonal matrix by inverse iteration. Algorithm III both provides more accurate eigenvectors and allows solution of larger problems than does TINVIT. These improvements follow from a new iteration stopping criterion and the use of random starting vectors. The latter allow us to formalize a portion of the algorithm formerly based solely on heuristics. Namely, the statistical analysis in § 7 proves that random vectors are usually linearly independent and have sufficient components in the directions of the eigenvectors being computed. The first property leads to linearly independent iterates, the second to fast convergence.

9. Acknowledgment. The authors thank Stan Eisenstat and Jim Demmel for many helpful discussions and the reviewers for their instructive comments.

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, LAPACK: *A portable linear algebra library for high-performance computers*, LAPACK Working Note 20, Computer Science Department, University of Knoxville, Knoxville, TN, 1990.

[2] J. Barlow and J. Demmel, *Computing accurate eigensystems of scaled diagonally dominant matrices*, SIAM J. Numer. Anal., 27 (1990), pp. 762-791.

[3] H. Bowdler, R. Martin, and J. Wilkinson, *The QR and QL algorithms for symmetric matrices*, Numer. Math., 11 (1968), pp. 227-240.

[4] S. Crump, *The estimation of variance components in analysis of variance*, Biometrics, 2 (1946), pp. 7-11.

[5] J. Cuppen, *A divide and conquer method for the symmetric tridiagonal eigenproblem*, Numer. Math., 36 (1981), pp. 177-95.

[6] P. Deift, J. Demmel, L.-C. Li, and C. Tomei, *The bidiagonal singular value decomposition and Hamiltonian mechanics*, LAPACK Working Note 11, Computer Science Dept. Tech. Report, Courant Institute, New York University, New York, NY, 1989.

[7] J. Demmel and W. Kahan, *Accurate singular values of bidiagonal matrices*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 873-912.

[8] J. Demmel and K. Veselić, *Jacobi's method is more accurate than QR*, LAPACK Working Note 15, Computer Science Dept. Tech. Report, Courant Institute, New York University, New York, NY, 1989.

[9] L. Devroye, *Non-Uniform Random Variate Generation*, Springer-Verlag, Berlin, New York, 1986.

[10] J. Dixon, *Estimating extremal eigenvalues and condition numbers of matrices*, SIAM J. Numer. Anal., 20 (1983), pp. 812-814.

[11] J. Dongarra and E. Grosse, *Distribution of mathematical software via electronic mail*, Comm. ACM, 30 (1987), pp. 403-407.

[12] J. Dongarra and D. Sorensen, *A fully parallel algorithm for the symmetric eigenvalue problem*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s139-s154.

[13] A. Edelman, *Eigenvalues and condition numbers of random matrices*, SIAM J. Matrix Anal. Appl., 9 (1988), pp. 543-560.

[14] G. Golub and C. Van Loan, *Matrix Computations*, Second Edition, The Johns Hopkins University Press, Baltimore, MD, 1989.

[15] R. Gregory and D. Karney, *A Collection of Matrices for Testing Computational Algorithms*, John Wiley & Sons, Inc., New York, 1969.

[16] I. Ipsen and E. Jessup, *Solving the symmetric tridiagonal eigenvalue problem on the hypercube*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 203-229.

[17] E. Jessup, *Parallel Solution of the Symmetric Tridiagonal Eigenproblem*, Ph.D. thesis, Dept. of Computer Science, Yale University, New Haven, CT, 1989.

[18] N. Johnson and S. Kotz, *Continuous Univariate Distributions*, Houghton Mifflin Company, Boston, MA, 1970.

[19] J. Kuczyński and H. Woźniakowski, *Estimating the largest eigenvalue by the power and Lanczos algorithms with a random start*, Tech. Report, Dept. of Computer Science, Columbia University, New York, NY, 1989.

[20] C. Moler, Personal communication, 1987.

[21] B. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[22] B. Smith, J. Boyle, J. Dongarra, B. Garbow, Y. Ikebe, V. Klema, and C. Moler, *Matrix Eigensystem Routines-EISPACK Guide*, Lecture Notes in Computer Science 6, Second Edition, Springer-Verlag, Berlin, New York, 1976.

[23] J. Wilkinson, *Rounding Errors in Algebraic Processes*, Prentice-Hall, Englewood Cliffs, NJ, 1963.

[24] ———, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, U.K., 1965.

[25] ———, *Inverse iteration in theory and practice*, Symposia Mathematica, Vol. X of the Institute Nationale di Alta Mathematica Monograf, Bologna, Italy, 19 (1972), pp. 361-379.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 2, pp. 573–595, March 1992

009

GENERALIZED SCHWARZ SPLITTINGS*

WEI PAI TANG†

Abstract. A classic mathematical technique, the Schwarz Alternating Method (SAM), has recently attracted much attention from researchers in the field of parallel computations, as well as theoreticians. Its advantages in parallelism, wide applicability and great flexibility in implementation make SAM a competitive choice in parallel computations. However, the computational performance of the classical SAM and its modern extensions strongly depend on the amount of overlap between the neighboring subregions. Introducing a large overlap has changed the image of SAM from an impractical theoretical technique to a rewarding numerical approach. However, the duplication of work in these overlapped regions is undesirable. Reducing the amount of overlap without affecting the speed of convergence has become an important performance issue.

Schwarz Splitting (SS) has been proposed as an extension of SAM in numerical linear algebra, and a generalized SS is presented in this paper. The new approach allows utilization of the flexibility of the splitting to further improve convergence speed and complexity. A fast convergence is obtained by choosing a good splitting instead of increasing the overlap. The best performance of our generalized SS is much better than that of a previously recommended SS, in which a large overlap is used. Both convergence analysis and numerical results are presented here.

Key words. Schwarz Alternating Method (SAM), Schwarz Splitting (SS), generalized Schwarz splitting, domain decomposition, parallel computation, overlap

AMS(MOS) subject classifications. 65F10, 65N10

1. Introduction.

Experience with the new generation of parallel computers has promoted efforts to search for *truly* parallel algorithms rather than parallelizing the existing sequential algorithms. For coarse grain parallelism, domain decomposition has become an increasingly important focus of research for the numerical solution of partial differential equations.

A classic mathematical approach, the Schwarz Alternating Method (SAM) (1869), [16] appears to offer promise for the parallel solution of the very large systems of linear or nonlinear algebraic equations that arise when elliptic problems in elasticity, fluid dynamics, or other important areas are discretized by finite elements or finite differences. With this approach, a large problem is decomposed into several coupled subproblems. If a proper ordering is used, these subproblems can be solved independently. Starting from a given initial guess, the solution is iterated in each subregion and new values are exchanged on these coupled artificial boundaries. This process will converge to the solution for the entire region. Flexibility in mapping these subproblems into different parallel computer topologies and the advantageous ratio between communication and computation make SAM a tempting choice in parallel processing. It is also crucial for some complex fluid flow calculations that different modelings or grids be applied to different subdomains of the flow. For example, in many applications we need to merge Euler's equation, the Navier–Stokes equations, potential flow, and other models in suitable subregions for a single large problem. There are also applications where composite meshes in regions with complicated boundaries are needed. SAM can provide a natural framework within which all these requirements are met.

The recognition of SAM's potential in numerical computations was a rather recent event [2], [14], [11], [12], [5], [17], [4], [10], [3]. This delay may have been caused by

* Received by the editors January 30, 1989; accepted for publication (in revised form) January 14, 1991. This research was supported by the Natural Sciences and Engineering Research Council of Canada.

† Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

some disappointing experiments with earlier implementations of SAM [15]. During the past several years, understanding of SAM's computational behavior has become clearer. If the area of the overlap regions is a constant fraction of the subregions, it has been shown that the convergence of SAM is independent of the mesh [12], [3].

Several *modern extensions* of SAM have been proposed. For example, a generalization of SAM in linear algebra—Schwarz splittings—was introduced in [14], [17], and an additive version of SAM is being investigated [4], [3]. It is now known that the convergence of the classical SAM and most of its extensions strongly depend on the amount of overlap between subregions. Introducing a larger overlap does considerably improve the performance of SAM [12], [8]. In conjunction with other acceleration techniques, such as multilevel techniques, preconditioning or SOR accelerations, SAM has proven to be a useful method in large scale scientific computations. However, one undesirable feature of SAM is the duplication of work on the overlapped regions at each iteration. Reducing the amount of overlap without affecting speed of convergence has become an important performance issue.

Schwarz originally proposed a coupling between subregions which requires only the continuity of the unknown. Kantorovich and Krylov ([9, pp. 617–626]) presented a rather general convergence result of SAM for a second order partial differential equation of the form

$$L(u) = F\left(x, y, u, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial^2 u}{\partial x^2}, \frac{\partial^2 u}{\partial y^2}\right) = 0,$$

with Dirichlet boundary condition. They also used Dirichlet boundary condition on the artificial boundaries. It is not difficult to observe that we can replace the continuity of the unknown on these artificial boundaries by other couplings, for example, the continuity of the unknown's derivative. Several attempts to use different boundary conditions on these overlapped regions were not very successful [17], [13], leading us to conjecture a few years ago that "For different problems the best choice of the type of boundary conditions may vary. It is a very interesting open problem for future research." For the model problem, it was shown that the Dirichlet boundary condition has a better convergence rate than the Neumann boundary condition [17].

In this paper, a new coupling between the overlap subregions is identified. If a successful coupling is chosen, a fast convergence of the alternating process can be achieved without a large overlap.

Before proceeding, let us introduce a generalized version of SAM. We consider the Dirichlet problem for a second order elliptic operator L

$$\begin{aligned} &L(u(\mathbf{x})) = 0, && \mathbf{x} \in \Omega, \\ &u(\mathbf{x})\,|_{\Gamma_\Omega} = \psi(\mathbf{x}), && \mathbf{x} \in \Gamma_\Omega, \end{aligned} \tag{1}$$

where Ω is a bounded region in k-dimensional space, Γ_Ω is the boundary of Ω, and $\mathbf{x} = \{x_1, x_2, \cdots, x_k\}$ is the independent variable (Fig. 1 shows an example of the two-dimensional case). To simplify the discussion, we consider a case for two subregions, although direct generalization can be made to more subregions. We also assume that the solution to this problem exists and is unique.

Split the solution domain Ω into two overlapping subdomains Ω_1 and Ω_2 (see Fig. 1), provided $\Omega_{12} = \Omega_1 \cap \Omega_2 \neq \emptyset$. Denote Γ_{Ω_1}, Γ_{Ω_2}, $\Gamma_{\Omega_{12}}$ the boundaries of Ω_1, Ω_2, and Ω_{12}, respectively. We denote Γ_i, $i = 1, 2$, the part of boundary which belongs to

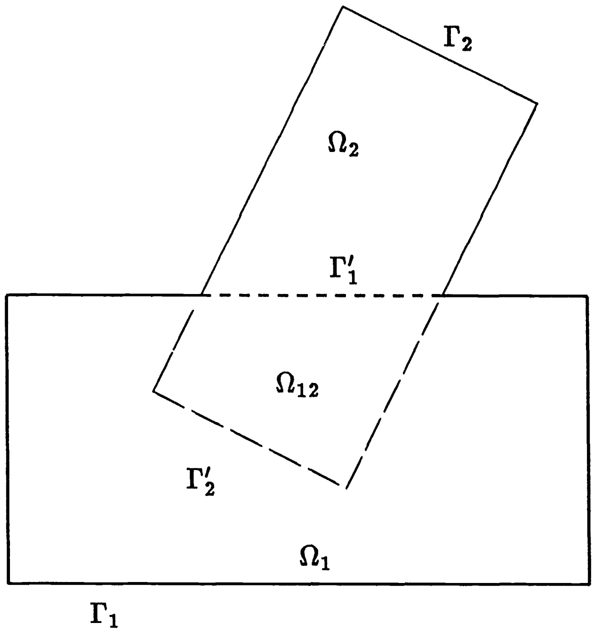


FIG. 1. *Two overlapping subregions.*

Γ_{Ω_i}. Let Γ_i' , $i = 1, 2$, be the part of the artificial boundary in Ω_i. We have

$$\Gamma_{\Omega_1} = \Gamma_1 \cup \Gamma_1',$$
$$\Gamma_{\Omega_2} = \Gamma_2 \cup \Gamma_2'.$$

Denote u_1 and u_2 as the solutions on subdomain Ω_1 and Ω_2, respectively. Then, the following couplings

$$g_1(u_1) \mid_{\Gamma_1'} = g_1(u_2) \mid_{\Gamma_1'}, \tag{2}$$

$$g_2(u_2) \mid_{\Gamma_2'} = g_2(u_1) \mid_{\Gamma_2'}, \tag{3}$$

are true on the artificial boundaries Γ_1' and Γ_2', where

$$g_i(u) = \omega_i u + (1 - \omega_i)\frac{\partial u}{\partial n}, \qquad i = 1, 2. \tag{4}$$

With these new couplings we can formulate two coupled subproblems

$$\begin{aligned} &L(u_1(\mathbf{x})) = 0, \qquad \mathbf{x} \in \Omega_1, \\ &u_1(\mathbf{x}) \mid_{\Gamma_1} = \psi(\mathbf{x}), \\ &g_1(u_1(\mathbf{x})) \mid_{\Gamma_1'} = g_1(u_2(\mathbf{x})) \mid_{\Gamma_1'}, \end{aligned} \tag{5}$$

$$\begin{aligned} &L(u_2(\mathbf{x})) = 0, \qquad \mathbf{x} \in \Omega_2, \\ &u_2(\mathbf{x}) \mid_{\Gamma_2} = \psi(\mathbf{x}), \\ &g_2(u_2(\mathbf{x})) \mid_{\Gamma_2'} = g_2(u_1(\mathbf{x})) \mid_{\Gamma_2'} . \end{aligned} \tag{6}$$

We have the following result.

THEOREM 1. *If the boundary value problem*

$$\begin{aligned} &L(w(\mathbf{x})) = 0, \qquad \mathbf{x} \in \Omega_{12}, \\ &g_1(w(\mathbf{x}))\,|_{\Gamma_1'} = 0, \\ &g_2(w(\mathbf{x}))\,|_{\Gamma_2'} = 0, \end{aligned} \tag{7}$$

has only trivial solution and the solutions u_1, u_2 of (5) *and* (6) *exist, then*

1. $u_1(\mathbf{x}) = u_2(\mathbf{x})$, $\mathbf{x} \in \Omega_{12}$.
2. $u(\mathbf{x}) = u_1(\mathbf{x})$, $\mathbf{x} \in \Omega_1$ *and* $u(\mathbf{x}) = u_2(\mathbf{x})$, $\mathbf{x} \in \Omega_2$,

where u, and u_1, u_2 are the solutions of (1) *and, respectively,* (5) *and* (6).

The proof of this theorem is straightforward and a direct generalization of this result to a finite number of overlapping subregions can be made. However, the same proof can work only for the case in which no three subregions have a common overlap region. We say that problem (1) is *equivalent* to (5) and (6), if 1 and 2 in Theorem 1 are true. A version of this result in linear algebra will be shown in the next section. Following Theorem 1, we can replace problem (1) by (5) and (6).

Since there are unknowns which are coupled in the boundary conditions of (5) and (6), we cannot solve the two problems independently. Given an initial guess $u_2^{(0)}\,|_{\Gamma_1'} = \psi_0$, we will then be able to construct a sequence $\{u_1^{(i)}, u_2^{(i)}\}$ as follows:

$$\begin{aligned} &L(u_1^{(i)}) = 0, \qquad \mathbf{x} \in \Omega_1, \\ &u_1^{(i)}\,|_{\Gamma_1} = \psi, \\ &g_1(u_1^{(i)})\,|_{\Gamma_1'} = g_1(u_2^{(i-1)})\,|_{\Gamma_1'}, \end{aligned} \tag{8}$$

$$\begin{aligned} &L(u_2^{(i)}) = 0, \qquad \mathbf{x} \in \Omega_2, \\ &u_2^{(i)}\,|_{\Gamma_2} = \psi, \\ &g_2(u_2^{(i)})\,|_{\Gamma_2'} = g_2(u_1^{(i)})\,|_{\Gamma_2'}, \qquad i = 1, 2, \cdots. \end{aligned} \tag{9}$$

A key question to ask is "Under what conditions will the sequence $\{u_1^{(i)}, u_2^{(i)}\}$ converge to the solutions $\{u_1, u_2\}$ of (5) and (6)?" If it converges, then the solution of (1) can be constructed from the solution of (5) and (6). An analysis for the model problem will be given in §3.

The generalized SAM provides us with a general framework. Several questions of implementation which affect efficiency are left open. We can effectively tailor this approach to different problems or a different computer architecture. In particular, the following issues have an important impact on performance in real applications.

- The choice of the couplings $g_i(u)$. A better choice of g_i can yield substantial improvement in the convergence rate (see numerical results in §3).
- The decomposition of the solution domain. The flexibility in decomposition makes it possible to choose the geometry of most of the subregions to meet the requirements imposed by fast solvers or by grids. A fast biharmonic solver on irregular domains using generalized SAM is studied in [1].
- The selection of the individual solution technique for each subdomain. We are able to use different solution techniques for different subproblems. It is also

possible to use different ways to obtain the solution of the same subproblem in the different stages of the computation, allowing us to use an optimal approach at any particular moment and in any particular location. Hierarchical grid and inexact solution strategies are typical examples here [18], [7].

- Numerical model for each subproblem. Special boundary shapes or local behavior of the solution may require different models in different subregions. The decoupled subproblems allow us to localize the special treatment to the place where it is needed. Composite grids are a good example of this case.

A particularly important application of SAM is for parallel computations. In the previous description of generalized SAM, the parallelism is not very obvious. When the number of the subregions is greater than or equal to the number of processors, we can color the subregions such that subregions with the same color can be solved independently. There are many issues which need to be considered in practical parallel implementation such as load balancing, communication, synchronization, and ordering of the solution of the subproblems. These are very important in terms of parallel efficiency; however, we shall not study them in depth here.

In the next section, a generalized Schwarz Splitting (generalized SS) and an equivalence theorem are presented. This generalization is an analogy of the generalization from SAM to Schwarz Splitting. Then, an application of this generalized SS to the solution of elliptic equations is shown in §3. The convergence analysis of the strip case and our numerical results indicate that the performance of a proposed generalized SS depends mostly on a coupling parameter α. A fast convergence rate based on a proper choice of α can be obtained with very little overlap, thus the concern about the duplication of computation in the traditional SAM can be alleviated.

2. Generalized Schwarz Splittings. In this section we present an extension of the generalized SAM to numerical linear algebra. For a matrix equation $Ax = f$, we first introduce a Schwarz enhanced equation $\widetilde{A}\widetilde{x} = \widetilde{f}$. The corresponding matrix $\widetilde{A}$ is called a Schwarz enhanced matrix. A necessary and sufficient condition for the equivalence of the original equation and the Schwarz enhanced equation is shown. The analogy of applying generalized SAM to the matrix equation is equivalent to applying a particular block Gauss–Seidel scheme to the Schwarz enhanced matrix. The corresponding splitting of the Schwarz enhanced matrix is called generalized Schwarz Splitting (generalized SS). With this extension, many classical results in numerical linear algebra can be applied to this problem.

2.1. Definitions. As we mentioned in the introduction, the approach of the generalized SAM to a problem is to create an equivalent problem which consists of several loosely coupled subproblems, then to solve the subproblems iteratively. It is not necessary to view SAM only as a way of solving elliptic partial differential equations, as it can also be viewed as a general method for problem solving. Here the generalized SAM is discussed in terms of matrix theory. This approach provides new opportunities for generalizing and improving the original SAM.

Consider a matrix problem:

$$Ax = f, \tag{10}$$

where A is an $N \times N$ nonsingular matrix, f and x are N vectors. A partitioned form of (10) will be used in the rest of this paper. A partition is defined by the integers $n_1, n_2, \cdots, n_{2k+1}$ such that

$$n_1 + n_2 + \cdots + n_{2k+1} = N, \tag{11}$$

$$n_{2i} > 0, \qquad n_{2i+1} \geq 0, \qquad i = 1, \cdots, k. \tag{12}$$

Given a set $\{n_i\}_{i=1}^{2k+1}$ which satisfies (11) and (12), the $(2k+1) \times (2k+1)$ partitioned form of the matrix A is then given by

$$A = \begin{bmatrix} A_{1\ 1} & A_{1\ 2} & \cdots & A_{1\ 2k+1} \\ A_{2\ 1} & A_{2\ 2} & \cdots & A_{2\ 2k+1} \\ \vdots & \vdots & \ddots & \vdots \\ A_{2k+1\ 1} & A_{2k+1\ 2} & \cdots & A_{2k+1\ 2k+1} \end{bmatrix}, \tag{13}$$

where $A_{i\ j}$ is an $n_i \times n_j$ submatrix. We always assume that the unknown vector x and the known vector f in the matrix equation $Ax = f$ are partitioned in a form consistent with A. Thus, if A is given by (13), then x is assumed to be partitioned as

$$x = [x_1, x_2, \cdots, x_{2k+1}]^T, \tag{14}$$

where x_i is an $n_i \times 1$ matrix (column vector). An *augmented vector* of x

$$\tilde{x} = [x_1, x_2, x_2, x_3, x_4, x_4, x_5, \cdots, x_{2k}, x_{2k}, x_{2k+1}]^T \tag{15}$$

is defined such that all even subvectors x_{2i}, $i = 1, \cdots, k$ are duplicated once in their places, and all odd subvectors remain the same.

We present the cases for $N = 3$ and 5 here for readability. The generalization to a large N is direct. A dense 3×3 partitioned matrix can be written as

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

If the operator $L(u)$ in (1) is a linear second order elliptic operator, then the discretized problem can be written as a matrix equation:

$$Ax = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} = f. \tag{16}$$

The order of the unknowns is arranged so that $[x_1, x_2]$ corresponds to the unknowns in Ω_1, $[x_2, x_3]$ corresponds to the unknowns in Ω_2, and $[x_2]$ corresponds to the unknowns in Ω_{12}, which is the overlapped region. The numerical generalized SAM for the above problem solves the following subproblems alternately:

$$\begin{aligned} \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & B_2 \end{bmatrix} \begin{bmatrix} x_1^{(i)} \\ x_2^{(i-1/2)} \end{bmatrix} &= \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} + \begin{bmatrix} 0 & A_{13} \\ C_2 & A_{23} \end{bmatrix} \begin{bmatrix} x_2^{(i-1)} \\ x_3^{(i-1)} \end{bmatrix}, \\ \begin{bmatrix} B_2' & A_{23} \\ A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} x_2^{(i)} \\ x_3^{(i)} \end{bmatrix} &= \begin{bmatrix} f_2 \\ f_3 \end{bmatrix} + \begin{bmatrix} A_{21} & C_2' \\ A_{31} & 0 \end{bmatrix} \begin{bmatrix} x_1^{(i)} \\ x_2^{(i-1/2)} \end{bmatrix}, \end{aligned} \tag{17}$$

where

$$A_{22} = B_2 + C_2 = B_2' + C_2'. \tag{18}$$

The splittings in (18) correspond to the couplings in (2) and (3). In the next section we will show that a good choice of the splitting of A_{22} can significantly affect the convergence of SAM. It is therefore a very interesting research problem for further improvement of SAM.

It is not difficult to observe that this procedure is equivalent to a 2 × 2 block Gauss–Seidel iteration for the following matrix equation:

$$\text{(19)} \qquad \widetilde{A}\widetilde{x} = \left[\begin{array}{cc|cc} A_{11} & A_{12} & 0 & A_{13} \\ A_{21} & B_2 & C_2 & A_{23} \\ \hline A_{21} & C_2' & B_2' & A_{23} \\ A_{31} & 0 & A_{32} & A_{33} \end{array}\right] \begin{bmatrix} \widetilde{x}_1 \\ \widetilde{x}_2 \\ \widetilde{x}_2' \\ \widetilde{x}_3 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_2 \\ f_3 \end{bmatrix} = \widetilde{f}.$$

Under certain conditions [14], [17], we know that the procedure (17) will converge, the solution of (19) satisfies $\widetilde{x}_2 = \widetilde{x}_2'$, and $[\widetilde{x}_1, \widetilde{x}_2, \widetilde{x}_3]^T$ is a solution of (16). This is to say, the augmented vector of the solution of (16) is the solution of (19) and vice versa. Later we will prove that this conclusion can be true only when $(B_2 - C_2')^{-1}$ exists. For most approximations of an elliptic partial differential equation, this restriction is not very difficult to satisfy. We shall call (19) the generalized Schwarz enhanced equation of (16), and the corresponding matrix $\widetilde{A}$ in (19) the generalized Schwarz enhanced matrix of the matrix A.

It may be observed that the second equation in (16) becomes a pair of dual equations in (19):

$$A_{21}\widetilde{x}_1 + B_2\widetilde{x}_2 + C_2\widetilde{x}_2' + A_{23}\widetilde{x}_3 = f_2,$$

$$A_{21}\widetilde{x}_1 + C_2'\widetilde{x}_2 + B_2'\widetilde{x}_2' + A_{23}\widetilde{x}_3 = f_2.$$

They are almost identical, except the term $A_{22}x_2$ of the second equation in (16) is split in two different ways:

$$\begin{aligned} A_{22}x_2 &\implies B_2\widetilde{x}_2 + C_2\widetilde{x}_2', \\ A_{22}x_2 &\implies C_2'\widetilde{x}_2 + B_2'\widetilde{x}_2'. \end{aligned}$$

Here is another example of a 5 × 5 block matrix equation and its generalized Schwarz enhanced equation:

$$Ax = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} & A_{15} \\ A_{21} & A_{22} & A_{23} & A_{24} & A_{25} \\ A_{31} & A_{32} & A_{33} & A_{34} & A_{35} \\ A_{41} & A_{42} & A_{43} & A_{44} & A_{45} \\ A_{51} & A_{52} & A_{53} & A_{54} & A_{55} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{bmatrix} = f,$$

$$\widetilde{A}\widetilde{x} = \left[\begin{array}{cc|ccc|cc} A_{11} & A_{12} & 0 & A_{13} & A_{14} & 0 & A_{15} \\ A_{21} & B_2 & C_2 & A_{23} & A_{24} & 0 & A_{25} \\ \hline A_{21} & C_2' & B_2' & A_{23} & A_{24} & 0 & A_{25} \\ A_{31} & 0 & A_{32} & A_{33} & A_{34} & 0 & A_{35} \\ A_{41} & 0 & A_{42} & A_{43} & B_4 & C_4 & A_{45} \\ \hline A_{41} & 0 & A_{42} & A_{43} & C_4' & B_4' & A_{45} \\ A_{51} & 0 & A_{52} & A_{53} & 0 & A_{54} & A_{55} \end{array}\right] \begin{bmatrix} x_1 \\ x_2 \\ x_2' \\ x_3 \\ x_4 \\ x_4' \\ x_5 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_2 \\ f_3 \\ f_4 \\ f_4 \\ f_5 \end{bmatrix} = \widetilde{f},$$

where

$$A_{22} = B_2 + C_2 = B_2' + C_2',$$
$$A_{44} = B_4 + C_4 = B_4' + C_4'.$$

For a general partitioned matrix (13), the splittings of the submatrices $A_{2i\ 2i}$ are

$$A_{2i\ 2i} = B_{2i} + C_{2i} = B_{2i}' + C_{2i}'.$$

From these examples, we can summarize the following rules for constructing the Schwarz enhanced equation. The odd-numbered equations in $Ax = f$ are changed to

$$\sum_{j=1}^{2k+1} A_{1\ j}x_j = f_1,$$

$$\sum_{j=1}^{i} A_{2i-1\ 2j-1}x_{2j-1} + \sum_{j=1}^{i-1} A_{2i-1\ 2j}x_{2j}' + \sum_{j=2i}^{2k+1} A_{2i-1\ j}x_j = f_{2i-1}, \quad 1 < i \le k, \quad k \ne 1,$$

$$\sum_{j=1}^{k+1} A_{2k+1\ 2j-1}x_{2j-1} + \sum_{j=1}^{k} A_{2k+1\ 2j}x_{2j}' = f_{2k+1},$$

while the even-numbered equations become a pair of dual equations in the generalized Schwarz enhanced equation:

$$\begin{aligned} &\sum_{j=1}^{i} A_{2i\ 2j-1}x_{2j-1} + \sum_{j=1}^{i-1} A_{2i\ 2j}x_{2j}' + B_{2i}x_{2i} + C_{2i}x_{2i}' + \sum_{j=2i+1}^{2k+1} A_{2i\ j}x_j = f_{2i},\\ &\sum_{j=1}^{i} A_{2i\ 2j-1}x_{2j-1} + \sum_{j=1}^{i-1} A_{2i\ 2j}x_{2j}' + C_{2i}'x_{2i} + B_{2i}'x_{2i}' + \sum_{j=2i+1}^{2k+1} A_{2i\ j}x_j = f_{2i},\\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i = 1, \cdots, k. \end{aligned} \tag{20}$$

Only two terms are different in the two dual equations. We will not describe the details of how to form the generalized Schwarz enhanced matrix in general cases, as it is similar to the Schwarz enhanced matrix described in [17]. From the construction of the generalized Schwarz enhanced equation, it is easy to see the following result.

LEMMA 1. *If the vector* $x = (x_1, x_2, \cdots, x_{2k+1})^T$ *is the solution of* (10), *then its augmented vector* $\tilde{x}$ *is the solution of the generalized Schwarz enhanced equation* $\tilde{A}\tilde{x} = \tilde{f}$, *where* $\tilde{f}$ *is the augmented vector of* f.

The matrices $A_{2i\ 2i}$, $i = 1, \cdots, k$, are also called overlapped blocks. Let two matrices $\tilde{B}$ and $\tilde{C}$ be the Schwarz enhanced matrices of the same matrix A and their overlapped blocks are $B_{2i\ 2i}$ and $C_{2i\ 2i}, i = 1, \cdots, k$, respectively. If $B_{2i\ 2i}$ and $C_{2i\ 2i}$ have a relationship such that each $B_{2i\ 2i}$ is a submatrix of the corresponding $C_{2i\ 2i}$, we then say $\tilde{C}$ has *more overlap* than $\tilde{B}$. This overlap is closely related to the overlap area of the solution regions for the subregions mentioned in the introduction. As we have shown in [17], for the continuous model problem, if the amount of overlap increases, then the convergence rate will increase if a traditional SAM is applied. For the matrix model we have a similar result [12].

2.2. Equivalence theorem. A necessary and sufficient condition for the equivalence of (10) and its Schwarz enhanced equation (19) is given in this section. Let A be the same partitioned matrix in (13) and $\widetilde{A}$ be its Schwarz enhanced matrix.

THEOREM 2. *Let $\lambda(A)$, $\lambda(\widetilde{A})$, and $\lambda(B_{2i} - C'_{2i})$, $i = 1, \cdots, k$, be the sets of eigenvalues of $A, \widetilde{A}$, and $(B_{2i} - C'_{2i})$, $i = 1, \cdots, k$, respectively. Then $\lambda(\widetilde{A}) \subset \lambda(A) \cup (\bigcup_{i=1}^{k} \lambda(B_{2i} - C'_{2i}))$.*

Proof. Let λ be an eigenvalue of $\widetilde{A}$ and

$$\tilde{x} = (\tilde{x}_1, \tilde{x}_2, \tilde{x}'_2, \cdots, \tilde{x}_{2k+1})$$

be the corresponding eigenvector. Substituting $\tilde{x}$ into the equation $2i$ and its dual equation, we have

$$\sum_{j=1}^{i} A_{2i\ 2j-1}\tilde{x}_{2j-1} + \sum_{j=1}^{i-1} A_{2i\ 2j}\tilde{x}'_{2j} + B_{2i}\tilde{x}_{2i} + C_{2i}\tilde{x}'_{2i} + \sum_{j=2i+1}^{2k+1} A_{2i\ j}\tilde{x}_j = \lambda\tilde{x}_{2i},$$

$$\sum_{j=1}^{i} A_{2i\ 2j-1}\tilde{x}_{2j-1} + \sum_{j=1}^{i-1} A_{2i\ 2j}\tilde{x}'_{2j} + C'_{2i}\tilde{x}_{2i} + B'_{2i}\tilde{x}'_{2i} + \sum_{j=2i+1}^{2k+1} A_{2i\ j}\tilde{x}_j = \lambda\tilde{x}'_{2i}.$$

As we mentioned in the last section, only two terms are different in the left-hand sides of the two equations. Subtracting the first equation from the second, we have

$$(B_{2i} - C'_{2i})(\tilde{x}_{2i} - \tilde{x}'_{2i}) = \lambda(\tilde{x}_{2i} - \tilde{x}'_{2i}), \qquad i = 1, \cdots, k.$$

If $\tilde{x}_{2i} - \tilde{x}'_{2i} \neq 0$ for some i, then we have $\lambda \in \bigcup_{i=1}^{k} \lambda(B_{2i} - C'_{2i})$. If $\lambda \notin \bigcup_{i=1}^{k} \lambda(B_{2i} - C'_{2i})$, then $\tilde{x}_{2i}$ has to be equal to $\tilde{x}'_{2i}$ for $i = 1, \cdots, k$. Therefore, $\tilde{x}$ is an augmented vector of $x = (\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \cdots, \tilde{x}_{2k+1})^T$, which will satisfy equation

$$Ax = \lambda x.$$

Thus $\lambda \in \lambda(A)$, which concludes the proof. □

Define the Schwarz enhanced equation (19) as *equivalent* to (10) if $\widetilde{A}^{-1}$ exists and the solution vector $\tilde{x}$ is an augmented vector of the solution x of (10). Similarly, we say that the Schwarz enhanced matrix $\widetilde{A}$ is *equivalent* to matrix A if $\widetilde{A}^{-1}$ exists. With this definition and the result from Theorem 2 we have the following theorem.

THEOREM 3. *If a matrix $\widetilde{A}$ is a Schwarz enhanced matrix of the nonsingular matrix A, then the following conditions are equivalent:*

1. *Matrix $\widetilde{A}$ is equivalent to matrix A.*
2. $0 \notin \bigcup_{i=1}^{k} \lambda(B_{2i} - C'_{2i})$.

Proof. If $0 \notin \bigcup_{i=1}^{k} \lambda(B_{2i} - C'_{2i})$, then from Theorem 2 we know $\widetilde{A}^{-1}$ exists. Applying the same strategy used in the previous proof, we can show that the solution $\tilde{x}$ of $\widetilde{A}\tilde{x} = \tilde{f}$ is an augmented vector of the solution x of $Ax = f$.

Now we show that $0 \notin \bigcup_{i=1}^{k} \lambda(B_{2i} - C'_{2i})$ is also a necessary condition. Suppose there is a j such that $0 \in \lambda(B_{2j} - C'_{2j})$. We know that $(B_{2j} - C'_{2j})$ is singular, hence so is $(B'_{2j} - C_{2j})$, since $(B_{2j} - C'_{2j}) = (B'_{2j} - C_{2j})$. Now, if we subtract row $2j$ from $2j'$ in matrix $\widetilde{A}$, we will have

$$0, \cdots, 0, (B_{2j} - C'_{2j}), -(B'_{2j} - C_{2j}), 0, \cdots, 0. \tag{21}$$

This means that $\widetilde{A}$ is singular. The proof is complete. □

If a matrix is a positive definite matrix or an M-matrix,[1] any principal minor of this matrix is also a positive definite matrix or an M-matrix, respectively. Thus, if we choose $C_{2i} = 0$ and $C'_{2i} = 0$,[2] we immediately have the following corollaries.

COROLLARY 1. *Any Schwarz enhanced matrix of a positive definite matrix A is equivalent to A if $C_{2i} = 0$ and $C'_{2i} = 0, i = 1, \cdots, k$.*

COROLLARY 2. *Any Schwarz enhanced matrix of an M-matrix A is equivalent to A if $C_{2i} = 0$ and $C'_{2i} = 0, i = 1, \cdots, k$.*

3. A parameterized generalized Schwarz Splitting. The general framework of a generalized SS is given in the last section. Here, the convergence behavior of a particular generalized SS for the elliptic equation, namely, parameterized generalized SS, is studied. In a traditional approach to SS, we choose $C_{2i} \equiv 0$. In this case, it is well understood that the amount of overlap is a key factor which affects the convergence rate. Even though a larger overlap means more duplication of work on these overlapping regions, the overall complexity is still better than with a smaller overlap. However, a natural question is raised: is a larger overlap the ultimate choice? The generalized Schwarz Splitting discussed in the previous section provides a way to explore possibilities of further improving the performance of SAM. In particular, we will examine the importance of splitting

$$A_{2i\ 2i} = B_{2i} + C_{2i} = B'_{2i} + C'_{2i}.$$

First, an application of generalized SS to a two-point boundary value problem is investigated. A similar approach can also be applied to two-dimensional problems.

Consider a two-point boundary value problem

$$\begin{aligned} U''(x) + qU(x) &= f(x), & x &\in (0,1), \\ U(0) &= a_0, & U(1) &= a_1, \end{aligned}$$

where $q \leq 0$. After discretization using a centered finite difference, the resulting linear system is

$$T_n(\beta)x = b, \tag{22}$$

where

$$T_n(\beta) = Tridiagonal\{-1, \quad \beta, \quad -1\}_{n \times n}$$

and $\beta \geq 2$. If there is no ambiguity, it will be abbreviated as T_n. Denote

$$T_n(x_1, x_2, x_3)$$

as the same $n \times n$ tridiagonal matrix $T_n(x_2)$, except the first diagonal element is x_1 and the last is x_3.

The generalized SAM for solving this problem divides the region into k overlapping subregions Ω_i $i = 1, \cdots, k$ as shown in Fig. 2. (To simplify the analysis we assume the overlap pattern is uniform). Let h be the grid size, ℓ the length of the overlap and η

[1] Any $n \times n$ matrix $A = (a_{ij})$ with $a_{ij} \leq 0$ for all $i \neq j$ is an M-matrix if A is nonsingular, and $A^{-1} \geq 0$.

[2] In the traditional approach of SAM, we always choose $C_{2i} = 0$ and $C'_{2i} = 0$.

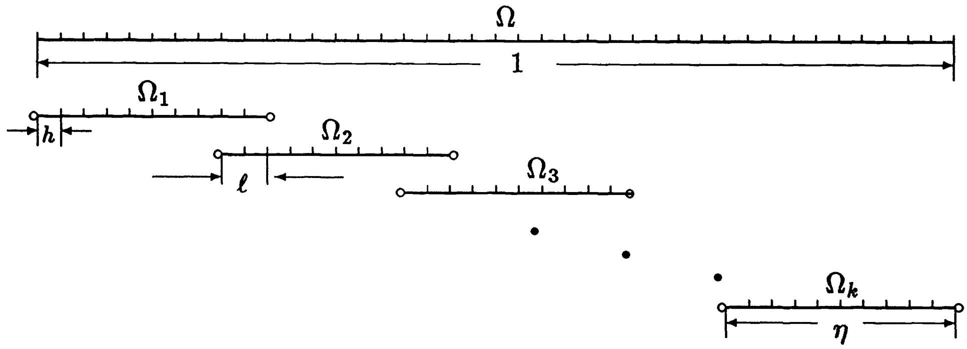


FIG. 2. *One-dimensional overlapping grid.*

the length of every subregion. Then let $n+1 = 1/h$, $l = \ell/h$ and $m+1 = \eta/h$. Here we assume

$$l < m/2,$$

which means no three subregions have a common overlap part. The open circle points in Fig. 2 are the boundaries of the subregions.

We display the case of $k = 3$ for readability. The general case can be easily extended from this case. The partitioned form of (22) is now

$$T_n x = \begin{bmatrix} T_{m-l} & -F_1 & 0 & 0 & 0 \\ -E_2 & T_l & -F_2 & 0 & 0 \\ 0 & -E_3 & T_{m-2l} & -F_3 & 0 \\ 0 & 0 & -E_4 & T_l & -F_4 \\ 0 & 0 & 0 & -E_5 & T_{m-l} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix} = b.$$

Its corresponding Schwarz enhanced equation is

$$\widetilde{T}_n \widetilde{x} = \begin{bmatrix} T_{m-l} & -F_1 & 0 & 0 & 0 & 0 & 0 \\ -E_2 & B_2 & C_2 & -F_2 & 0 & 0 & 0 \\ -E_2 & C_2' & B_2' & -F_2 & 0 & 0 & 0 \\ 0 & 0 & -E_3 & T_{m-2l} & -F_3 & 0 & 0 \\ 0 & 0 & 0 & -E_4 & B_4 & C_4 & -F_4 \\ 0 & 0 & 0 & -E_4 & C_4' & B_4' & -F_4 \\ 0 & 0 & 0 & 0 & 0 & -E_5 & T_{m-l} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_2' \\ x_3 \\ x_4 \\ x_4' \\ x_5 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_2 \\ b_3 \\ b_4 \\ b_4 \\ b_5 \end{bmatrix} = \widetilde{b}.$$

The quantities above are defined as

- F_1: an $(m-l) \times l$ matrix with zero elements everywhere except for a 1 in position $(m-l, 1)$.
- F_2: an $l \times (m-2l)$ matrix with zero elements everywhere except for a 1 in position $(m-2l, 1)$.
- F_3: an $(m-2l) \times l$ matrix with zero elements everywhere except for a 1 in position $(m-2l, 1)$.

- F_4: an $l \times (m-l)$ matrix with zero elements everywhere except for a 1 in position $(m-l, 1)$.
- E_2: an $l \times (m-l)$ matrix with zero elements everywhere except for a 1 in position $(1, m-l)$.
- E_3: an $(m-2l) \times l$ matrix with zero elements everywhere except for a 1 in position $(1, l)$.
- E_4: an $l \times (m-2l)$ matrix with zero elements everywhere except for a 1 in position $(1, m-2l)$.
- E_5: an $(m-l) \times l$ matrix with zero elements everywhere except for a 1 in position $(1, l)$.

There are many ways to split the matrix T_l. We will introduce a parameterized generalized SS for this problem below. Let

- F be an $l \times l$ matrix with zero elements everywhere except for a 1 in position (l, l) and
$$C = C_2 = C_4 = \alpha F, \qquad B = B_2 = B_4 = T_l - C.$$
- E be an $l \times l$ matrix with zero elements everywhere except for a 1 in position $(1, 1)$ and
$$C' = C_2' = C_4' = \alpha E, \qquad B' = B_2' = B_4' = T_l - C',$$

where $0 \le \alpha < 1$. It is not difficult to show that

$$\det(B - C') = \det(T_l(\beta - \alpha, \beta, \beta - \alpha)) \neq 0,$$

provided $\beta \ge 2$. The resulting Schwarz enhanced equation is equivalent to (22). Then the parameterized generalized SS of (22) is defined as

$$\begin{aligned}\widetilde{T}_n &= M(\alpha) - N(\alpha)\\ &= \begin{bmatrix} T_1 & 0 & 0 \\ 0 & T_2 & 0 \\ 0 & 0 & T_3 \end{bmatrix} - \begin{bmatrix} 0 & U_1 & 0 \\ L_2 & 0 & U_2 \\ 0 & L_3 & 0 \end{bmatrix},\end{aligned}$$

where

$$T_1 = T_m(\beta, \beta, \beta - \alpha), \quad T_2 = T_m(\beta - \alpha, \beta, \beta - \alpha), \quad T_3 = T_m(\beta - \alpha, \beta, \beta),$$
$$L_2 = \begin{bmatrix} E_2 & \alpha E \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \qquad L_3 = \begin{bmatrix} 0 & E_4 & \alpha E \\ 0 & 0 & 0 \end{bmatrix},$$
$$U_1 = \begin{bmatrix} 0 & 0 & 0 \\ \alpha F & F_2 & 0 \end{bmatrix}, \qquad U_2 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \alpha F & F_4 \end{bmatrix}.$$

A simple calculation can show that the relationship between α and ω in (4) is

$$\omega = \frac{1-\alpha}{1-\alpha+h\alpha}.$$

When $\alpha = 0$, we have $\omega = 1$. Thus, this parameterized generalized SS reduced to the traditional SS, namely, a Dirichlet boundary condition, is used on these artificial boundaries. If $\beta > 2$ and $\alpha = 1$, we have $\omega = 0$. It is equivalent to using a Neumann condition on the artificial boundaries. (If $\beta = 2$, we can use a Neumann condition on only one of the boundaries for interior subregions. Otherwise, the resulting Schwarz enhanced matrix is singular. For two-dimensional problems, if a strip decomposition is employed, then a Neumann boundary condition can be used for both artificial boundaries.) For $0 < \alpha < 1$, this generalized SS corresponds to

$$g_1(u) = g_2(u) = \omega u + (1-\omega)\frac{\partial u}{\partial n}.$$

The convergence analysis of this parameterized generalized SS is therefore reduced to calculating the spectral radius of the block Jacobi matrix $J = M^{-1}N$. Notice that the matrix $N(\alpha)$ has only eight nonzero elements. So the matrix $J = M^{-1}N$ has only eight nonzero columns, provided $l < m/2$. They are related only to the elements in the last or first columns of the matrices T_1^{-1}, T_2^{-1}, and T_3^{-1}. Let $t_{i,j}$ be the elements of the matrix $T_n^{-1}(\beta)$ and $D_i(\beta) = \det T_i(\beta)$. We have the following results (see [6]):

$$D_k(\beta) = \begin{cases} \sinh(n+1)\theta/\sinh\theta, & \beta > 2, \quad 2\cosh\theta = \beta, \\ n+1, & \beta = 2, \\ \sin(n+1)\theta/\sinh\theta, & \beta < 2, \quad 2\cos\theta = \beta, \end{cases}$$

$$t_{i,j} = \begin{cases} D_{j-1}(\beta)D_{n-i}(\beta)/D_n(\beta), & i \geq j, \\ D_{i-1}(\beta)D_{n-j}(\beta)/D_n(\beta), & i < j. \end{cases}$$

Based on this result, the elements of T_1^{-1} and T_2^{-1} can be easily derived from the Sherman–Morrison formula. We will not elaborate on the detailed derivation here.

Denote the last columns of the matrices T_1^{-1} and T_2^{-1} by $t^{(1)}$ and $t^{(2)}$, respectively:

$$t^{(1)} = (t_1^{(1)}, t_2^{(1)}, \cdots, t_m^{(1)}),$$
$$t^{(2)} = (t_1^{(2)}, t_2^{(2)}, \cdots, t_m^{(2)}).$$

Note that elements $t_i^{(j)}$ are functions of α. Since matrix T_3^{-1} is a permuted matrix of T_1^{-1}, the first column of the inverse T_3^{-1} can be derived from the last column of T_1^{-1} by a simple permutation. Let P^T be a permutation matrix; permute columns $m-l$, $m-l+1$; $m+l$, $m+l+1$; $2m-l$, $2m-l+1$; $2m+l$, $2m+l+1$ to $3m-k+1$, $k = 8, 7, \cdots, 1$, respectively. J can be similarly transformed to $\tilde{J}$:

$$\tilde{J} = PJP^T = \begin{bmatrix} 0 & K \\ 0 & G \end{bmatrix},$$

where

$$G = \begin{pmatrix} 0 & 0 & -\alpha t_{m-l}^{(1)} & t_{m-l}^{(1)} & 0 & 0 & 0 & 0 \\ 0 & 0 & -\alpha t_{m-l+1}^{(1)} & t_{m-l+1}^{(1)} & 0 & 0 & 0 & 0 \\ t_{m-l+1}^{(2)} & -\alpha t_{m-l+1}^{(2)} & 0 & 0 & 0 & 0 & -\alpha t_{l-1}^{(2)} & t_{l-1}^{(2)} \\ t_{m-l}^{(2)} & -\alpha t_{m-l}^{(2)} & 0 & 0 & 0 & 0 & -\alpha t_l^{(2)} & t_l^{(2)} \\ t_l^{(2)} & -\alpha t_l^{(2)} & 0 & 0 & 0 & 0 & -\alpha t_{m-l}^{(2)} & t_{m-l}^{(2)} \\ t_{l-1}^{(2)} & -\alpha t_{l-1}^{(2)} & 0 & 0 & 0 & 0 & -\alpha t_{m-l+1}^{(2)} & t_{m-l+1}^{(2)} \\ 0 & 0 & 0 & 0 & t_{m-l+1}^{(1)} & -\alpha t_{m-l+1}^{(1)} & 0 & 0 \\ 0 & 0 & 0 & 0 & t_{m-l}^{(1)} & -\alpha t_{m-l}^{(1)} & 0 & 0 \end{pmatrix}.$$

Note that matrix G has only four independent columns. After a simple reduction, we know the following matrix includes four nonzero eigenvalues of G:

$$G' = \begin{bmatrix} 0 & t^{(1)}_{m-l} - \alpha t^{(1)}_{m-l+1} & 0 & 0 \\ t^{(2)}_{m-l} - \alpha t^{(2)}_{m-l+1} & 0 & 0 & t^{(2)}_{l} - \alpha t^{(2)}_{l-1} \\ t^{(2)}_{l} - \alpha t^{(2)}_{l-1} & 0 & 0 & t^{(2)}_{m-l} - \alpha t^{(2)}_{m-l+1} \\ 0 & 0 & t^{(1)}_{m-l} - \alpha t^{(1)}_{m-l+1} & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & g_1 & 0 & 0 \\ g_2 & 0 & 0 & g_3 \\ g_3 & 0 & 0 & g_2 \\ 0 & 0 & g_1 & 0 \end{bmatrix}.$$

Let

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}.$$

We have

$$HG'H = \begin{bmatrix} 0 & g_1 & 0 & 0 \\ g_2 + g_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & g_2 - g_3 \\ 0 & 0 & g_1 & 0 \end{bmatrix}.$$

Thus, the four eigenvalues of G' are:

$$\lambda_{1,2} = \pm\sqrt{g_1(g_2 + g_3)},$$
$$\lambda_{3,4} = \pm\sqrt{g_1(g_2 - g_3)}.$$

Here we present two figures to show the relationship between the spectral radius and the parameter α. For both cases, the size of each subproblem is $m = 10$ and $\beta = 2$. When $\beta > 2$, the generalized SS has a faster convergence rate. Figures 6 and 7 will show the results for the latter case. The x-axis is the parameter α, while the y-axis is the spectral radius of the Jacobi matrix for the generalized SS. The top figure shows the case of one overlapping node while the lower one demonstrates the case of overlapping half the subregion. In Fig. 3, we can see that the traditional SS (when $\alpha = 0$) has a very poor convergence rate, since the overlap is so small. The convergence factor is 0.9. To reduce l_2 norm of the residual by a factor of 10^6 requires more than 60 iterations. When the parameter α approaches 1, an amazing improvement of the convergence rate appears. For $\alpha = 0.9$, the convergence factor is less than 10^{-4}. That is to say, only very few iterations are needed for any particular computation. From the second picture, we can observe when $\alpha = 0.85$ the convergence rate of the generalized SS approaches the optimum. But the optimal convergence rate in this case is even worse than having minimum overlap, and the only positive point here is the sensitivity of the convergence rate with the parameter α.

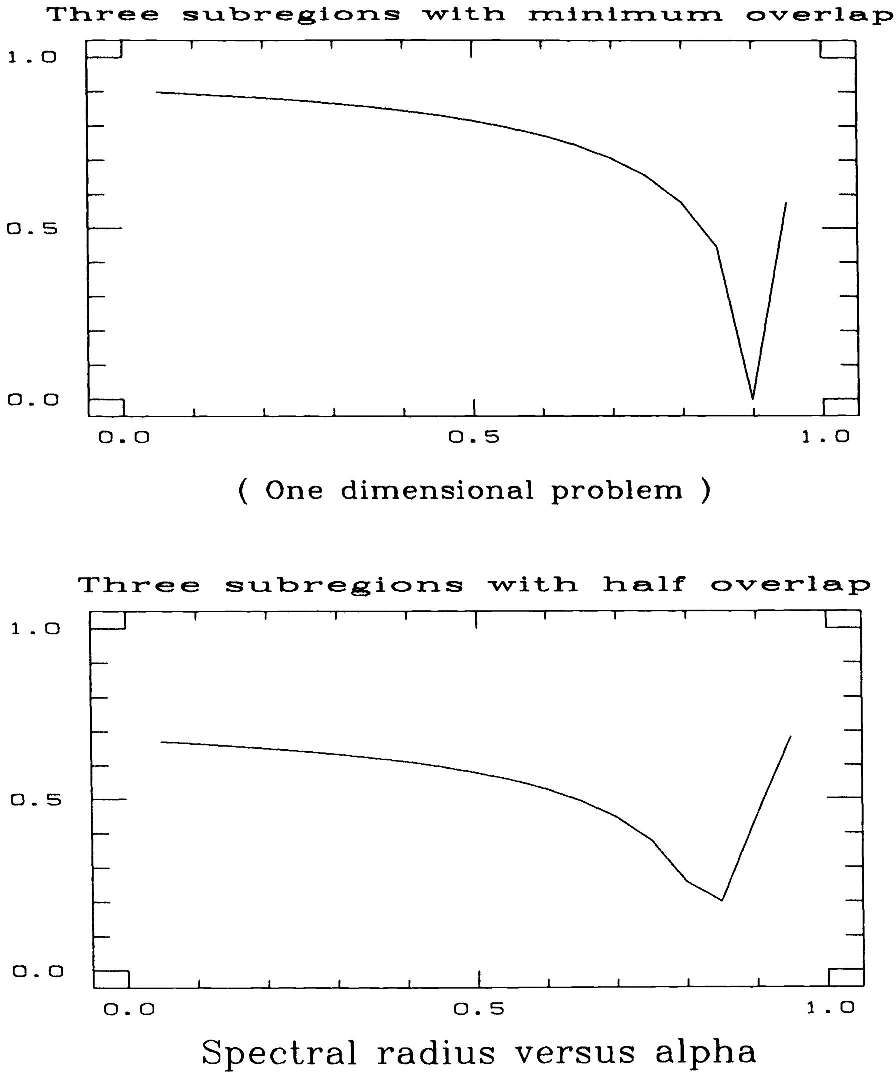


FIG. 3.

A numerical test has verified this analysis. The problem we are testing is

$$y''(x) = 2e^x \cos x, \qquad x \in (0,1),$$
$$y(0) = 0, \qquad y(1) = e\sin(1),$$

which has a solution $y(x) = e^x \sin(x)$. The solution region is covered by three overlap subregions with m unknowns each. Two neighboring subregions have one overlapping grid node. A random initial guess is used when the iteration starts and the l_2 norm is used to measure convergence. For $\alpha = 0.948, m = 20$, and $l = 2$, the residual is reduced by a factor of 10^{14} after three iterations. The results are the same for different mesh sizes. For $\alpha = 0.9, m = 10$, and $l = 2$, the residual is reduced by a factor of 10^{15} after three iterations. By comparison, the traditional SS with the same overlap will take 60 iterations to reduce the residual by only a factor of 10^5.

We apply the traditional SS to the same problem with an overlap of half of the subregion. Numerical testing shows that 11 iterations are needed to achieve a reduction of the initial residual by a factor of 10^5. If an optimal α is used, 4 iterations are needed. This again verifies the analysis shown in Fig. 3. The above results are summarized in Table 1.

TABLE 1

	Minimum overlap		Half overlap	
	Convergence factor	Number of iterations	Convergence factor	Number of iterations
Classical SAM	0.91	60	0.68	11
Generalized SAM	10^{-4}	3	0.06	4

In general, for k overlapping subregions, the nonzero eigenvalues of the Jacobi matrix are included in those of the following $(k-1) \times (k-1)$ block matrix:

$$G_k = \begin{bmatrix} D_1 & L & 0 & \cdot & \cdot & \cdot & 0 & 0 \\ U & D_2 & L & 0 & \cdot & \cdot & 0 & 0 \\ & & \ddots & \ddots & & & & \\ & & & \ddots & \ddots & & & \\ 0 & 0 & \cdot & \cdot & \cdot & U & D_2 & L \\ 0 & 0 & \cdot & \cdot & & 0 & U & D_3 \end{bmatrix}.$$

There will not be a closed form for the eigenvalues of G_k when $k > 4$. In Fig. 4 we present a numerical result for the spectral radius of J where five overlapping subregions exist. The first picture is for the minimum overlap case, while the second is for the overlapping half of the subregions. Again, a significant improvement can be obtained by choosing a good parameter α.

For the two-dimensional problem

$$\Delta U(x,y) - qU(x,y) = f(x,y), \qquad (x,y) \in (0,1) \times (0,1),$$
$$U(x,y)\,|_\Gamma = g(x,y),$$

where $q < 0$, a matrix equation

$$Ax = b$$

can be derived using the centered finite difference. Given grid size $h = 1/(n+1)$, A can be written as

$$A = T_n(\beta) \otimes I_n + I_n \otimes T_n(2),$$

where $\beta > 2$. Decompose the solution region into three overlapping subregions (in strip). The overlap pattern in the x-direction is exactly the same as in the one-dimensional case. The corresponding Schwarz enhanced matrix is

$$\widetilde{A} = \begin{bmatrix} A_1 & -F_1 & 0 \\ -E_1 & A_2 & -F_1 \\ 0 & -E_1 & A_3 \end{bmatrix},$$

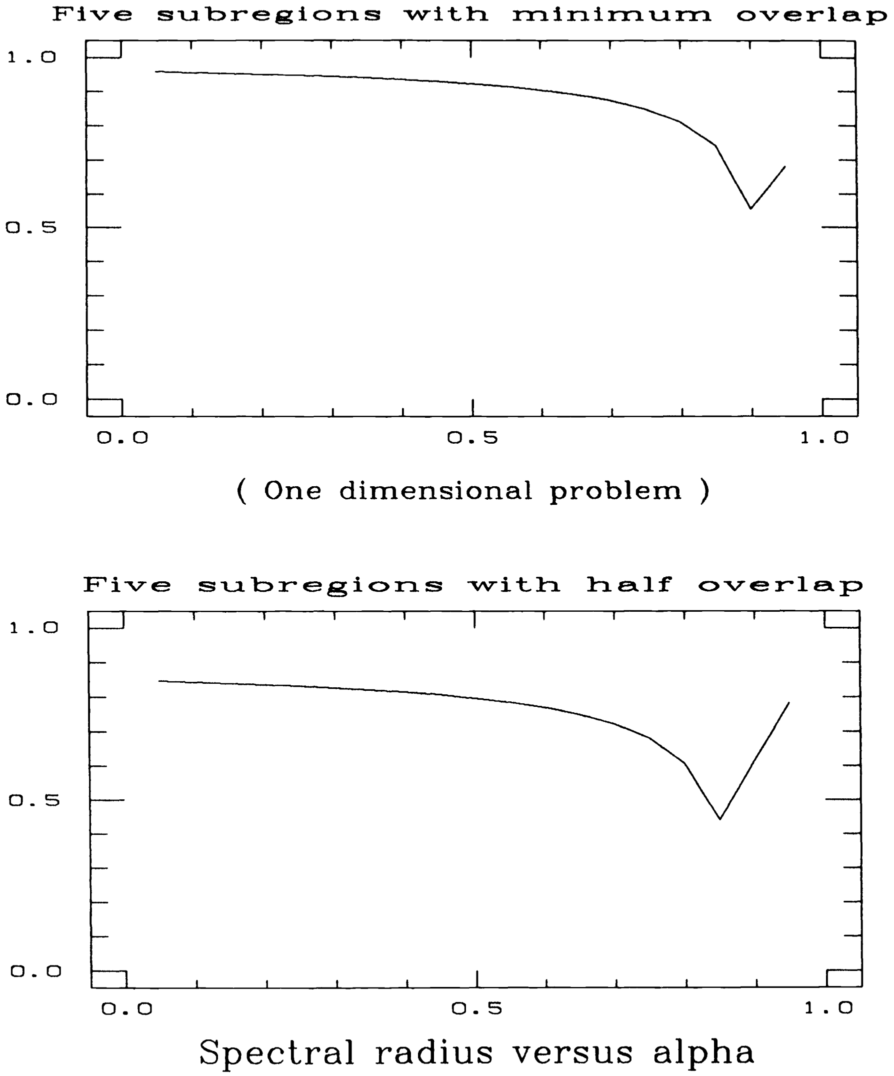


FIG. 4.

where

$$
\begin{aligned}
A_1 &= T_1 \otimes I_n + I_m \otimes T_n(2), \\
A_2 &= T_2 \otimes I_n + I_m \otimes T_n(2), \\
A_3 &= T_3 \otimes I_n + I_m \otimes T_n(2), \\
E_1 &= E_m \otimes I_n, \\
F_1 &= F_m \otimes I_n,
\end{aligned}
$$

and

- E_m is an $m \times m$ matrix with zero elements everywhere except for a 1 in position $(1, m-l)$ and α in position $(1, m-l+1)$.

- F_m is an $m \times m$ matrix with zero elements everywhere except for an α in position $(m, m-l)$ and 1 in position $(m, l-1)$.

The Jacobi iterative matrix for the generalized Schwarz splitting is

$$\begin{aligned} J &= \begin{bmatrix} A_1^{-1} & 0 & 0 \\ 0 & A_2^{-1} & 0 \\ 0 & 0 & A_3^{-1} \end{bmatrix} \begin{bmatrix} 0 & F_1 & 0 \\ E_1 & 0 & F_1 \\ 0 & E_1 & 0 \end{bmatrix} \\ &= M^{-1}N. \end{aligned}$$

Let

$$U = \begin{bmatrix} I_m \otimes X_n & 0 & 0 \\ 0 & I_m \otimes X_n & 0 \\ 0 & 0 & I_m \otimes X_n \end{bmatrix},$$

where X_n is an orthogonal matrix. Each column in X_n corresponds to an eigenvector of matrix $T_n(2)$ and $X_n T_n(2) X_n = D_n =$ diag $\{d_i\}$, $d_i = 2 + 2\cos(i\pi/(n+1))$, $i = 1, \cdots, n$. Note that U is orthogonal and $UNU = N$. So

$$\begin{aligned} J' &= UJU^T \\ &= (UMU^T)^{-1}N \\ &= \widetilde{M}N \end{aligned}$$

where

$$\begin{aligned} \widetilde{M} &= \begin{bmatrix} \widetilde{A}_1 & 0 & 0 \\ 0 & \widetilde{A}_2 & 0 \\ 0 & 0 & \widetilde{A}_3 \end{bmatrix}, \\ \widetilde{A}_i &= (I_m \otimes X_n) A_i (I_m \otimes X_n)^T \\ &= T_i \otimes I_n + I_m \otimes D_n. \end{aligned}$$

Let P be the permutation matrix which permutes row $(k-1)n+i$ to $(i-1)3m+k$, $k = 1, \cdots, 3m$, $i = 1, \cdots, n$. Then

$$PJ'P^T = \begin{bmatrix} J(d_1) & & & \\ & J(d_2) & & \\ & & \ddots & \\ & & & J(d_n) \end{bmatrix},$$

where each $J(d_i)$ is the Jacobi iterative matrix of the generalized SAM for matrix $T_n(d_i)$ in the one-dimensional case. Similar to the traditional SAM, we found that the convergence of the lower frequency components are slower than that of higher frequencies. We present two pictures in Fig. 5 to show how $\rho_{J(d_i)}$ changes when d_i changes. The first represents three subregions with minimum overlap, while the second shows the same number of subregions with half overlap.

Another two sets of figures (Figs. 6 and 7) present the relations between the spectral radius $\rho_{J(d_i)}$ and the parameter α. The first set is for $J(2.01)$ and the other is

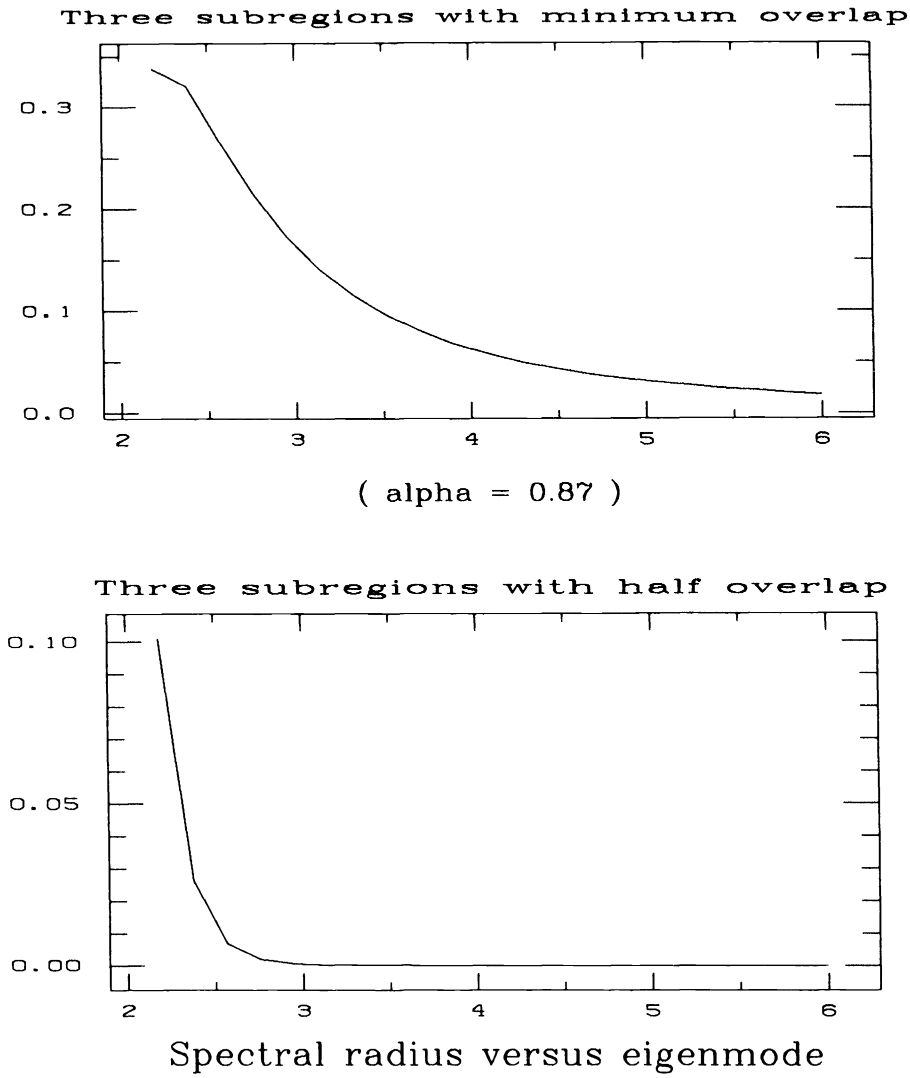


FIG. 5.

for $J(5.99)$, which represent the lowest and the highest frequencies of the eigenmodes, respectively. Both sets have one figure for minimum overlapping (one grid line) and another for overlapping half of the subregion. We can see that the sensitivity of the convergence rate with the parameter α is better in a two-dimensional problem. It is also noticeable that the convergence of the higher frequency mode is very fast for all α.

Numerical testing results for the model problem

$$\begin{aligned}&\Delta U(x,y) = -2x(1-x) - 2y(1-y), \qquad (x,y) \in (0,1) \times (0,1),\\ &U(x,y)\mid_\Gamma = 0\end{aligned}$$

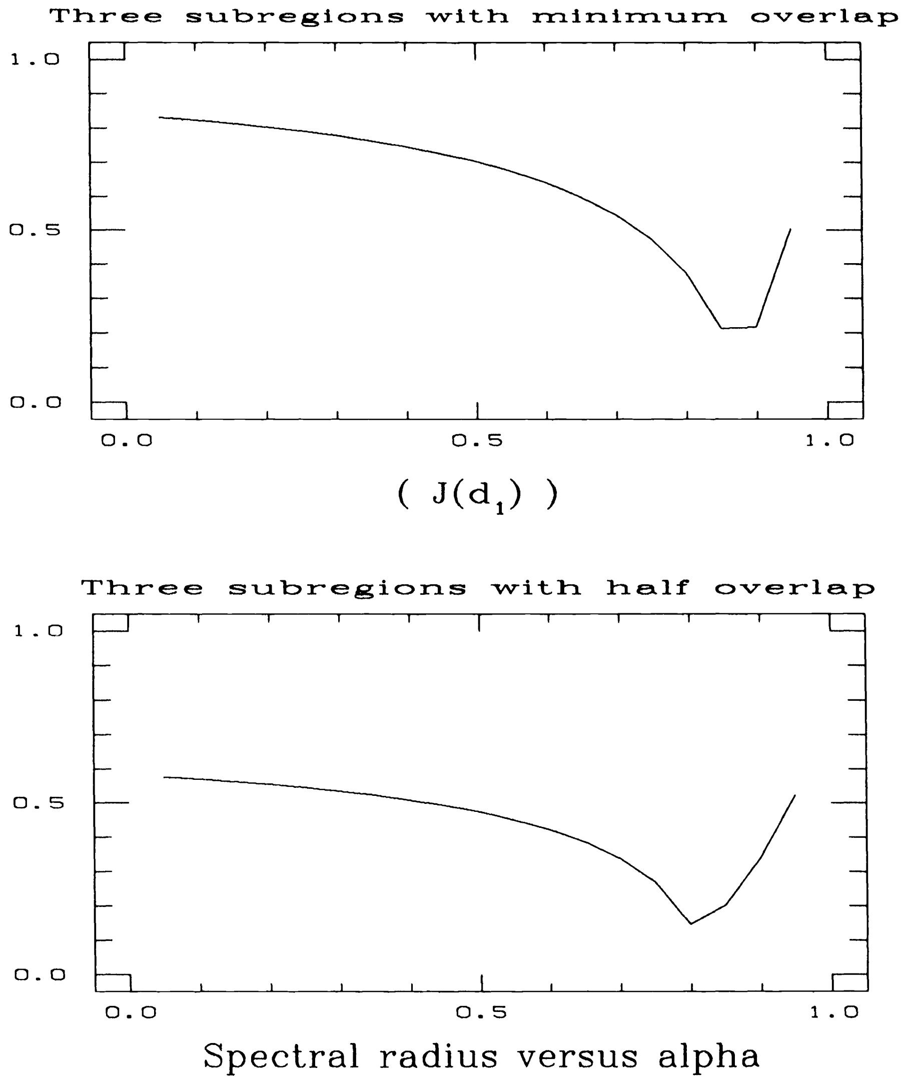


FIG. 6.

are given in Fig. 8. We present the relations between the number of iterations and the parameter α in these figures. Testing is carried out for three and five subregion cases, and for each decomposition, both minimum overlap and half overlap are tested. The initial guess is randomly generated. To make the programming easier, the grid size is slightly different for each case. h is between 1/40 and 1/50. The results plainly verify our analysis. The x-axis is the parameter α while the y-axis is the number of iterations needed for reducing the initial error by a factor of 10^5.

4. Conclusion. From the above analysis, a generalization of the traditional SAM is presented and the improvement of its performance is significant. The results of this study suggest that there may be other interesting splittings or couplings with good or even better performance characteristics. So far, our analysis has been

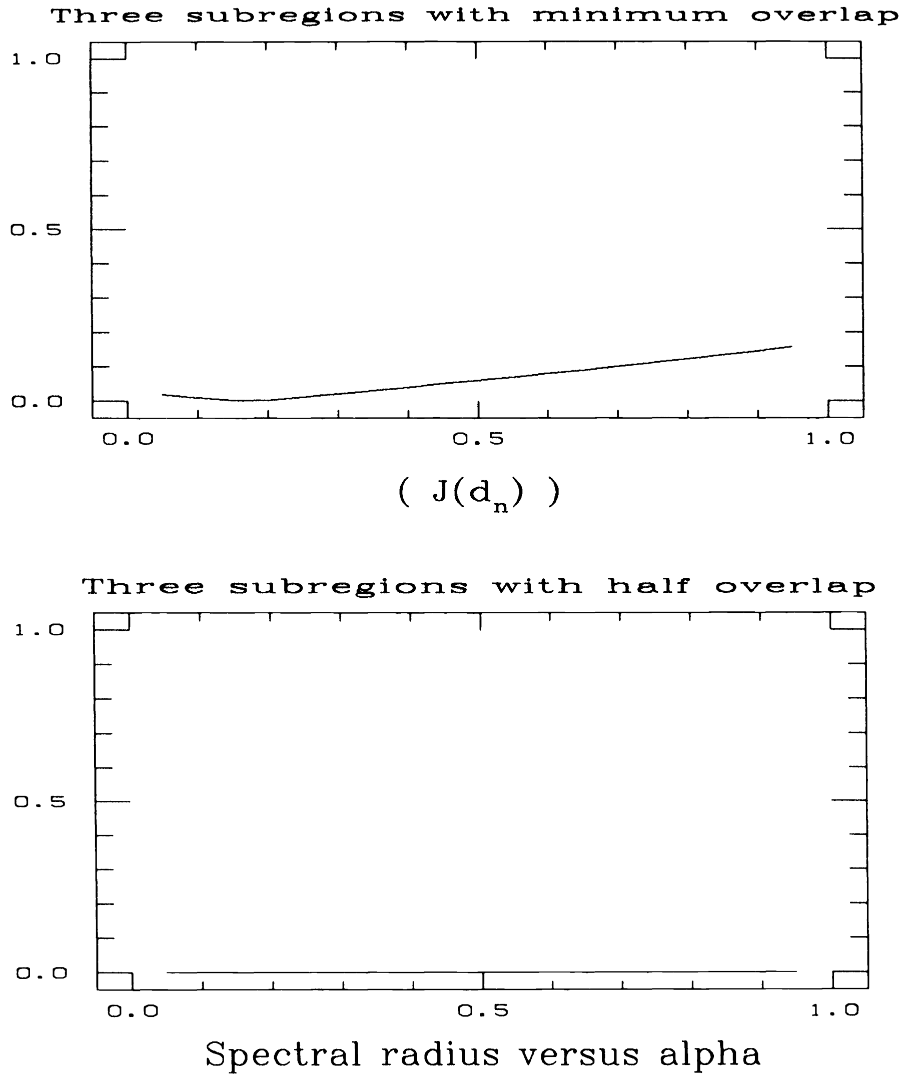


FIG. 7.

restricted to a simple case, namely, the strip decomposition. In particular, the close form of the spectral radius did not provide us with an insight of how the convergence is related to the decomposition and the parameter α. We do not have the same intuitive understanding we had for the classical SAM. More interesting problems remain to be studied. The effects of different couplings on the convergence of SAM have also been observed in a study for the fourth order equation [1].

We should also indicate that there is an extreme sensitivity between the parameter α and the convergence rate of generalized SS. A better understanding of this sensitivity is needed to make this generalized SS a practical technique.

Acknowledgment. The author would like to thank Gérard Meurant and referees for their valuable comments on this paper.

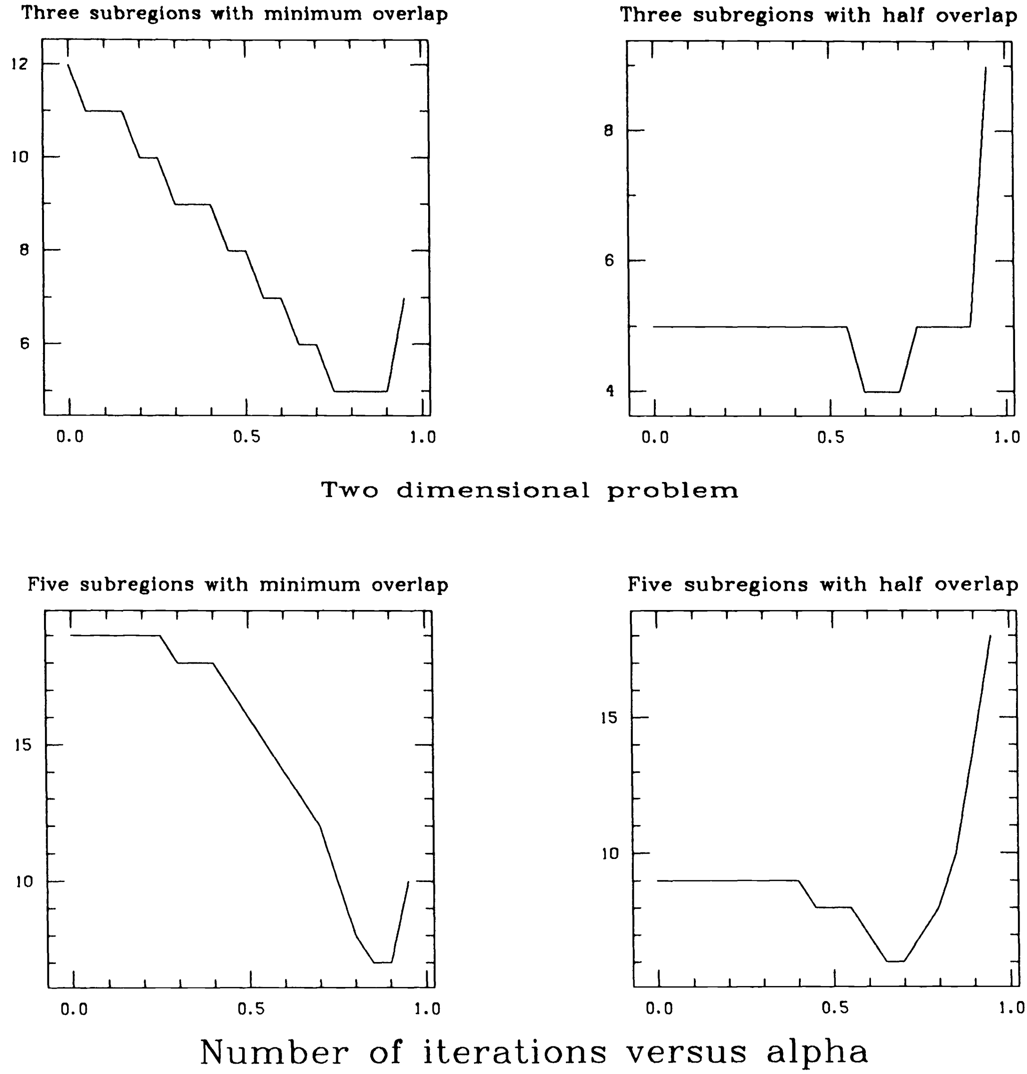


FIG. 8.

REFERENCES

[1] T. CHAN, D. GOOVAERTS, AND W. P. TANG, *A fast biharmonic solver on irregular domains using a generalized Schwarz alternating method*, Tech. Report 37, University of Waterloo, Waterloo, Ontario, Canada, 1989.

[2] R. V. DINH, R. GLOWINSKI, AND J. PERIAUX, *Applications of domain decomposition techniques to the numerical solution of the Navier-Stokes equations*, Tech. Report, INF-LAB, France, 1980.

[3] M. DRYJA, *An additive Schwarz algorithm for two- and three-dimensional finite element elliptic problems*, in Domain Decomposition Methods, T. Chan, R. Glowinski, J. Periaux, and O. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989, pp. 168–172.

[4] M. DRYJA AND O. WIDLUND, *An additive variant of Schwarz alternating method for the case of many subregions*, Tech. Report 339, New York University, New York, NY, 1987.

[5] L. W. EHRLICH, *The numerical Schwarz alternating procedure and SOR*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 989–993.

[6] C. F. FISCHER AND R. A. USMANI, *Properties of some tridiagonal matrices and their application to boundary value problems*, SIAM J. Numer. Anal., 6 (1969), pp. 127–142.

[7] G. H. GOLUB AND M. L. OVERTON, *The convergence of inexact Chebychev and Richardson iterative methods for solving linear systems*, Tech. Report, Stanford University, Stanford, CA, 1987.

[8] A. GREENBAUM, C. LI, AND H. Z. CHAO, *Parallelizing preconditioned conjugate gradient algorithms*, Tech. Report, New York University, New York, NY, 1988.

[9] L. V. KANTOROVICH AND V. I. KRYLOV, *Approximate methods of higher analysis*, Fourth Edition, P. Noordhoff LTD, Groningen, the Netherlands, 1958.

[10] P. L. LIONS, *On the Schwarz alternating method.* I, in First International Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. Golub, G. Meurant, and J. Periaux, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1988, pp. 1–42.

[11] U. MEIER, *Two parallel SOR variants of the Schwarz alternating procedure*, Tech. Report, Zentralinstitut für Angewandte Mathematik, 5170 Jülich, West Germany, 1986.

[12] J. OLIGER, W. SKAMAROCK, AND W. TANG, *Convergence analysis and acceleration of the Schwarz alternating method*, Tech. Report, Computer Science Dept., Stanford University, Stanford, CA, 1986.

[13] G. RODRIGUE, Personal communication, 1987.

[14] G. RODRIGUE AND J. SIMON, *A generalization of the numerical Schwarz algorithm*, in Computing Methods in Applied Sciences and Engineering VI, R. Glowinski and J. Lions, eds., North-Holland, Amsterdam, New York, Oxford, 1984, pp. 273–282.

[15] R. SCHREIBER, Personal communication, 1986.

[16] H. A. SCHWARZ, *Ueber einige abbildungsaufgaben*, J. Reine Angew. Math., 70 (1869), pp. 105–120.

[17] W. P. TANG, *Schwarz splitting and template operators*, Ph.D. thesis, Computer Science Dept., Stanford University, Stanford, CA, 1987.

[18] H. YSERENTANT, *Hierarchical bases give conjugate type method a multigrid speed of convergence*, Appl. Math. Comput., 19 (1986), pp. 147–158.

SIAM J. Sci. Stat. Comput.
Vol. 13, No. 2, pp. 596–610, March 1992

FOURIER ANALYSIS OF FINITE ELEMENT PRECONDITIONED COLLOCATION SCHEMES*

MICHEL O. DEVILLE† AND ERNEST H. MUND ‡

Abstract. This paper investigates the spectrum of the iteration operator of some finite element preconditioned Fourier collocation schemes. The first part of the paper analyses one-dimensional elliptic and hyperbolic model problems and the advection-diffusion equation. Analytical expressions of the eigenvalues are obtained with use of symbolic computation. The second part of the paper considers the set of one-dimensional differential equations resulting from Fourier analysis (in the transverse direction) of the two-dimensional Stokes problem. All results agree with previous conclusions on the numerical efficiency of finite element preconditioning schemes.

Key words. finite element, collocation method, eigenvalue analysis

AMS(MOS) subject classifications. 65N30, 65N35

1. Introduction.

In the recent past, Chebyshev collocation schemes have been applied extensively to the numerical integration of the Navier–Stokes equations [1], [3], [4]. For scalar elliptic problems, it is well known that the condition number of the matrix system of discrete algebraic equations increases rapidly with N, the number of degrees of freedom of the problem at hand. Therefore, the preconditioning technique seems to be the only adequate tool in order to overcome this numerical burden. The present authors [5], [6] demonstrated that finite elements (FE) constitute powerful preconditioners for general second-order elliptic equations. In [3], several fluid flow elements in velocity-pressure formulation were investigated. From the analysis of the eigenspectrum of the iteration operator, it was shown that the Q2–Q1 element is the best choice for the steady Stokes problem. As all the previous analyses on finite element preconditioning were carried out numerically, the present note aims at analytical results through use of symbolic manipulation languages (cf. [10]).

For the sake of simplicity, we will restrict ourselves to the study of a finite element preconditioned Fourier collocation scheme. Although the Fourier collocation scheme has a condition number of the same order of magnitude as finite element methods and therefore does not require preconditioning per se, the Fourier case provides uniform mesh and a much simpler algebraic treatment; moreover, there is numerical evidence that the Fourier results extend to the more difficult cases like Chebyshev or Legendre collocation [1]. In §2, a one-dimensional elliptic model is considered. The collocation process is preconditioned by Lagrangian linear, quadratic, cubic, and Hermite cubic elements, respectively. The Richardson iteration method is set up with these FE preconditioners as approximate operators and algebraic solvers. Using the spatial structure of the eigenvectors of the Fourier solutions, one may perform a full analysis of the eigenvalues of the iteration operator. This theoretical investigation corroborates

*Received by the editors January 16, 1990; accepted for publication January 9, 1991.

†Université Catholique de Louvain, Unité de Mécanique Appliquée, Louvain-La-Neuve, Belgium. The research of this author was partially supported by National Aeronautics and Space Administration (NASA) contract NAS-1-18605 while he was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, Virginia.

‡Université Libre de Bruxelles, Service de Métrologie Nucléaire, Brussels, Belgium, and Université Catholique de Louvain, Unité de Thermodynamique et Turbomachines, Louvain-La-Neuve, Belgium. This author acknowledges the continuous financial support of the Fonds National de la Recherche Scientifique (FNRS).

the previous numerical analyses [6]. In §3, a one-dimensional hyperbolic model is investigated using linear and quadratic FE preconditioning. The upwinding technique is also examined. A further model consists in an advection-diffusion equation. In §4, the Stokes problem is reduced to a one-dimensional incompressible flow model amenable to Fourier discretization. The Q2-Q1 and Q1-P0 elements are candidates as preconditioners. A similar Fourier analysis is done. The results corroborate numerical experiments carried out in the framework of Chebyshev collocation [3].

2. Elliptic model. Let us first consider the simple elliptic problem:

$$-u_{xx} = f, \qquad 0 \le x \le 2\pi, \tag{2.1}$$

with periodic boundary conditions. The subscript indicates partial derivative. The Fourier approximation of the dependent variable u is

$$u_N = \sum_{p=-N/2}^{(N/2)-1} \hat{u}_p e^{ipx_j}, \qquad 0 \le j < N, \tag{2.2}$$

where $\hat{u}_p$ are the discrete Fourier coefficients and x_j the collocation points defined by

$$x_j = \frac{2\pi j}{N}, \qquad j \in [0, N[. \tag{2.3}$$

The linear system corresponding to (2.1) may be found in [1] and will be denoted by L_c. The eigenfunctions of (2.1) are

$$\xi_j(p) = e^{ipx_j}, \qquad 0 \le j < N, \tag{2.4}$$

with the corresponding eigenvalues

$$\lambda(p) = p^2, \qquad p \in \left[-\frac{N}{2}, \frac{N}{2} - 1\right]. \tag{2.5}$$

The collocation problem will be preconditioned by finite elements. Introducing the approximate FE operator $\hat{L}$, the preconditioned Richardson iteration is written as

$$\bar{u}^{k+1} = \bar{u}^k - \alpha_k \hat{L}^{-1}(L_c \bar{u}^k - \bar{f}), \tag{2.6}$$

where k is an iteration index; α_k a relaxation factor; and $\bar{u}, \bar{f}$ the vectors corresponding to the unknowns and source terms at the collocation points. The convergence of (2.6) is governed by the spectral radius $\rho(A)$ of the iteration operator defined by $A = I - \alpha_k \hat{L}^{-1} L_c$. The optimal value of the relaxation factor is

$$\alpha_{\text{opt}} = \frac{2}{\lambda_{\min} + \lambda_{\max}}, \tag{2.7}$$

where $\lambda_{\min}$ and $\lambda_{\max}$ are the minimum and maximum eigenvalues of $\hat{L}^{-1} L_c$. An approximate estimate of the number of iterations n needed to reduce the error norm by a factor ζ is given by

$$n = -\log \zeta / R_\infty(A), \tag{2.8}$$

where $R_\infty(A) = -\log \rho(A)$ is the asymptotic rate of convergence of the iteration matrix. The spectral radius $\rho(A)$, which is involved in the error reduction process with the use of α_{opt} (2.7), is given by

$$\rho(A) = \frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}}. \tag{2.9}$$

In order to investigate this quantity for various preconditioners, we have to define the finite element problem more precisely.

Lagrangian linear elements, Hermite cubic elements (i.e., Q1, P3 in Ciarlet's notations [2]) as well as higher-order Lagrangian interpolants have their vertices at the Fourier collocation grid (2.3). However, for Lagrangian quadratics (Q2), midpoints are added at

$$x_{j+\frac{1}{2}} = \frac{2\pi}{N}\left(j + \frac{1}{2}\right), \qquad j \in [0, N[, \tag{2.10}$$

while for Lagrangian cubics (Q3), we have additional grid nodes located at

$$x_{j+1/3}, x_{j+2/3}, \quad j \in [0, N[, \tag{2.11}$$

with obvious definitions. For the Lagrangian case, the FE unknowns are the nodal values, while for the Hermite case, the unknowns are u_j and u'_j , the prime denoting the first-order derivative of u. In [5], the iteration operator is written as

$$A \equiv I - S_h^{-1} M_h I_h L_c, \tag{2.12}$$

where S_h is the stiffness matrix, M_h the mass matrix, and I_h an interpolation matrix evaluating the Fourier interpolant of the collocation operator at the FE nodes. The mesh size of the FE grid is defined by

$$h = 2\pi/N.$$

For Q1 elements, I_h reduces to the unit matrix; for higher-order interpolants, however, the structure of this matrix is more complicated. In order to avoid writing the details of I_h, we will systematically assume the use of static condensation. Consequently, the iteration operator may be written as follows:

$$A = I - \hat{S}_h^{-1} \hat{M}_h L_c, \tag{2.13}$$

where $\hat{S}_h$ and $\hat{M}_h$ refer to stiffness and mass matrices after static condensation.

2.1. Linear Lagrangian elements. For an interior node, the expressions for the stiffness and mass matrices are well known:

$$S_h u_j = \frac{1}{h}(-u_{j-1} + 2u_j - u_{j+1}), \tag{2.14}$$

$$M_h f_j = \frac{h}{6}(f_{j-1} + 4f_j + f_{j+1}). \tag{2.15}$$

Fourier analysis of (2.14), (2.15) with the eigenfunction (2.4) leads to the expression of the eigenspectrum of $S_h^{-1} M_h L_c$, denoted by $\sigma(p)$,

$$\sigma(p) = \frac{(ph/2)^2}{\sin^2 ph/2}\frac{(2 + \cos ph)}{3}, \qquad -\frac{N}{2} \le p \le \frac{N}{2} - 1. \tag{2.16}$$

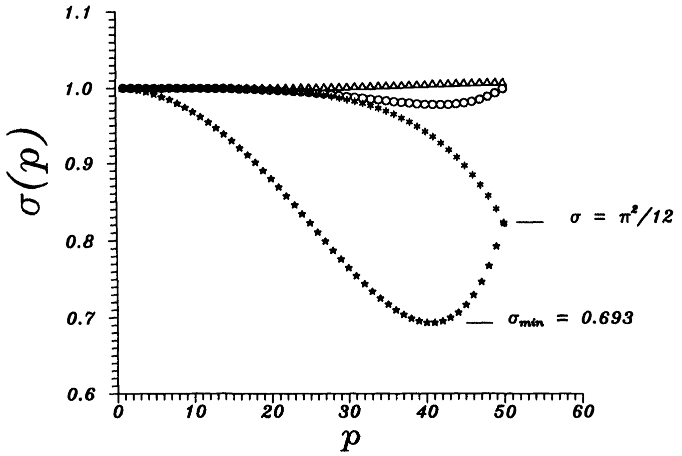


FIG. 1. *Eigenspectrum of* $\hat{S}_h^{-1}\hat{M}_h L_c$ *for an elliptic model. Preconditioners:* Q1 (★); Q2 (∗); Q3 (△); *Cubic Hermite* (○).

Typically, the second factor in the right-hand side of (2.16) comes from the contribution of the mass matrix. In the case of finite difference (FD) preconditioning, this factor is one. For $p = 0$, $\sigma(p) = 1$, while for $p = -N/2, \sigma(p) = \pi^2/12$. This last value should be compared to the FD equivalent, which is $\sigma(p) = \pi^2/4$ [7]. The eigenvalue spectrum of the FE preconditioning is reduced because of the beneficial presence of the mass matrix. Figure 1 shows the behaviour of $\sigma(p)$ with respect to p for $h = 2\pi/100$. The function has a minimum value equal to 0.693. Therefore, the optimum value for α is

$$\alpha_{\text{opt}} \simeq 1.18, \tag{2.17}$$

and overrelaxation is possible for FE preconditioning unlike the FD preconditioning where underrelaxation is required to converge. In practice, the Q1 preconditioning with a spectral radius of 0.18 converges twice as fast as the FD preconditioner whose spectral radius is of the order of 0.42.

2.2. Quadratic Lagrangian elements. The equations related to nodes j and $j \pm \frac{1}{2}$ may be cast in the following matrix form:

$$\frac{1}{3h}\begin{pmatrix} -8 & 16 & -8 & 0 & 0 \\ 1 & -8 & 14 & -8 & 1 \\ 0 & 0 & -8 & 16 & -8 \end{pmatrix}\begin{pmatrix} u_{j-1} \\ u_{j-1/2} \\ u_j \\ u_{j+1/2} \\ u_{j+1} \end{pmatrix} = \frac{h}{15}\begin{pmatrix} 1 & 8 & 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 4 & 1 & -\frac{1}{2} \\ 0 & 0 & 1 & 8 & 1 \end{pmatrix}\begin{pmatrix} f_{j-1} \\ f_{j-1/2} \\ f_j \\ f_{j+1/2} \\ f_{j+1} \end{pmatrix}. \tag{2.18}$$

The use of static condensation eliminates the contribution of $u_{j-1/2}$ and $u_{j+1/2}$ and (2.18) reduces to only one relationship for node j on the collocation grid

$$\frac{1}{h}(-u_{j-1} + 2u_j - u_{j+1}) = \frac{h}{3}(f_{j-1/2} + f_j + f_{j+1/2}). \tag{2.19}$$

Note that for $\hat{S}_h$ on the left-hand side of (2.19), we recover the stiffness matrix S_h associated to Q1 elements, whereas in the right-hand side, $\hat{M}_h$ corresponds to a different quadrature rule. Carrying out the Fourier analysis of (2.19), we obtain

$$\sigma(p) = \frac{(ph/2)^2}{\sin^2(ph/2)}\frac{1}{3}(1 + 2\cos(ph/2)), \qquad -\frac{N}{2} \leq p \leq \frac{N}{2} - 1. \tag{2.20}$$

For the particular values $p = 0$ and $p = -N/2$, $\sigma(p)$ is equal to 1 and $\pi^2/12$, respectively. As $\sigma(p)$ is a monotically decreasing function with respect to p (Fig. 1), the optimum value of α is

$$\alpha_{\text{opt}} = 2/(1+\pi^2/12) = 1.0974, \tag{2.21}$$

and the corresponding spectral radius $\rho(A)$ is equal to 0.0974.

2.3. Cubic Lagrangian elements. For the sake of compactness, we give the local stiffness and mass matrices over the uniform mesh:

$$S_h = \frac{2}{h}\begin{pmatrix} 37/20 & -189/80 & 27/40 & -13/80 \\ -189/80 & 27/5 & -297/80 & 27/40 \\ 27/40 & -297/80 & 27/5 & -189/80 \\ -13/80 & 27/40 & -189/80 & 37/20 \end{pmatrix}. \tag{2.22}$$

$$M_h = \frac{h}{2}\begin{pmatrix} 16/105 & 33/280 & -3/70 & 19/840 \\ 33/280 & 27/35 & -27/280 & -3/70 \\ -3/70 & -27/280 & 27/35 & 33/280 \\ 19/840 & -3/70 & 33/280 & 16/105 \end{pmatrix}. \tag{2.23}$$

Assembling the matrices of (2.22), (2.23) over two adjacent elements and eliminating the four unknowns attached to the interior nodes $u_{j\pm1/3}$, $u_{j\pm2/3}$, we are left with the relation:

$$\begin{aligned} \frac{1}{h}(-u_{j-1}+2u_j-u_{j+1}) = \frac{h}{10}\Bigg(&\frac{f_{j-1}}{6}+\frac{3}{4}f_{j-2/3}+3f_{j-1/3} \\ &+\frac{13}{6}f_j+3f_{j+1/3}+\frac{3}{4}f_{j+2/3}+\frac{f_{j+1}}{6}\Bigg). \end{aligned} \tag{2.24}$$

Here again, as in the previous case, we recover in the left-hand side of (2.24) the stiffness matrix of Q1 elements. Fourier analysis of (2.24) leads to the eigenvalue spectrum $\sigma(p)$:

$$\begin{aligned} &\sigma(p) = \frac{(ph/2)^2}{\sin^2(ph/2)}\frac{1}{10}\left(\frac{4}{3}\cos^3\frac{ph}{3}+3\cos^2\frac{ph}{3}+5\cos\frac{ph}{3}+\frac{2}{3}\right), \\ &-\frac{N}{2}\le p\le\frac{N}{2}-1. \end{aligned} \tag{2.25}$$

The particular values of $\sigma(p)$ corresponding to $p = 0$ and $p = -N/2$ are $\sigma(p) = 1$ and $49\pi^2/480 \simeq 1.007522$. The optimal value of the relaxation factor is given by $2/(1+49\pi^2/480) \simeq 0.9963$ and the corresponding spectral radius $\rho(A)$ is equal to 0.00375.

Looking back at the results in the previous subsections, one observes that the spectral radius $\rho(A)$ diminishes with increasing polynomial degrees. This does not mean, however, that one should use higher-order elements in the preconditioning because they involve more computational work as the bandwidth of the algebraic system increases.

2.4. Cubic Hermite elements. At node j, the discrete equations are

$$\begin{aligned} &-\frac{6}{5h}(u_{j-1}-2u_j+u_{j+1})-\frac{1}{10}(u'_{j-1}-u'_{j+1}) \\ &\quad=\frac{13h^2}{420}(f'_{j-1}-f'_{j+1})+\frac{h}{7}\left(\frac{9}{10}f_{j-1}+\frac{26}{5}f_j+\frac{9}{10}f_{j+1}\right), \end{aligned} \tag{2.26}$$

$$\begin{aligned}(2.27)\qquad &\frac{1}{10}(u_{j-1} - u_{j+1}) - \frac{h}{30}(u'_{j-1} - 8u'_j + u'_{j+1}) \\ &= -\frac{13}{420}h^2(f_{j-1} - f_{j+1}) - \frac{h^3}{140}\left(f'_{j-1} - \frac{8}{3}f'_j + f'_{j+1}\right).\end{aligned}$$

Fourier analyzing (2.26), one gets the spectrum
(2.28)

$$\sigma(p) = \frac{(ph/2)^2}{6\sin^2(ph/2) - (ph/2)\sin(ph/2)\cos(ph/2)}\frac{1}{7}\left(26 + 9\cos ph + \frac{13}{6}ph\sin ph\right), \qquad |ph| \in [0, \pi].$$

For $p = 0$ and $-N/2$, $\sigma(p) = 1$ and $17\pi^2/168 \simeq 0.9987$, respectively. Figure 1 displays the behaviour of $\sigma(p)$, which is first slightly decreasing with respect to p, achieving its minimum value 0.97722 and then increasing to 1. The optimal value of α is 1.01152 and the corresponding spectral radius 0.0115, an intermediate value between those of Q2 and Q3 preconditioning.

3. Hyperbolic problems.

We now turn our attention to the first-order differential equation

$$(3.1)\qquad u_x = f,$$

in the periodic case. The eigenfunctions of (3.1) are again (2.4) with the eigenvalues

$$\lambda(p) = ip, \qquad p \in \left[-\frac{N}{2}, \frac{N}{2} - 1\right].$$

3.1. Linear Lagrangian elements.

Using the standard Galerkin approach, a centered scheme is produced and yields the discrete equation

$$(3.2)\qquad \frac{u_{j+1} - u_{j-1}}{2} = \frac{h}{6}(f_{j-1} + 4f_j + f_{j+1}).$$

By Fourier analysis, we obtain

$$(3.3)\qquad \sigma(p) = \frac{ph}{\sin ph}\frac{2 + \cos ph}{3}, \qquad |ph| \in [0, \pi].$$

For $p = 0$, $\sigma(p) = 1$, while $\sigma(-N/2)$ is obviously unbounded as in the FD case. The presence of the mass matrix does not help to circumvent the difficulty.

One may proceed, however, using an upwinding technique. This has been a key step in treating hyperbolic problems. In finite elements, the method uses separate test and trial function spaces, i.e., the Petrov–Galerkin method. There are several ways to implement upwinding. Let us introduce the weight functions $w^i(r)$, $(i = 1, 2)$, defined on the reference interval $[-1, +1]$ by Heinrich and Zienkiewicz [8]:

$$(3.4)\qquad w^i(r) = \varphi^i(r) + (-1)^i\epsilon F(r), \qquad -1 \le r \le 1,$$

where $\varphi^i(r)$ are the linear Lagrangian trial functions, $F(r)$ an auxiliary quadratic element vanishing at both end points

$$(3.5)\qquad F(r) = \frac{3}{4}(1 - r^2),$$

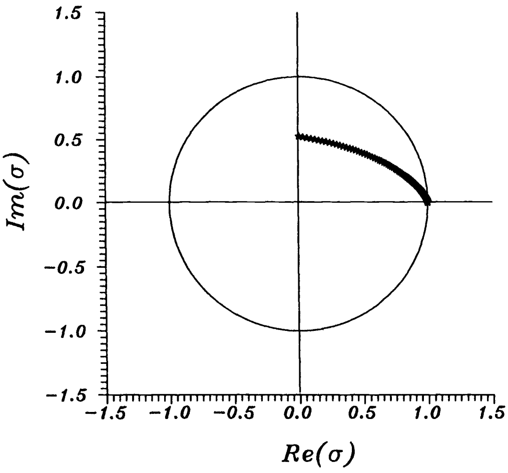


FIG. 2. *Eigenspectrum of $\hat{S}_h^{-1}\hat{M}_h L_c$ for a first-order operator. Upwinding with a parameter $\epsilon = 1$ is used.*

and ϵ the upwinding parameter to be given independently. Using φ^i and w^i as trial and test functions, respectively, one gets

$$(3.6) \qquad -\frac{1+\epsilon}{2h}u_{j-1} + \frac{\epsilon}{h}u_j + \frac{1-\epsilon}{2h}u_{j+1} = \frac{2+3\epsilon}{12}f_{j-1} + \frac{2}{3}f_j + \frac{2-3\epsilon}{12}f_{j+1}.$$

This equation reduces to (3.2) when $\epsilon = 0$ (no upwinding). With a value of ϵ as yet undefined, the Fourier analysis of (3.6) leads to complex eigenvalues given by

$$(3.7) \qquad \begin{aligned} &\sigma(p) = \frac{1}{6}\frac{1}{\epsilon(1-\cos ph) + i\sin ph}(3\epsilon ph \sin ph + 2iph(2+\cos ph)), \\ &|ph| \in [0, \pi]. \end{aligned}$$

For $p = 0, \sigma(p) = 1$, while for $p = -N/2, \sigma(p) = -i\pi/6\epsilon$. Upwinding forces the spectrum of $S_h^{-1}M_hL_c$ almost entirely inside the unit circle, as shown in Fig. 2, where the eigenvalues (3.7) have been plotted for $\epsilon = 1$ and positive values of ph. The spectral radius of the matrix A in this case is approximately equal to $\sqrt{5}/2$ and underrelaxation is required in order to ensure convergence of the preconditioning iterations. The eigenvalues (3.7) being complex, the evaluations of α_{opt} and the spectral radius $\rho(A)$ are no longer given by (2.7) and (2.9).

3.2. Quadratic Lagrangian elements. Another way to solve (3.1) consists of using a FE preconditioner based on quadratic Lagrangian elements. Applying the Galerkin approach and assembling the contributions of two adjacent elements at node j, one obtains a set of three equations related to nodes j and $j \pm \frac{1}{2}$ similar to (2.18), which are cast in matrix form:

(3.8)

$$\begin{pmatrix} -\frac{2}{3} & 0 & \frac{2}{3} & 0 & 0 \\ \frac{1}{6} & -\frac{2}{3} & 0 & \frac{2}{3} & -\frac{1}{6} \\ 0 & 0 & -\frac{2}{3} & 0 & \frac{2}{3} \end{pmatrix} \begin{pmatrix} u_{j-1} \\ u_{j-1/2} \\ u_j \\ u_{j+1/2} \\ u_{j+1} \end{pmatrix} = \frac{h}{30}\begin{pmatrix} 2 & 16 & 2 & 0 & 0 \\ -1 & 2 & 8 & 2 & -1 \\ 0 & 0 & 2 & 16 & 2 \end{pmatrix} \begin{pmatrix} f_{j-1} \\ f_{j-1/2} \\ f_j \\ f_{j+1/2} \\ f_{j+1} \end{pmatrix}.$$

If static condensation is carried out through Gaussian elimination, the contribution of the exterior nodes u_{j-1}, u_{j+1} disappears and one is left with a staggered scheme:

$$(3.9) \qquad \frac{2}{3}(u_{j-1/2} - u_{j+1/2}) = \frac{h}{60}(-f_{j-1} + 12f_{j-1/2} + 18f_j + 12f_{j+1/2} - f_{j+1}),$$

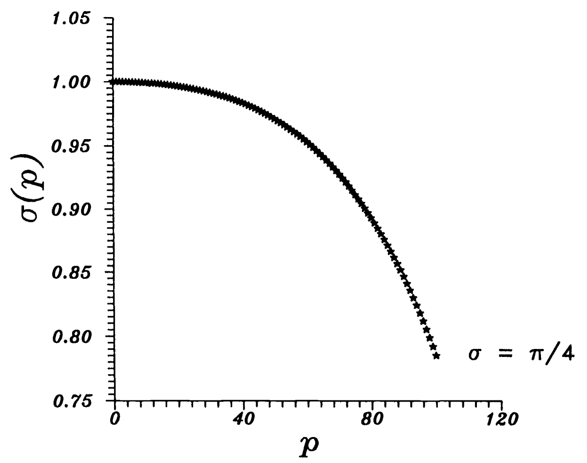


FIG. 3. *Eigenspectrum of a first-order operator preconditioned by* Q2 *elements.*

suitable for Fourier analysis. Its spectrum is given by

$$\sigma(p) = \frac{\frac{ph}{2}}{\sin\frac{ph}{2}} \frac{9 + 12\cos\frac{ph}{2} - \cos ph}{20}, \qquad |ph| \in [0, \pi]. \tag{3.10}$$

It is monotonically decreasing and bounded by $\sigma(0) = 1$ and $\sigma(-N/2) = \pi/4$ as shown in Fig. 3. Note that the first factor in the right-hand side of (3.10) is identical to the spectrum obtained in the FD case where the function is computed on the main grid while the derivative is evaluated by first-order differences on a staggered grid. The second factor whose maximum value is equal to one is induced by the presence of the mass matrix and reduces to unity in the FD case. The optimal value of α is equal to

$$\alpha_{\text{opt}} = 2/(1 + \pi/4) = 1.1202,$$

and the corresponding spectral radius $\rho(A)$ is equal to 0.1202. This staggered scheme generated by Q2 elements is the key to success for FE preconditioning of Navier–Stokes problems. This excellent behaviour explains the reason why in Demaret and Deville [4], the relaxation parameter was almost independent of the Reynolds number.

3.3. Advection-diffusion model. The last scalar model analyzed in this paper is the one-dimensional advection-diffusion problem. The differential equation writes

$$-\kappa u_{xx} + c u_x = f(x), \tag{3.11}$$

where κ is the diffusion coefficient and c the constant advection velocity. Of particular interest are advection-dominated problems, which impose severe conditions on the element mesh size (cf. [9]). With the eigenvectors (2.4), the eigenvalues of (3.11) are

$$\lambda(p) = p^2 + \frac{i\gamma p}{h}, \qquad p \in \left[-\frac{N}{2}, \frac{N}{2} - 1\right],$$

where γ is the cell Reynolds number defined by $\gamma = ch/\kappa$.

Using the linear FE basis and upwinding introduced in the previous section, one gets the discrete equations

$$\begin{aligned} &-\left(1 + \gamma\frac{1+\epsilon}{2}\right)u_{j-1} + (2 + \gamma\epsilon)u_j - \left(1 - \gamma\frac{1-\epsilon}{2}\right)u_{j+1} \\ &= \frac{h^2}{12}\left[(2 + 3\epsilon)f_{j-1} + 8f_j + (2 - 3\epsilon)f_{j+1}\right], \end{aligned} \tag{3.12}$$

where ϵ is the upwinding parameter of (3.4). With no upwinding (i.e., $\epsilon = 0$), stability requirements of FD and FE schemes restrict γ to values ≤ 2 [9]. The Fourier analysis of (3.12) is straightforward. The eigenvalues of (3.12) are complex and given by

$$(3.13) \qquad \sigma(p) = \frac{\frac{(ph)^2}{3}(2+\cos ph) + \epsilon\gamma\frac{ph}{2}\sin ph + i[\gamma\frac{ph}{3}(2+\cos ph) - \epsilon\frac{(ph)^2}{2}\sin ph]}{2(2+\epsilon\gamma)\sin^2\frac{ph}{2} + i\gamma\sin ph}, \quad |ph| \in [0,\pi].$$

In absence of upwinding, (3.13) reduces to an analytical expression whose real and imaginary parts may be written in compact form:

$$(3.14) \qquad \begin{aligned} \mathrm{Re}(\sigma(p)) &= ph\frac{4ph\sin^2\frac{ph}{2} + \gamma^2\sin ph}{16\sin^4\frac{ph}{2} + \gamma^2\sin ph}\frac{2+\cos ph}{3}, \\ \mathrm{Im}(\sigma(p)) &= \gamma ph\frac{4\sin^2\frac{ph}{2} - ph\sin ph}{16\sin^4\frac{ph}{2} + \gamma^2\sin ph}\frac{2+\cos ph}{3}. \end{aligned}$$

The factor $(2+\cos ph)/3$ in the right-hand side of (3.14) is another example of the contribution of the mass matrix in FE preconditioning. As in (2.16), this factor reduces to unity in the expression of the eigenvalues corresponding to FD preconditioning. One can draw similar conclusions to the diffusion problem, except for the complex nature of the eigenvalues. Introducing $ph = 0$ and $ph = \pi$ into the eigenvalues of the FD case gives the bounds of the spectrum:

$$(3.15) \qquad 1 \leq \mathrm{Re}(\sigma^{FD}) \leq \frac{\pi^2}{4}, \quad 0 \leq \mathrm{Im}(\sigma^{FD}) \leq \frac{\gamma\pi}{4}.$$

In the FE case, the upper bounds are reduced by a factor of 3 because of the presence of the mass matrix. Figure 4 displays the result (3.14) for both FD and FE preconditionings and for two different values of γ: 0.2 and 2. Provided the value of γ is less than the stability limit, the spectrum of A for FE preconditioning lies inside the unit circle. Reducing the value of the cell Reynolds number brings the eigenvalues closer to the real axis. Figure 5 exhibits the spectrum (3.13) for FE preconditioning and for γ exceeding the stability limit: 5 and 10. Without upwinding in this case, the spectrum lies outside the unit circle and blows up linearly with the value of γ. This is to be expected on the grounds that we approach a true hyperbolic problem (3.1). Introducing upwinding brings the spectrum inside the unit circle. For increasing values of γ one gets back the results displayed on Fig. 2, since in the limit where γ goes to infinity, (3.13) reduces to (3.7).

4. Stokes equations. Let us write the Stokes equations in stress formulation:

$$(4.1) \qquad div\underline{\underline{\sigma}} + \rho\underline{f} = 0,$$

$$(4.2) \qquad div\underline{v} = 0.$$

The symbol $\underline{\underline{\sigma}}$ denotes the stress tensor, ρ is the density, $\underline{f}$ the body force term and $\underline{v}$ is the velocity field. Equation (4.1) is the momentum equation and (4.2) enforces the continuity constraint. The two-dimensional Stokes problem may be reduced to a one-dimensional problem if the solution of (4.1), (4.2) is sought as a Fourier mode:

$$(4.3) \qquad \underline{v} = \underline{v}(x)e^{iky}, p = p(x)e^{iky}.$$

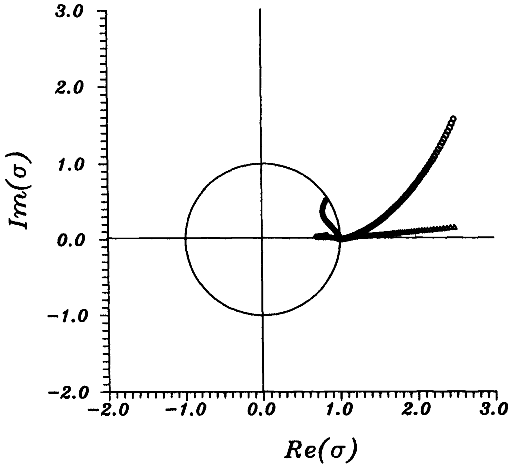


FIG. 4. *Eigenspectrum of an advection-diffusion problem. The curves outside the unit circle are concerned with* FD *preconditioning, while the inside curves are related to* FE *preconditioning. Both top curves are obtained for the cell Reynolds number* $\gamma = 2$. *The bottom curves are gotten for* $\gamma = 0.2$

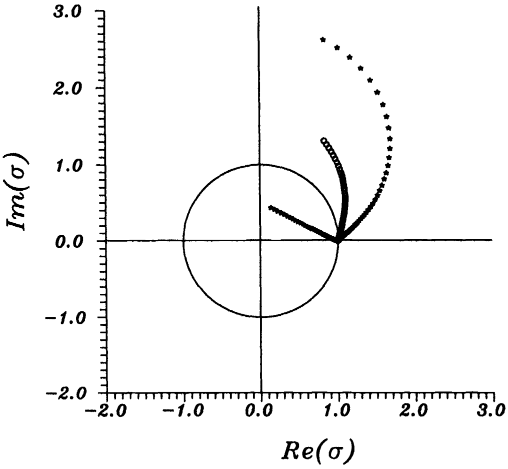


FIG. 5. *Eigenspectrum of an advection-diffusion problem preconditioned by* Q1 *elements for* $\gamma = 5(\bigcirc)$ *and* 10★. *Without upwinding* ($\epsilon = 0$), *the spectrum lies outside the unit circle. Upwinding* ($\epsilon = 1$) *brings the spectrum inside the unit circle.*

Introducing (4.3) in (4.1), (4.2), we get:

$$-\frac{\partial p}{\partial x} + 2\mu\frac{\partial^2 u}{\partial x^2} + ik\mu\left(iku + \frac{\partial v}{\partial x}\right) + \rho f_x = 0, \tag{4.4}$$

$$\mu\frac{\partial}{\partial x}\left(iku + \frac{\partial v}{\partial x}\right) - ikp - 2\mu k^2 v + \rho f_y = 0, \tag{4.5}$$

$$\frac{\partial u}{\partial x} + ikv = 0, \qquad 0 \le x \le 2\pi. \tag{4.6}$$

The velocity and pressure fields are assumed to be 2π-periodic. This one-dimensional problem is discretized in the x direction using Fourier series of type (2.2) for each variable. The discrete collocation equations are preconditioned by finite elements such as the Q2-Q1 and Q1-P0 elements. The FE equations come from Galerkin projection. Introducing ψ_l and φ_l, the trial functions for the FE approximations of the velocity and pressure fields, respectively, such that

$$\underline{v}(x) = \sum_{l=1}^{M} \underline{v}_l \psi_l, \qquad p(x) = \sum_{l=1}^{N} p_l \varphi_l, \tag{4.7}$$

the discrete FE equations are obtained by use of the divergence theorem as a tool for the integration by parts with the notation $f_x = f$, $f_y = g$:

$$\sum_l [2\mu A_{jl} + \mu k^2 B_{jl}] u_l - ik\mu \sum_l C_{jl} v_l - \sum_l D_{jl} p_l = \sum_l B_{jl} f_l, \qquad 0 \le j \le M, \tag{4.8}$$

$$ik\mu \sum_l C_{jl}^T u_l + \sum_l [2\mu k^2 B_{jl} + \mu A_{jl}] v_l + ik \sum_l E_{jl} p_l = \sum_l B_{jl} g_l, \qquad 0 \le j \le M, \tag{4.9}$$

$$-\sum_l D_{jl}^T u_l - ik \sum_l E_{jl}^T v_l = 0, \qquad 0 \le j \le N. \tag{4.10}$$

In (4.8)–(4.10), the various matrices are defined by the relationships:

$$\begin{gathered} A_{jl} = \int \frac{\partial \psi_j}{\partial x} \frac{\partial \psi_l}{\partial x} dx, \qquad B_{jl} = \int \psi_j \psi_l dx, \qquad C_{jl} = \int \psi_j \frac{\partial \psi_l}{\partial x} dx, \\ D_{jl} = \int \frac{\partial \psi_j}{\partial x} \varphi_l dx, \quad E_{jl} = \int \varphi_j \psi_l dx. \end{gathered} \tag{4.11}$$

4.1. Q2-Q1 elements. For this element, $M = 2N$. Carrying through the algebra involved by the quadratures (4.11) and assembling by direct stiffness the contributions of the two elements connected to node j, we obtain

$$\begin{aligned} &\frac{2\mu}{3h}(u_{j-1} - 8u_{j-1/2} + 14u_j - 8u_{j+1/2} + u_{j+1}) \\ &\quad + \frac{\mu k^2 h}{30}(-u_{j-1} + 2u_{j-1/2} + 8u_j + 2u_{j+1/2} - u_{j+1}) \\ &\quad - \frac{ik\mu}{6}(v_{j-1} - 4v_{j-1/2} + 4v_{j+1/2} - v_{j+1}) \\ &\quad - \frac{1}{6}(p_{j-1} - p_{j+1}) = \frac{h}{30}(-f_{j-1} + 2f_{j-1/2} + 8f_j + 2f_{j+1/2} - f_{j+1}), \end{aligned} \tag{4.12}$$

$$\begin{aligned} &\frac{4\mu}{3h}(-4u_j + 8u_{j+1/2} - 4u_{j+1}) + \frac{\mu k^2 h}{30}(2u_j + 16u_{j+1/2} + 2u_{j+1}) \\ &\quad - \frac{2ik\mu}{3}(-v_j + v_{j+1}) - \frac{2}{3}(p_j - p_{j+1}) = \frac{h}{30}(2f_j + 16f_{j+1/2} + 2f_{j+1}), \end{aligned} \tag{4.13}$$

$$
\begin{aligned}
&\frac{ik\mu}{6}(-u_{j-1}+4u_{j-1/2}-4u_{j+1/2}+u_{j+1})\\
&\quad-\frac{\mu k^2h}{15}(-v_{j-1}+2v_{j-1/2}+8v_j+2v_{j+1/2}-v_{j+1})\\
&\quad+\frac{2\mu}{6}(v_{j-1}-8v_{j-1/2}+14v_j-8v_{j+1/2}+v_{j+1})-\frac{ikh}{3}p_j\\
&=\frac{h}{30}(-g_{j-1}+2g_{j-1/2}+8g_j+2g_{j+1/2}-g_{j+1}),
\end{aligned}
\tag{4.14}
$$

$$
\begin{aligned}
&\frac{2ik\mu}{3}(u_j-u_{j+1})+\frac{8\mu}{3h}(-v_j+2v_{j+1/2}-v_{j+1})-\frac{2\mu k^2h}{15}(v_j+8v_{j+1/2}+v_{j+1})\\
&\quad-\frac{ikh}{3}(p_j+p_{j+1})=\frac{h}{15}(f_j+8f_{j+1/2}+f_{j+1}),
\end{aligned}
\tag{4.15}
$$

$$
\frac{1}{6}(-u_{j-1}-4u_{j-1/2}+4u_{j+1/2}+u_{j+1})+\frac{2ik}{3}(v_{j-1/2}+v_j+v_{j+1/2})=0.
\tag{4.16}
$$

Equations (4.12), (4.14), and (4.16) correspond to momentum and incompressibility relations, while (4.13) and (4.15) are the momentum equations associated to mid-node $x_{j+1/2}$. Similar expressions hold for mid-node $x_{j-1/2}$ with appropriate shifts for the indices.

Now, static condensation represents a formidable task and is greatly helped by the symbolic manipulation program. Elimination of $u_{j\pm1/2}, v_{j\pm1/2}$ leads to a matrix system of order three.

The full Fourier solution gives the collocation matrix L_c:

$$
L_c=\begin{pmatrix}-2\mu l^2-k^2\mu & -\mu kl & -il\\ -\mu kl & -\mu l^2-2\mu k^2 & -ik\\ -il & -ik & 0\end{pmatrix},
\tag{4.17}
$$

where l is the wavenumber in the x direction.

The analytical computation of $\hat{S}_h^{-1}\hat{M}_hL_c$ is performed as far as the symbolic program can handle tractable expressions. Then numerical evaluation of the eigenspectrum is done. Because of the divergence-free constraint, a zero eigenvalue is systematically obtained. In Figs. 6 and 7, the eigenspectrum of $\hat{S}_h^{-1}\hat{M}_hL_c$ is plotted for two cases $k=1$ and $k=10$, respectively. In these two figures, the lower curve shows the same behaviour as the eigenspectrum of the elliptic problem solved by Q2 elements. For $l=-N/2$, $\sigma(l)$ is equal to $\pi^2/12$. For the other curve, $\sigma(0)=1$ and $\sigma(-N/2)$ is close to 2.08. Therefore, the optimal α value is given by

$$
\alpha_{\text{opt}}=2/(2.08+\pi^2/12)\simeq 0.69,
$$

a value close to $\frac{2}{3}$ obtained by Demaret and Deville [3] for a two-dimensional Chebyshev collocation discretization of the Stokes problem preconditioned by Q2-Q1 elements.

4.2. Q1-P0 element. The quadratures (4.11) provide less complicated discrete equations in this case:

$$
\begin{aligned}
&\frac{2\mu}{h}(-u_{j-1}+2u_j-u_{j+1})-\frac{\mu hk^2}{6}(u_{j-1}+4u_j+u_{j+1})+\frac{ik\mu}{2}(v_{j+1}-v_{j-1})\\
&\quad-p_j+p_{j+1}=\frac{h}{6}(f_{j-1}+4f_j+f_{j+1}),
\end{aligned}
\tag{4.18}
$$

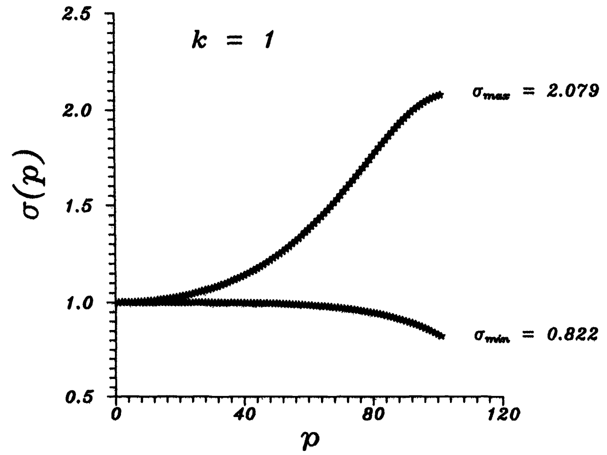


FIG. 6. *Eigenspectrum of the Stokes problem preconditioned by* Q2-Q1 *element. Here* $k = 1$.

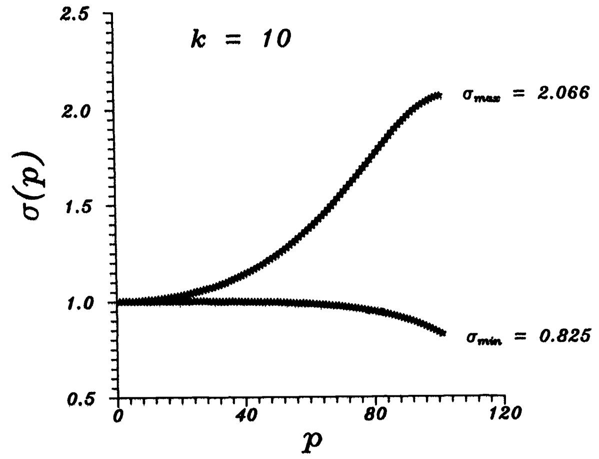


FIG. 7. *Eigenspectrum of the Stokes problem preconditioned by* Q2-Q1 *element. Here* $k = 10$.

$$\begin{aligned}(4.19)\qquad &\frac{ik\mu}{2}(u_{j-1}-u_{j+1})-\frac{\mu k^2 h}{3}(v_{j-1}+4v_j+v_{j+1})+\frac{\mu}{h}(-v_{j-1}+2v_j-v_{j+1})\\ &\quad-\frac{ikh}{2}(p_j+p_{j+1})=\frac{h}{6}(g_{j-1}+4g_j+g_{j+1}),\end{aligned}$$

$$(4.20)\qquad -u_{j-1}+u_{j+1}+\frac{ikh}{2}(v_{j-1}+2v_j+v_{j+1})=0.$$

Obviously, this element generates second-order differences for partial derivatives. When the mass matrix is involved, the standard weighted mean between three adjacent nodes appears in the expressions. No static condensation is needed in this case. Fourier analyzing (4.18)–(4.20), the stiffness and mass matrices are now

$$S_h=\begin{pmatrix} 8\frac{\mu}{h}\sin^2(\frac{hl}{2})+\frac{hk^2\mu}{3}(2+\cos hl) & \mu k\sin hl & 2i\sin(\frac{hl}{2})\\ \mu k\sin hl & 4\frac{\mu}{h}\sin^2(\frac{hl}{2})+\frac{2\mu k^2 h}{3}(2+\cos hl) & ikh\cos(\frac{hl}{2})\\ 2i\sin hl & ikh(1+\cos hl) & 0\end{pmatrix},$$

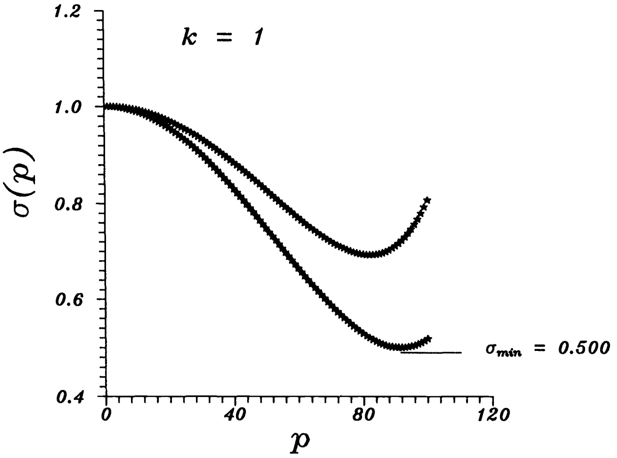


FIG. 8. *Eigenspectrum of the Stokes problem preconditioned by* Q1-P0 *element. Here* $k = 1$.

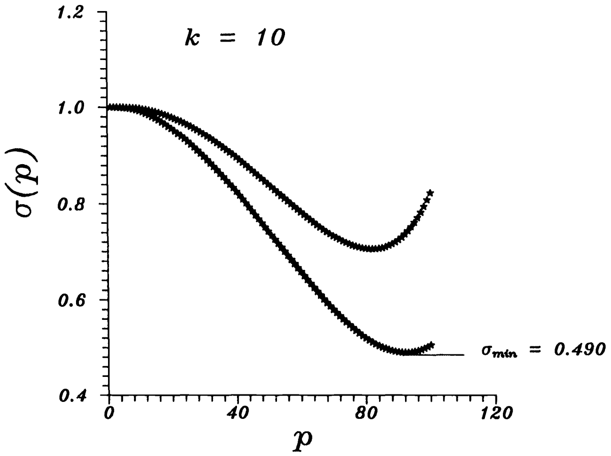


FIG. 9. *Eigenspectrum of the Stokes problem preconditioned by* Q1-P0 *element. Here* $k = 10$.

$$M_h = \operatorname{diag}\left(\frac{h}{3}(2+\cos hl), \frac{h}{3}(2+\cos hl), 0\right). \tag{4.21}$$

In Figs. 8 and 9, the eigenspectrums of $S_h^{-1}M_hL_c$ are displayed for $k = 1$ and 10, respectively. In these two figures, the top curve is that of the elliptic model preconditioned by the Q1 element. The bottom curve starts from 1 for $l = 0$, decreases till a minimum value close to 0.49 and then increases to reach $\sigma(-N/2) = 0.5$. The optimum value is

$$\alpha_{\text{opt}} = 2/(1+0.5) = \tfrac{4}{3}.$$

5. Conclusions. In this paper, we have Fourier analyzed the eigenspectrum of the iteration operator for finite element preconditioning of Fourier collocation applied to one-dimensional problems. For elliptic models, this theoretical analysis confirms previous numerical findings, especially the beneficial presence of the mass matrix which reduces the bounds of the eigenspectrum. For first-order problems, linear elements without and with upwinding are considered. With quadratic elements, a staggered scheme is produced. Its eigenspectrum is bounded and ranges between 1 and $\pi/4$. Finally, a Stokes problem is reduced to a one-dimensional approach. Two types of elements are examined. The Q2-Q1 element leads to an optimum value of the relaxation parameter close to the value obtained by numerical analysis of preconditioned Chebyshev collocation. For Q1-P0 element, the method can be overrelaxed.

REFERENCES

[1] C. Canuto, M. Y. Hussaini, A. Quarteroni, T. A. Zang, *Spectral Methods in Fluid Dynamics*, Springer-Verlag, New York, 1988.

[2] P. G. Ciarlet, *The Finite Element Method*, North-Holland, Amsterdam, 1979.

[3] P. Demaret and M. O. Deville, *Chebyshev pseudospectral solution of the Stokes equations using finite element preconditioning*, J. Comp. Phys., 83 (1989), pp. 463–484.

[4] ———, *Chebyshev collocation solutions of the Navier–Stokes equations using multidomain decomposition and finite element preconditioning*, J. Comp. Phys., 95 (1991), pp. 359–386.

[5] M. Deville and E. Mund, *Chebyshev pseudospectral solution of second-order elliptic equations using finite element preconditioning*, J. Comp. Phys., 60 (1985), pp. 517–533.

[6] ———, *Finite element preconditioning for pseudospectral solutions of elliptic problems*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 311–342.

[7] P. Haldenwang, G. Labrosse, S. Abboudi, and M. O. Deville, *Chebyshev 3-D spectral and 2-D pseudospectral solvers for the Helmholtz equation*, J. Comp. Phys., 55 (1984), pp. 115–128.

[8] J. C. Heinrich and O. C. Zienkiewicz, *The finite element method and upwinding techniques in the numerical solution of convection dominated flow problems*, in Finite Element Methods for Convection Dominated Flows, T. J. R. Hughes, ed., ASME, New York, 1979, pp. 105–136.

[9] F. Thomasset, *Implementation of Finite Element Methods for Navier–Stokes Equations*, Springer-Verlag, New York, 1981.

[10] S. Wolfram, *SMP, A Symbolic Manipulation Program: Reference Manual*, Inference Corporation, Los Angeles, CA, 1983.

SIAM J. Sci. Stat. Comput.
Vol. 13, No. 2, pp. 611–630, March 1992

011

SONIC FLUX FORMULAE*

P. L. ROE†

Abstract. The numerical solution of hyperbolic partial differential equations is studied, with special reference given to behaviour near a sonic point. A thorough treatment of the scalar case generalises to yield a satisfactory modification of upwind schemes for systems of equations.

Key words. initial-value problems, sonic points, upwind differencing

AMS(MOS) subject classifications. 65M05, 76N15

1. Introduction. This paper deals with the calculation of numerical solutions to hyperbolic differential equations. Initially, it concentrates on the scalar conservation law

$$u_t + f_x = 0, \tag{1.1}$$

where $f(u)$ is some nonlinear function. The particular aspect of interest is the behaviour of the solution near sonic points ($u \simeq \bar{u}, f'(\bar{u}) = 0$). It is well known that details of the numerical flux calculation under these conditions are very critical. If a wrong choice is made, the algorithm may not converge (under mesh refinement, or as time increases) to the correct (entropy-satisfying) weak solution. "Sonic flux formulae" that do guarantee such convergence are developed in [11], [5]. Even when convergence does take place, it may be very slow, and the choice of flux formula has a pronounced effect [8], [9].

Also, even though the local errors may be small, they tend to accumulate into some kind of local pathology (variously named "glitches" or "doglegs") as noted in [14], for example. It will be explained in §2 that such glitches are to be expected from *any* first-order upwind method in which only the sonic flux (the flux at the interface across which the characteristic speed changes sign) is modified. This is because the various design criteria that might be used to obtain a sonic flux formula cannot all be satisfied simultaneously. In §3, a more accurate treatment is presented that involves modifying the sonic flux and both its neighbours. Section 4 reports some numerical experiments. Section 5 demonstrates that the analysis is actually simpler for second-order schemes, and §6 contains the generalisation to systems of equations. The one-dimensional Euler equations are investigated in §7, and §8 gives the numerical results. Section 9 comments briefly on the behaviour of schemes designed using the MUSCL approach.

The emphasis throughout is on the accuracy of the approximation. Nevertheless, convergence to the correct entropy solution follows, at least for convex problems, because we are able to show that sonic gradients always decay [3]. The fact that they decay at the correct rate may be expected to accelerate convergence also. We do not, at this stage, attempt to relate this work to the more empirical "entropy fixes" that are found necessary in many practical calculations, both to smooth the solution and to enable the application of Newton-like iterations in seeking a steady state. For these purposes, the entropy modification due to Harten [4] has proved popular.

* Received by the editors September 5, 1989; accepted for publication (in revised form) January 21, 1991.

† College of Aeronautics, Cranfield Institute of Technology, Cranfield, Bedford MK43 0AL, United Kingdom. Present address, Department of Aerospace Engineering, University of Michigan, Ann Arbor, Michigan 48109-2140.

2. The origin of glitches. Consider (1.1) rewritten as

$$u_t + a(u)u_x = 0, \tag{2.1}$$

where $a(u) = f'(u)$. To solve (1.1) or (2.1) to first order on a mesh $(j\Delta x, n\Delta t)$, the generic conservative method is

$$u_j^{n+1} = u_j^n - \lambda \left[F_{j+\frac{1}{2}}^{n+\frac{1}{2}} - F_{j-\frac{1}{2}}^{n+\frac{1}{2}} \right], \tag{2.2}$$

where $\lambda = \Delta t/\Delta x$. For an upwind scheme away from shock or sonic points, the interface flux is defined by

$$F_{j+\frac{1}{2}}^{n+\frac{1}{2}} = \begin{cases} f(u_j^n) & \text{if } a_j^n, a_{j+1}^n > 0, \\ f(u_{j+1}^n) & \text{if } a_j^n, a_{j+1}^n < 0. \end{cases} \tag{2.3}$$

If

$$a(u_j^n) \geq 0 \geq a(u_{j+1}^n)$$

we have a *shock point*, and if

$$a(u_j^n) \leq 0 \leq a(u_{j+1}^n)$$

we have a *sonic point*. In these two cases, various upwind schemes offer various prescriptions for calculating the flux $F_{j+(1/2)}^{n+(1/2)}$. We are concerned here with the sonic case, illustrated diagrammatically in Fig. 1. Henceforward, subscripts $(\cdot)_L$ and $(\cdot)_R$ will denote quantities in two consecutive cells for which $a_L \leq 0 \leq a_R$. The fluxes across the three interfaces involved in updating u_L, u_R will be labelled F_L, F_S, F_R. Conventional upwind practice will always take

$$\begin{aligned} F_L &= f(u_L), \\ F_R &= f(u_R) \end{aligned} \tag{2.4}$$

and the question will simply be the choice of F_S. To make this choice, we can attempt to enforce either of two properties characteristic of sonic points.

For any sonic point $\bar{x}$ such that $u(\bar{x}) = \bar{u}$, we have from (2.1) that $u_t(\bar{x}) = 0$. Thus $x = \bar{x}$ remains a sonic point for all time (or at least until the arrival of a shockwave). To implement this first sonic point property as a design criterion, we may note that both a and u_t are, to first order, proportional to the distance from the (fixed) sonic point. Hence

$$\frac{u_R^{n+1} - u_R^n}{u_L^{n+1} - u_L^n} = \frac{a_R}{a_L} \tag{2.5}$$

or

$$\frac{F_R - F_S}{F_S - F_L} = \frac{a_R}{a_L},$$

so that

$$F_S = \frac{1}{2}\left[f(u_L) + f(u_R)\right] - \frac{a_R - a_L}{a_R + a_L}\left[f(u_R) - f(u_L)\right].$$

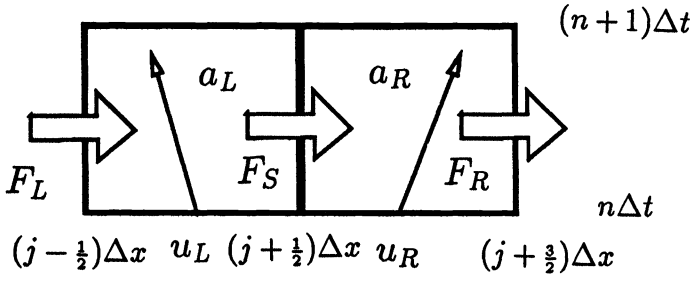

$(n+1)\Delta t$
a_L
a_R
F_L
F_S
F_R
$n\Delta t$
$(j-\frac{1}{2})\Delta x$
u_L
$(j+\frac{1}{2})\Delta x$
u_R
$(j+\frac{3}{2})\Delta x$

FIG. 1. *Sketch used to define notation near a sonic point.*

If we approximate

$$\frac{1}{2}(a_R + a_L) = \frac{f(u_R) - f(u_L)}{u_R - u_L}, \tag{2.6}$$

we arrive at

$$F_S = \frac{1}{2}[f(u_L) + f(u_R)] - \frac{1}{4}(a_R - a_L)(u_R - u_L). \tag{2.7}$$

Comparing this with the standard "viscosity form" for a flux formula

$$F^{n+\frac{1}{2}}_{j+\frac{1}{2}} = \frac{1}{2}\left[f(u^n_j) + f(u^n_{j+1})\right] - \frac{q}{2}\left[u^n_{j+1} - u^n_j\right] \tag{2.8}$$

we see that to enforce the first sonic point property, we should choose

$$q_S = \frac{1}{2}\Delta a.$$

The second property comes from differentiating (2.1) with respect to x:

$$u_{xt} + a_x u_x + a u_{xx} = 0. \tag{2.9}$$

Let $s = u_x(\bar{x})$. Then

$$s_t + a_u s^2 = 0, \tag{2.10}$$

where $a_u = f_{uu}(u)$ is evaluated at $u = \bar{u}$. Thus we know the rate at which the solution gradient decays near sonic points. Indeed, we can integrate (2.10) to obtain

$$\frac{1}{s(t)} - \frac{1}{s(0)} = a_u t. \tag{2.11}$$

Discretising in time, we have the *exact* result

$$s^{n+1} - s^n = -\Delta t\, a_u s^n s^{n+1}. \tag{2.12}$$

To implement a design based on the second sonic property, we note that

$$(u_R - u_L)^{n+1} - (u_R - u_L)^n = -\lambda\left[F_R + F_L - 2F_S\right]. \tag{2.13}$$

The left-hand side (LHS) can be approximated (see (2.12)) as

$$-\lambda(a_R - a_L)^n(u_R - u_L)^n$$

which leads to

$$F_S = \frac{1}{2}\left[f(u_L) + f(u_R)\right] - \frac{1}{2}(a_R - a_L)(u_R - u_L). \tag{2.14}$$

Therefore we now have

$$q_S = \Delta a,$$

exactly twice the previous value.

The formulae (2.7), (2.14) are due to this author [13] and Goodman and Leveque [3], respectively. Specialised to Burgers' equation $f(u) = \frac{1}{2}u^2$ they yield

$$F_S = \frac{1}{2}u_L u_R$$

and

$$F_S = u_L u_R - \frac{1}{4}(u_L^2 + u_R^2).$$

To ask which of these two formulae is correct is to miss the point; both are wrong! The reason can be traced back to the use of the basic upwind formulae (2.4) for F_L and F_R. This results in the solution inside the pair of cells concerned being updated according to

$$\frac{(u_L + u_R)^{n+1} - (u_L + u_R)^n}{\Delta t} = \frac{f(u_L) - f(u_R)}{\Delta x}.$$

The LHS is an approximation to *twice* u_t; the right-hand side (RHS) is an approximation to $-f_x$. Displacing the fluxes from the cell centres to the cell interfaces creates only a first-order error if done consistently. Here the displacements are in opposite directions, which means that locally we are solving

$$u_t + \frac{1}{2}f(u)_x = 0. \tag{2.15}$$

Alternatively, we can say that although the flux still has only a first-order error in its absolute value, there is a zero-order relative error. It is because of this inconsistency that we cannot enforce both of the properties that ought to hold, so that a glitch of some kind is inevitable. This simple trap built into first-order upwinding seems so far to have gone unobserved.

Before giving the correct treatment, this may be a convenient place to insert the observation that according to (2.13) the sonic gradient $u_R - u_L$ will always diminish in magnitude, provided

$$\frac{f(u_R) + f(u_L) - 2F_S}{u_R - u_L} > 0.$$

Now for a convex (concave) flux function, u_R is greater than (less than) u_L, so that F_S must be less than (greater than) the average of $f(u_L), f(u_R)$. For such flux functions, this is enough to guarantee convergence to the entropy-satisfying weak solution [3]. This may be compared with Osher's [11] definition of an E-scheme as one for which F_S, whenever u_R is greater than (less than) u_L, is less than (greater than) the minimum (maximum) value of f in the interval $u \in [u_L, u_R]$. This condition ensures, for any flux function, that any value of u that should appear in the solution will eventually do so. Taking F_S actually to be the minimum (maximum) value is, as Osher observes, equivalent to Godunov's technique of obtaining interface fluxes from the exact solution to the Riemann problem $[u_L, u_R]$. It is possible that there exist entropy-satisfying schemes that are not E-schemes, since Osher proved sufficiency, but not necessity, of his condition. The present technique does not, however, yield such a scheme for nonconvex flux functions. Basically, this is because it is designed around sonic points; it successfully breaks up stationary rarefaction shocks, but can be fooled by ones that move.

3. A consistent sonic point treatment. A scheme will be devised that both satisfies a consistent conservation equation, and also yields the correct decay rate. Both conditions are met to second order. The conservation equation will be

$$\frac{\delta u_L + \delta u_R}{2} = -\lambda \left[f(u_R) + \frac{1}{2} a_R \delta u_R - f(u_L) - \frac{1}{2} a_L \delta u_L \right] \tag{3.1}$$

which is obtained by integrating around the control volume ABCD in Fig. 2, and approximating the flux values by Taylor expansions. The notation $\delta u_L = u_L^{n+1} - u_L^n$ has been used and variables without superscripts (like u_R) are at time level n. The decay equation will be (see (2.12))

$$\delta u_R - \delta u_L = -\lambda(a_R - a_L)\left[u_R + \delta u_R - u_L - \delta u_L\right]. \tag{3.2}$$

This can be written as

$$(\delta u_R - \delta u_L)(1 + \Delta\nu) = -(u_R - u_L)\Delta\nu, \tag{3.3}$$

where the difference of Courant numbers is

$$\Delta\nu = \lambda(a_R - a_L). \tag{3.4}$$

Now (3.1) can be rearranged, using Δf for $f(u_R^n) - f(u_L^n)$, as

$$\delta u_L + \delta u_R = -2\lambda\Delta f - \frac{1}{2}\lambda(\delta u_L + \delta u_R)(a_R - a_L) - \frac{1}{2}\lambda(\delta u_R - \delta u_L)(a_R + a_L). \tag{3.5}$$

Combining (3.3) and (3.5) leads to

$$(\delta u_L + \delta u_R)\left(1 + \frac{1}{2}\Delta\nu\right) = -\lambda\left[2\Delta f - \frac{(a_L + a_R)(u_R - u_L)\Delta\nu}{2(1 + \Delta\nu)}\right].$$

Using the approximation (2.6), the RHS of this expression simplifies, and a factor $(1 + \frac{1}{2}\Delta\nu)$ cancels, leaving

$$\delta u_L + \delta u_R = \frac{-\lambda}{1 + \Delta\nu}(a_L + a_R)(u_R - u_L). \tag{3.6}$$

Combining this with (3.3) leads to the simple results

$$\delta u_L = -\frac{\lambda a_L (u_R - u_L)}{1 + \Delta\nu}, \tag{3.7}$$

$$\delta u_R = -\frac{\lambda a_R(u_R - u_L)}{1+\Delta\nu}. \tag{3.8}$$

Note that by specifying *consistent* conservation and decay equations we have achieved as a free bonus the condition (2.5) requiring sonic points not to move. However, when stationarity was the only criterion the changes obtained differed from those above by a factor $(1+\Delta\nu)/2$.

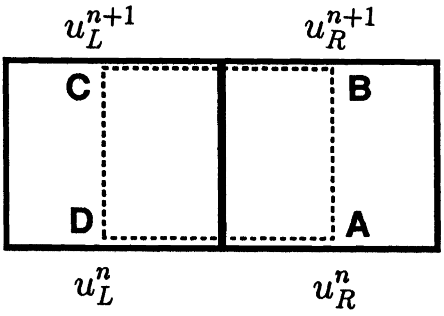


FIG. 2. *The control volume used to define equations* (3.1), (3.2).

Next, the interface fluxes that bring about these changes will be found. We must have

$$-\lambda^{-1}\,\delta u_L = \frac{a_L(u_R - u_L)}{1+\Delta\nu} = F_S - F_L, \tag{3.9}$$

$$-\lambda^{-1}\,\delta u_R = \frac{a_R(u_R - u_L)}{1+\Delta\nu} = F_R - F_S. \tag{3.10}$$

Clearly there is no unique solution to these equations, because the same constant could be added to each flux, but a particular solution having more symmetry than the alternatives is

$$F_L = \frac{f(u_L) - \frac{1}{8}(5a_L - a_R)(u_R - u_L)}{1+\Delta\nu}, \tag{3.11}$$

$$F_S = \frac{\frac{1}{2}[f(u_L) + f(u_R)] - \frac{1}{8}(a_R - a_L)(u_R - u_L)}{1+\Delta\nu}, \tag{3.12}$$

$$F_L = \frac{f(u_R) - \frac{1}{8}(5a_R - a_L)(u_R - u_L)}{1+\Delta\nu}. \tag{3.13}$$

In this case, the semidiscrete fluxes ($\Delta\nu = 0$) can be obtained by the sort of geometrical argument typically used to construct second-order schemes. Assume that a and u both vary linearly over the interval $[j-\frac{1}{2}, j+\frac{3}{2}]$, where the sonic interface is at $j+\frac{1}{2}$. Then the average value of a in the interval $[j+1, j+\frac{3}{2}]$ is $(5a_R - a_L)/4$. Extrapolating f from $j+1$ to $j+\frac{3}{2}$ using $df = a\,du$ now gives (3.11) and similar arguments give (3.13). The same technique applied to the intervals $[j, j+\frac{1}{2}]$ and $[j+\frac{1}{2}, j+1]$ gives two alternative formulae for F_S,

$$F_S = f(u_L) + \frac{3a_L + a_R}{8}(u_R - u_L),$$

$$F_S = f(u_R) - \frac{3a_R + a_L}{8}(u_R - u_L).$$

Approximate equality of these expressions follows from (2.6), and taking the average gives (3.12). Note that this formula, even without the factor $(1+\Delta\nu)^{-1}$, is different from both (2.7) and (2.14).

4. Numerical experiments. The test problem chosen in this section is the inviscid Burgers' equation

$$u_t + \left(\frac{1}{2}u^2\right)_x = 0,$$

with initial data (see Fig. 3)

$$\begin{aligned} u &= -1, && 0 \le x \le 2.9, \\ u &= 1 - \frac{(x-4.1)^2}{0.72}, && 2.9 \le x \le 4.1, \\ u &= 1, && 4.1 \le x \le 7. \end{aligned}$$

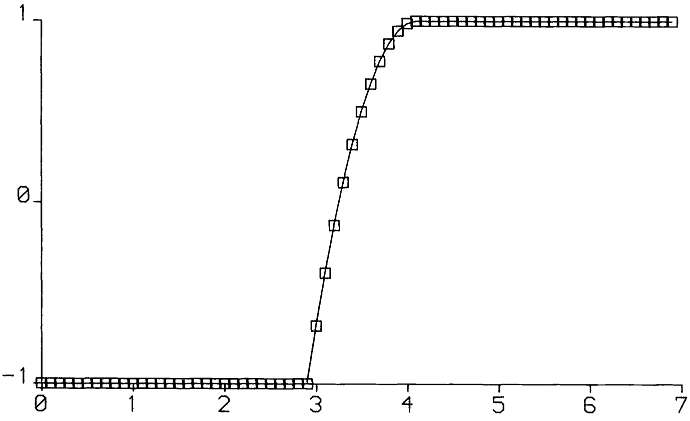


FIG. 3. *Data at $t=0$ for the test problem using Burgers' equation.*

It was thought that a nonuniform gradient might make a more demanding task than the usual ramp test. The numbers were juggled to produce a sonic point roughly halfway between mesh points; this appears to be the hardest case. The problem above has an analytical solution, the nontrivial part of which is

$$u = \frac{4xt - 1 + (8t^2 - 8xt + 1)^{\frac{1}{2}}}{4t^2}, \qquad -(1+t) \le x \le t,$$

where x has been scaled to place the initial data on $[-1, 1]$.

Solutions were obtained at t=2.25, by taking 45 timesteps with $\Delta t = 0.05$. Figures 4(a)–(d) show results from various first-order methods. Figure 4(a) uses the simple version of the author's flux formula,

$$F_S = \frac{1}{2}[f(u_L) + f(u_R)] - \frac{1}{2}\left|\frac{f(u_R) - f(u_L)}{u_R - u_L}\right|(u_R - u_L) \tag{4.1}$$

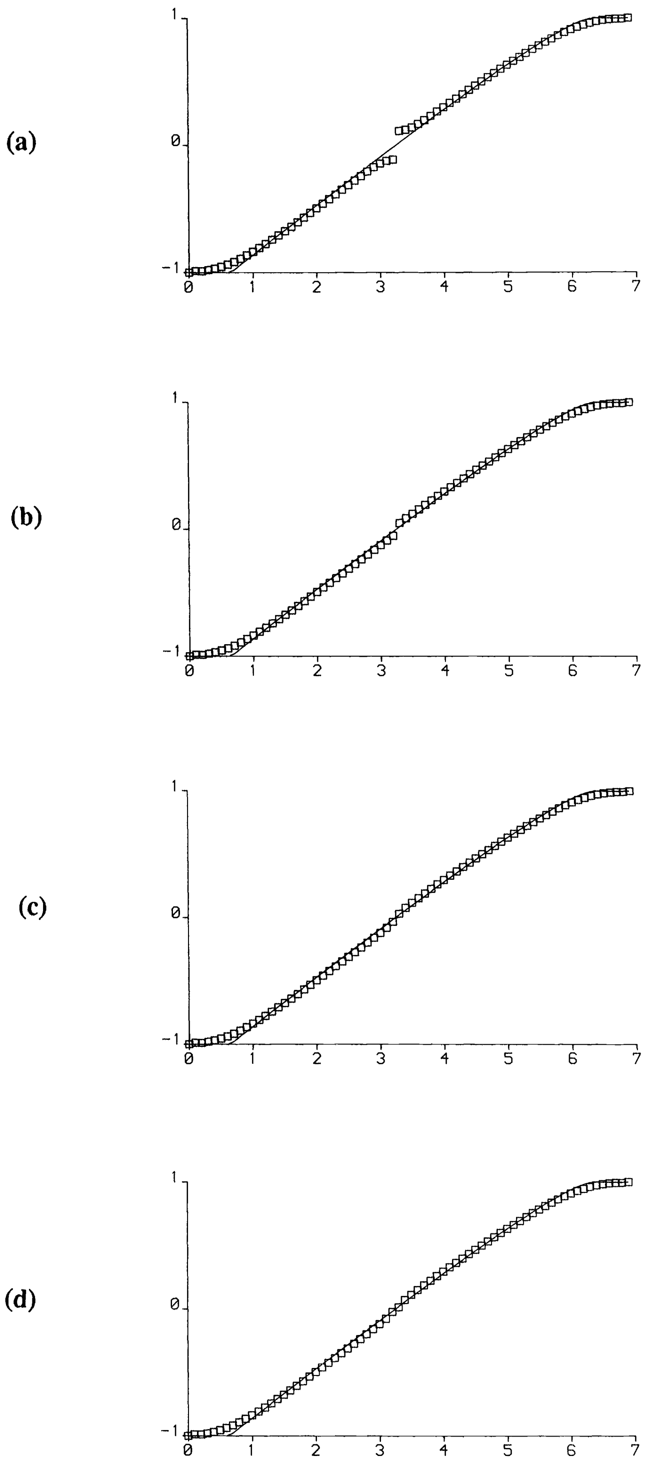


FIG. 4. *Results at* $t = 2.25$ *for the Burgers' equation test problem from four first-order schemes.* (a) *The flux formula* (4.1), *used everywhere, gives these results.* (b) *At the sonic interface,* (4.1) *is replaced by the sonic flux* $f = 0$. (c) *At the sonic interface,* (4.1) *is replaced by* (2.7). (d) *At the sonic interface,* (4.1) *is replaced by* (2.14).

and clearly shows an entropy-violating shockwave. This will not decay if the solution is continued to later time. Figure 4(b) shows the results from Godunov's method (for Burgers' equation this means taking $F_S = 0$). The central gradient does decay, but far too slowly. Figure 4(c) shows the result of keeping the sonic point stationary, (2.7). The solution no longer appears discontinuous, but still suffers a marked inflection. In fact it is easy to show that, at the sonic point, Godunov's flux causes the slope to decay too slowly by a factor of four, and (2.7) by a factor of two. The flux (2.14), due to Goodman and Leveque [3], produces the results in Fig. 4(d). The two symbols straddling the sonic point are now almost exact because the true decay rate is guaranteed, but on either side there are noticeable errors. These are due to using the locally inconsistent upwind values of F_L and F_R, which provide poor boundary conditions for the solution away from the sonic point. However, this is the best of the first-order methods.

Some second-order results are shown in Figs. 5 (a)–(c). The solution has been noticeably improved, compared to the first-order solutions, in its outer parts (it seems that this problem is sufficiently smooth that no overshoots are created in the solution), but there is still a rather irregular passage through the sonic point. The code was then amended so that the fluxes $F_S, F_{S\pm 1}$ were overwritten with the formulae (3.11)–(3.13). This produced the results shown in Fig. 5(b). The sonic region is now very smooth and accurate. Finally, in Fig. 5(c), we see the results of modifying the Lax–Wendroff scheme with the Superbee flux limiter [13]. The antidiffusive effects of the limiter produce a further marked improvement in the outer regions, and close examination of the computer printout reveals that there is even a consequent slight improvement near the sonic point.

5. Simplified second-order schemes. Although the flux formulae (3.11)–(3.13) give excellent results, they are rather complex, especially for systems of equations, so a simplified strategy was sought. Recall that for first-order schemes a major source of error was the fluxes on either side of the sonic flux; when these were made more accurate the discrepancies of §2 disappeared.

If we assume that in a second-order scheme these neighbouring fluxes are sufficiently accurate anyway, then only the sonic flux has to be chosen. It now does make sense to ask which of the two sonic properties should be enforced, because choosing either of them can be shown to imply the other with second-order accuracy. However, there are numerical reasons to prefer one over the other, and other reasons too, as will emerge.

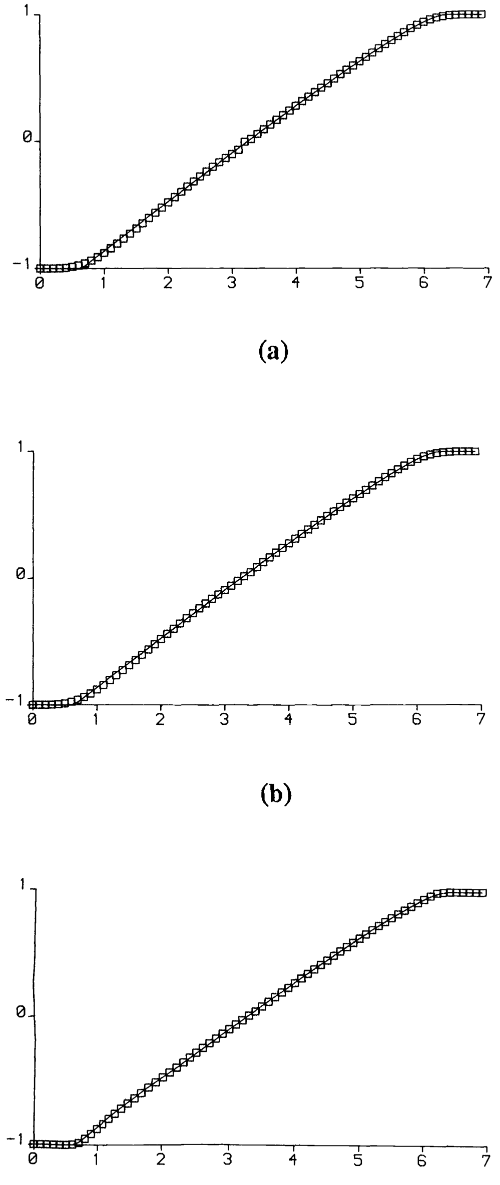
To keep the sonic point stationary, we should impose

$$\frac{F_R - F_S}{F_S - F_L} = \frac{a_R}{a_L},$$

where F_L, F_R are assumed given by some regular second-order scheme. Then

$$F_S = \frac{F_L a_R + F_R a_L}{a_R + a_L}. \tag{5.1}$$

This is an ill-formed expression, since near a sonic point both the numerator and denominator will be small.



(a)

(b)

(c)

FIG. 5. *Results at $t = 2.25$ for the Burgers' equation test problem from three second-order schemes.* (a) *Results from the unmodified Lax–Wendroff scheme.* (b) *For F_L, F_S, F_R the Lax–Wendroff flux is overwritten with* (3.11)–(3.13). (c) *As* (b), *but the Superbee limiter is added to the Lax–Wendroff scheme.*

To ensure the correct decay rate, we should return to (3.12) to obtain

$$F_S = \frac{1}{2}\left[F_L + F_R - \frac{\Delta a \Delta u}{1 + \Delta \nu}\right] \tag{5.2}$$

which will not cause any numerical problems.

As an experimental confirmation, the flux limited solution was rerun, using (5.2) to overwrite the sonic flux F_S only. To graphical accuracy the results were identical with those in Fig. 5(c). In fact, the numbers were even a little better. However, using (5.1) instead produced the results in Fig. 6 with a small but noticeable reduction of slope near the sonic point. In other runs, with different timesteps, the slope obtained from (5.1) was too large. This might be anticipated, since no control is being exerted over the slope, and it is open to accidental causes of any kind.

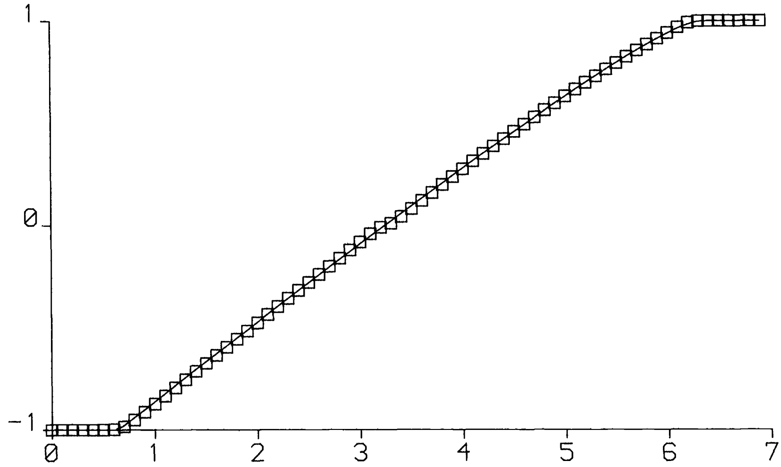


FIG. 6. *Results at $t = 2.25$ for Burgers' equation test problem from a simplified second-order scheme.*

Experiments were also made on the celebrated test case due to Harten, Hyman, and Lax [6]. Here the flux function is

$$f(u) = u - 3\sqrt{3}u^2(u-1)^2,$$

which is illustrated in Fig. 7. The solution with Riemann data $u = 1, x < 0; u = 0, x > 0$ has the entropy-satisfying solution denoted by the tangent OI, which represents a single jump moving to the right with unit speed. Harten, Hyman, and Lax [6] demonstrated that the Lax–Wendroff scheme admits entropy-violating solutions. The oscillations created by the Lax–Wendroff scheme send u outside the range $[0, 1]$ into regions where $f'(u)$ is negative. A stable numerical solution forms, represented by the path OABI. Here, OA is a weak, entropy-satisfying, right-moving shock; BI is a strong, entropy-satisfying, left-moving shock; and AB is a strong, entropy-violating, stationary shock. Overwriting the sonic flux values with (5.1) did nothing to cure the problem, but using (5.2) had a dramatic effect, producing the solution OPI. This time, OP is a right-moving, entropy-violating shock that nevertheless closely approximates the true solution, whereas PI is a weak, right-moving, entropy-satisfying shock. Although not completely successful, this was a big step in the right direction. But, clearly, at least one more chapter of this story remains to be written.

6. Extension to systems. We consider now the solution of a one-dimensional system of conservation laws written in the form

$$\mathbf{u}_t + \mathbf{f}(\mathbf{u})_x = 0, \tag{6.1}$$

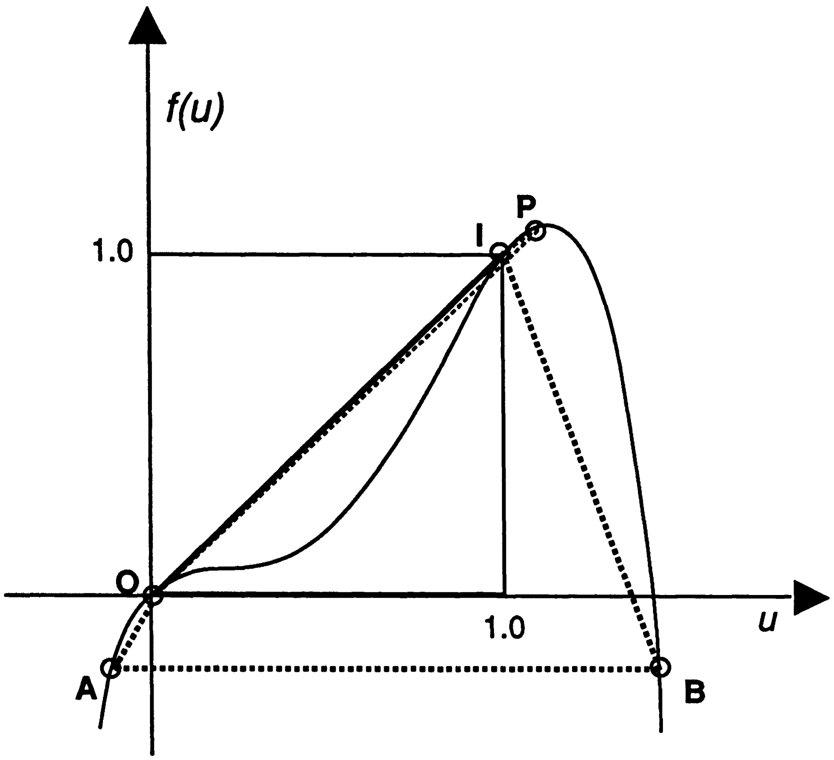


FIG. 7. *Diagram to illustrate solution to the Harten, Hyman, and Lax* [6] *test problem.*

or, more conveniently here, in matrix form

$$\mathbf{u}_t + A(\mathbf{u})\mathbf{u}_x = 0. \tag{6.2}$$

The question now is what statements can be made analogous to the two properties of sonic points given in the scalar case. In fact, we can make *no* general statements about sonic points remaining stationary. For a system of equations, sonic points in general move, unless they occur within simple waves. Thus, all we have to work with is the decay rate, but since this proved the more reliable criterion in the scalar case, this is not too distressing. A result corresponding to (2.10) can be found by differentiating (6.2) with respect to x, and premultiplying with ℓ, where $\ell(\mathbf{u})$ is a left eigenvector of $A(\mathbf{u})$. The result is

$$\ell \cdot [\mathbf{u}_{xt} + A_x\mathbf{u}_x + A\mathbf{u}_{xx}] = 0. \tag{6.3}$$

Let λ be the eigenvalue associated with ℓ. Since $\ell A = \lambda A$, we can write, at a sonic point where λ vanishes,

$$\ell \cdot [\mathbf{u}_{xt} + A_x\mathbf{u}_x] = 0, \tag{6.4}$$

and we have a prediction for the scalar quantity $\ell \cdot \mathbf{u}_{xt}$. This quantity is only in some loose sense a measure of the rate at which the wave of this particular family is decaying. For that we would need something like $(\ell \cdot \mathbf{u_x})_\mathbf{t}$, with a careful normalisation of ℓ. However, it is (6.4) that is the computationally useful result, although there seems to be no simple rule that transforms it from one set of unknowns to another. Therefore, we evaluate (6.4) for a particular case, that of the Euler equations in conservation form, $\mathbf{w} = (\rho, m, E)^T$, for which the matrix $A(\mathbf{u})$ is

$$\begin{vmatrix} 0 & 1 & 0 \\ \frac{1}{2}(\gamma-3)u^2 & -(\gamma-3)u & (\gamma-1) \\ -\frac{uc^2}{\gamma-1} + \frac{\gamma-2}{2}u^3 & \frac{c^2}{\gamma-1} + \frac{3-2\gamma}{2}u^2 & \gamma u \end{vmatrix}, \tag{6.5}$$

of which the left and right eigenvectors are contained in the rows of L and the columns of R, respectively, where

$$L = \frac{1}{2c^2}\begin{vmatrix} \frac{\gamma-1}{2}u^2 + uc & -(\gamma-1)u - c & \gamma-1 \\ 2c^2 - (\gamma-1)u^2 & 2(\gamma-1)u & -2(\gamma-1) \\ \frac{\gamma-1}{2}u^2 - uc & -(\gamma-1)u + c & \gamma-1 \end{vmatrix}, \tag{6.6}$$

$$R = \begin{vmatrix} 1 & 1 & 1 \\ u-c & u & u+c \\ \frac{1}{2}u^2 - uc + \frac{c^2}{\gamma-1} & \frac{1}{2}u^2 & \frac{1}{2}u^2 + uc + \frac{c^2}{\gamma-1} \end{vmatrix}. \tag{6.7}$$

Note that A, L, R can be written entirely in terms of the fluid velocity $u = m/\rho$ and the sound speed c given by $c^2 = \gamma(\gamma-1)[E/\rho - \frac{1}{2}m^2/\rho^2]$. L and R are given here in a normalised form such that $LR = RL = I$. From the above we can evaluate

$$A_x\mathbf{u}_x = \begin{bmatrix} 0 \\ (3-\gamma)\rho u_x^2 \\ \frac{4}{\gamma-1}\rho c u_x c_x + (3-\gamma)\rho u u_x^2 \end{bmatrix}. \tag{6.8}$$

By taking the inner product of this with each left eigenvector of A in turn, we find the astonishingly simple results,

$$\ell \cdot \mathbf{u}_{xt} = \frac{u_x}{2\rho c}[(3-\gamma)u_x - 4c_x], \tag{6.9}$$

$$\ell \cdot \mathbf{u}_{xt} = \frac{4u_x c_x}{\rho c}, \tag{6.10}$$

$$\ell \cdot \mathbf{u}_{xt} = \frac{u_x}{2\rho c}[(3-\gamma)u_x + 4c_x], \tag{6.11}$$

for sonic points associated with waves of the families $(u-c), u, (u+c)$, respectively.

For the special case of a simple wave, these results take interesting special forms. Inside a simple wave of the (u-c) family we have

$$u_x = -\frac{2}{\gamma-1}c_x = -\frac{1}{\rho c}p_x.$$

Under these circumstances, (6.7) can be written

$$\ell \cdot \mathbf{u}_{xt} = \frac{1}{2\rho c}\left(u_x - \frac{1}{\rho c}p_x\right)(u_x - c_x). \tag{6.12}$$

Strikingly, the brackets here do represent wavestrength and the gradient of wavespeed, directly analogous to the $\Delta u \Delta a$ term in the scalar case. For a simple $(u+c)$-wave we have

$$\ell \cdot \mathbf{u}_{xt} = -\frac{1}{2\rho c}\left(u_x + \frac{1}{\rho c}p_x\right)(u_x + c_x), \tag{6.13}$$

and for a simple u-wave, since $u_x = 0$, we have

$$\ell \cdot \mathbf{u}_{xt} = 0. \tag{6.14}$$

Actually, for a simple wave of any family, we can write, obtaining brevity at the expense of physical interpretation,

$$\ell \cdot \mathbf{u}_{xt} = -\frac{\gamma+1}{\gamma-1}\frac{c_x u_x}{\rho c}. \tag{6.15}$$

However, the fact that for nonsimple waves the equations (6.9)–(6.11) are not of the form $\Delta u \Delta a$ does seem to indicate that the sonic flux problem cannot be dealt with field by field. Despite this, we find experimentally that a field-by-field treatment is very successful.

7. Implementation for systems. Implementing these ideas in a code is straightforward. In the course of each timestep any sonic interface can be flagged, and the neighbouring cells corrected at the end without destroying the flow of any vectorisation. Let $\delta \mathbf{u}_L$ be the change brought about in the left cell by whatever preferred method is being used, and $\delta \mathbf{u}_R$ the change in the right cell. Let $\delta^* \mathbf{u}_L, \delta^* \mathbf{u}_R$ be changes that will bring about the proper decay. Then

$$\ell \cdot [\delta^* \mathbf{u}_R - \delta^* \mathbf{u}_L] = \ell \cdot \mathbf{u_{xt} \Delta x \Delta t}. \tag{7.1}$$

Since the neighbouring fluxes are not going to be altered (in a second-order scheme), we have

$$\delta^* \mathbf{u}_R + \delta^* \mathbf{u}_L = \delta \mathbf{u}_R + \delta \mathbf{u}_L. \tag{7.2}$$

Combining these gives

$$\ell \cdot (\delta \mathbf{u}_L^* - \delta \mathbf{u}_L) \;=\; \frac{1}{2}\left[\delta \mathbf{u}_R - \delta \mathbf{u}_L - \mathbf{u}_{xt} \Delta x \Delta t\right], \tag{7.3}$$

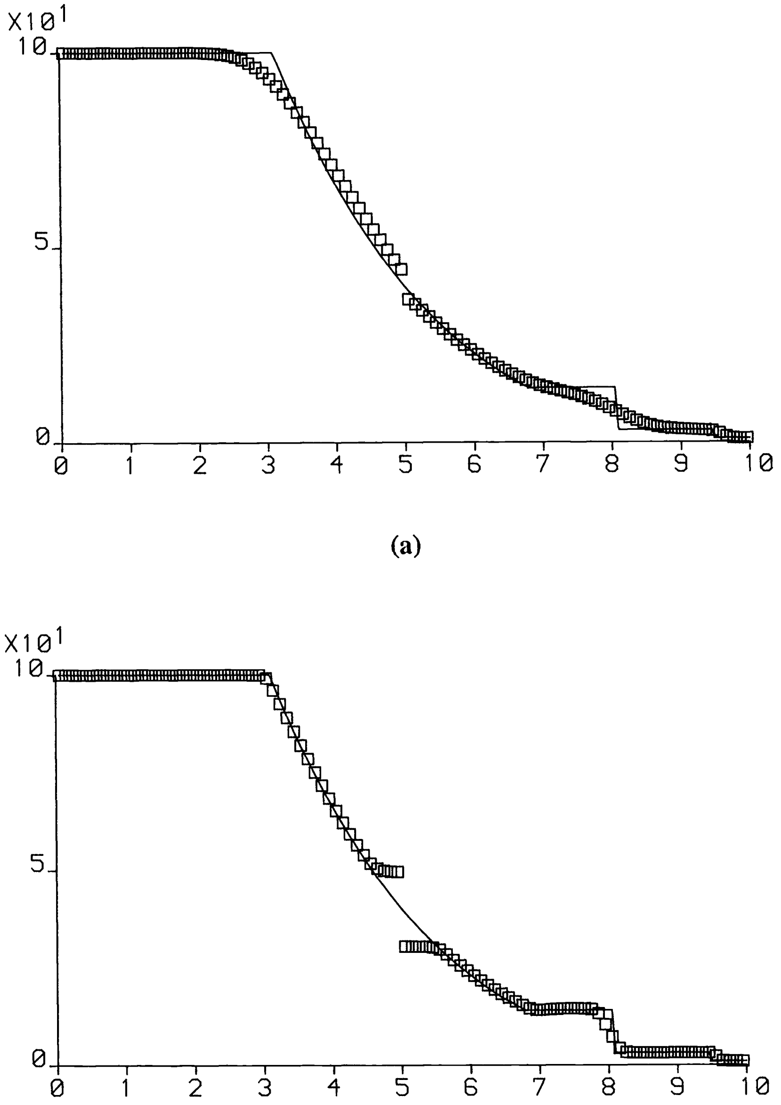
$$\ell \cdot (\delta \mathbf{u}_R^* - \delta \mathbf{u}_R) \;=\; -\frac{1}{2}\left[\delta \mathbf{u}_R - \delta \mathbf{u}_L - \mathbf{u}_{xt} \Delta x \Delta t\right]. \tag{7.4}$$

These conditions can be met, and conservation retained, if certain conserved quantities are transferred from the left cell to the right cell. It is natural to suppose that these should be proportional to $\mathbf{r}$, the right eigenvector corresponding to the sonic field, evaluated at the sonic interface. If they are taken to be $\alpha \mathbf{r}$, then by orthogonality of the left and right eigenvectors, and the normalisation used in (6.6), (6.7),

$$\alpha = \frac{1}{2}\ell \cdot \mathbf{u}_{xt} \Delta x \Delta t - \frac{1}{2}\ell \cdot \left[\delta \mathbf{u}_R - \delta \mathbf{u}_L\right], \tag{7.5}$$

where the first term is obtained from the appropriate equation of (6.9)–(6.11), or one of its simplifications.

8. Experiments with the Euler equations. As a first test for the Euler equations, the shock tube problem $\rho_L = 100$, $\rho_R = 1.0$; $p_L = 100, p_R = 1.0$; $u_L = u_R = 0$ was chosen. Figure 8(a) shows the solution to this problem on a mesh of 100 points using the author's upwind scheme [12] with no special treatment of the sonic point; 80 timesteps have been taken with $\Delta t / \Delta x = 0.20$. Apart from the usual defects of a first-order method, there is a very noticeable expansion shock. Figure 8(b) shows a



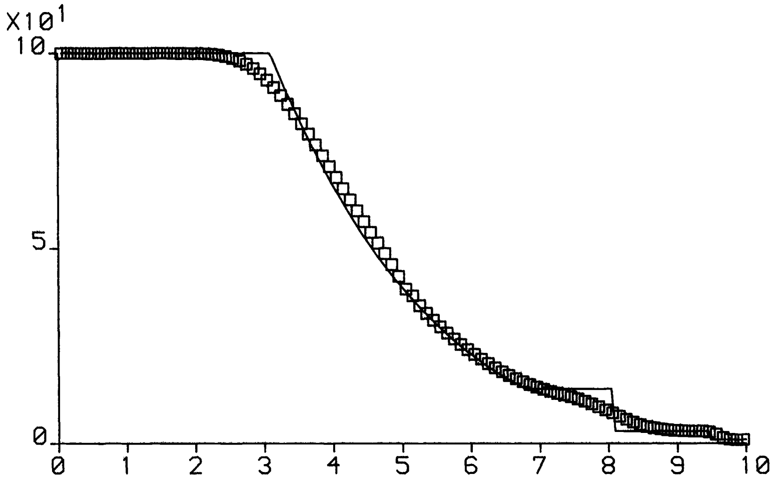
(a)

(b)

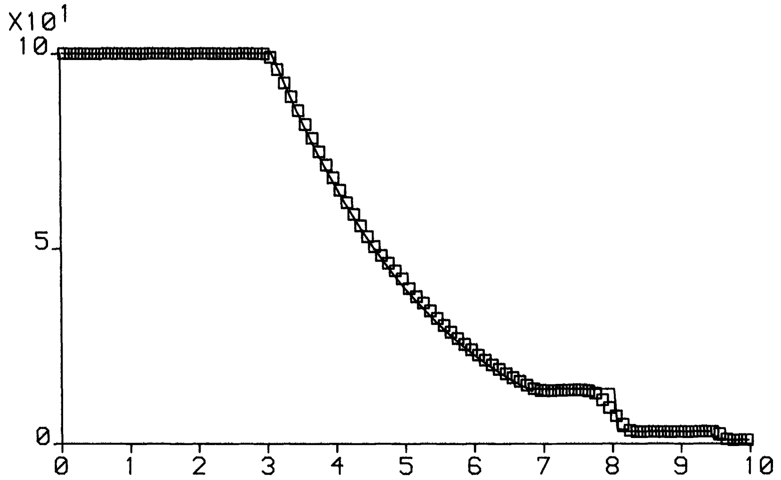
FIG. 8. *Results for a shock tube problem with no special treatment of the sonic point.* (a) *The basic first-order scheme.* (b) *The second-order scheme with Superbee limiter.*

second-order solution featuring the Superbee limiter. All the discontinuities are very much better resolved, including, unfortunately, the expansion shock. Clearly, this is an excellent solution to the wrong problem.

In Fig. 9(a) the first-order algorithm has been supplemented with the correction (7.5) evaluating the spreading rate to first order from (6.12). A very small disturbance can be observed near the sonic point, but this is by now the least of the errors. To have removed this completely we should have, as in the scalar case, modified the neighbouring fluxes as well. Figure 9(b) shows the second-order results, again with



(a)



(b)

FIG. 9. *As Fig.* 8 *but with the sonic cells modified according to* (7.3)–(7.4). (a) *The first-order scheme.* (b) *The second-order scheme with Superbee limiter.*

the spreading rate evaluated to the first order. The results are nearly as good as in the scalar case. For this test, no discernible change to the graphical results followed from using the more complicated evaluation of the spreading rate (6.9)–(6.11), or the simplified version (6.15). Indeed, since the flow is actually a simple wave, there should be no difference.

To investigate a more complex situation, the penetration of two rarefactions was considered. This is a classical problem solved by Riemann. A very full account of it is given by von Mises [10] and the main results are quoted in [2, pp. 191–196]. Unfortunately, the solution is inverse giving x and t in terms of u and c. Instead, accurate results were found from a numerical characteristics solution for data

$$u = 0, \qquad c = 1.2, \qquad |x| < 1, \tag{8.1}$$

$$u = 1.5\text{sgn}(x), \qquad c = 0.9, \quad 1 \leq |x| \leq 5. \tag{8.2}$$

The pattern of characteristics in Fig. 10 was produced. In this figure, the limiting characteristics of each expansion intersect at $t = 12.31$, which compares well with the exact value of $295/24 = 12.29$. The two lines marked by small symbols are the loci of the sonic points $|u| = c$. Initially these are fixed at $x = \pm 1$, but move outward as the waves penetrate. Figures 11(a), (b) show the distribution of a Mach number at $t = 3.0$. In the exact solution, the gradients are different (but not quite piecewise constant) in the central interaction region, and in the simple wave regions outside. In the unmodified second-order solution (Fig. 11(a)), the sonic points have remained in their original positions, and all parts of the solution are badly in error. Figure 11(b) shows that the modification (7.5) produces good results.

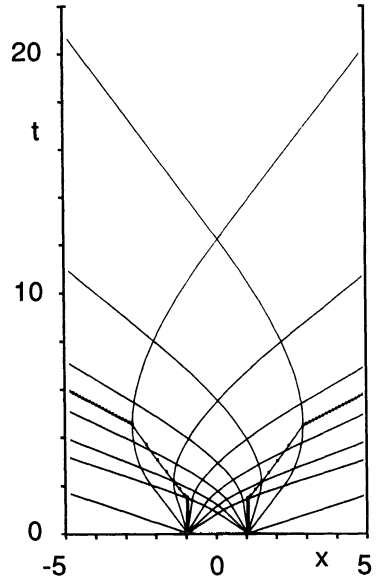


FIG. 10. *Wave diagram for two interpenetrating rarefactions.*

It had been intended to try and use the double expansion test as a way of discriminating between the various formulae (6.9)–(6.15). Since only (6.9)–(6.11) are derived without assuming simple waves, it was anticipated that they would do much better on this test than their simplified rivals. Surprisingly, very little difference was observed, and the following explanation(s) are advanced. The errors committed at a sonic point are usually very small. They become significant only by accumulation. However, in a compound wave the sonic points move and the errors do not accumulate at one point. Following this line of reasoning, the sonic modification was turned off entirely for $t > 1.2$ (when the sonic points first begin to move), and little change was seen. Thus moving sonic points seem able to take care of themselves, and it only seems necessary to modify the sonic behaviour for simple, and hence stationary, waves. Another possibility is that a completely correct treatment of the interaction would require modification of waves belonging to the nonstationary families. This would presumably be a second-order effect, but would need to be included before the

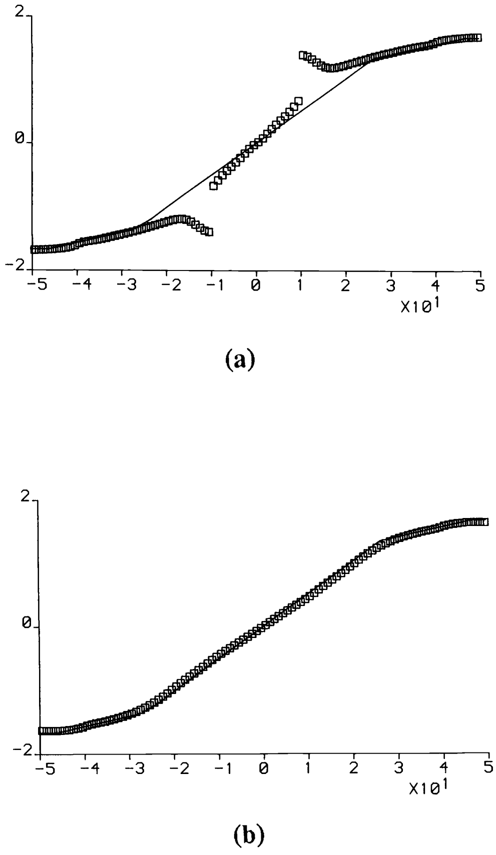


FIG. 11. *Two numerical solutions at* $t = 3$. (a) *No special treatment of the sonic points.* (b) *Sonic points treated according to* (7.3)–(7.4).

benefit of other second-order terms could be felt. In fact, an attempt was made to work out a second-order estimate for $\ell \cdot \mathbf{u}_{xt}$ along the lines adopted for the scalar problem in §3. However, the resulting formulae were very complicated, and resulted in no improvement. The conclusion is, therefore, that the very simple estimate (6.15) is satisfactory for practical application, and leads to excellent results.

9. Remarks on MUSCL-type schemes. It has often been noted that schemes using the MUSCL formulation [7], its higher order extensions [1], and to a lesser extent higher order schemes generally, have fewer difficulties near sonic points. Compare, for example, [15, Fig. 7a], which shows a near-discontinuous rarefaction wave in a solution generated using Godunov's method, with [15, Fig. 7e] where a MUSCL-type scheme generates a satisfyingly smooth solution to the same problem. This is because, in a region of fairly continuous expansion, the second-order fluxes at the interfaces either side of the sonic interface will avoid the errors discussed in connection with first-order upwind schemes, leading to equation (2.15).

If the data are not smooth, and limiters are called into play, the reconstructed data may be only first-order accurate, and the problem will recur. Recovery from this situation will eventually take place (provided the Riemann problems are solved using an entropy-satisfying scheme) in the sense that the solution will begin to decay, and may even come to decay at the correct rate. However the scheme cannot "know" at later times that its data derives from earlier errors, so some local distortion will persist. Whenever this happens, the present technique of enforcing the correct decay rate at all times should cure the problem.

10. Concluding remarks. The proper treatment of sonic regions in the context of upwind differencing has been the source of much confusion. This seems to be because all existing theory (which is limited to scalar problems) is based on the assumption that each interface flux away from the sonic interface will be computed by simple first-order upwinding. It is pointed out here that this necessarily creates errors at those interfaces next to the sonic interface, and that tinkering with the sonic interface alone cannot cure all the problems. For a first-order method to meet all of the natural design criteria, the adjacent fluxes must also be modified. For second-order schemes there is a choice between "fixing" the stationary characteristic, or forcing the proper decay of the stationary wave. The second choice is numerically better posed, gave better results, and is also the only one that can be generalised to systems of equations. The algebra for the one-dimensional Euler equations leads to fairly simple conditions that are straightforward to implement to first order. Although there is some indication that a field-by-field treatment is not valid to second order, the first-order implementation gives excellent practical results.

Acknowledgment. The shock tube data used for the first Euler test case were suggested by Dr. E.F. Toro, of the College of Aeronautics, Cranfield. For this, and for interesting discussions, I am most grateful.

REFERENCES

[1] P. Colella and P. R. Woodward, *The piecewise parabolic method (PPM) for gas dynamical simulations*, J. Comput. Phys., 54 (1984), pp. 174–201.

[2] R. Courant and K. O. Friedrichs, *Supersonic Flow and Shock Waves*, Interscience, New York, 1948; reprinted by Springer-Verlag, Berlin, New York, 1976.

[3] J. B. Goodman and R. J. Leveque, *A geometric approach to high-resolution TVD schemes*, SIAM J. Numer. Anal., 25 (1984), pp. 268–284.

[4] A. Harten, *High resolution schemes for hyperbolic conservation laws*, J. Comput. Phys., 49 (1983), pp. 357–393.

[5] A. Harten and J. M. Hyman, *Self-adjusting grid methods for one-dimensional hyperbolic conservation laws*, J. Comput. Phys., 50 (1985), pp. 235–269.

[6] A. Harten, J. M. Hyman, and P. D. Lax, *On finite difference approximation and entropy conditions for shocks*, Comm. Pure Appl. Math., 29 (1976), pp. 297–322.

[7] B. van Leer, *Toward the ultimate conservative differencing scheme, V, a second-order sequel to Godunov's method*, J. Comput. Phys., 32 (1979), pp. 101–136.

[8] ———, *On the relationship between the upwind differencing schemes of Godunov, Engquist-Osher, and Roe*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 1–20.

[9] B. van Leer, W.-T. Lee, and K. G. Powell, *Sonic-point capturing*, AIAA paper 89-1945-CP, AIAA 9th CFD Conference, Buffalo, NY, 1989.

[10] R. von Mises, *Mathematical Theory of Compressible Fluid Flow*, Academic Press, New York, 1958.

[11] S. Osher, *Riemann solvers, the entropy condition, and difference approximations*, SIAM J. Numer. Anal., 25 (1984), pp. 217–235.

[12] P. L. Roe, *Approximate Riemann solvers, parameter vectors, and difference schemes*, J. Comput. Phys., 43 (1981), pp. 357–372.

[13] ———, *Some contributions to the modelling of discontinuous flows*, in Lectures in Applied

Mathematics, 22, Part 2, B. E.Engquist, S. Osher, and R. J. Somerville, eds., American Mathematical Society, Providence, RI, 1985, pp. 163–194.

[14] P. K. SWEBY, *High-resolution schemes using flux limiters for hyperbolic conservation laws*, SIAM J. Numer. Anal., 21 (1984), pp. 995–1011.

[15] P. R. WOODWARD AND P. COLELLA, *The numerical simulation of two-dimensional flow with strong shocks*, J. Comput. Phys., 54 (1984), pp. 115–173.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 2, pp. 631–644, March 1992

BI-CGSTAB: A FAST AND SMOOTHLY CONVERGING VARIANT OF BI-CG FOR THE SOLUTION OF NONSYMMETRIC LINEAR SYSTEMS*

H. A. VAN DER VORST†

Abstract. Recently the Conjugate Gradients-Squared (CG-S) method has been proposed as an attractive variant of the Bi-Conjugate Gradients (Bi-CG) method. However, it has been observed that CG-S may lead to a rather irregular convergence behaviour, so that in some cases rounding errors can even result in severe cancellation effects in the solution. In this paper, another variant of Bi-CG is proposed which does not seem to suffer from these negative effects. Numerical experiments indicate also that the new variant, named Bi-CGSTAB, is often much more efficient than CG-S.

Key words. Bi-CG, CG-S, nonsymmetric linear systems, iterative solver, preconditioning

AMS(MOS) subject classification. 65F10

1. Introduction.

In recent years the Conjugate Gradients-Squared (CG-S) method [8] has been recognized as an attractive variant of the Bi-Conjugate Gradient (Bi-CG) iterative method for the solution of certain classes of nonsymmetric linear systems. Recent studies indicate that the method is often competitive with other established methods, such as GMRES [7]. For such comparative studies see, e.g., [1], [2], [6].

In many situations, however, one is faced with a quite irregular convergence behaviour of CG-S, in particular in situations when starting the iteration close to the solution. This is a common situation in, e.g., the final stages of some nonlinear solvers and in time dependent problems. The then often occurring irregularities in the iteration process may even lead to severe cancellation, spoiling the solution delivered by the process.

This motivated our search for a more smoothly converging variant of Bi-CG, without giving up the attractive speed of convergence of CG-S. It appeared that an early predecessor of CG-S, named IDR [10], could be reformulated to a method that shows the desired properties (at least for a large number of test cases).

In this paper we give some relevant background for the CG-S method in §2, which provides a suitable framework for the presentation of the new variant Bi-CGSTAB in §3. The various ways in which preconditioning can be incorporated in the algorithm are discussed in §4. In §5 we show, by suitable examples, that Bi-CGSTAB may be very attractive in comparison with CG-S in many situations.

2. Conjugate gradients-squared (CG-S).

The residual vectors r_i, generated by the CG method, satisfy a 3-term recurrence relation. This 3-term recurrence relation is not used explicitly in CG, but it is fundamental for the efficiency of the method. When A is not symmetric we lose this property for the r_i's. Moreover, when A is not positive definite, then $\| \; \|_A$ does not define a norm, so that it does not make sense to minimize $\| x - x_i \|_A$.

In the Bi-CG method [3], the approximations are constructed in such a way that the residual r_j is orthogonal with respect to another row of vectors $\hat{r}_0, \hat{r}_1, \cdots, \hat{r}_{j-1}$,

* Received by the editors May 21, 1990; accepted for publication (in revised form) February 18, 1991.

† Mathematical Institute, University of Utrecht, Budapestlaan 6, NL-3584 CD Utrecht, the Netherlands (vorst@math.ruu.nl).

and, vice versa, $\hat{r}_j$ is orthogonal with respect to $r_0, r_1, \cdots, r_{j-1}$. This can be accomplished by two 3-term recurrence relations for the rows $\{r_j\}$ and $\{\hat{r}_j\}$. The Bi-CG method also terminates within n steps at most (when A is an n by n matrix), but there is no minimization property as in CG for the intermediate steps.

Sonneveld [8] made the observation that, in the case of convergence, both the rows $\{r_j\}$ and $\{\hat{r}_j\}$ converge towards zero, but that only the convergence of the $\{r_j\}$ is exploited. He proposes the following modification to Bi-CG by which all the convergence effort is concentrated in the r_i vectors. For the Bi-CG vectors it is well known that they can be written as $r_j = P_j(A)r_0$ and $\hat{r}_j = P_j(A^T)\hat{r}_0$, and because of the bi-orthogonality relation we have that

$$\begin{aligned}(r_j, \hat{r}_i) &= (P_j(A)r_0, P_i(A^T)\hat{r}_0)\\ &= (P_i(A)P_j(A)r_0, \hat{r}_0) = 0 \quad \text{for } i < j.\end{aligned}$$

The iteration parameters for Bi-CG are computed from innerproducts like the above. Sonneveld observed that we can also construct the vectors $\tilde{r}_j = P_j^2(A)r_0$, using only the latter form of the innerproduct for recovering the Bi-CG parameters (which implicitly define the polynomial P_j). By doing so, it can be avoided that the vectors $\hat{r}_j$ have to be formed, nor is there any multiplication with the matrix A^T.

The resulting algorithm can be represented by the following scheme.

Unpreconditioned CG-S Algorithm.

```
$x_0$ is an initial guess; $r_0 = b - Ax_0$;
$\hat{r}_0$ is an arbitrary vector, such that $(r_0, \hat{r}_0) \neq 0$,
    e.g., $\hat{r}_0 = r_0$;
$\rho_0 = 1; p_0 = q_0 = 0$;
for $i = 1, 2, 3, \cdots$,
    $\rho_i = (\hat{r}_0, r_{i-1})$;
    $\beta = \rho_i / \rho_{i-1}$;
    $u = r_{i-1} + \beta q_{i-1}$;
    $p_i = u + \beta(q_{i-1} + \beta p_{i-1})$;
    $v = Ap_i$;
    $\alpha = \rho_i / (\hat{r}_0, v)$;
    $q_i = u - \alpha v$;
    $\hat{u} = u + q_i$;
    $x_i = x_{i-1} + \alpha w$;
    if $x_i$ is accurate enough then quit;
    $r_i = r_{i-1} - \alpha Aw$;
end
```

One iteration step of CG-S involves about as many arithmetical operations as one step of Bi-CG.

The last line of the Unpreconditioned CG-S Algorithm could be replaced by the computation of the true residual vector $r_i = b - Ax_i$. However, in many of the experiments, especially those with the typical jumping convergence behaviour of CG-S, it has been seen that this has a negative effect on the iteration process, i.e., it may take many more iteration steps for this *true residual* process to get at an x_i of similar accuracy as in the CG-S algorithm in its form described above. In some cases the *true residual* process did even not converge whereas CG-S did.

An example of such a situation is discussed in §5.4. A possible explanation might be that locally large variations in a current update direction overshadow variations in other almost converged directions, so that the true residual vector does not necessarily satisfy the underlying orthogonality relations for the updated vectors $P_i(A)r_0$. This aspect needs further research.

If the Bi-CG polynomial $P_i(A)$ is viewed as a reductor working on r_0, then this reductor is now applied twice in the new method and this explains the name *conjugate gradients-squared.*

The weak point in this way of reasoning is that the reduction operator $P_i(A)$ in Bi-CG is very dependent on the starting initial residual r_0, and that it is not likely to be a reduction operator for any other vector, not even for the particular vector $P_i(A)r_0$ itself. Indeed, it is not very difficult to construct examples for which $P_i(A)r_0$ is small in norm and for which ${P^2}_i(A)r_0$ is even much bigger in norm then r_0 is.

Such a phenomenon may happen in the following situation. Suppose that r_0 has small coordinates in some eigenvector directions of the matrix A. Then $P_i(\lambda)$ may take very large values in the corresponding eigenvalues, in particular when these eigenvalues are more or less isolated from the others, without leading to a significant contribution of that eigenvector direction to the norm of r_i in Bi-CG. However, the value of ${P^2}_i(\lambda)$ may be so large that the contribution of the corresponding eigenvector direction even dominates in the residual r_i of CG-S.

These situations occur quite often, and even during one convergence history we may observe many local peaks in the convergence curve for CG-S (see, e.g., the examples in §5). These peaks do not seem to delay the convergence of CG-S. However, in practical situations they may have quite adverse effects, since they may be so large that the local corresponding corrections to the current iterate result in cancellation. As a result, the final solution may have little significance, which can be checked by computing the real residual (instead of the updated residual as is done usually in these processes). This is a very serious problem with CG-S and in our examples in §5 we will encounter significant losses in accuracy.

For a more detailed discussion on the convergence behaviour of CG-S, see [9].

3. A more smoothly converging variant of CG-S. We have mentioned that the residuals r_i in CG-S satisfy the relation $r_i = P_i(A)^2r_0$, in which $P_i(A)r_0$ just defines the residual $r_{bi-CG,i}$ in the Bi-CG method:

$$r_{bi-CG,i} = P_i(A)r_0.$$

By construction we have that $(P_i(A)r_0, P_j(A^T)\hat{r}_0) = 0$ for $j < i$, which expresses the fact that $P_i(A)r_0$ is perpendicular to the subspace $K^i(A^T; \hat{r}_0)$, spanned by the vectors

$$\hat{r}_0, A^T\hat{r}_0, \cdots, (A^T)^{i-1}\hat{r}_0.$$

This implies that, in principle, we can also recover the Bi-CG iteration parameters by requiring that, e.g., r_i is perpendicular to $\tilde{P}_j(A^T)\hat{r}_0$, or, equivalently, that $(\tilde{P}_j(A)P_i(A)r_0, \hat{r}_0) = 0$, for another suitable set of polynomials $\tilde{P}_j$ of degree j. In Bi-CG one takes $\tilde{P}_j = P_j$, namely, $\hat{r}_j = P_j(A^T)\hat{r}_0$. This is exploited in CG-S, as we have indicated before, since recursion relations for the vectors $P_j^2(A)r_0$ can be derived from those for $P_j(A)r_0$.

Of course, we can construct other iteration methods, by which x_i are generated so that $r_i = \tilde{P}_i(A)P_i(A)r_0$ with other ith degree polynomials, like, e.g., Chebychev polynomials, which might be more suitable. Unfortunately, the optimal parameters for the Chebychev polynomials are in general not easily obtainable and also the recurrence relations for the resulting method are more complicated than for CG-S. Another possibility is to take for $\tilde{P}_j$ a polynomial of the form

$$(1) \qquad Q_i(x) = (1-\omega_1 x)(1-\omega_2 x)\cdots(1-\omega_i x),$$

and to select suitable constants ω_j. This expression leads to an almost trivial recurrence relation for the Q_i.

An obvious possibility to select ω_j in the jth iteration step is to minimize r_j, with respect to ω_j, for residuals that can be written as $r_j = Q_j(A)P_j(A)r_0$. This leads to a method which is mathematically equivalent with the IDR method described in [10]. IDR, however, in the form as it is described in [10] may also suffer from severe cancellation and can therefore lead to unreliable results. These effects can be even much worse than in CG-S. In fact, the CG-S method has been derived as an improvement to IDR and the IDR method was not given further attention.

However, it is possible to rewrite the scheme, in which the residuals $Q_i(A)P_i(A)r_0$ are generated, to an apparently rather stable and more efficient one. Because of its similarity to CG-S, its favourable stability properties, and its relation with Bi-CG, we have named the method Bi-CGSTAB.

We will now derive the (unpreconditioned) Bi-CGSTAB Algorithm. This will be done in a similar way as followed in [8] for the derivation of CG-S.

The polynomial P_i and related polynomials are implicitly defined by the Bi-CG scheme.

Unpreconditioned Bi-CG Algorithm.

```
$x_0$ is an initial guess; $r_0 = b - Ax_0$;
$\hat{r}_0$ is an arbitrary vector, such that
    $(\hat{r}_0, r_0) \neq 0$, e.g., $\hat{r}_0 = r_0$;
$\rho_0 = 1$;
$\hat{p}_0 = p_0 = 0$;
for $i = 1, 2, 3, \cdots$,
    $\rho_i = (\hat{r}_{i-1}, r_{i-1}); \beta_i = (\rho_i/\rho_{i-1})$;
    $p_i = r_{i-1} + \beta_i p_{i-1}$;
    $\hat{p}_i = \hat{r}_{i-1} + \beta_i \hat{p}_{i-1}$;
    $v_i = Ap_i$;
    $\alpha_i = \rho_i/(\hat{p}_i, v_i)$;
    $x_i = x_{i-1} + \alpha_i p_i$;
    if $x_i$ is accurate enough then quit;
    $r_i = r_{i-1} - \alpha_i v_i$;
    $\hat{r}_i = \hat{r}_{i-1} - \alpha_i A^T \hat{p}_i$;
end
```

From this scheme it is straightforward to show that $r_i = P_i(A)r_0$ and $p_{i+1} = T_i(A)r_0$, in which $P_i(A)$ and $T_i(A)$ are ith degree polynomials in A. The Bi-CG Algorithm then defines the relations between these polynomials:

$$T_i(A)r_0 = (P_i(A) + \beta_{i+1}T_{i-1}(A))r_0,$$

and

$$P_i(A)r_0 = (P_{i-1}(A) - \alpha_i A T_{i-1}(A))r_0.$$

In the Bi-CGSTAB Algorithm we wish to have recurrence relations for

$$r_i = Q_i(A)P_i(A)r_0.$$

With Q_i as in (1) and the Bi-CG relation for the factor P_i and T_i, it then follows that

$$\begin{aligned}Q_i(A)P_i(A)r_0 &= (1-\omega_i A)Q_{i-1}(A)(P_{i-1}(A)-\alpha_i AT_{i-1}(A))r_0\\ &= \{Q_{i-1}(A)P_{i-1}(A)-\alpha_i AQ_{i-1}(A)T_{i-1}(A)\}r_0\\ &\quad -\omega_i A\{(Q_{i-1}(A)P_{i-1}(A)-\alpha_i AQ_{i-1}(A)T_{i-1}(A))\}r_0.\end{aligned}$$

Clearly, we also need a relation for the product $Q_i(A)T_i(A)r_0$. This can also be obtained from the Bi-CG relations:

$$\begin{aligned}Q_i(A)T_i(A)r_0 &= Q_i(A)(P_i(A)+\beta_{i+1}T_{i-1}(A))r_0\\ &= Q_i(A)P_i(A)r_0+\beta_{i+1}(1-\omega_i A)Q_{i-1}(A)T_{i-1}(A)r_0\\ &= Q_i(A)P_i(A)r_0+\beta_{i+1}Q_{i-1}(A)T_{i-1}(A)r_0\\ &\quad -\beta_{i+1}\omega_i AQ_{i-1}(A)T_{i-1}(A)r_0.\end{aligned}$$

Finally we have to recover the Bi-CG constants ρ_i, β_i, and α_i by innerproducts in terms of the new vectors that we now have generated. For example, β_i can be computed as follows. First we compute

$$\tilde{\rho}_{i+1} = (\hat{r}_0, Q_i(A)P_i(A)r_0) = (Q_i(A^T)\hat{r}_0, P_i(A)r_0).$$

By construction, $P_i(A)r_0$ is orthogonal with respect to all vectors $U_{i-1}(A^T)\hat{r}_0$, where U_{i-1} is an arbitrary polynomial of degree $i-1$ at most. This means that we have to consider only the highest order term of $Q_i(A^T)$ when computing $\tilde{\rho}_{i+1}$. This term is given by $(-1)^i\omega_1\omega_2\cdots\omega_i(A^T)^i$. We actually wish to compute

$$\rho_{i+1} = (P_i(A^T)\hat{r}_0, P_i(A)r_0),$$

and since the highest order term of $P_i(A^T)$ is given by $(-1)^i\alpha_1\alpha_2\cdots\alpha_i(A^T)^i$, it follows that

$$\beta_i = (\tilde{\rho}_i/\tilde{\rho}_{i-1})(\alpha_{i-1}/\omega_{i-1}).$$

The other constants can be derived similarly.

Note that in our discussion we have focused on the recurrence relations for the vectors r_i and p_i, while in fact our main goal is to determine x_i. As in all CG-type methods, x_i itself is not required for continuing the iteration, but it can easily be determined as a "sideproduct" by realizing that an update of the form $r_i = r_{i-1} - \gamma Ay$ corresponds to an update $x_i = x_{i-1} + \gamma y$ for the current approximated solution.

By writing r_i for $Q_i(A)P_i(A)r_0$ and p_i for $Q_{i-1}(A)T_{i-1}(A)r_0$, we obtain the following scheme for Bi-CGSTAB (we trust that, with the foregoing observations, the reader will now be able to verify the relations in Bi-CGSTAB). In this scheme we have computed the ω_i so that $r_i = Q_i(A)P_i(A)r_0$ is minimized in 2-norm as a function of ω_i.

UNPRECONDITIONED BI-CGSTAB ALGORITHM.

```
$x_0$ is an initial guess; $r_0 = b - Ax_0$;
$\hat{r}_0$ is an arbitrary vector, such that
    $(\hat{r}_0, r_0) \neq 0$, e.g., $\hat{r}_0 = r_0$;
$\rho_0 = \alpha = \omega_0 = 1$;
```

```
$v_0 = p_0 = 0$;
for $i = 1, 2, 3, \cdots$,
    $\rho_i = (\hat{r}_0, r_{i-1}); \beta = (\rho_i/\rho_{i-1})(\alpha/\omega_{i-1})$;
    $p_i = r_{i-1} + \beta(p_{i-1} - \omega_{i-1}v_{i-1})$;
    $v_i = Ap_i$;
    $\alpha = \rho_i/(\hat{r}_0, v_i)$;
    $s = r_{i-1} - \alpha v_i$;
    $t = As$;
    $\omega_i = (t, s)/(t, t)$;
    $x_i = x_{i-1} + \alpha p_i + \omega_i s$;
    if $x_i$ is accurate enough then quit;
    $r_i = s - \omega_i t$;
end
```

In order to restrict on memory traffic, we have carried out both updates to the current solution x in one single step, while the updates to the residual r had to be done separately ($s = r_{i-1} - \alpha v_i$ and $r_i = s - \omega_i t$). So s represents the residual after a "Bi-CG step" and one might terminate the iteration as soon as $\|s\|$ is small enough, but in that case, before stopping the algorithm, the current solution has to be updated appropriately as $x_i = x_{i-1} + \alpha p_i$ in order to be compatible with the current residual s (and the computation of t, ω_i, as well as the second update $\omega_i s$ should be skipped).

From the orthogonality property $(P_i(A)r_0, Q_j(A^T)\hat{r}_0) = 0$, for $j < i$, it follows that Bi-CGSTAB is also a finite method, i.e., in exact arithmetic it will terminate after $m \leq n$ iteration steps. In this case we get $s = 0$ at iteration step m and ω_m is then not defined. This represents a lucky breakdown of the algorithm and the process should be terminated as indicated in the previous paragraph.

In the above form Bi-CGSTAB requires, for the solution of an N by N system $Ax = b$, evaluation of two matrix vector products with A, $12N$ flops for vector updates, and four innerproducts. This has to be compared with (unpreconditioned) CG-S which requires also two matrix vector products with A, and $13N$ flops, but only two innerproducts. In practical situations, however, the two additional innerproducts lead to only a small increase in computational work per iteration step and this is readily undone by almost any reduction in the number of iteration steps (especially on vector computers on which innerproducts are usually fast operations).

Except for memory locations for x, b, r, and A, we need for Bi-CGSTAB memory space for four additional N-vectors $\hat{r}_0, p, v$, and t (note that r may be overwritten by s). This is the same as for CG-S.

In §5 we will see, for some examples, the more smooth convergence properties of Bi-CGSTAB in comparison with CG-S. We will also see that, at least for our experiments, the new method is often more efficient than CG-S, in terms of the amount of computational work necessary to achieve a specified accuracy in the final result.

We have not discussed convergence problems that one can encounter with Bi-CG, and, hence, with CG-S and Bi-CGSTAB. These problems stem basically from the fact that for general matrices the bilinear form

$$[x, y] \equiv (P(A^T)x, P(A)y),$$

which is used to form the bi-orthogonality, does not define an innerproduct. In particular, it may occur that, by an unfavourable choice for $\hat{r}_0$, an iteration parameter ρ_i or $(\hat{r}_0, v_i)$ is zero (or very small), without convergence having taken place. In an actual code, one should test for such situations and take appropriate measures, e.g., restart with a different $\hat{r}_0$ or switch to another method (for example, GMRES [7]).

Moreover, one should keep in mind that Bi-CGSTAB, in the form as presented, is one out of many possible variants, which are all equivalent in exact arithmetic but which may have different behaviour in finite precision arithmetic. One such variant, which is less straightforward than the presented one, is obtained by the following changes to the scheme:

- include $\rho_1 = (\hat{r}_0, r_0)$ in the initialization part of the scheme;
- skip the computation of ρ_i in the first line of the iteration part;
- add the computation of $\rho_{i+1} = -\omega_i(\hat{r}_0, t)$ immediately after the computation of ω_i.

We have seen, in some of our experiments, a markedly better convergence behaviour for this variant, but further research is necessary in order to determine the most satisfying variant.

4. Preconditioning. If we want to use preconditioning with a suitable preconditioning matrix K, i.e., $K \approx A$, then we write $K = K_1K_2$ and we may apply any of the previously discussed iteration schemes to the explicitly preconditioned system

$$\tilde{A}\tilde{x} = \tilde{b}, \tag{2}$$

with $\tilde{A} = K_1{}^{-1}AK_2{}^{-1}, x = K_2{}^{-1}\tilde{x}$, and $\tilde{b} = K_1{}^{-1}b$. For example, for $K_1 = I$ we have preconditioning from the right (or postconditioning), for $K_2 = I$ we have preconditioning from the left, and for $K_1 = L, K_2 = U$, we have the well-known preconditioning from both sides.

Now we write the CG-S scheme (§2) for (2), and denote all the occurring vectors by $\tilde{}$, e.g., $\tilde{p}_i$. With the change of variables:

$$\begin{aligned}
&\tilde{p}_i \Rightarrow K_1{}^{-1}p_i, \tilde{q}_i \Rightarrow K_1{}^{-1}q_i, \tilde{v}_i \Rightarrow K_1{}^{-1}v_i,\\
&\tilde{r}_i \Rightarrow K_1{}^{-1}r_i, \tilde{u}_i \Rightarrow K_1{}^{-1}u_i, \tilde{x}_i \Rightarrow K_2x_i,\\
&\hat{r}_0 \Rightarrow K_1{}^T\bar{r}_0,
\end{aligned}$$

this leads to the following scheme for *preconditioned CG-S*.

PRECONDITIONED CG-S ALGORITHM.

```
$x_0$ is an initial guess; $r_0 = b - Ax_0$;
$\bar{r}_0$ is an arbitrary vector, such that $(r_0, \bar{r}_0) \neq 0$,
    e.g., $\bar{r}_0 = r_0$;
$\rho_0 = 1; p_0 = q_0 = 0$;
for $i = 1, 2, 3, \cdots$,
    $\rho_i = (\bar{r}_0, r_{i-1})$;
    $\beta = \rho_i/\rho_{i-1}$;
    $u = r_{i-1} + \beta q_{i-1}$;
    $p_i = u + \beta(q_{i-1} + \beta p_{i-1})$;
    Solve $y$ from $Ky = p_i$;
    $v = Ay$;
    $\alpha = \rho_i/(\bar{r}_0, v)$;
    $q_i = u - \alpha v$;
    Solve $z$ from $Kz = u + q_i$
    $x_i = x_{i-1} + \alpha z$;
```

```
        if $x_i$ is accurate enough then quit;
        $r_i = r_{i-1} - \alpha Az$;
    end
```

Note that this scheme delivers the variables x_i and r_i corresponding to the original system $Ax = b$, i.e., $r_i = b - Ax_i$.

A remarkable observation is that K_1 and K_2 play no explicit role in the scheme, and that any of the forms of preconditioning (i.e., any of the choices for K_1 and K_2) *correspond only with a different choice for* $\hat{r}_0$ in the original, explicitly preconditioned scheme. Hence, if one of the forms of explicit preconditioning in the original scheme might lead to an advantage, then the same advantage can be obtained by the above scheme through an appropriate choice of $\bar{r}_0$. Or, in still other words, instead of studying the effect of the different forms of applying the preconditioner, one may as well study the effect of the choice for $\bar{r}_0$.

When we rewrite the Bi-CGSTAB scheme for equation (2), similarly as previously for CG-S, then with the change of variables:

$$\tilde{p}_i \Rightarrow {K_1}^{-1}p_i, \tilde{v}_i \Rightarrow {K_1}^{-1}v_i, \tilde{r}_i \Rightarrow {K_1}^{-1}r_i,$$
$$\tilde{s} \Rightarrow {K_1}^{-1}s, \tilde{t} \Rightarrow {K_1}^{-1}t, \tilde{x}_i \Rightarrow K_2 x_i,$$
$$\hat{r}_0 \Rightarrow {K_1}^T \bar{r}_0,$$

we obtain the following scheme for *preconditioned Bi-CGSTAB*.

PRECONDITIONED BI-CGSTAB ALGORITHM.

```
    $x_0$ is an initial guess; $r_0 = b - Ax_0$;
    $\bar{r}_0$ is an arbitrary vector, such that
        $(\bar{r}_0, r_0) \neq 0$, e.g., $\bar{r}_0 = r_0$;
    $\rho_0 = \alpha = \omega_0 = 1$;
    $v_0 = p_0 = 0$;
    for $i = 1, 2, 3, \cdots$,
        $\rho_i = (\bar{r}_0, r_{i-1}); \beta = (\rho_i/\rho_{i-1})(\alpha/\omega_{i-1})$;
        $p_i = r_{i-1} + \beta(p_{i-1} - \omega_{i-1}v_{i-1})$;
        Solve $y$ from $Ky = p_i$;
        $v_i = Ay$;
        $\alpha = \rho_i/(\bar{r}_0, v_i)$;
        $s = r_{i-1} - \alpha v_i$;
        Solve $z$ from $Kz = s$;
        $t = Az$;
        $\omega_i = ({K_1}^{-1}t, {K_1}^{-1}s)/({K_1}^{-1}t, {K_1}^{-1}t)$;
        $x_i = x_{i-1} + \alpha y + \omega_i z$;
        if $x_i$ is accurate enough then quit;
        $r_i = s - \omega_i t$;
    end
```

This scheme too delivers the variables x_i and r_i corresponding to the original system $Ax = b$.

For preconditioned CG-S we have seen that any of the forms of preconditioning can be regarded as a suitable choice for $\bar{r}_0$. However, for Bi-CGSTAB there is an explicit difference between the different forms, which cannot be attributed to a suitable choice for $\bar{r}_0$, because of the expression for ω_i. This expression does not seem to make the preconditioned scheme very attractive, but we might as well compute another value for ω_i:

$$\omega_i = (t, s)/(t, t). \tag{3}$$

When computing ω_i as in (3), we are minimizing the current residual for the original system rather than the preconditioned one. In this case we have in fact the same effect as by explicit postconditioning in the original scheme (though for a different $\bar{r}_0$), and hence we lose potential (near-)symmetry of the preconditioned operator. In such a case we might prefer to apply the Unpreconditioned Bi-CGSTAB Algorithm, as in §3, to the explicitly preconditioned system $K_1^{-1}AK_2^{-1}\tilde{x} = K_1^{-1}b$. However, this has the obvious disadvantage that we have to construct a (near-)symmetric splitting of the preconditioner K.

We will refer to the scheme with expression (3) for the ω_i as Bi-CGSTAB-P.

Hence, if one compares Bi-CGSTAB-P with preconditioned CG-S then one can view this as a comparison between the two explicitly postconditioned schemes in §§2 and 3 (each with a different $\hat{r}_0$).

5. Numerical results. In our experiments we consider four different, but representative situations. These experiments have been carried out in double precision floating point arithmetic (about 15 decimal places) on a CONVEX C-240 computer. In order to avoid all confusion about the definition of $\bar{r}_0$ (see §4), all experiments have been carried out with CG-S and Bi-CGSTAB applied to the explicitly preconditioned system $L^{-1}AU^{-1}\tilde{x} = L^{-1}b$. However, based upon our experiments so far, our conclusions would be about the same when we compare the preconditioned CG-S scheme with Bi-CGSTAB-P (§4).

In all cases the iteration was started with $x_0 = 0$.

5.1. Example 1. The first situation is one in which Bi-CG converges quite smoothly and in which it was, apparently, a good idea to square the Bi-CG polynomial. This happens sometimes in the early phases of the iteration process or in situations where the eigenvalue distribution is quite uniform.

As an example we show in Fig. 1 the convergence behaviour for the discretized Poisson equation in two dimensions over a 150 × 150 grid (leading to a symmetric positive definite system), preconditioned by Modified Incomplete Choleski decomposition [4]. In this case CG-S converged about twice as fast as Bi-CG, as we might expect from heuristic arguments.

We see that in this case, though Bi-CGSTAB converges more smoothly, it does not improve the iteration process with respect to efficiency. In similar experiments Bi-CGSTAB requires roughly the same number of iteration steps as CG-S, sometimes slightly more and sometimes slightly less.

5.2. Example 2. Our second example is a preconditioned symmetric positive definite system for which Bi-CG (=Conjugate Gradients in this case) loses orthogonality among the residuals in a very early phase. For a discussion on this effect and its consequences, see [9]. In such a case one would expect some strong effects when squaring the Bi-CG polynomial, as is done in CG-S.

The linear system comes from a 5-point finite difference discretization of the partial differential equation

$$-(Du_x)_x - (Du_y)_y = 1$$

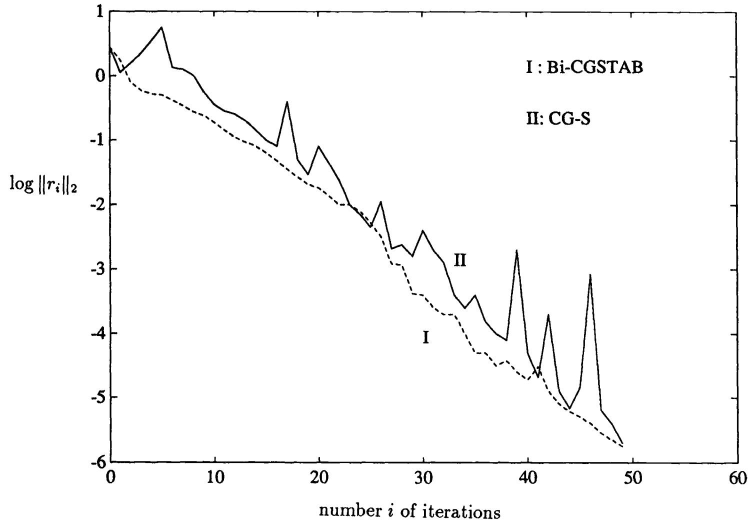


FIG. 1. *CG-S and Bi-CGSTAB for an ideal CG-S situation.*

over the unit square, with Dirichlet boundary conditions along $y = 0$ and Neumann conditions along the other parts of the boundary. Meshsizes have been chosen so that the resulting linear system has $150 \times 150 = 22,500$ unknowns.

The function D is defined as

$$D = 1000 \quad \text{for } 0.1 \leq x, y \leq 0.9, \quad \text{and } D = 1 \quad \text{elsewhere.}$$

Modified Incomplete Choleski Decomposition [4] was used as preconditioner.

In Fig. 2 we see the loss of orthogonality reflected in the convergence behaviour of Bi-CG by small irregularities and by the fact that superlinear convergence does not take place. CG-S converges faster eventually, but we see that the local effects in this method are much more violent.

Bi-CGSTAB seems to have about the same "asymptotical" speed of convergence as CG-S (if we discard the peaks in the CG-S curve), but its convergence behaviour is definitely smoother. In situations like these, we often see that the updated residual in Bi-CGSTAB has more significance than the one in CG-S. See also the discussion of example 4 (§5.4) for this effect.

5.3. Example 3. In our third example the nonsymmetric linear system comes from discretization of the partial differential equation

$$-u_{xx} - u_{yy} + ((au)_x + au_x)/2 = 1$$

over the unit square, with $a = 20 \exp(3.5(x^2 + y^2))$. Along the boundaries we have Dirichlet conditions (this equation was taken from [11]). The equation was discretized over a rectangular grid, with central differences for the first order terms, with stepsize $1/201$ in each direction, leading to a system with $40,000$ unknowns. The linear system was preconditioned by a standard incomplete LU factorization [5].

In Fig. 3 we have displayed the norms of the iterated vectors r_i for both CG-S and Bi-CGSTAB, and again we note the much smoother convergence behaviour of

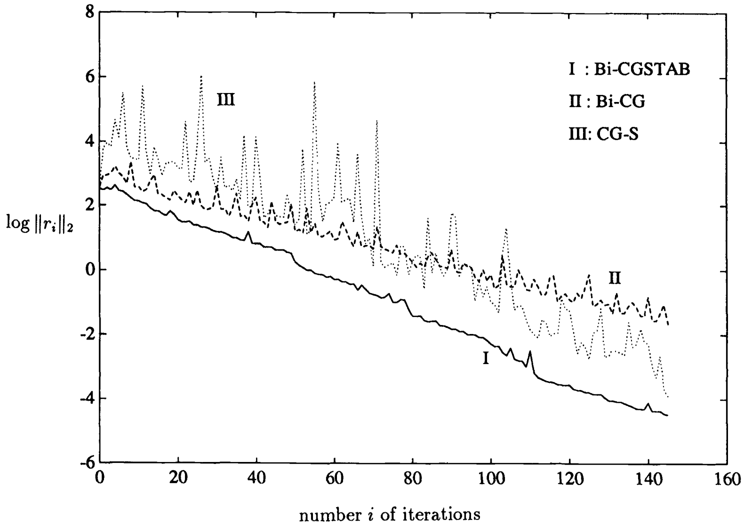


FIG. 2. *Less smooth convergence behaviour of Bi-CG and CG-S.*

Bi-CGSTAB. Also note that some residuals in CG-S may be quite large as compared with the starting residual. Since the updates to the corresponding iterate vectors x_i must have been large too, we may expect cancellation effects in the iterated vector. Indeed, after 200 CG-S iterations, the explicitly computed residual (using the delivered iteration vector x_i) was larger (in norm) than the updated residual vector r_i by three orders of magnitude. This implies that the approximated solution had an error which was about 1000 times as large as we might have thought. For Bi-CGSTAB both residuals were virtually equal to each other.

Finally note that in this case Bi-CGSTAB, apart from being more stable, also converges faster than CG-S. This was true for most of our test cases. We have also tested Bi-CG for this problem, but this method showed hardly any progress within 450 iteration steps.

5.4. Example 4. The fourth example is typical for a situation in which Bi-CGSTAB is much more efficient than CG-S. The nonsymmetric linear system comes from discretization of the partial differential equation

$$-(Au_x)_x - (Au_y)_y + B(x,y)u_x = F$$

over the unit square, with $B(x,y) = 2\exp(2(x^2+y^2))$. Along the boundaries we have Dirichlet conditions: $u = 1$, for $y = 0, x = 0$ and $x = 1$, and $u = 0$ for $y = 1$.

The function A is defined as shown in Fig. 4; $F = 0$ everywhere, except for the small subsquare in the centre where $F = 100$.

The equation was discretized over a rectangular grid, with central differences for the first order term, with stepsize $1/128$ in each direction, leading to a system with 127^2 unknowns. The linear system was preconditioned by an incomplete LU factorization [5].

Figure 5 shows the norms of the iteration vectors r_i and we observe similar effects as in example 3. At the 151st iteration Bi-CGSTAB was terminated with $\|r_i\|_2 \approx 10^{-8}$; at that point the true residual $\|b - Ax_i\|_2$ is virtually the same.

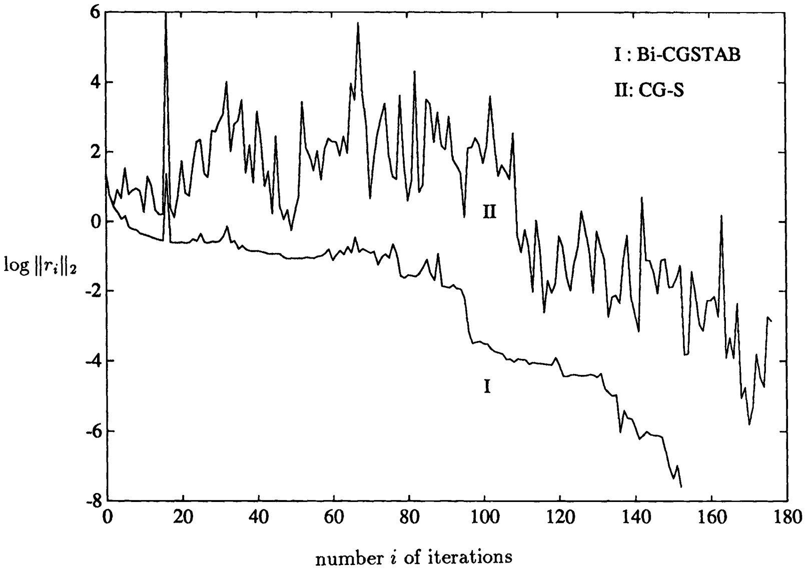


FIG. 3. *CG-S and Bi-CGSTAB for a nonsymmetric problem (example* 3).

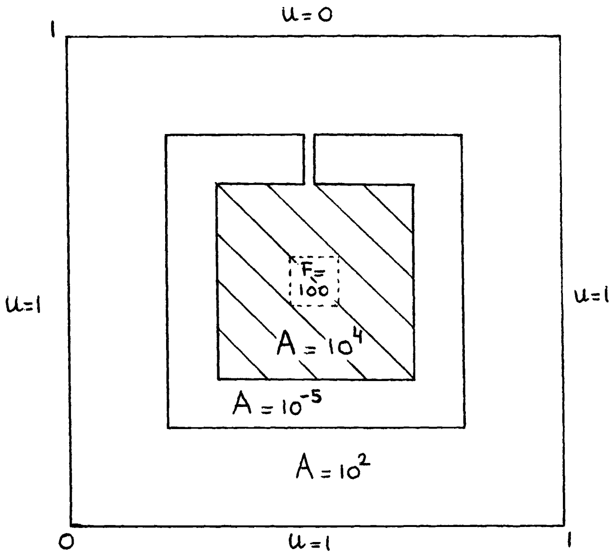


FIG. 4. *The coefficients for example* 4.

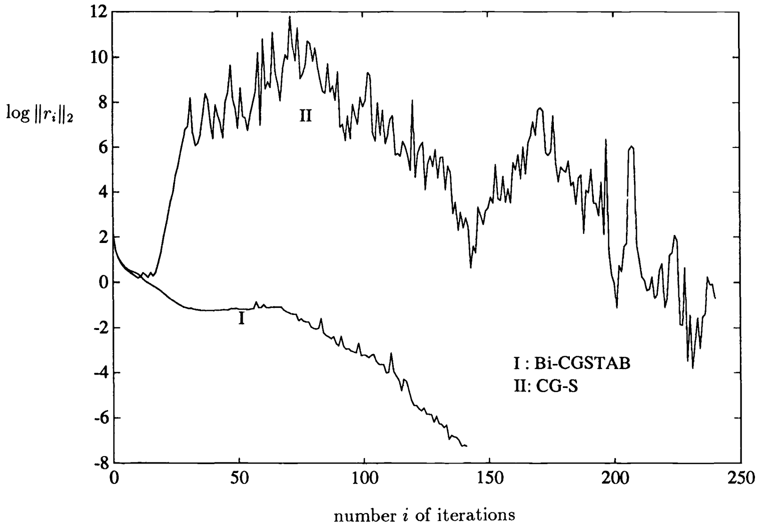


FIG. 5. *CG-S and Bi-CGSTAB for a nonsymmetric problem (example* 4).

It takes CG-S 382 iteration steps to obtain $\|r_i\|_2 \approx 10^{-8}$, but at that point the true residual $\|b - Ax_i\|_2$ is only about 10^{-4}. This means that in this case we have to iterate even further with CG-S, possibly after restarting the process, in order to get a result similar to that with Bi-CGSTAB.

For CG-S we have also checked what happens if we replace the *updated* vector r_i in the process by $r_i = b - Ax_i$ (see §2). It then turns out that both CG-S processes produce about the same residual vectors up to the 70th iteration step. But from then on the *updated* r_i process and the *true residual* process produce r_i vectors that differ more and more. After a while the *true residual* process delivers even much less accurate x_i vectors than Bi-CGSTAB. For example, at the 382nd iteration step the norm of the true residual $b - Ax_i$ in the *updated* r_i process is, as has been mentioned before, about 10^{-4}, whereas $\|b - Ax_i\|_2$ in the *true residual* process is about 10^2 for $i = 382$.

For this problem Bi-CG required 282 iteration steps to get the residual in norm below 10^{-8}. This example represents one of those rare examples in which CG-S does much worse than Bi-CG.

6. Conclusions. From many experiments we have learned that Bi-CGSTAB (and Bi-CGSTAB-P) is an attractive alternative for CG-S. Its convergence behaviour is much smoother so that it often produces much more accurate residual vectors (and, hence, more accurate solutions), and in most cases it converges considerably faster than CG-S.

The method has recently been tested for problems coming from semi-conductor device simulation and oil reservoir simulation and also in these circumstances our earlier findings have been confirmed (for many of these problems CG-S was the method of choice so far).

Therefore, we conclude that Bi-CGSTAB (with preconditioning, of course) is a very competitive method for solving relevant classes of nonsymmetric linear systems.

Acknowledgements. Peter Sonneveld (TU-Delft) suggested that I reconsider his old method IDR again, which turned out to be the key for Bi-CGSTAB, and I learned a lot from discussions with him.

I wish to thank Wolfgang Fichtner (ETH-Zürich), Koos Meijerink (Shell-Rijswijk), Marjan Driessen (Philips-Eindhoven), and Shun Doi (NEC-Kawasaki) for testing Bi-CGSTAB extensively on industrial problems and for reporting their results to me.

The referees have helped me to improve the presentation of this paper.

I further wish to express my thanks to Philips-Eindhoven for kindly permitting me access to their IBM 3090 system. Most of the numerical experiments, leading to understanding CG-S and to learning about the advantages of Bi-CGSTAB, have been carried out on that computer.

REFERENCES

[1] R. E. Bank, W. M. Coughran, M. A. Driscoll, W. Fichtner, and R. K. Smith, *Iterative methods in semiconductor device simulation*, Comput. Phys. Comm., 53 (1989), pp. 201–212.

[2] G. Brussino and V. Sonnad, *A comparison of direct and preconditioned iterative techniques for sparse unsymmetric systems of linear equations*, Internat. J. Numer. Methods Engrg., 28 (1989), pp. 801–815.

[3] R. Fletcher, *Conjugate gradient methods for indefinite systems*, Lecture Notes in Math., 506 (1976), pp. 73–89.

[4] I. Gustafsson, *A class of* 1*st order factorization methods*, BIT, 18 (1978), pp. 142–156.

[5] 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.

[6] G. Radicati di Brozolo and Y. Robert, *Parallel conjugate gradient-like algorithms for solving sparse non-symmetric systems on a vector multiprocessor*, Parallel Comput., 11 (1989), pp. 223–239.

[7] Y. Saad and M. H. Schultz, *GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.

[8] P. Sonneveld, *CGS: a fast Lanczos-type solver for nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 36–52.

[9] H. A. van der Vorst, *The convergence behaviour of preconditioned CG and CG-S in the presence of rounding errors*, Lecture Notes in Math., 1457 (1990), pp. 126–136.

[10] P. Wesseling and P. Sonneveld, *Numerical experiments with a multiple grid and a preconditioned Lanczos type method*, in Approximation Methods for Navier-Stokes Problems, Lecture Notes in Mathematics, R. Rautmann, ed., Springer-Verlag, Berlin, 1980.

[11] O. Widlund, *A Lanczos method for a class of nonsymmetric systems of linear equations*, SIAM J. Numer. Anal., 15 (1978), pp. 801–812.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 2, pp. 645–653, March 1992

013

TIMELY COMMUNICATION

Under the "timely communications" policy for the SIAM Journal on Scientific and Statistical Computing, papers that have significant timely content and do not exceed five pages automatically will be considered for a separate section of the journal with an accelerated reviewing process. It will be possible for the note to appear approximately six months after the date of acceptance.

A FAST REORDERING ALGORITHM FOR PARALLEL SPARSE TRIANGULAR SOLUTION*

ALEX POTHEN† AND FERNANDO L. ALVARADO‡

Abstract. A space-efficient partitioned representation of the inverse of a unit lower triangular matrix L may be used for efficiently solving sparse triangular systems on massively parallel computers. The number of steps required in the parallel triangular solution is equal to the number of subsets of elementary triangular matrices in the partitioned representation of the inverse. Alvarado and Schreiber have recently described two partitioning algorithms that compute the minimum number of subsets in the partition over all permutations of L which preserve the lower triangular structure of the matrix. Their algorithms require space linear and time nonlinear in the number of nonzeros in L. This paper describes a partitioning algorithm that requires only $\mathcal{O}(n)$ time and space for computing an optimal partition, when L is restricted to be a Cholesky factor. (Here n is the order of L.) The savings result from the observation that instead of working with the structure of L, it is sufficient to work with its transitive reduction, the elimination tree of L. Experimentally the new partitioning algorithm requires negligible time in comparison to the previous partitioning algorithms and to the Multiple-Minimum-Degree ordering algorithm.

Key words. directed acyclic graph, elimination tree, massively parallel computers, reordering algorithm, sparse Cholesky factorization, sparse triangular systems, transitive reduction

AMS(MOS) subject classifications. 65F50, 65F25, 68R10

1. The problem.

A unit lower triangular matrix L of order n can be expressed as a product of elementary lower triangular matrices $L = \prod_{i=1}^{n-1} L_i$, where $L_i = I + \underline{m}_i \, \underline{e}_i^{\,T}$, $\underline{m}_i$ has its first i components equal to zero, and $\underline{e}_i$ is the ith coordinate vector. Assume that L is sparse. Consider a representation of L in which the elementary lower triangular matrices are grouped together to form m unit lower triangular factors $L = \prod_{i=1}^{m} P_i$, where each factor P_i has the property that P_i^{-1} can be represented in the same space as P_i. We say that P_i is invertible in place. Here each $P_i = \prod_{k=e_i}^{e_{i+1}-1} L_k$, with $e_1 \equiv 1 < e_2 < \cdots < e_m < e_{m+1} \equiv n$. This leads to a partitioned representation of the inverse of L of the form $L^{-1} = \prod_{i=m}^{1} P_i^{-1}$, that can be stored in just the space required for L. This partitioned inverse representation is advantageous when a triangular system $L\underline{y} = \underline{b}$ must be solved repeatedly with different right-hand-side vectors $\underline{b}$ on a massively parallel computer such as the Connection Machine; on such a machine, the solution $\underline{y}$ can be computed as $\underline{y} := \prod_{i=m}^{1} P_i^{-1}\underline{b}$ in m steps. This representation has been considered by Alvarado et al. in the Power Engineering literature [4], [8], and by Alvarado and Schreiber [3]. It has been observed that a

* Received by the editors April 8, 1991; accepted for publication (in revised form) August 6, 1991.

† Department of Computer Science, Whitmore Laboratory, Pennsylvania State University, University Park, Pennsylvania 16802 (pothen@cs.psu.edu, na.pothen@na-net.ornl.gov). A part of this work was done while the author was visiting the Departments of Computer Science and Mathematics at the University of Wisconsin, Madison. The research of this author was supported by National Science Foundation grant CCR-9024954, by U. S. Department of Energy grant DE-FG02-91ER25095, and by U.S. Air Force Office of Scientific Research grant AFOSR-88-0161.

‡ Electrical and Computer Engineering Department, 1425 Johnson Dr., University of Wisconsin, Madison, Wisconsin 53706 (alvarado@ece.wisc.edu). The research of this author was supported by National Science Foundation grants ECS-8822654 and ECS-8907391.

reduction in the number of factors m is advantageous, and that this number can be reduced by permuting L into a new lower triangular matrix L_Π, and determining the partitioned representation of L_Π^{-1}.

Henceforth, without loss of generality, we will assume that L is irreducible. Alvarado and Schreiber [3] considered the following problem:

(P1) Given a unit lower triangular matrix $L = \prod_{i=1}^{n-1} L_i$, find a permutation $L_\Pi = \Pi L \Pi^T$ and a representation $L_\Pi = \prod_{i=1}^{m} P_i$, where

(1) each $P_i = \prod_{k=e_i}^{e_{i+1}-1} L_k$, with $e_1 = 1 < e_2 < \cdots e_m < e_{m+1} = n$,

(2) each P_i is invertible in place, and

(3) m is minimum over all permutations Π such that L_Π is lower triangular.

They designed two algorithms to solve (P1), which they called RP1 and RP2 in their paper. Both algorithms require time nonlinear in the number of nonzeros in L, and space proportional to the number of nonzeros in L.

In this paper, we consider the restriction of (P1) to unit lower triangular matrices, which arise in the LDL^T factorization of symmetric, positive definite matrices. (Henceforth we call this the Cholesky factorization.) Several applications of this problem in Power Engineering are described in [8]. We describe an $\mathcal{O}(n)$-time algorithm to compute the minimum number of factors in the partitioned inverse representation of L. The algorithm requires only the elimination tree and the number of nonzeros in each column of L as input, and *not* the nonzero structure of L. Thus the space requirement of the proposed algorithm is $\mathcal{O}(n)$. Further, since the elimination tree and the nonzero counts of the columns of L may be computed directly from the original matrix A, the space requirement of the overall algorithm to compute the factors in the partitioned inverse representation from A is $\mathcal{O}(n + \tau(A))$, where $\tau(A)$ is the number of nonzeros in the strict lower triangle of A.

In (P1), the action of the permutation Π on L is to reorder the elementary matrices whose product is L; however, these elementary matrices cannot be arbitrarily reordered, since we require the resulting matrix L_Π to be lower triangular. From the equation $L_i = I + \underline{m}_i \, \underline{e}_i{}^T$, it can be verified that the elementary matrices L_i and L_{i+1} can be permuted if and only if $l_{i+1,i} = 0$. These precedence constraints on the order in which the elementary matrices may appear is captured in a graph model of the problem.

2. A graph model.

Let $G(L) = (V, E)$ denote the directed graph with vertex set V equal to the set of columns of L, and an edge $(j, i) \in E$ (with $i > j$) if and only if $l_{i,j} \neq 0$. The edge (j, i) is directed from the lower-numbered vertex j to the higher-numbered vertex i. Hence $G(L)$ is a directed acyclic graph (DAG). Since we have assumed that L is irreducible, $G(L)$ is weakly connected, i.e., there is a path joining every pair of vertices in $G(L)$ when $G(L)$ is viewed as an undirected graph. If there is a directed path from a vertex j to a vertex i in $G(L)$, we will say that j is a *predecessor* of i, and that i is a *successor* of j. In particular, if $(j, i) \in E$, then j is a predecessor of i and i is a successor of j.

Given a subset P of the columns of L, the concept of the subgraph corresponding to the nonzeros in P will be useful in the remainder of this paper. Accordingly, we define the *subgraph of* $G(L)$ *induced by a set of vertices* P as the graph that contains all edges directed from vertices in P to all vertices in $G(L)$, and all the vertices that are the endpoints of such edges.

A *topological ordering* of $G(L)$ is an ordering of its vertices in which predecessors are numbered lower than successors; i.e., for every edge $(j, i) \in E$, $i > j$. By construction, the original vertex numbering of $G(L)$ is a topological ordering. A permutation

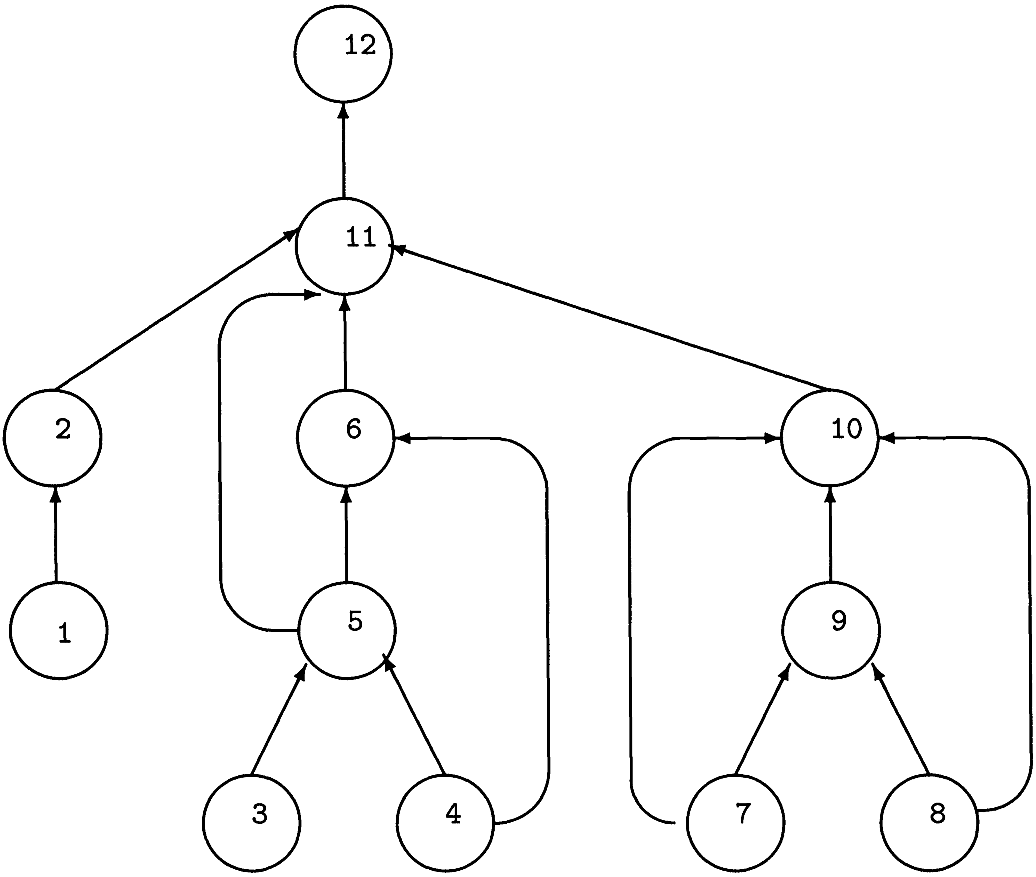


FIG. 1. *A directed acyclic graph $G(L)$ corresponding to a Cholesky factor L.*

Π that leaves L_{Π} lower triangular corresponds to a topological reordering of the vertices of $G(L)$. A topologically ordered DAG corresponding to a Cholesky factor L is shown in Fig. 1.

In what follows, we identify a subset of columns P with the factor formed by multiplying, in order of increasing column number, the elementary matrices corresponding to columns in P. The condition that the nonzero structure of a factor P should be the same as the structure of its inverse corresponds in the graph model to the requirement that the subgraph induced by P should be transitively closed [3], [9]. (A DAG G is *transitively closed* if for every pair of vertices j and i such that there is a directed path in G from j to i, the edge (j, i) is present in G.)

A graph model of (P1) is provided in the following problem:

(P2) Find an ordered partition $P_1 \prec P_2 \prec \cdots \prec P_m$ of the vertices $\{1, 2, \cdots, n-1\}$ of a topological ordering of $G(L)$ such that
(1) for every $v \in \{1, 2, \cdots, n-1\}$, if $v \in P_i$ then all predecessors of v belong to $P_1, \cdots, P_i$,
(2) the subgraph induced by each P_i is transitively closed, and
(3) m is minimum subject to the two conditions above.

The permutation Π in (P1) can be obtained by renumbering the vertices in the ordered partition P_1 to P_m in increasing order. The first condition follows from the fact that the factors are formed by grouping the elementary matrices of L_Π in order from lowest-numbered to highest-numbered. The other conditions follow from the discussion in the preceding paragraphs. The graph model (P2) is not explicitly stated in Alvarado and Schreiber [3], although it is implicit in the description of their algorithm RP1.

3. Cholesky factorization. Now we consider the restriction of (P1) to Cholesky factors. Then the graph $G(L)$, viewed as an undirected graph, is a chordal graph. The gist of this section is that the chordality of $G(L)$ simplifies the problem a great deal, and enables the design of an $\mathcal{O}(n)$ algorithm for computing the partition, whereas previous algorithms [3] required time nonlinear in the number of edges of $G(L)$. The savings result from the fact that it suffices to consider the transitive reduction of $G(L)$, the elimination tree of L, instead of all the edges in $G(L)$.

The *elimination tree* of L is a directed tree $T = (V, E_T)$, whose vertices are the columns of L, with a directed edge $(j,i) \in E_T$ if and only if the lowest-numbered row index of a subdiagonal nonzero in the jth column of L is i. (The edge is directed from j to i.) The vertex i is the *parent* of j, and j is a *child* of i.

We define the *higher adjacency set* $hadj(j)$ to be the set of all vertices k adjacent to j in $G(L)$ such that k is numbered higher than j. If (j,i) is an edge in the elimination tree, the lowest-numbered vertex in $hadj(j)$ is i. The reader can verify that the elimination tree of the graph $G(L)$ in Fig. 1 is obtained by omitting the edges $(4,6)$, $(5,11)$, $(7,10)$, and $(8,10)$ from the graph.

A comprehensive survey of the role of elimination trees in sparse Cholesky factorization has been provided by Liu [13]. We will assume some knowledge of the properties of elimination trees, and in particular, the following result will be useful.

LEMMA 3.1. *If v is the parent of a vertex u in the elimination tree T, then* $hadj(u) \subseteq \{v\} \cup hadj(v)$. □

Our partitioning algorithm will require as input the elimination tree with vertices numbered in a topological ordering. It also requires the subdiagonal nonzero counts of each column of L, stored in an array $hd(v)$. The algorithm uses a variable *level* to partition the vertices; $level(v) = l$ implies that v belongs to the set P_l.

The idea of the algorithm is as follows. It examines the vertices of the elimination tree in increasing order. If a vertex v is a leaf of the tree, then it is included in the first *level*, which constitutes the vertices in P_1. Otherwise, it divides the children of v into two sets: C_1 is the subset of the children u such that the subgraph of $G(L)$ induced by u and v is transitively closed, and C_2 denotes the subset of the remaining children. Let l_1 denote the maximum *level* of a child in C_1 and l_2 denote the maximum *level* of a child in C_2. Set $l_i = 0$ if $C_i = \emptyset$. If C_1 is empty, or if $l_1 \leq l_2$, then v cannot be included in the same *level* as any of its children, and hence begins a new *level* $(l_2 + 1)$. Otherwise, $l_1 > l_2$, and v can be included together with some child $u \in C_1$ such that $level(u) = l_1$.

We now describe the details of an implementation. The vertices of the elimination tree are numbered in a topological ordering from 1 to n. The descendant relationships in the elimination tree are represented by two arrays of length n, *child* and *sibling*. $child(v)$ represents the first child of v, and $sibling(v)$ represents the right sibling of v, where the children of each vertex are ordered arbitrarily. If $child(v) = 0$, then v has no child and is a leaf of the elimination tree; if $sibling(v) = 0$, then v has no right sibling. The array $hd(.)$, also of length n, contains the *higher degree* of a vertex v (equal to $|hadj(v)|$). Our partitioning algorithm, Algorithm RPtree, is shown in Fig. 2. The reader can verify that $P_1 = \{1,3,4,7,8,9\}$, $P_2 = \{2,5,6,10\}$, and $P_3 = \{11\}$ for the

```
for $v := 1$ to $n \rightarrow$
    if $child(v) = 0$ then {$v$ is a leaf}
        $level(v) := 1$;
    else {$v$ is not a leaf}
        $u := child(v)$; $l_1 := 0$; $l_2 := 0$;
        while $u \neq 0$ do
            if $hd(u) = 1 + hd(v)$ then
                $l_1 := \max\{l_1, level(u)\}$;
            else {$hd(u) < 1 + hd(v)$}
                $l_2 := \max\{l_2, level(u)\}$;
            fi
            $u := sibling(u)$;
        od
        if $l_1 \leq l_2$ then {$v$ begins a new level}
            $level(v) := l_2 + 1$;
        else {$l_1 > l_2$, $v$ can be included in level $l_1$ }
            $level(v) := l_1$;
        fi
    fi
rof
```

FIG. 2. *Algorithm RPtree.*

graph in Fig. 1.

The complexity of the algorithm is easily analyzed. For a given vertex v, we examine all of its children, and the operations associated with examining a child u can be performed in constant time. If we charge the cost of examining a child u of v to u, then each vertex in the elimination tree is charged at most once, since each child has a unique parent. Thus the time complexity of the algorithm is $\mathcal{O}(n)$. The space complexity is also $\mathcal{O}(n)$, since the elimination tree, the higher degrees, and the *level* information are all stored using arrays of length n.

4. Correctness of the algorithm.

We now prove that Algorithm RPtree correctly solves (P1).

THEOREM 4.1. *Algorithm RPtree correctly solves* (P1) *when L is the unit lower triangular matrix in the LDL^T factorization of a symmetric, positive definite matrix.*

Proof. We consider problem (P2), the graph model of (P1), and the DAG $G(L)$, which viewed as an undirected graph is chordal. We will show that the ordered partition obtained by the algorithm satisfies the three conditions in (P2).

First we show that a vertex $v \in P_i$ only if all predecessors of v belong to $P_1, \cdots, P_i$. Since the elimination tree T is the transitive reduction of $G(L)$, any predecessor of v in the latter graph is a predecessor of v in T as well. Thus it suffices to consider the descendants of v in the elimination tree. Further, since the vertices in the elimination tree are topologically ordered, and the algorithm assigns *level* values to the vertices in increasing order, it suffices to consider the children of v in T. Finally, since the algorithm assigns *level* values to a vertex v such that

$$level(v) \geq \max\{level(u) : u \text{ is a child of } v\},$$

the result is true.

Second, we show that the subgraph induced by the vertices in a *level* is transitively closed. In the preceding paragraph, it was seen that *level* values are nondecreasing along any path from a leaf to the root on the elimination tree. Hence it follows that the vertices included in a *level* l by Algorithm RPtree can be expressed as the union (not necessarily disjoint) of certain paths in the elimination tree, where vertices on a path are constrained to have *level* values equal to l. Let u_0, u_1, $\cdots$, u_p be such a path, which is maximal with respect to the property of having the same *level* value; then since the algorithm includes all these vertices in a single *level*,

$$hd(u_0) = 1 + hd(u_1) = 2 + hd(u_2) = \cdots = p + hd(u_p).$$

It follows from Lemma 3.1 that

$$hadj(u_0) = \{u_1\} \cup hadj(u_1) = \cdots = \{u_1, \ldots, u_p\} \cup hadj(u_p).$$

Hence the subgraph induced by the vertices in this path is transitively closed. Since the set of vertices in a *level* is the union of such paths, the result now follows.

It remains to show that m is the minimum number of ordered sets in the partition of $G(L)$ subject to the above conditions. We prove the result by induction on $(n-1)$, the number of vertices to be partitioned. The base case when this number is one is trivial. Assume inductively that Algorithm RPtree optimally partitions all chordal graphs with at most $n-1$ vertices (hence there are at most $n-2$ vertices to be partitioned).

Consider the partition $P_1 \prec P_2 \prec \cdots \prec P_m$ with m levels obtained by the algorithm on a chordal graph $G(L)$ with n vertices. Let T denote the elimination tree of $G(L)$ with vertices in a topological ordering. As shown in the proof of the second condition, P_1 consists of the union of vertices on certain paths on the elimination tree T, each path beginning from a leaf and ending at a vertex w such that *level*(w) is one, and *parent*(w) has *level* greater than one. (Thus these paths are maximal with respect to the condition that their *level* values are equal to one.) Let $u_0 < u_1 \ldots < u_p$ be such a path, and denote by u_{p+1} the *parent* of u_p in T.

The fact that the algorithm did not include u_{p+1} in P_1 together with u_p could be due to one of two reasons. If u_{p+1} has some child x with $level(x) > level(u_p) = 1$, then the inclusion of u_{p+1} in P_1 would destroy the first condition of problem (P2). Otherwise, all the children of u_{p+1} belong to P_1. Now the algorithm would not include u_{p+1} in P_1 only if some child x satisfied $hd(x) < 1 + hd(u_{p+1})$. If this were the case, there is some vertex $w \in hadj(u_{p+1}) \setminus hadj(x)$. Thus the inclusion of u_{p+1} into P_1 would destroy the property that the subgraph induced by P_1 is transitively closed. Thus P_1 contains the maximum number of vertices possible from the vertices in $G(L)$.

Now consider the set of vertices Q which consists of those vertices of $G(L)$ not included in P_1. Then the induced subgraph $G(Q)$ has fewer than $n-1$ vertices, and by the inductive hypothesis, is partitioned optimally by Algorithm RPtree into $m-1$ ordered sets. Let T_Q denote the elimination tree of the subgraph $G(Q)$ obtained by removing vertices in P_1 from the elimination tree T of $G(L)$. From the preceding paragraph, since vertices in P_1 cannot be included in a *level* which contains the leaf vertices of T_Q, it follows that any partition of $G(L)$ requires at least $(m-1)+1 = m$ levels. This completes the proof. □

5. Experimental results. We implemented Algorithm RPtree and compared its performance with the RP1 and RP2 algorithms of Alvarado and Schreiber on eleven problems from the Boeing–Harwell collection [6]. All the algorithms were implemented in C, within Alvarado's Sparse Matrix Manipulation System [1]. Each problem was initially ordered using the Multiple-Minimum-Degree ordering of Liu [12], and the structure of the resulting lower triangular factor L was computed. We call this the primary ordering step. Then Algorithms RP1, RP2, or RPtree were used in a secondary ordering step to reorder the structure of L to obtain the minimum number of partitions over reorderings that preserve the DAG $G(L)$. All three algorithms lead to the same number of levels in the partition since they solve the same problem.

The experiments were performed on a Sun SPARCstation IPC with 24 Mbytes of main memory and a 100 Mbyte swap space running SunOS 4.1 version of the Unix operating system. The unoptimized standard system compiler was used to compile the code. Recall that $\tau(A)$ is the number of nonzeros in the strict lower triangle of A; $\tau(L)$ is similarly defined. We scale these numbers by a thousand for convenience. In Table 1, we report the scaled values of $\tau(A)$ and $\tau(L)$, the CPU times taken by the primary and secondary ordering algorithms (in seconds), and the height of the elimination tree obtained from the primary ordering. (The fill and the etree height reported here are somewhat different from previously published values for the MMD ordering because of our use of SMMS. In SMMS, the problem datum is first converted to an element list from the Boeing–Harwell format before it is stored using sparse matrix data structures. This changes the initial matrix ordering which is input to the MMD algorithm, with the consequent change in the fill and etree height.)

Table 1 also reports the number of factors in the partitioned inverse of L. The number in the column "levels(new)" corresponds to the number of factors in the solution of the problem (P1), i.e., in the partition of the permuted matrix L_Π. The number in the column "levels(orig)" indicates the number of factors obtained when the unpermuted Cholesky factor L is partitioned. In the graph model (P2), this corresponds to replacing the first condition with the stricter condition:

For every $v \in \{1, \cdots, n-1\}$, if $v \in P_i$, then all vertices numbered lower than v belong to $P_1, \cdots, P_i$.

TABLE 1

Comparison of execution times on a Sun SPARC*station* IPC *for three secondary reordering schemes with the* MMD *primary ordering. The parameters $\tau(A)$ and $\tau(L)$ have been scaled by a thousand for convenience.*

	Original data		MMD		Etree	CPU time (sec)			Levels	
Problem	n	$\tau(A)$	Time	$\tau(L)$	Height	RP1	RP2	RPtree	Orig	New
BCSPWR10	5,300	8.27	1.72	23.2	128	1.07	1.26	0.10	70	32
BCSSTK13	2,003	40.9	4.74	264	654	61.1	22.1	0.05	53	24
BCSSTM13	2,003	9.97	1.12	42.6	261	5.08	2.63	0.03	25	16
BLCKHOLE	2,132	6.37	0.73	53.8	224	3.15	2.58	0.05	24	15
CAN1072	1,072	5.69	0.72	19.4	151	0.78	0.92	0.02	21	16
DWT2680	2,680	11.2	1.82	49.9	371	2.43	2.45	0.05	50	36
LSHP3466	3,466	10.2	1.03	81.2	341	4.48	4.14	0.07	37	25
NASA1824	1,824	18.7	1.42	72.2	259	6.01	3.88	0.03	34	16
NASA4704	4,704	50.0	3.92	275	553	33.8	16.1	0.12	41	17
39x39 9pt	1,521	10.9	0.50	31.6	185	1.35	1.50	0.02	19	15
79x79 9pt	6,241	45.9	2.17	190	429	12.7	11.4	0.12	30	23

Alvarado and Schreiber [2] have shown that when the partitioned inverse is employed on a massively parallel computer such as the CM-2, the number of levels in the partitioned inverse representation determines the complexity of parallel triangular solution. On the other hand, the complexity of a conventional triangular solution algorithm is governed by the height of the elimination tree. Table 1 shows both these quantities, and it is seen that the number of levels in the partitioned inverse is many times smaller (by a factor of sixteen on the average) than the elimination tree height. Hence the use of the partitioned inverse leads to much faster parallel triangular system solution on massively parallel computers.

An interesting feature in these results is that the number of levels seems to be only weakly dependent on the problem, the order of A, or the number of nonzeros in A or L. This number is between ten and forty and is significantly smaller than the order of the matrix A for most of these problems. If this observation holds true for larger instances of a wide collection of problems, then it will have a significant impact on the application of the ideas in this paper to parallel computing. For the $k \times k$ model grid problem ordered by the optimal nested dissection ordering, the height of the elimination tree is $3k + \Theta(1)$, while the number of levels is $2 \log_2 k + \Theta(1)$.

Algorithm RPtree has $\mathcal{O}(n)$ time complexity while RP1 and RP2 are both nonlinear in the number of nonzeros in L. This is confirmed by the experiments: on the average problem in this test set, RPtree is more than *a hundred* times faster than RP1 or RP2, and the advantage increases with increasing problem size. From a practical perspective, the time needed by Algorithm RPtree is quite small when compared to the cost of computing the initial MMD ordering. This is not true of either the RP1 or the RP2 algorithm. An equally important advantage of Algorithm RPtree is that it requires only $\mathcal{O}(n)$ additional space, whereas both RP1 and RP2 require $\mathcal{O}(\tau(L))$ additional space.

We have also experimented with a variant of the Minimum-Length-Minimum-Degree (MLMD) ordering [5] as the primary ordering, but we do not report detailed results because Timely Communications are by definition brief. The MLMD ordering incurs a great deal more fill in L than the MMD algorithm, and its current, fairly straightforward implementation is quite slow compared to the MMD algorithm. We believe an implementation comparable in sophistication to the MMD algorithm should not be significantly slower than MMD, and may also reduce the number of fills. In spite of the larger number of fills, the MLMD ordering is more effective in almost all cases than MMD in reducing the number of levels in the partition of both L and L_{Π}. In some cases, the initial number of levels obtained when MLMD is used as the primary ordering is lower than the final number of levels obtained with MMD after the secondary reordering.

When the MLMD ordering is used, Algorithm RPtree has an even greater time advantage over the RP1 and RP2 algorithms, since the former works with the elimination tree, while the latter algorithms require the structure of the Cholesky factor. For instance, on BCSSTK13, RP2 takes 461 seconds, while RPtree requires only 0.07 seconds.

6. Extensions.

There are two directions in which the results in this paper may be extended.

Given the factorization $A = LDL^T$ of a symmetric, positive definite matrix, consider the filled matrix $F = L + L^T$ and the corresponding chordal undirected graph $G_u(F)$. In problem (P3) we ask for the minimum number of factors m in the partitioned inverse representation of L over all vertex orderings that preserve the structure of the filled graph $G_u(F)$ (rather than the DAG $G(L)$ and the corresponding elimination tree of L as (P2) does). A solution to this problem would reduce the number of factors in the partitioned inverse over the number required in (P2). Such an ordering would have to be applied to the original matrix A, *before* the computation of the factorization. This problem turns out to be much harder than (P2), and many subtle issues are involved in the solution of this problem. It can be solved by a *generalization* of the Jess and Kees ordering [11]. (However, the Jess and Kees ordering by itself will not work.) An efficient algorithm to solve (P3) that makes use of the clique tree data structure will be reported in [14]. It is heartening that the above ordering is also appropriate for efficiently computing the factorization in parallel on massively parallel machines.

The ideas in this paper can also be applied to the general unsymmetric problem to obtain a more efficient partitioning algorithm than RP2. It turns out that the transitive reduction of the directed graph $G(L)$ could be used instead of $G(L)$ at several places in the RP2 algorithm. It is necessary to implement this idea to see the net computational savings the use of transitive reduction may bring in this context. Other applications of transitive reduction in unsymmetric sparse factorizations have been recently considered by Eisenstat, Gilbert, and Liu [7], [10].

REFERENCES

[1] F. L. ALVARADO, *Manipulation and visualization of sparse matrices*, ORSA J. Comput., 2 (1990), pp. 180–207.

[2] F. L. ALVARADO AND R. SCHREIBER, *Fast parallel solution of sparse triangular systems*, 13th IMACS World Congress on Computation and Applied Mathematics, Dublin, July 1991.

[3] ———, *Optimal parallel solution of sparse triangular systems*, SIAM J. Sci. Statist. Comput., 13 (1992), to appear.

[4] F. L. ALVARADO, D. C. YU, AND R. BETANCOURT, *Partitioned sparse A^{-1} methods*, IEEE Trans. Power Systems, 5 (1990), pp. 452–459.

[5] R. BETANCOURT, *An efficient heuristic ordering algorithm for partial matrix refactorization*, IEEE Trans. Power Systems, 3 (1988), pp. 1181–1187.

[6] I. S. DUFF, R. G. GRIMES, AND J. G. LEWIS, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14.

[7] S. C. EISENSTAT AND J. W.-H. LIU, *Exploiting structural symmetry in unsymmetric sparse symbolic factorizations*, Tech. Report 90-12, Computer Science, York University, North York, Ontario, Canada, 1990.

[8] M. K. ENNS, W. F. TINNEY, AND F. L. ALVARADO, *Sparse matrix inverse factors*, IEEE Trans. Power Systems, 5 (1990), pp. 466–472.

[9] J. R. GILBERT, *Predicting structure in sparse matrix computations*, Tech. Report 86-750, Computer Science, Cornell University, Ithaca, NY, 1986.

[10] J. R. GILBERT AND J. W-H. LIU, *Elimination structures for unsymmetric sparse LU factors*, Tech. Report 90-11, Computer Science, York University, North York, Ontario, Canada, 1990.

[11] J. G. LEWIS, B. W. PEYTON, AND A. POTHEN, *A fast algorithm for reordering sparse matrices for parallel factorization*, SIAM J. Sci. Statist. Comput., 6 (1989), pp. 1146–1173.

[12] J. W.-H. LIU, *Modification of the minimum-degree algorithm by multiple elimination*, ACM Trans. Math. Software, 11 (1985), pp. 141–153.

[13] ———, *The role of elimination trees in sparse factorization*, SIAM J. Matrix. Anal. Appl., 11 (1990), pp. 134–172.

[14] A. POTHEN AND X. YUAN, *A clique tree algorithm for optimally reordering sparse Cholesky factors for parallel triangular solution*, work in preparation, 1991.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 3, pp. 655-665, May 1992

001

COMPUTATION OF CONSTANTS IN THE STRENGTHENED CAUCHY INEQUALITY FOR ELLIPTIC BILINEAR FORMS WITH ANISOTROPY*

PANAYOT S. VASSILEVSKI† AND MAYA H. ETOVA‡

Abstract. Numerical results are presented that give dependence of the constants in the strengthened Cauchy inequality, which is a basic tool for constructing two-level and multilevel preconditioning matrices on the anisotropy of second-order elliptic bilinear forms for a number of finite element piecewise polynomial spaces over triangular and rectangular elements.

Key words. strengthened Cauchy inequality, second-order elliptic problems, anisotropy, finite elements

AMS(MOS) subject classifications. 65N20, 65N30

1. Introduction. Consider the following elliptic bilinear form:

$$a(u, \phi) = \int_\Omega (\nabla u)^T \begin{pmatrix} \varepsilon & \delta \\ \delta & 1 \end{pmatrix} (\nabla \phi),$$

where $\varepsilon > 0$ and $\delta \geqq 0$ are parameters, $\varepsilon > \delta^2$, and Ω is, say, a two-dimensional polygon. By ∇u we denote $(\partial u/\partial x, \partial u/\partial y)^T$.

We consider two nested piecewise polynomial finite element spaces

$$\tilde{\mathbf{V}} \subset \mathbf{V}.$$

Let the corresponding triangulations of Ω be $\tilde{\tau}$ and τ, with τ obtained from $\tilde{\tau}$ by uniform refinement; that is, the elements of τ are obtained by dividing the elements of $\tilde{\tau}$ into a fixed number of congruent parts. We denote the nodes in $\tilde{\tau}$ and τ by $\tilde{N}$ and N, respectively. We have by construction that $\tilde{N} \subset N$.

It is well known (cf. Bank and Dupont [5]; Axelsson and Gustafsson [2]; Axelsson [1]; Braess [6], [7]; and Maitre and Musy [9]) that the following strengthened Cauchy (or Cauchy-Bunyakowskii-Schwarz) inequality is valid: There exists a constant $\gamma \in (0, 1)$ independent of the mesh parameter $\tilde{h}$ of $\tilde{\tau}$, such that

$$a(\phi, \tilde{\phi}) \leqq \gamma (a(\phi, \phi))^{1/2} (a(\tilde{\phi}, \tilde{\phi}))^{1/2} \quad \text{for all } \tilde{\phi} \in \tilde{\mathbf{V}}, \tag{1.1}$$
$$\text{and } \phi \in \mathbf{V}, \quad \phi(x) = 0, \quad x \in \tilde{N}.$$

As shown in Maitre and Musy [9] (see also Axelsson [1] or Eijkhout and Vassilevski [8]), this inequality can be verified element by element by first deriving inequalities of the form

$$a_{\tilde{E}}(\phi, \tilde{\phi}) \leqq \gamma_{\tilde{E}} (a_{\tilde{E}}(\phi, \phi))^{1/2} (a_{\tilde{E}}(\tilde{\phi}, \tilde{\phi}))^{1/2} \quad \text{for all } \tilde{\phi} \in \tilde{\mathbf{V}},$$
$$\text{and } \phi \in \mathbf{V}, \quad \phi(x) = 0, \quad x \in \tilde{N} \cap \tilde{E},$$

* Received by the editors September 17, 1990; accepted for publication January 16, 1991. This research was supported in part by the Bulgarian Committee for Science, under grant 55-26-03-87, and by National Science Foundation grant INT-8914472.

† Department of Mathematics, University of Wyoming, P.O. Box 3036, University Station, Laramie, Wyoming 82071-3036 (PANAYOT@CORRAL.UWYO.EDU). This work was performed at the University of Wyoming, while the author was on leave from the Center of Computer Science and Technology, Bulgarian Academy of Sciences, "Acad. G. Bontchev" Street, Block 25A, 1113 Sofia, Bulgaria.

‡ Institute of Mathematics, Bulgarian Academy of Sciences, "Acad. G. Bontchev" Street, Block 8, 1113 Sofia, Bulgaria.

for each element $\tilde{E} \in \tilde{\tau}$. Here

$$a_{\tilde{E}}(\phi, \tilde{\phi}) = \int_{\tilde{E}} (\nabla \phi)^T \begin{pmatrix} \varepsilon & \delta \\ \delta & 1 \end{pmatrix} (\nabla \tilde{\phi})\, dx. \tag{1.2}$$

Then γ is defined as

$$\gamma = \max_{\tilde{E} \in \tilde{\tau}} \gamma_{\tilde{E}}.$$

It is clear then that γ is independent of the grid parameter $\tilde{h}$ and on possible jumps of the coefficients, in the case of more general elliptic form

$$a(u, \phi) = \int_{\Omega} (\nabla u)^T \begin{pmatrix} a_{11}(x) & a_{12}(x) \\ a_{21}(x) & a_{22}(x) \end{pmatrix} (\nabla \phi)\, dx,$$

if they are continuous or, say, constants within each element from the initial triangulation $\tilde{\tau}$. In this case, when we have piecewise constant coefficients, the constant γ can be shown to be independent of the number of refinement levels, since by construction we have that γ depends on the form of the element $\tilde{E}$ but not on its size. In other words, if we keep the elements at every discretization level geometrically similar to a fixed set of elements and the piecewise polynomials are of fixed degree throughout the refinement, it is clear that γ will remain bounded away from unity uniformly with respect to the number of discretization levels used (cf. Maitre and Musy [9], Eijkhout and Vassilevski [8]).

The purpose of this communication is to supply some computational information about the dependence of γ on possible anisotropy and mixed derivatives in the considered second-order elliptic bilinear form. Such information is useful since the above-defined constant γ is needed explicitly for the construction of the recently proposed algebraic multilevel iterative methods in Axelsson and Vassilevski [3], [4]. For the case of coefficient matrix

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

that is, for diffusion problems, it has been demonstrated in [9] that for arbitrary initial triangles and piecewise linear polynomials the above constant γ is strictly less than $\frac{3}{4}$ and tends to $\frac{3}{4}$ if some of the triangles degenerate. For the algebraic multilevel methods proposed in [3], [4], there is an important inequality upon which the optimality of the methods are valid: $\nu > 1/\sqrt{1-\gamma^2}$, where ν is the degree of a properly scaled and shifted Chebyshev polynomial involved in the construction of the multilevel preconditioning matrices. For details, see [3], [4]. We see that for the above-mentioned example of diffusion problems, this inequality is satisfied with $\nu = 2$. We cannot choose ν, the degree of the Chebyshev polynomials, very large since this causes the computational labor per iteration, that is, solving a system of equations defined by the multilevel preconditioning matrix becomes too costly. For two-dimensional problems, we consider uniform refinement, that is, when we partition the elements at a given discretization level into four congruent ones we have the restriction $\nu < 4$ in order to ensure that the arithmetic operations needed for solving a system with the multilevel preconditioning matrix are proportional to the number of unknowns on the finest discretization level. In other words, we have to satisfy the following inequalities:

$$4 > \nu > 1/\sqrt{1-\gamma^2},$$

or since ν can be 2 or 3, the inequalities:

$$\begin{aligned} &\text{for } \nu=2, \quad \gamma<\gamma^{(2)}=\sqrt{3}/2=0.8660\cdots, \\ &\text{for } \nu=3, \quad \gamma<\gamma^{(3)}=2\sqrt{2}/3=0.9428\cdots. \end{aligned} \tag{1.3}$$

The remainder of the paper is organized as follows. In the next section we briefly describe the algorithm for the computation of local γ-constants. In § 3 the computed constants γ for linear and quadratic piecewise polynomials on triangular elements and for bilinear, quadratic, and cubic serendipity piecewise polynomials on square elements are shown. Finally some conclusions are drawn.

A main observation from the presented numerical results is that increasing the degree of the piecewise polynomials used can cause severe deterioration of γ constants in certain cases of anisotropy, especially in the presence of mixed derivatives in the considered elliptic bilinear form. However, in most of the cases we have constants γ that satisfy the required inequality at least for $\nu=3$.

2. Computation of γ-constants. Consider the local bilinear form (1.2). In Figs. 1(a) and 1(b) a refined triangular element with linear and quadratic piecewise polynomials, respectively, are shown.

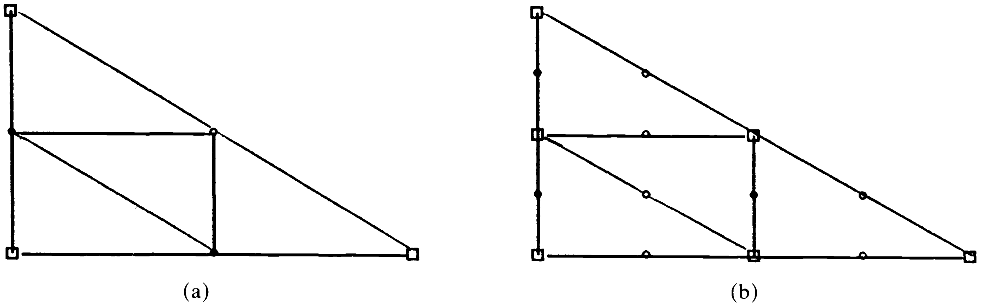


FIG. 1. *A refined triangular element.* (a) *Piecewise linear polynomials.* (b) *Piecewise quadratic polynomials.*

The nodes (degrees of freedom) are denoted by $\tilde{N}$ and N_1, where $\tilde{N}$ are the nodes from the coarser triangulation and N_1 the nodes added after the refinement. In Figs. 1(a) and (b) the $\tilde{N}$ nodes are denoted by "□," whereas the N_1 nodes are denoted by "○." Similarly, in Figs. 2(a)-(c) refined rectangular elements with piecewise bilinear polynomials (Fig. 2(a)), piecewise quadratic serendipity polynomials (Fig. 2(b)), and piecewise cubic serendipity polynomials (Fig. 2(c)) are shown. Here the splitting of the nodes (degrees of freedom) into two sets $\tilde{N}$ (the nodes from the coarser grid) and N_1 (those added in the refinement nodes) is also denoted by "□" and "○," respectively.

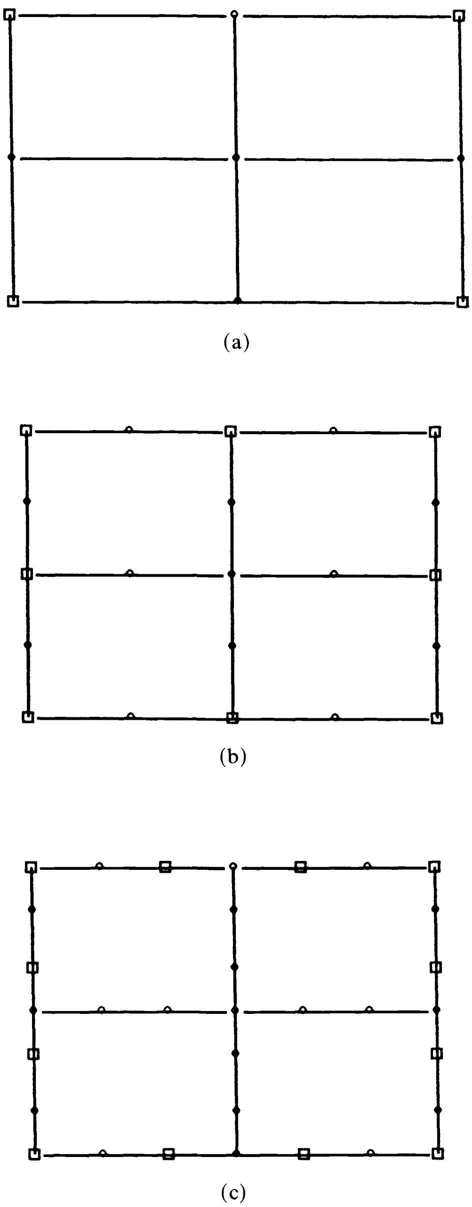
Below we list the nodal basis functions for a reference element $\hat{E}$: a triangle with baricentric coordinate functions λ_1, λ_2, and λ_3, and the square $(-1,1)\times(-1,1)$.

Consider first the triangular element $\hat{E}$ with vertices $A_1=(0,0)$, $A_2=(1,0)$, and $A_3=(X, Y)$, and angles α and β. We have

$$X=\cot\alpha/(\cot\alpha+\cot\beta), \qquad Y=1/(\cot\alpha+\cot\beta).$$

Linear polynomials:

$$\phi_i=\lambda_1, \qquad i=1,2,3,$$



(a)

(b)

(c)

FIG. 2. (a) *Refined rectangular element with piecewise bilinear polynomials.* (b) *Refined rectangular element with piecewise quadratic serendipity polynomials.* (c) *Refined rectangular element with piecewise cubic serendipity polynomials.*

where

$$\lambda_i = (a_i + b_i x + c_i y)/(2S).$$

S is the area of $\hat{E}$, that is, $S = \sin\alpha \sin\beta/(2\sin(\alpha+\beta))$. If we denote the coordinates of the vertices A_i by (x_i, y_i), the coefficients of the baricentric functions λ_i are given by

$$a_i = x_j y_k - x_k y_j, \quad b_i = y_j - y_k, \quad c_i = x_k - x_j,$$

where the triple (i, j, k) is the corresponding transposition of $(1, 2, 3)$. In our case we obtain

$$\begin{array}{lll} a_1 = 1/(a+b), & b_1 = -1/(a+b), & c_1 = -b/(a+b), \\ a_2 = 0, & b_2 = 1/(a+b), & c_2 = -a/(a+b), \\ a_3 = 0, & b_3 = 0, & c_3 = 1, \end{array}$$

with $a = \cot\alpha$, $b = \cot\beta$.

Quadratic polynomials are

$$\begin{aligned} \phi_1 &= \lambda_1(2\lambda_1 - 1), \\ \phi_2 &= 4\lambda_1\lambda_2, \\ \phi_3 &= \lambda_2(2\lambda_2 - 1), \\ \phi_4 &= 4\lambda_2\lambda_3, \\ \phi_5 &= \lambda_3(2\lambda_3 - 1), \\ \phi_6 &= 4\lambda_1\lambda_3. \end{aligned}$$

Consider now the rectangular element $\tilde{E} = (-1, 1) \times (-1, 1)$.

Bilinear polynomials:

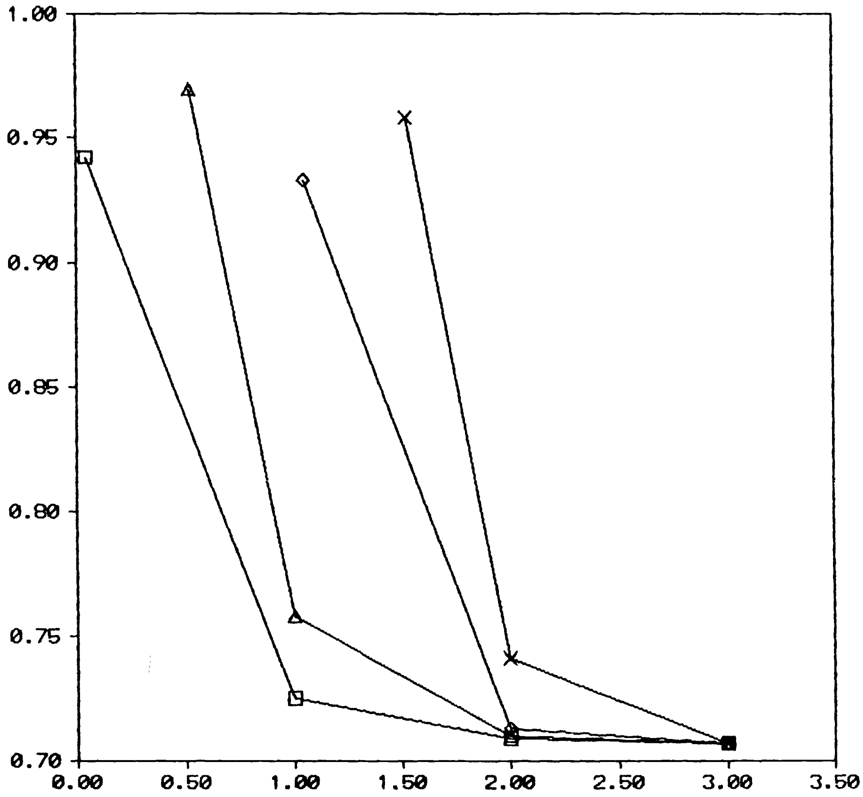
$$\begin{aligned} \phi_1 &= \tfrac{1}{4}(1-x)(1-y), \\ \phi_2 &= \tfrac{1}{4}(1+x)(1-y), \\ \phi_3 &= \tfrac{1}{4}(1+x)(1+y), \\ \phi_5 &= \tfrac{1}{4}(1-x)(1+y). \end{aligned}$$

Quadratic serendipity polynomials:

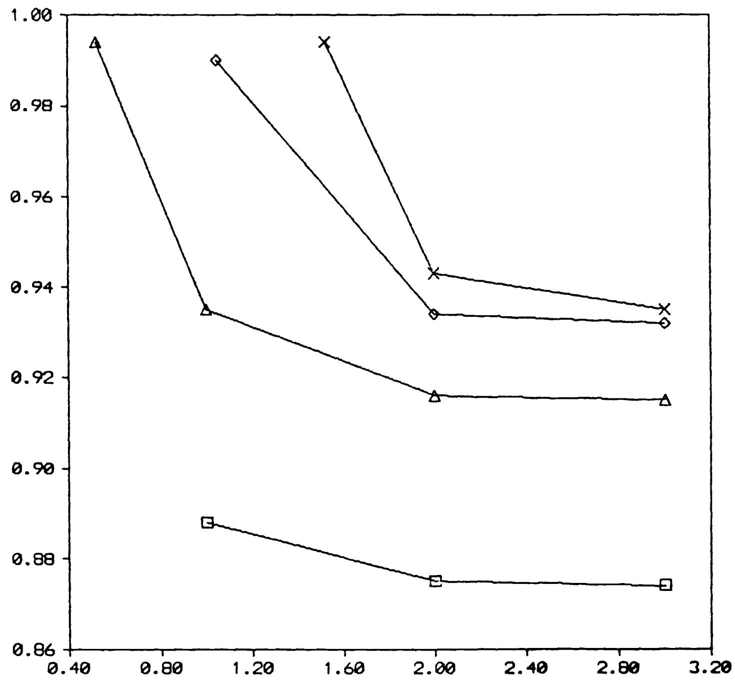
$$\begin{aligned} \phi_1 &= -\tfrac{1}{4}(1-x)(1-y)(1+x+y), \\ \phi_2 &= \tfrac{1}{2}(1-x^2)(1-y), \\ \phi_3 &= \tfrac{1}{4}(1+x)(1-y)(x-y-1), \\ \phi_4 &= \tfrac{1}{2}(1+x)(1-y^2), \\ \phi_5 &= \tfrac{1}{4}(1+x)(1+y)(x+y-1), \\ \phi_6 &= \tfrac{1}{2}(1-x^2)(1+y), \\ \phi_7 &= -\tfrac{1}{4}(1-x)(1+y)(x-y+1), \\ \phi_8 &= \tfrac{1}{2}(1-x)(1-y^2). \end{aligned}$$

Cubic serendipity polynomials:

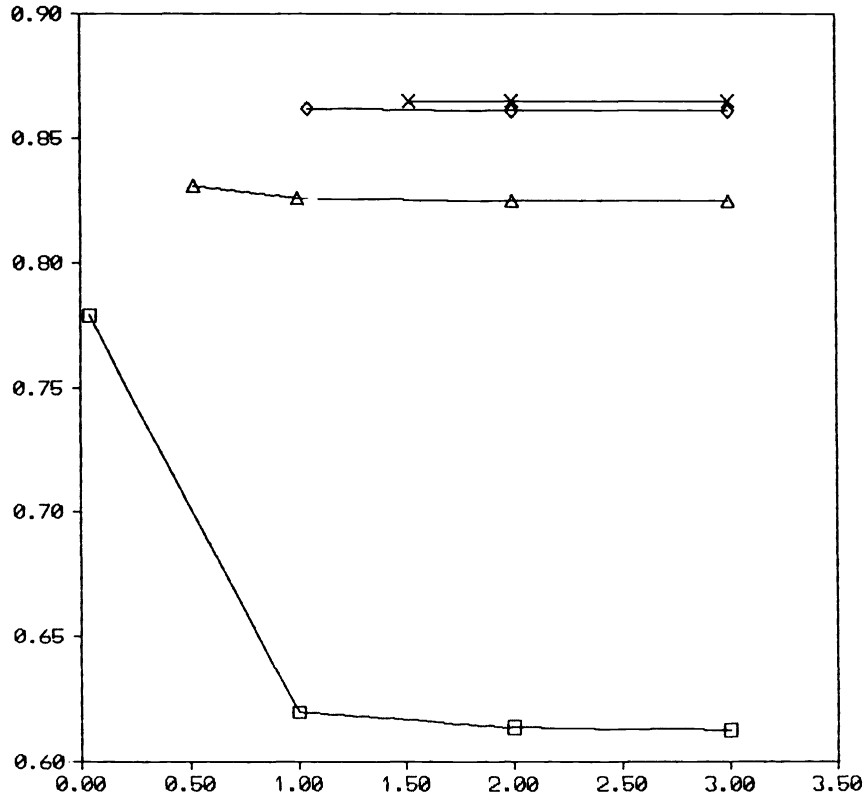
$$\begin{aligned} \phi_1 &= \tfrac{1}{32}(1-x)(1-y)(-10+9x^2+9y^2), \\ \phi_2 &= \tfrac{9}{32}(1-x^2)(1-y)(1-3x), \\ \phi_3 &= \tfrac{9}{32}(1-x^2)(1+3x)(1-y), \\ \phi_4 &= \tfrac{1}{32}(1+x)(1-y)(-10+9x^2+9y^2), \\ \phi_5 &= \tfrac{9}{32}(1+x)(1-y^2)(1-3y), \\ \phi_6 &= \tfrac{9}{32}(1+x)(1-y^2)(1+3y), \\ \phi_7 &= \tfrac{1}{32}(1+x)(1+y)(-10+9x^2+9y^2), \\ \phi_8 &= \tfrac{9}{32}(1-x^2)(1+3x)(1+y), \\ \phi_9 &= \tfrac{9}{32}(1-x^2)(1-3x)(1+y), \\ \phi_{10} &= \tfrac{1}{32}(1-x)(1+y)(-10+9x^2+9y^2), \\ \phi_{11} &= \tfrac{9}{32}(1-x)(1+3y)(1-y^2), \\ \phi_{12} &= \tfrac{9}{32}(1-x)(1-3y)(1-y^2). \end{aligned}$$



GRAPH 1. *Triangular elements*: $\alpha = \beta = \pi/4$; *piecewise linear polynomials*.



GRAPH 2. *Triangular elements*: $\alpha = \beta = \pi/4$; *piecewise quadratic polynomials*.



GRAPH 3. *Rectangular elements: piecewise bilinear polynomials.*

The bilinear form $a_{\tilde{E}}(\cdot,\cdot)$ generates the assembled stiffness matrix $A_{\tilde{E}}$ obtained by assembling the corresponding element stiffness matrices for all four finer elements contained in $\tilde{E}$ (see Figs. 1(a), (b) and Figs. 2(a)-(c)). Using the partitioning of the nodes into $\tilde{N}$ and N_1 as described above, the assembled stiffness matrix admits the following two-level block form:

$$A_{\tilde{E}}=\begin{pmatrix} A_{\tilde{E};1,1} & A_{\tilde{E};1,2} \\ A_{\tilde{E};2,1} & A_{\tilde{E};2,2}\end{pmatrix}\begin{matrix}\}N_1 \\ \}\tilde{N}\end{matrix},$$

that is, first are ordered the nodes from N_1 and then the nodes from $\tilde{N}$. We will need the element Schur complement

$$S_{\tilde{E}}=A_{\tilde{E};2,2}-A_{\tilde{E};2,1}(A_{\tilde{E};1,1})^{-1}A_{\tilde{E};1,2}.$$

We also consider the coarse grid element stiffness matrix $\tilde{A}_{\tilde{E}}$.

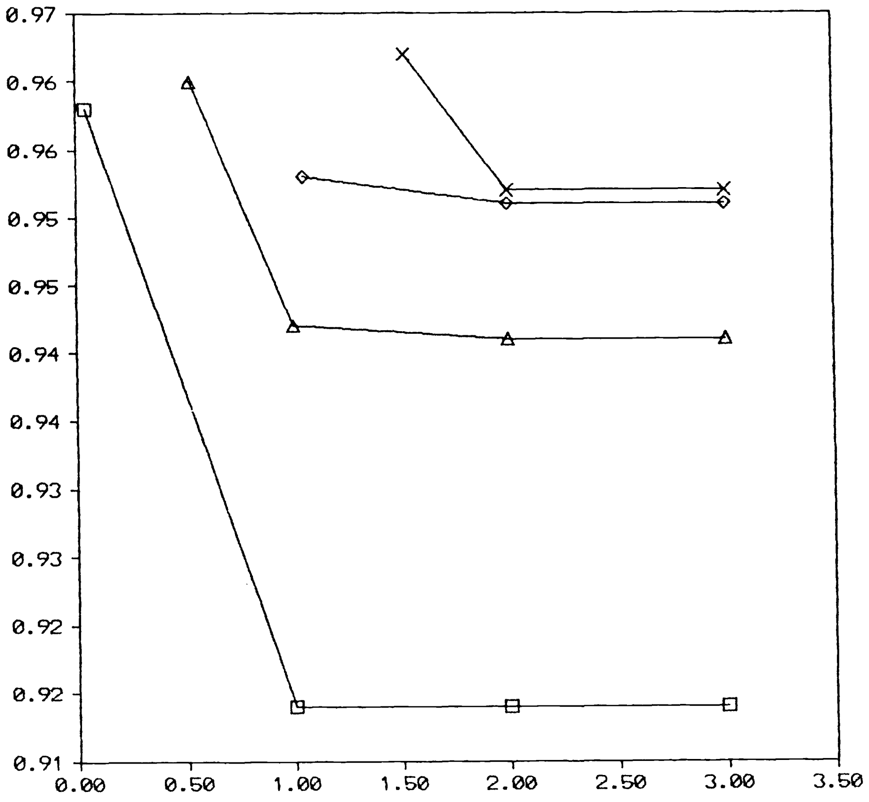
It has been demonstrated (e.g., in [8]) that the local γ-constant, $\gamma_{\tilde{E}}$, and the minimal eigenvalue, $\lambda_{\tilde{E},\min}$, of the generalized eigenvalue problem

$$\lambda_{\tilde{E}}\tilde{A}_{\tilde{E}}\mathbf{q}=S_{\tilde{E}}\mathbf{q}, \tag{2.1}$$

are related as follows:

$$\lambda_{\tilde{E},\min}=1-\gamma_{\tilde{E}}^2.$$

Note that $\tilde{A}_{\tilde{E}}$ and $S_{\tilde{E}}$ have the same nullspace $\{C(1,1,\cdots,1)^T\}$, that is, the constant vectors.



GRAPH 4. *Rectangular elements: piecewise quadratic serendipity polynomials.*

For the actual computation of $\lambda_{\tilde{E},\min}$ we use matrices obtained from $\tilde{A}_{\tilde{E}}$ and $S_{\tilde{E}}$ by deleting, say, the last row and the last column of both matrices. This is readily seen (cf. [8]) since (2.1) implies for $\mathbf{e}=(1,\cdots,1)^{\mathrm{T}}$

$$\lambda_{\tilde{E}}\tilde{A}_{\tilde{E}}(\mathbf{q}-q_{i_0}\mathbf{e})=S_{\tilde{E}}(\mathbf{q}-q_{i_0}\mathbf{e}), \tag{2.2}$$

where q_{i_0} is the i_0th component of $\mathbf{q}$ for arbitrary i_0.

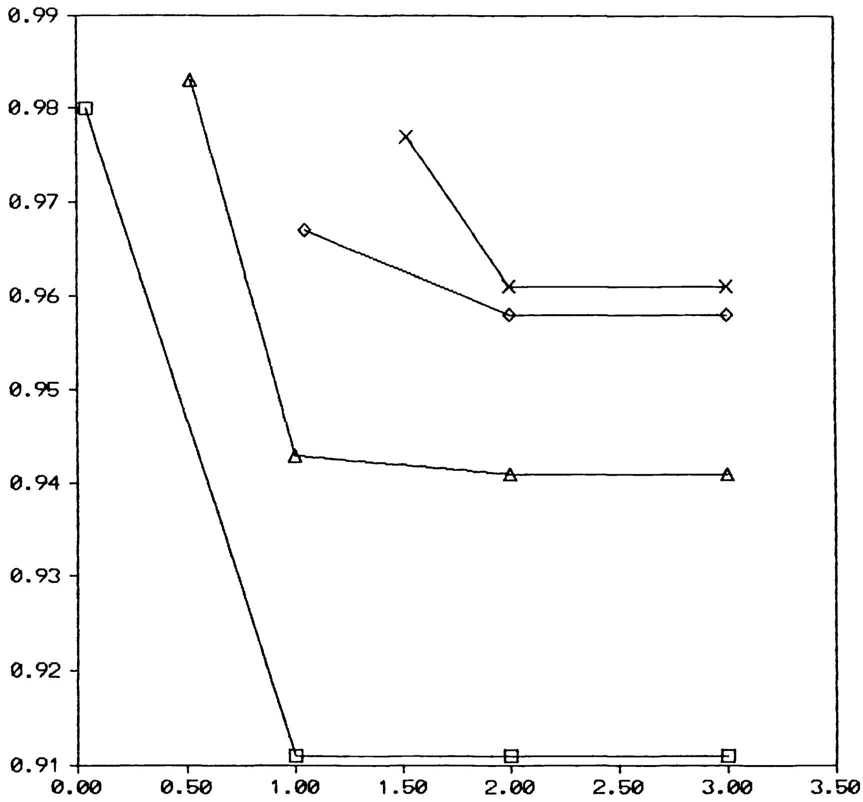
Now by leaving out the i_0th equation of (2.2) we obtain

$$\lambda_{\tilde{E}}\tilde{A}'_{\tilde{E}}\mathbf{q}'=S'_{\tilde{E}}\mathbf{q}', \tag{2.3}$$

where by "′" we denote the matrices and vectors obtained by deleting the i_0th column and the i_0th row of the matrices and by deleting the i_0th component of the corresponding vectors.

Now $\tilde{A}'_{\tilde{E}}$ and $S'_{\tilde{E}}$ are nonsingular, hence positive definite (and symmetric) and any algorithm for computing the minimal eigenvalue of (2.3) can be applied. We have chosen the simple bisection algorithm.

3. Numerical results. The numerical results are collected in Graphs 1-5. The results are listed for $\varepsilon=1$, 0.1, 0.01, and 0.001 for the cases of triangular elements: linear and quadratic piecewise polynomials, Graphs 1 and 2, respectively. Results are also listed for rectangular elements: piecewise bilinear, piecewise quadratic serendipity, and piecewise cubic serendipity polynomials on Graphs 3, 4, and 5, respectively. Recall that $\gamma^{(2)}$ and $\gamma^{(3)}$ are defined by (1.3).



GRAPH 5. *Rectangular elements*: *piecewise cubic serendipity polynomials.*

For all the graphs we have used the following notation:

□ $\varepsilon = 1$,

△ $\varepsilon = 0.1$,

◇ $\varepsilon = 0.01$,

× $\varepsilon = 0.001$,

x-axis: $\log_{10} 1/\delta$,

y-axis: γ.

The actual computed values of the γ-constants are shown in Tables 1-5, corresponding to Graphs 1-5.

4. Conclusion. It is clearly demonstrated by our numerical test that for problems without mixed derivatives ($\delta = 0$) all the computed γ-constants are less than the critical value $\gamma^{(2)}$ (see (1.3)) in the case of piecewise linear or bilinear polynomials. This is also the case for the same piecewise polynomials in the presence of mixed derivatives for all tested values δ such that δ^2 is not very close to ε, that is, when the bilinear form is clear of indefiniteness. However, increasing δ and the degree of the piecewise polynomials lead to deterioration in γ and in some specific cases (piecewise cubic serendipity polynomials, Graph 5 or Table 5) γ becomes larger than the critical $\gamma^{(3)}$. Note that the most stable results are obtained for piecewise bilinear polynomials (Table 3). We have in this case that all the computed values of γ are less than $\gamma^{(2)}$.

Table 1

$\varepsilon = 1$	δ	0.9	0.1	0.01	0.001
	γ	0.942	0.725	0.709	0.707
$\varepsilon = 0.1$	δ	0.3	0.1	0.01	0.001
	γ	0.969	0.758	0.710	0.707
$\varepsilon = 0.01$	δ		0.09	0.01	0.001
	γ		0.933	0.713	0.707
$\varepsilon = 0.001$	δ		0.03	0.01	0.001
	γ		0.958	0.741	0.707

Table 2

$\varepsilon = 1$	δ		0.1	0.01	0.001
	γ		0.888	0.875	0.874
$\varepsilon = 0.1$	δ	0.3	0.1	0.01	0.001
	γ	0.994	0.935	0.916	0.915
$\varepsilon = 0.01$	δ		0.09	0.01	0.001
	γ		0.990	0.934	0.932
$\varepsilon = 0.001$	δ		0.03	0.01	0.001
	γ		0.994	0.943	0.935

Table 3

$\varepsilon = 1$	δ	0.9	0.1	0.01	0.001
	γ	0.779	0.619	0.613	0.612
$\varepsilon = 0.1$	δ	0.3	0.1	0.01	0.001
	γ	0.831	0.826	0.825	0.825
$\varepsilon = 0.01$	δ		0.09	0.01	0.001
	γ		0.862	0.861	0.861
$\varepsilon = 0.001$	δ		0.03	0.01	0.001
	γ		0.865	0.865	0.865

Table 4

$\varepsilon = 1$	δ	0.9	0.1	0.01	0.001
	γ	0.963	0.919	0.919	0.919
$\varepsilon = 0.1$	δ	0.3	0.1	0.01	0.001
	γ	0.965	0.947	0.946	0.946
$\varepsilon = 0.01$	δ		0.09	0.01	0.001
	γ		0.958	0.956	0.956
$\varepsilon = 0.001$	δ		0.03	0.01	0.001
	γ		0.967	0.957	0.957

TABLE 5

$\varepsilon=1$	δ	0.9	0.1	0.01	0.001
	γ	0.980	0.911	0.911	0.911
$\varepsilon=0.1$	δ	0.3	0.1	0.01	0.001
	γ	0.983	0.943	0.941	0.941
$\varepsilon=0.01$	δ		0.09	0.01	0.001
	γ		0.967	0.958	0.958
$\varepsilon=0.001$	δ		0.03	0.01	0.001
	γ		0.977	0.961	0.961

We can draw the conclusion that for problems with anisotropy, mixed derivatives, and piecewise polynomials of high degree, the straightforward application of the algebraic multi-level iterative (AMLI) methods from Axelsson and Vassilevski [3], [4] can be troublesome. A remedy has been suggested in Vassilevski [10] by allowing the polynomials involved in the construction of the AMLI preconditioning matrices to have degrees that vary with the discretization levels, that is, to have $\nu=\nu_k$, $k=1,2,\cdots,l$, where l is the number of discretization levels used. It was demonstrated in [10] that at most of the levels one can use polynomials of first degree, which leads to a hybrid V-cycle multilevel iteration. This latter approach does not impose such strong restriction on the degree $\nu=\nu_k$ of the Chebyshev polynomials involved in the construction of the AMLI preconditioning matrices, and it preserves their asymptotic optimality (when l, the number of discretization levels used, is sufficiently large). For details, see [10].

REFERENCES

[1] O. AXELSSON, *On multigrid methods of two-level type*, in Multigrid Methods, Proceedings, Köln-Porz, 1981, W. Hackbusch and U. Trottenberg, eds., Lecture Notes in Mathematics 960, Springer-Verlag, Berlin, New York, 1982, pp. 352-367.

[2] O. AXELSSON AND I. GUSTAFSSON, *Preconditioning and two-level multigrid methods of arbitrary degree of approximation*, Math. Comp., 40 (1983), pp. 219-242.

[3] O. AXELSSON AND P. S. VASSILEVSKI, *Algebraic multilevel preconditioning methods*, I, Numer. Math., 56 (1989), pp. 157-177.

[4] ———, *Algebraic multilevel preconditioning methods*, II, SIAM J. Numer. Anal., 27 (1990), pp. 1569-1590.

[5] R. BANK AND T. DUPONT, *Analysis of a two-level scheme for solving finite element equations*, Report CNA-159, Center for Numerical Analysis, University of Texas at Austin, Austin, TX, 1980.

[6] D. BRAESS, *The contraction number of a multigrid method for solving the Poisson equation*, Numer. Math., 37 (1981), pp. 387-404.

[7] ———, *The convergence rate of a multigrid method with Gauss-Seidel relaxation for the Poisson equation*, in Multigrid Methods, Proceedings, Köln-Porz, 1981, W. Hackbusch and U. Trottenberg, eds., Lecture Notes in Mathematics 960, Springer-Verlag, Berlin, New York, 1982, pp. 368-387.

[8] V. EIJKHOUT AND P. S. VASSILEVSKI, *The role of the strengthened C.-B.-S. inequality in multilevel methods*, SIAM Rev., 33 (1991), pp. 405-419.

[9] J.-F. MAITRE AND F. MUSY, *The contraction number of a class of two-level methods; an exact evaluation for some finite element subspaces and model problems*, in Multigrid Methods, Proceedings, Köln-Porz, 1981, W. Hackbusch and U. Trottenberg, eds., Lecture Notes in Mathematics 960, Springer-Verlag, Berlin, New York, 1982, pp. 535-544.

[10] P. S. VASSILEVSKI, *Hybrid V-cycle algebraic multilevel preconditioners*, Report #9, Center of Informatics and Computer Technology, Bulgarian Academy of Sciences, Sofia, Bulgaria, 1990; Math. Comp., to appear.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 3, pp. 666-675, May 1992

APPROXIMATION OF THE INVARIANT MANIFOLD IN THE JOSEPHSON EQUATION*

M. VAN VELDHUIZEN†

Abstract. This paper considers the application of the algorithm described in [M. van Veldhuizen, *Math. Comp.*, 51 (1988), pp. 677-697] to approximate the average slope of solutions of the Josephson equation. The potential for parallelism of this algorithm has been a motivation for its study. In this work this potential is realized by presenting results obtained on a multiprocessor environment.

Key words. invariant curve, parallel computation

AMS(MOS) subject classifications. primary 65L99; secondary 65H10, 34C40

1. Introduction. Recently there has been some interest in numerical algorithms for the approximation of an invariant curve (cf. Chan [2]; Kevrekidis et al. [9]; Kaas-Peterson [6]-[8]; van Veldhuizen [16], [18]; Dieci, Lorenz, and Russell [4] and the references therein). In this paper we concentrate on collocation-type methods for the Josephson equation

$$\beta\ddot{\phi}+(1+\gamma\cos\phi)\dot{\phi}+\sin\phi=p(t), \tag{1.1}$$

where $\beta>0$ and γ are parameters, and $p(t)$ is periodic of period $T=2\pi/\omega$. This equation is sometimes used as a model for the Kamper-Soulen resistively shunted Josephson thermometer (cf. [14], [15]). The quantity of interest is the average slope of the solution $\langle\dot{\phi}\rangle=\lim(\phi(t)/t)$, $t\to\infty$. It can be measured, since it is proportional to the voltage over the junction. A straightforward computation of this quantity leads to a sequential algorithm. In this paper we describe a parallel algorithm based on the invariant curve of the Josephson equation and its properties.

It is convenient to rewrite this second-order differential equation as a system of two first-order differential equations

$$\beta\dot{\phi}=\psi-\phi-\gamma\sin\phi, \tag{1.2a}$$

$$\psi=-\sin\phi+p(t). \tag{1.2b}$$

Define the Poincaré map P as the period map taking $x=(\phi(0),\psi(0))$ into Px defined as the solution of (1.2a,b) at $t=T=2\pi/\omega$ with initial vector x at $t=0$. Since (1.2a,b) are essentially defined on the cylinder obtained by identifying each point (ϕ,ψ) with its 2π translates $(\phi+2\pi j,\psi+2\pi j)$, so is the Poincaré map P. The following characterization is due to Levi [10].

THEOREM 1.1. *For all T-periodic functions $p(t)$ and all parameters β and γ satisfying*

$$0\leqq\gamma\leqq 1,\qquad 0<\beta<\tfrac{1}{4}(1-\gamma)^2, \tag{1.3}$$

the Poincaré map P possesses a globally attracting invariant circle Γ given by $\psi=f(\phi)$, where $f(\phi)$ tends to $\phi+\gamma\sin\phi$ for $\beta\to 0$. Also, f is strictly increasing.

Observe that f is invertible. Thus, the invariant manifold may also be written as $\phi=f^{-1}(\psi)$. This simplifying observation is used in the algorithm described below.

* Received by the editors November 28, 1990; accepted for publication (in revised form) December 20, 1990.

† Department of Mathematics and Computer Science, Vrije Universiteit, De Boelelaan 1081, 1081HV Amsterdam, the Netherlands.

The Poincaré map P induces a circle map P_Γ on the invariant circle Γ. It is easily seen that $\langle\dot{\phi}\rangle = \omega\rho$, where ρ is the rotation number of the circle map induced by P on Γ. In particular, $\langle\dot{\phi}\rangle$ exists. See Levi [10]. For the definition of the rotation number ρ of a circle map, see Guckenheimer and Holmes [5] or Coddington and Levinson [3].

The restrictions (1.3) are discussed in Levi [10]. In particular, the constraint $\beta < 1/4$ for $\gamma = 0$ has a physical interpretation.

Computationally we have two ways of approximating the average slope $\langle\dot{\phi}\rangle$. First, we may approximate the limit by the quotient $\phi(t)/t$ for large t. For a slightly more sophisticated way of doing this, see van Veldhuizen and Fowler [15]. These processes are purely sequential in nature. Second, we might approximate the invariant manifold Γ and the circle map P_Γ induced on it by P (see Algorithm 1.2 below), followed by the numerical approximation of the rotation number; see [17]. We shall see that the computation of the approximate invariant manifold can be done in parallel. Hence, the larger part of the work involving the integration of the differential equations (1.2a,b) can be done in parallel. This is advantageous on a multiprocessor machine (cf. § 3).

We shall now describe a numerical algorithm for the approximation of Γ and the circle map P_Γ. This algorithm is very much in the spirit of Kevrekidis et al. [9], and has been investigated in [18].

ALGORITHM 1.2. Let Δ be a partitioning of the circle $[0, 2\pi)$, consisting of the N points

$$0 = \psi_0 < \psi_1 < \cdots < \psi_{N-1} < 2\pi. \tag{1.4}$$

Approximate Γ by a polygon Γ_Δ described by $\phi = g_\Delta(\psi)$, where g_Δ is a piecewise linear (with respect to Δ) continuous function. Determine g_Δ, i.e., determine the $g_\Delta(\psi_i)$, by the equations, $i = 0, 1, \cdots, N-1$,

$$(g_\Delta(\psi_i), \psi_i) \in \bar{\Gamma}_\Delta, \tag{1.5}$$

where $\bar{\Gamma}_\Delta$ is the polygon on the cylinder with vertices $P(g_\Delta(\psi_i), \psi_i)$. The circle map P_Γ is approximated by the piecewise linear continuous function induced by $\psi_i \to \bar{\psi}_i$, $\bar{\psi}_i$ the ψ-coordinate of $P(g_\Delta(\psi_i), \psi_i)$, $i = 0, 1, \cdots, N-1$.

The set of equations is solved by successive substitution, as in [18]. In order to clarify the programming of this algorithm, let us describe the iterative process separately. For all i, let $\phi_i^{(k)}$ denote the actual approximation to $g_\Delta(\psi_i)$. First, one integrates the ordinary differential equations (1.2a,b) with initial vector $(\phi_i^{(k)}, \psi_i)$ from $t = 0$ to $t = 2\pi/\omega$, $i = 0, 1, \cdots, N-1$. Or, in a formal notation,

$$P: \quad (\phi_i^{(k)}, \psi_i) \to (\bar{\phi}_i^{(k)}, \bar{\psi}_i^{(k)}).$$

After the integration step comes the interpolation step. For all i separately determine the index l (depending on i) such that $\psi_i \in I_l = [\bar{\psi}_l^{(k)}, \bar{\psi}_{l+1}^{(k)}]$ on the circle $[0, 2\pi)$. Then $\phi_i^{(k+1)}$ is obtained by linear interpolation on the segment I_l in the point ψ_i. The process stops if the norm of the difference between two successive vectors of iterates is smaller than a prescribed tolerance.

Observe the parallel nature of this algorithm; the images under the Poincaré map P can be computed in parallel. This computation can be done by a high quality numerical code for the integration of ordinary differential equations.

2. Summary of results.

In this section we summarize the results obtained by Levi [10] for the Josephson equation. In addition, we summarize the results from the convergence analysis in [18]. We omit the details, since no essentially new arguments are used in deriving the results.

A remarkable feature of the global existence result by Levi [10] (cf. Theorem 1.1 above) is the explicit estimate for the domain in parameter space for which the result holds true. In fact, by carefully redoing Levi's proof, one can extend this domain in parameter space somewhat. Instead of restricting β and γ by (1.3), it suffices to restrict β and γ to the union of the regions I, II, and III in Fig. 1, a somewhat larger domain. Nevertheless, even this larger domain still belongs to the strip $0 \leqq \gamma < 1$. On the other hand, numerical evidence suggests that the properties of the Josephson equation do not dramatically change at $\gamma = 1$, although the character of the solution followed in time may do so (internal layers for $\gamma > 1$). An existence result for $\gamma > 1$ in an open neighborhood of the line $\beta = \gamma$ in parameter space follows from the following simple observation.

PROPOSITION 2.1. *If $\beta = \gamma$, then for all T-periodic functions $p(t)$ and all values of $\beta > 0$, the Poincaré map P possesses an invariant circle Γ given by*

$$\psi - \phi = \bar{p} = \int_{-\infty}^{0} e^{s/\beta} p(s)\, ds. \tag{2.1}$$

Thus Γ is a straight line parallel to the line $\psi = \phi$.

Proof. If $\beta = \gamma$ one easily obtains from (1.2a,b) the differential equation

$$\beta(\dot{\psi} - \dot{\phi}) = \phi - \psi + \beta p(t). \tag{2.2}$$

If $\psi(0) - \phi(0) = \bar{p}$, then so is $\psi(T) - \phi(T)$. This proves the assertion. □

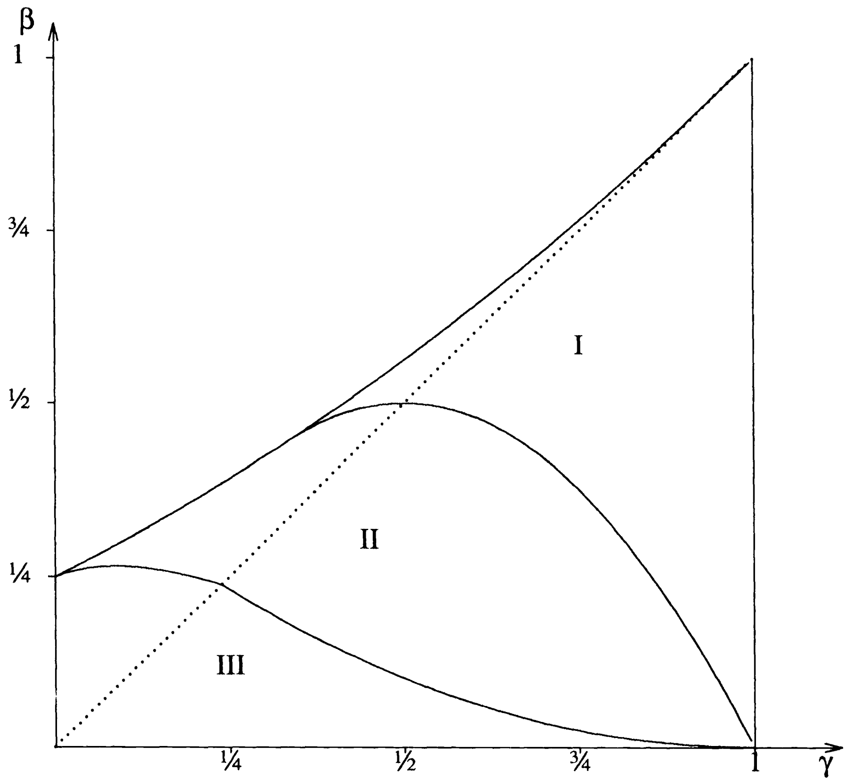


FIG. 1. *Regions in parameter space where Theorem* 1.1 *is valid* (I + II + III), *and where convergence results for Algorithm* 1.2 *are available* (II + III).

If $\beta=\gamma$ we may compute the restriction of P to Γ, the circle map corresponding to the invariant manifold. Let Δ denote the unique periodic solution of (2.2), which satisfies (2.1) in the points jT, $j \in \mathbb{N}$. Let ϕ solve

$$\beta\dot{\phi}=\Delta-\beta \sin \phi. \tag{2.3}$$

Then $\sigma: \phi(0) \rightarrow \phi(T)$ describes the circle map σ on Γ.

One might ask for the existence of an invariant manifold in the neighborhood of the line $\beta=\gamma$ in the (β, γ)-plane. Typical perturbation results for invariant manifolds require that the action on the manifold is small in relation to the attraction to the manifold. See Sacker [12] for typical results and nice counterexamples. In the language of Sacker [12] (we use his symbols on the left of the equality sign), one has

$$\bar{\beta}=1, \qquad \underline{\lambda}=\beta.$$

An invariant manifold exists in an open neighborhood of the point (β, β) in the (β, γ)-plane if $\bar{\beta}<\underline{\lambda}$ (cf. Sacker [12]). This condition is satisfied for all $\beta>1$. Hence, *the invariant manifold exists in an open neighborhood of the line* $\beta=\gamma$, $\beta>1$.

The convergence of Algorithm 1.2 follows from the results in [18], at least if some conditions are fulfilled. Actually, the convergence theory in [18] contains two related criteria. We give the interpretation adapted to Algorithm 1.2.

The first criterion given in [18] is based on the following idea. The Poincaré map P cannot be a contraction, but its restriction to lines parallel to the ϕ-axis might be a contraction. The first criterion of [18] is based on this possibility in combination with the linear interpolation process of Algorithm 1.2. For the Josephson equation, it is possible to estimate the region in parameter space for which this criterion assures convergence of the algorithm: just adapt the proofs in [18] to this differential equation. Of course, the estimates give part of the region. The region of convergence in parameter space obtained in this way is shown as region III in Fig. 1. Observe that this criterion requires no assumption about the mesh.

The second criterion given in [18] requires that the Poincaré map reduces the horizontal distance to the invariant curve. Here, the horizontal distance of $x=(\phi, \psi)$ to Γ is defined by $|f^{-1}(\psi)-\phi|$ (cf. Theorem 1.1). If the Poincaré map reduces the horizontal distance to Γ, then Algorithm 1.2 converges for a sufficiently fine mesh in a sufficiently narrow tube along the invariant curve. In regions III and II in Fig. 1, convergence can be proved in view of this criterion. Again, this region is obtained by carefully redoing the estimates given in the convergence proofs in [18].

These criteria, when satisfied, result in convergence by a contraction principle. In actual computations, the iterative process for solving the nonlinear equations is based on this contracting map. Hence, if the contraction factor is sufficiently small, say $<\frac{1}{2}$, the algorithm is stable with respect to small perturbations. This is important, since small perturbations are to be expected. Rounding errors cause small perturbations, but we also have the errors caused by numerical integration of the initial value problems. In actual computations the contraction factor is easily estimated, and often as small as $\frac{1}{10}$. For large values of ω the contraction factor tends to unity. This is easily avoided by using an mth iterate of the Poincaré map instead of the Poincaré map itself.

3. Numerical experiments. In this section we mention a few numerical results obtained by Algorithm 1.2. We compare the results with earlier ones. Also, we shortly discuss the parallel nature of Algorithm 1.2, and we give some examples.

In actual computations, Algorithm 1.2 performs much better than suggested by the theory of [18]. For instance, the algorithm converges rapidly in the union of the regions I, II, and III in Fig. 1, not only in the regions II and III. Also, the algorithm converges for $\gamma > 1$, and small $\beta > 0$. In addition, one may even interchange the role of ψ and ϕ in Algorithm 1.2, and this new algorithm behaves very much like Algorithm 1.2. Results for γ as large as 3.0 have been obtained, for both algorithms.

The algorithm uses linear interpolation, a rather crude type of interpolation. However, higher-order interpolation may cause problems. From the explicit description in § 1, it is obvious that the ordering of the ψ-coordinate of the images plays an important part. With linear interpolation no problems have been observed, due to the shape-preserving properties of linear interpolation and the existence of a global P-invariant foliation of the plane (cf. Levi [11]). With piecewise quadratic or cubic interpolation the images may get out of order, making the computation worthless. For the moment, linear interpolation seems a reasonable choice.

As mentioned in § 1, one is interested in the numerical values of $\langle\dot{\phi}\rangle = \omega\rho$, where ρ is the rotation number of the circle map, i.e., the restriction of P to Γ. In [15] results have been reported in which $\langle\dot{\phi}\rangle$ is approximated directly. This requires the integration of the differential equations (1.2a,b) over large intervals, say $2000T$ or even larger. This is a sequential process, and there is a real danger of accumulation of integration errors.

Some idea about the loss of accuracy is obtained by considering an artificial example. By Proposition 2.1 we may construct an example in which the invariant curve Γ is known. Choose $\beta = \gamma = 0.1$, and choose $\phi(t) = \omega t + A \sin(\Omega t)$, Ω an integer multiple of $\omega \sim 1$. Consequently, the function $\Delta(t) = \beta(\dot{\phi} + \sin\phi)$ (cf. (2.3)) is ω-periodic. Since Δ is the periodic solution of (2.2), we obtain $p(t)$ from (2.2). Hence $p(t)$ is ω-periodic. However, in a numerical experiment designed to estimate the accumulation of truncation errors, the problem behaves rather like a Ω-periodic function. Therefore, we use $T = 2\pi/\Omega$. In using this smaller value, we can be sure not to exaggerate. Typically, $\Omega = 100\,\omega$. Even with $T = 2\pi/\Omega$, there is a significant loss in accuracy after integration over 2000 periods. Using the multistep code from Shampine and Gordon [13] with initial values on Γ the errors in ϕ are of the order 10^3 TOL, where TOL is the tolerance used, $\text{TOL} < 1.0\text{E}-06$ (equal absolute and relative error tolerance). The deviation from the stable manifold Γ is much smaller, say in the order of 10 TOL. This is what one would expect. The relation between the error in the solution ϕ and the average slope $\langle\dot{\phi}\rangle$ is a complicated one. For tolerances as small as $1.0\text{E}-08$, the effect of the accumulation of errors really shows in the obtained approximation. However, for a tolerance like $1.0\text{E}-11$ the accumulation of errors is hardly noticeable. In one typical example, the error in the approximation for $\langle\dot{\phi}\rangle$ increases by $4\text{E}-12$ in every 500 periods, with integration by the Shampine–Gordon code and tolerances $1.0\text{E}-11$. In this particular example we had $\gamma = 1.5$, $\beta = 0.05$, $\omega = 38.0$. Clearly, direct approximation of ρ is possible, but only with limited accuracy due to accumulation of errors, unless very small tolerances are being used. For instance, the results reported in [15, Fig. 2] have an error of up to 0.1 percent. A better accuracy, if possible, would increase the computing time considerably.

As an alternative, one might compute an approximation to the circle map by Algorithm 1.2, and compute the rotation number of this approximate circle map. In this way, one also integrates the differential equations (1.2a,b), many times over short intervals typically of length T and there is hardly any serious accumulation of rounding errors and local errors. Afterwards, one approximates ρ by means of [17, Algorithm 3.2], an algorithm for the numerical approximation of the rotation number. In the invariant manifold approach, there are two main error sources: the approximation error in the approximate invariant manifold and the truncation error in the approximate rotation number. This latter error is hard to assess theoretically (cf. the discussion in

[17]). Practical experience indicates the reliability of the algorithm used, with truncation errors less than 1.0E − 07. It is much harder to estimate the error in the approximate invariant manifold. Typically, with $N \sim 100$, i.e., a step size of $2\pi/N$, the invariant manifold approach results in approximations with an error of about 1.0E − 05 in the rotation number (comparison with results for $N = 800$). In resonance regions the method performs even better, sometimes giving the exact result with $N = 50$. In many instances the invariant manifold approach is highly competitive with direct integration over extremely long intervals.

As an example, consider Fig. 2. We have shown the rotation number computed from the approximate circle map obtained by Algorithm 1.2, with $N = 100$, $p(t) = 11 + 0.9\,\omega \cos(\omega t)$, $\beta = 0.01$, $\gamma = 3/2$. Results are shown for $\omega = 37.853\ (0.01)\ 40.653$. The results show the familiar devil's staircase picture. In the upper part of Fig. 2 we show the difference between the results from Algorithm 1.2 and direct integration (data from [15]). The maximal error is 3.6E − 04 on a rotation number $\rho \sim 2/7$. The upper graph shows larger errors at the borders of the steps. Actually, a step in the devil's staircase is just a resonance region, and the difference between the two results tends to be larger at the borders of these regions. These places correspond to places where

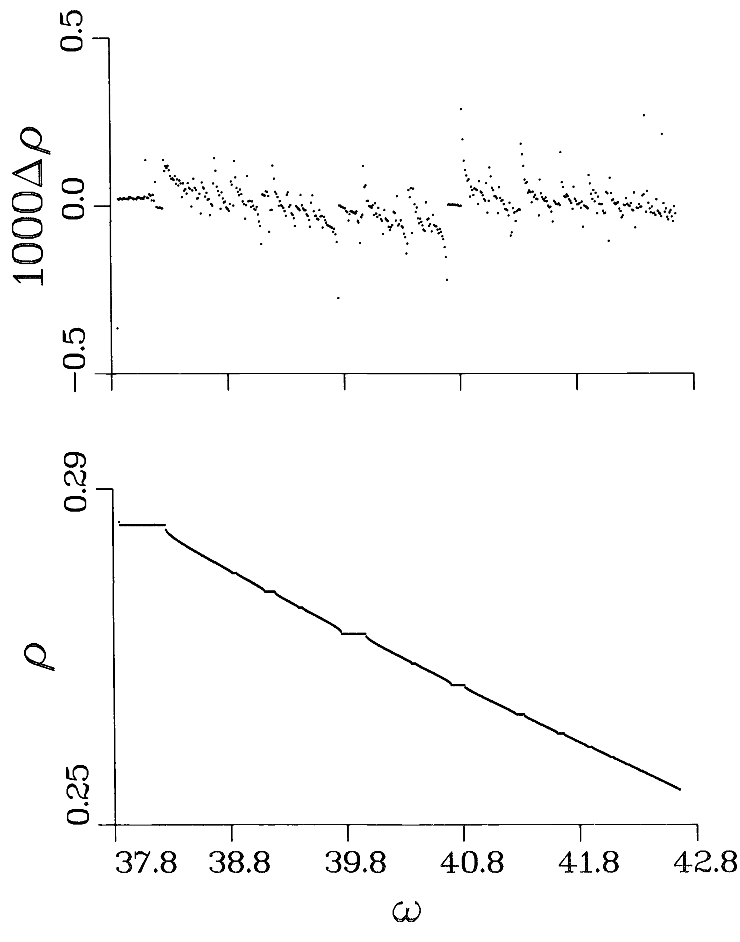


FIG. 2. *The rotation number of the circle map of the Josephson equation as computed with Algorithm* 1.2 *and* [17, *Algorithm* 3.2]. *The upper part shows the difference between these values and the values obtained in* [15]. *For the parameter values, see text.*

ρ has a square root behavior. One might also say that the different methods result in slightly different resonance regions. Relatively large errors at the borders of the resonance regions are then to be expected. For this specific example the computation based on Algorithm 1.2 is approximately twice as fast as a comparable (in tolerances) direct computation. In order to get comparable timings, all computations have been done on one mainframe computer (CDC Cyber 990), with recomputation of the results reported in [15].

A further speed-up is to be expected if one exploits the parallel nature of Algorithm 1.2. To that end we have implemented this algorithm on the department's shared memory machine, best described as a set of 16 boards, each of them with 4Mbyte local memory, an M68030 CPU and an M68882 FPU, and mutually connected by a VME-bus. The programming language implemented on this system is Orca (cf. Bal and Tanenbaum [1]). Orca is based on Modula-2, with a fork statement for the creation of new asynchronous processes, and a guard mechanism for synchronizing these processes. Both the hardware and the software are still in an experimental stage, mainly used for research in distributed systems.

The parallelism in Algorithm 1.2 comes from the parallel computation of the images under the Poincaré map P. So let us consider this computation in some detail. In one mode, whether in parallel or not, one computes many evaluations of P simultaneously by integrating one big system of differential equations. In this way one distributes the overhead of the code over many evaluations. This works well if the step sizes in the integration do not vary that much. This is typically the case for $0 < \gamma < 1$. However, if γ is significantly larger than 1, the solution contains small areas with internal layers along the t-axis. These areas require small steps in the integration routine. The position of the internal layer strongly depends on the initial condition. Thus, if one would combine the computation of many images under P in one integration, one would be forced at some point in time to use small internal layer steps for all equations together, while only a few equations require them. Hence, combination of differential equations into one larger set may not be economical for large values of γ. For $\gamma = 1.5$ both processes require about the same amount of time.

We have implemented Algorithm 1.2 in many ways on the multiprocessor. In the first implementation, all processors, the master and the slave processors, did their share in the computation of images under the Poincaré map. This turned out to be inefficient, resulting in a lot of idle time and bad load balancing. Therefore, we changed the strategy. The master now distributes the tasks and performs the interpolation required for the computation of new iterates. The slave processors do the actual integration and report to the master. For small p, $p = 1, 2, 3$, one wastes some time on the master. For larger values of p, this process is by far superior to a process in which the master does some integration as well. Of course, in all implementations one should carefully minimize the number of bus accesses. Also, one should minimize the amount of data shared between master and slave. This is partly due to the hardware, partly to the software overhead caused by Orca.

Many implementations with a master processor have been tried. Two implementations of Algorithm 1.2 capture the whole range of possibilities. In the implementation A-1, the master assigns the computation of just one image of P to an idle processor, until all evaluations have been done. The master then does the interpolation, and goes on with the next step. Actually, part of the interpolation is done in parallel. If there are p slaves, the master does the interpolation for the first p assignments. Then, while the slaves are at work, the master does the next p interpolations, and so on. At the other end of the spectrum, in implementation A-2, we compute as many images as are feasible together on a slave processor. Suppose one wants to compute N evaluations of the Poincaré map P on $p+1$ processors, p slaves, and 1 master. Divide the N evaluations in p sets of $N \operatorname{div} p$ and $(N \operatorname{div} p)+1$ equations. Integrate these equations together, each of them on one slave processor. Of course, the equations are collected in such a way that the variation in the initial values is as small as possible. This implies that the initial value problems are as similar as possible. Thus, if internal layers are present, there is as much overlap of these layers on the t-axis as possible.

In Table 1 we give results for the problem with equidistant mesh, and

$$(3.1) \qquad N=128, \quad \beta=0.25, \quad \gamma=0.7, \quad \omega=19.0, \quad \alpha=5, \quad \kappa=0.$$

The total number of processors, p slaves plus 1 master, is mentioned under NCPU. A Runge-Kutta code with variable step size is used as integrator, tolerance 1.0E−07. The invariant curve algorithm uses 12 iteration steps. The efficiency "eff" in Table 1 is defined as follows. On one processor the best time obtained is 293 seconds (same compiler, identical CPU plus FPU). The efficiency is defined by

$$(3.2) \qquad \text{eff} = \frac{\text{time one processor}}{\text{NCPU} * \text{time}}$$

expressed as a percentage. Note that implementation A-2 is very much superior to implementation A-1 (both implementations of Algorithm 1.2, and mathematically equivalent).

In a second example we choose

$$(3.3) \qquad N=128, \quad \beta=0.02, \quad \gamma=2.5, \quad \omega=19.0, \quad \alpha=2, \quad \kappa=5.$$

Now we have a large value of γ and a small β. Algorithm 1.2 converges in six steps, starting from the straight line $\phi=\psi$ as an initial guess. Again we look at the parallel implementations, implementation A-1 and implementation A-2. Since γ is quite large, we expect implementation A-1 to outperform implementation A-2. The results confirm this for a small number of processors. Indeed, with a few processors implementation

TABLE 1
Results for (3.1).

	Implementation A-1		Implementation A-2	
NCPU	Time in seconds	eff	Time in seconds	eff
2	440.0	33%	282.0	52%
3	220.0	44%	149.5	65%
4	148.0	49%	101.0	73%
5	111.3	53%	78.2	75%
6	89.8	54%	62.6	78%
7	75.8	55%	53.2	78%
8	65.6	56%	46.9	78%
9	55.9	58%	42.4	77%
10	52.4	56%	38.0	77%
11	45.6	58%	35.1	76%
12	42.5	57%	31.9	77%
13	39.3	57%	30.0	75%
14	35.8	58%	28.6	73%
15	34.9	56%	27.0	72%
16	34.6	53%	25.0	73%

A-2 lumps many equations together into one rather large differential equation; this is inefficient for large γ as outlined above. However, the number of differential equations lumped together decreases if the number of processors increases. Hence, in implementation A-2 the bad effects of large γ diminish for larger p. In order to show this effect as clearly as possible, we have added a few additional columns in Table 2. Under the heading "size" we give the maximal number of equations lumped together. The minimal number equals the maximal number if p divides 128, else the minimal number equals the maximal number minus 1. Under the heading "time/eq" we give the time per equation, i.e., the time divided by the maximal number of lumped equations. These results of Table 2 strongly indicate the extra cost of lumping together too many equations. The table also shows that implementation A-2 outperforms implementation A-1 for many slave processors. Because of the internal layers we solve this problem with a tolerance of 1.0E−10.

TABLE 2
Results for (3.3).

	Implementation A-1	Implementation A-2		
NCPU	Time in seconds	Time in seconds	Size	Time/eq
2	810.0	1376.5	128	10.7
3	406.0	641.3	64	10.0
4	272.7	359.0	43	8.3
5	205.8	242.2	32	7.6
6	164.8	187.5	26	7.2
7	138.5	147.7	22	6.7
8	118.9	122.3	19	6.4
9	105.0	102.6	16	6.4
10	94.7	90.5	15	6.0
11	83.9	83.1	13	6.4
12	76.6	74.1	12	6.2
13	69.8	67.5	11	6.1
14	64.8	61.5	10	6.2
15	62.4	55.1	10	5.5
16	57.6	54.7	9	6.1

The results in Table 2 for 14 and 15 processors reflect the programming of the master. The maximum size of a job is ten equations in each case. By a job we mean the task assigned to a slave processor, which consists of the integration of a number of differential equations lumped together. With 14 processors, there are 11 jobs consisting of ten equations, and two jobs consisting of nine equations. The master assigns the larger jobs first, then the smaller ones. In the case of 15 processors two jobs of ten equations and 12 jobs of nine equations result. The maximum job consists of ten differential equations, as in the previous case. But the total time spent is more reminiscent of jobs of nine equations, as with 16 processors. The explanation seems to be the overlap caused by distributing the larger jobs first.

4. Conclusion. The results of § 3 show the reliability of Algorithm 1.2. The algorithm performs well, even far outside the domain in parameter space in which convergence can be proved. Second, Algorithm 1.2 may serve as the basis for the approximation of rotation numbers. As such, the approach via Algorithm 1.2 is at least competitive with a direct approach. In addition, Algorithm 1.2 can be implemented efficiently on parallel architectures.

Acknowledgments. I am indebted to M. F. Kaashoek, H. E. Bal and C. J. H. Jacobs for implementing the floating point hardware, and for help with Amoeba, Orca, and the use of C functions in Orca.

REFERENCES

[1] H. E. Bal and A. S. Tanenbaum, *Distributed programming with shared data*, Proc. IEEE CS 1988 International Conference on Computer Languages, Miami, FL, Oct. 1988, pp. 82-91.

[2] T. N. Chan, *Numerical bifurcation analysis of simple dynamical systems*, Ph.D. thesis (unpublished), Department of Computer Science, Concordia University, Montreal, Canada, 1983.

[3] E. A. Coddington and N. Levinson, *Theory of Ordinary Differential Equations*, McGraw-Hill, New York, 1955.

[4] L. Dieci, J. Lorenz, and R. D. Russell, *Numerical calculation of invariant tori*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 607-647.

[5] J. Guckenheimer and Ph. Holmes, *Nonlinear Oscillations, Dynamical Systems and Bifurcations of Vector Fields*, Springer-Verlag, New York, 1983.

[6] Chr. Kaas-Petersen, *Computation of quasi-periodic solutions of forced dissipative systems*, J. Comput. Phys., 58 (1985), pp. 395-408.

[7] ———, *Computation of quasi-periodic solutions of forced dissipative systems* II, J. Comput. Phys., 64 (1986), pp. 433-442.

[8] ———, *Computation, continuation and bifurcation of torus solutions*, Phys. D, 25 (1987), pp. 288-306.

[9] 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.

[10] M. Levi, *Nonchaotic behavior in the Josephson junction*, Phys. Rev. A, (1988), pp. 927-931.

[11] ———, *Invariant curves and invariant foliations in forced oscillations*, preprint, Boston University, 1988.

[12] R. J. Sacker, *A perturbation theorem for invariant manifolds and Hölder continuity*, J. Math. Mech., 18 (1969), pp. 705-762.

[13] L. Shampine and M. K. Gordon, *Computer Solution of Ordinary Differential Equations*, W. H. Freeman, San Francisco, 1975.

[14] R. J. Soulen, Jr. and R. P. Giffard, *Josephson-effect absolute noise thermometer: Resolution of unmodeled errors*, Appl. Phys. Lett., (1978), pp. 770-772.

[15] M. van Veldhuizen and H. A. Fowler, *Subharmonic frequency locking in the resistive Josephson thermometer*, Phys. Rev. B, (1985), pp. 5805-5810.

[16] M. van Veldhuizen, *A new algorithm for the numerical approximation of an invariant curve*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 951-962.

[17] ———, *On the numerical approximation of the rotation number*, J. Comput. Appl. Math., 21 (1988), pp. 203-212.

[18] ———, *Convergence results for invariant curve algorithms*, Math. Comp., 51 (1988), pp. 677-697.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 3, pp. 676-686, May 1992

A GENERALIZED PRIME FACTOR FFT ALGORITHM FOR ANY $N = 2^p 3^q 5^r$ *

CLIVE TEMPERTON†

Abstract. Prime factor fast Fourier transform (FFT) algorithms have two important advantages: they can be simultaneously self-sorting and in-place, and they have a lower operation count than conventional FFT algorithms. The major disadvantage of the prime factor FFT has been that it was only applicable to a limited set of values of the transform length N. This paper presents a generalized prime factor FFT, which is applicable for any $N = 2^p 3^q 5^r$, while maintaining both the self-sorting in-place capability and the lower operation count. Timing experiments on the Cray Y-MP demonstrate the advantages of the new algorithm.

Key words. fast Fourier transform (FFT), prime factor algorithm (PFA), self-sorting FFT, in-place FFT

AMS(MOS) subject classification. 65T05

1. Introduction. Fast Fourier transform (FFT) algorithms can be defined whenever the transform length N can be factorized as $N = N_1 N_2 \cdots N_k$, where the factors N_i are integers. Though there are many variants of these algorithms, they fall into two basic categories: those based on the prime factor algorithm (PFA) of Good [5], which are only applicable if the factors N_i are mutually prime, and those descended from the algorithm of Cooley and Tukey [3], for which there is no such restriction (indeed the most familiar case is $N_i = 2$ for all i).

The prime factor algorithms have two important advantages. For a given value of N, the operation count is lower than that for the corresponding Cooley–Tukey algorithm. Moreover, the PFA can be made both self-sorting (input and output both naturally ordered) and in-place (requiring no work space) [2], [10], [14]. Their principal disadvantage is that in order to achieve this, a program to implement the PFA requires an explicit section of code (a "discrete Fourier transform (DFT) module") which performs a self-sorting in-place transform of length N_i for each of the factors. (The DFT module is the radix-N_i generalization of the familiar radix-2 "butterfly.") In most implementations of the PFA, the factors N_i are thus assumed to be members of the set $\{2, 3, 4, 5, 7, 8, 9, 16\}$. Coupled with the requirement that the factors be mutually prime, this severely limits the set of practicable transform lengths N.

Johnson and Burrus [7] demonstrated that the radix-2 Cooley–Tukey algorithm could also be made self-sorting and in-place, and in [18] the principle was extended to radix-3, radix-4, and radix-5 transforms, and finally to the mixed-radix case. However, as hinted in [18] the operation count for the resulting algorithm can be improved if N contains a mixture of factors.

In this paper we combine ideas from the PFA and from the self-sorting in-place form of the Cooley–Tukey algorithm, to derive a new generalized prime factor FFT algorithm with the following nice properties:

(1) It works for any transform length of the form $N = 2^p 3^q 5^r$;

(2) It is always self-sorting and in-place (the only work space required is for an optional list of precomputed twiddle factors);

* Received by the editors April 30, 1990; accepted for publication (in revised form) March 19, 1991. Much of this work was done at Recherche en Prévision Numérique, Atmospheric Environment Service, Canada.

† European Centre for Medium Range Weather Forecasts, Shinfield Park, Reading, Berkshire RG2 9AX, United Kingdom.

(3) For values of N suitable for the PFA, the new algorithm reduces to the PFA and has the same operation count;

(4) For $N=2^p$, 3^q, or 5^r, the new algorithm reduces to that described in [18], and has the same operation count;

(5) If N contains a mixture of factors but is unsuitable for the PFA, the new algorithm has a lower operation count than for the Cooley-Tukey algorithm.

Section 2 of this paper reviews the essentials of the prime factor algorithm. In § 3 we show how the PFA can be generalized to yield the new algorithm, removing the restriction on the transform lengths. Section 4 describes the implementation on a Cray Y-MP, and includes timing results to illustrate the properties of the new algorithm. In § 5, we describe a refinement of the algorithm which gives an even lower operation count for suitable values of N. Finally, § 6 includes a brief summary and discussion.

2. Prime factor algorithms. A thorough derivation of the family of prime factor FFT algorithms has been given by Burrus [1], [2]. Briefly, the DFT of length N is defined by

$$x(n)=\sum_{k=0}^{N-1} z(k)\omega_N^{kn}, \qquad 0\leqq n\leqq N-1, \tag{2.1}$$

where $x(n)$ and $z(k)$ are complex, and we use the notation

$$\omega_N=\exp(\pm 2i\pi/N). \tag{2.2}$$

Either sign may be taken in the definition (2.2).

We illustrate the derivation of the PFA by means of an example. Suppose that $N=N_1N_2$, where N_1 and N_2 are mutually prime. In this case [8, p. 250] we can find integers p, q, r, s $(0<p<N_1,\ 0<q<N_2,\ 0<r<N_2,\ 0<s<N_1)$ such that

$$pN_2=rN_1+1,\ qN_1=sN_2+1. \tag{2.3}$$

We use this "Chinese Remainder Theorem" (CRT) to define a mapping between the integers n, k $(0\leqq n\leqq N-1,\ 0\leqq k\leqq N-1)$ and the corresponding integer pairs (n_1, n_2) and (k_1, k_2) where $0\leqq n_1<N_1$, $0\leqq n_2<N_2$, $0\leqq k_1<N_1$, $0\leqq k_2<N_2$. As described in [14], we have a choice of two such mappings, the CRT map itself and the "Ruritanian" map [6]. In [14] the CRT map was chosen; this time we choose the Ruritanian map, for reasons which will become clear later.

Thus, the mapping is defined by

$$n_1=(pn) \bmod N_1, \quad n_2=(qn) \bmod N_2; \tag{2.4}$$

$$k_1=(pk) \bmod N_1, \quad k_2=(qk) \bmod N_2, \tag{2.5}$$

where p, q are defined in (2.3).

The inverse map is given by

$$n=(N_2n_1+N_1n_2) \bmod N; \tag{2.6}$$

$$k=(N_2k_1+N_1k_2) \bmod N. \tag{2.7}$$

An example for $N=40$ $(N_1=8,\ N_2=5)$ is shown in Table 1. The integer solutions of (2.3) are $p=5$, $q=2$, $r=3$, $s=3$. In fact, it is easy to construct the mapping in the form of a table without having to find these solutions. The entries in the first column increase from 0 in steps of $N/N_1(=N_2)$, while those in the first row increase from 0 in steps of $N/N_2(=N_1)$. The remaining columns (or rows) can then be filled in by using the same increment as in the first column (or row), and taking the results modulo N. We will use this technique later for indexing in the transform algorithm.

TABLE 1
The Ruritanian map for $N_1=8$, $N_2=5$.

			n_2		
n_1	0	1	2	3	4
0	0	8	16	24	32
1	5	13	21	29	37
2	10	18	26	34	2
3	15	23	31	39	7
4	20	28	36	4	12
5	25	33	1	9	17
6	30	38	6	14	22
7	35	3	11	19	27

As shown in [14], if we substitute (2.6) and (2.7) into (2.1), we obtain

$$x(n_1, n_2) = \sum_{k_2=0}^{N_2-1} \left[\sum_{k_1=0}^{N_1-1} z(k_1, k_2)\omega_{N_1}^{N_2 k_1 n_1} \right] \omega_{N_2}^{N_1 k_2 n_2}. \tag{2.8}$$

If it were not for the appearance of N_2 and N_1 multiplying the exponents, (2.8) would be exactly in the form of a two-dimensional DFT of dimension $N_1 \times N_2$, and the transform could be computed simply by performing N_2 DFTs of length N_1 in one dimension, followed by N_2 DFTs of length N_1 in the other (or vice versa, without changing the results). There are no "twiddle factors" between the two stages (hence the lower operation count than for the Cooley-Tukey algorithm). The output of each of the short one-dimensional transforms can overwrite the corresponding input, and the whole computation can thus be done in place. (Here we assume that N_1 and N_2 are "small" so that explicit DFT modules can be coded for these transform lengths.)

As further shown in [14], the appearance of N_2 and N_1 in the exponents simply *rotates* the transforms (applying a rotation r to a transform of length N_i means that instead of appearing in the original order $0, 1, 2, \cdots, N_i - 1$, the same results appear in the order $0, r, 2r, \cdots, (N_i - 1)r$, where these indices are to be interpreted modulo N_i). Moreover, these rotations can be incorporated by modifying certain constants which appear in the definitions of the short DFT algorithms of length N_1 and N_2. Detailed algorithms for rotated "small-n" DFTs are given in [14], [16].

The generalization to the case $N = N_1 N_2 \cdots N_k$, where all the factors are mutually prime, is straightforward; the one-dimensional transform of length N is equivalent to a k-dimensional transform in which the DFTs in each dimension are rotated as appropriate.

For future reference, it will be helpful to express these results in matrix form. We define W_N to be the DFT matrix of order N; thus element (j, k) of W_N is ω_N^{jk} (rows and columns of W_N are indexed from 0 to $N-1$), and (2.1) can be rewritten as $\underset{\sim}{x} = W_N \underset{\sim}{z}$. Further, we define $W_N^{[r]}$ to be the matrix W_N with all its elements raised to the power r. Then (2.8) corresponds to the factorization

$$W_N = R^{-1}(W_{N_2}^{[N_1]} \times W_{N_1}^{[N_2]})R \tag{2.9}$$

where × denotes the Kronecker (tensor) product and R is the permutation matrix which maps the integers $0 \leqq n \leqq N-1$ to the corresponding integer pairs (n_1, n_2) via the Ruritanian map (2.4). In general, if $N = N_1 N_2 \cdots N_k$, where all the N_i's are mutually prime, then the factorization is

$$W_N = R^{-1}(W_{N_k}^{[N/N_k]} \times \cdots \times W_{N_2}^{[N/N_2]} \times W_{N_1}^{[N/N_1]})R \tag{2.10}$$

where R now maps the array $x(n)$ to the corresponding k-dimensional array $x(n_1, n_2, \cdots, n_k)$ via the appropriate Ruritanian mapping. Notice that, to compute the transform, it is not necessary to reorder the input (or output) physically; the required mapping can be implemented implicitly via the indexing logic.

3. The generalized PFA. Using the results of the previous section, we can derive self-sorting in-place PFAs for certain values of N. For example, if $N = 60 = 3.4.5$, (2.10) becomes

$$W_{60} = R^{-1}(W_5^{[12]} \times W_4^{[15]} \times W_3^{[20]})R. \tag{3.1}$$

Since the rotation $[r]$ in $W_{N_i}^{[r]}$ can be taken modulo N_i, (3.1) simplifies to

$$W_{60} = R^{-1}(W_5^{[2]} \times W_4^{[3]} \times W_3^{[2]})R. \tag{3.2}$$

Algorithms are available for each of the short transforms W_5, W_4, W_3, including the rotations [14], [16], and these short transforms can be self-sorting and in-place.

Suppose now that $N = 129600 = 3^4 \times 4^3 \times 5^2$. If we arrange the factors in the form of a palindrome, for example

$$N = 3 \times 3 \times 4 \times 5 \times 4 \times 5 \times 4 \times 3 \times 3,$$

then we can use the results of [18, § 5] to obtain a self-sorting in-place algorithm for a transform of this length. However, the operation count would be the same as that for the corresponding mixed-radix Cooley–Tukey algorithm.

Alternatively, by (2.10) we have

$$W_{129600} = R^{-1}(W_{25}^{[5184]} \times W_{64}^{[2025]} \times W_{81}^{[1600]})R, \tag{3.3}$$

which, on reducing the rotations modulo N_i, becomes

$$W_{129600} = R^{-1}(W_{25}^{[9]} \times W_{64}^{[41]} \times W_{81}^{[61]})R. \tag{3.4}$$

In previous work on the PFA, it appeared that a self-sorting in-place implementation of (3.4) was not possible, since there was no way to perform self-sorting in-place transforms of lengths 25, 64, and 81. This is no longer true, thanks to the Johnson–Burrus self-sorting in-place radix-2 algorithm [7] and its generalization to other radices [18]. We need one final ingredient, so that the necessary rotations can be incorporated.

The required lemma was given in [14], in connection with deriving a rotated DFT module for $N_i = 9$. It can be stated as follows: if we have a radix-p algorithm for a transform of length $N = p^m$, then we can apply a rotation r by the following:

(1) Applying the rotation r (modulo p) to each radix-p module (e.g., by changing the multiplier constants); and

(2) raising all the twiddle factors to the power r.

For example, $W_{25}^{[9]}$ in (3.4) can be implemented by rotating each radix-5 module by $r' = 4$ ($=9$ modulo 5), and by raising all the twiddle factors to the power 9 (note that if the twiddle factors are stored in a precomputed list, this simply reorders them).

Thus, the generalized self-sorting in-place prime factor FFT algorithm (GPFA) for any $N = 2^p 3^q 5^r$ is constructed as follows:

(1) Use the Ruritanian mapping to convert to a three-dimensional transform of size $2^p \times 3^q \times 5^r$;

(2) Do the component one-dimensional transforms of length 2^p, 3^q, and 5^r using the generalized Johnson–Burrus scheme [18] with rotations incorporated as described above.

If the factors 2^p, 3^q, and 5^r are all "small," then the above algorithm is equivalent to the original PFA. If $N=2^p$, 3^q, or 5^r (i.e., only one of p, q, r is nonzero), then the transform remains one-dimensional and the above algorithm is equivalent to that in [18], with the same operation count as the Cooley-Tukey algorithm. In other cases, we have a new self-sorting in-place algorithm which takes advantage of the splitting of N into its mutually prime factors to reduce the operation count.

3.1. Operation counts. In [13], it was shown how to compute the operation counts for the mixed-radix Cooley-Tukey FFT algorithm. Here we adapt the formulae given in [13] to the case $N=2^p3^q5^r$, where the factors of 2 are treated in pairs (i.e., using a radix-4 algorithm). As in [13], the formulae assume that redundant multiplications are avoided when the twiddle factor is 1, but that there is no special treatment of other "simple" twiddle factors. The number of real additions is then given by

$$\mathcal{A}(N)=2N(1.375p+2.67q+4r-1)+2, \tag{3.5}$$

while the number of real multiplications is

$$\mathcal{M}(N)=2N(0.75p+2q+2.8r-2)+4. \tag{3.6}$$

In the case of the GPFA, we sum the operation counts from the transforms in each of the three dimensions: if $N=N_1N_2N_3$, where $N_1=2^p$, $N_2=3^q$, $N_3=5^r$, then the numbers of real additions and multiplications are given, respectively, by

$$\mathcal{A}(N)=\sum_{i=1}^{3}(N/N_i)\mathcal{A}(N_i), \tag{3.7}$$

$$\mathcal{M}(N)=\sum_{i=1}^{3}(N/N_i)\mathcal{M}(N_i), \tag{3.8}$$

where $\mathcal{A}(N_i)$ and $\mathcal{M}(N_i)$ are obtained from (3.5) and (3.6).

Examples for $N=3600=3^2\cdot 4^2\cdot 5^2$ are given in Table 2. In comparison with the Cooley-Tukey algorithm, the GPFA saves 10 percent of the additions and 33 percent of the multiplications. Thus, besides being both self-sorting and in-place, the GPFA has a significantly lower operation count than the conventional FFT when N contains a mixture of factors. Further examples of operation counts will be given in § 4. (The GPFA+algorithm is described in § 5.)

As an additional bonus, there is a useful saving in the storage required for tables of precalculated twiddle factors. Since a separate table is now used for the transforms in each of the three dimensions, the storage required becomes $2^p+3^q+5^r$ rather than $2^p\times 3^q\times 5^r$. For example, in the case $N=216000=3^3\times 4^3\times 5^3$ the twiddle factor storage requirement falls from 216000 to 216.

3.2. Indexing. To illustrate the indexing logic, we present a section of Fortran code which implements the first half of the radix-2 part of the algorithm for general

TABLE 2
Real operation counts for $N=3600$.

	+	*
Cooley-Tukey	128402	76324
GPFA	115538	50596
GPFA+	110250	40020

$N = 2^p 3^q 5^r$. Set $NI = 2^p$, $IP = p$. We first compute the required rotation and set up the table of twiddle factors:

```
   COMPLEX TRIGS(NI)
   DEL=4.0*ASIN(1.0)/FLOAT(NI)
   IROT=MOD((N/NI),NI)
   KK=0
   DO 10 K=1, NI
   ANGLE=FLOAT(KK)*DEL
   TRIGS(K)=CMPLX(COS(ANGLE),SIN(ANGLE))
   KK=KK+IROT
   IF (KK.GT.NI) KK=KK-NI
10 CONTINUE
```

The first $(p+1)/2$ radix-2 passes are then performed by the following code:

```
   COMPLEX X(N), W, Z
   NH=N/2
   INC=N/NI
   DO 50 L=1, (IP+1)/2
   LA=2**(L-1)
   JA=0
   JB=NH/LA
   KK=1
   DO 40 K=0, JB-1, INC
   W=TRIGS(KK)
   DO 30 J=K+1, N, N/LA
   IA=JA+J
   IB=JB+J
   DO 20 I=1, INC
   Z=W*(X(IA)-X(IB))
   X(IA)=X(IA)+X(IB)
   X(IB)=Z
   IA=IA+NI
   IF (IA.GT.N) IA=IA-N
   IB=IB+NI
   IF (IB.GT.N) IB=IB-N
20 CONTINUE
30 CONTINUE
   KK=KK+LA
40 CONTINUE
50 CONTINUE
```

The details of the indexing may be understood by comparing this code with Table 1 ($N=40$, $NI=8$). The three outer loops are very similar to the three loops of the code presented in [18], which performed the first half of a self-sorting in-place radix-2 algorithm. In the present case these loops set up base addresses in the first column of Table 1. The advantage of the Ruritanian map is that the entries in the first column (or row) increase monotonically in steps of N/NI; there is no need to compute these addresses modulo N. The innermost loop (DO 20) then steps *across* the table, performing one "butterfly" from each of the N/NI transforms of length NI. Only in this innermost loop is it necessary to update the addresses modulo N.

The code for the second half of the radix-2 part of the algorithm for general $N = 2^p 3^q 5^r$ may be obtained by similarly adapting the corresponding code given in [18], again inserting a new innermost loop which "traverses" the table. Notice that the indexing for the radix-2 part of the algorithm depends only on NI and N/NI; it is immaterial whether the Ruritanian map is one-, two- or three-dimensional. The same is true of the corresponding radix-3 and radix-5 parts.

4. Implementation on Cray Y-MP. The three Cray Assembly Language (CAL) routines described in [18], which implemented multiple self-sorting in-place complex FFTs for $N = 2^p$, 3^q and 5^r, have been generalized to implement the GPFA. The input parameter list for the modified version of each of the three routines now includes both $N = 2^p 3^q 5^r$ and the appropriate $NI = 2^p$, 3^q, or 5^r. The sequences of floating-point vector instructions are essentially unchanged, but the addressing and loop control now have to "navigate" the Ruritanian map as described in § 3.2, and are more complicated. However, this extra complexity has no impact on the vectorization, since the innermost loops (within the individual vector instructions) still step across the M simultaneous transforms being performed, with constant stride. For general $N = 2^p 3^q 5^r$, each of the three routines is now called in turn to implement the complete transform algorithm.

For the timing experiments presented below, 64 transforms were performed simultaneously. The experiments were run using a single processor of a Cray Y-MP with clock cycle 6.4 nsec. The new algorithm was compared with three other FFT routines, all CAL-coded and vectorized in the same way. In Tables 3-5, CFFT refers to the "old

TABLE 3
Timing comparisons for self-sorting transforms of length N.

	Real operations (+/*) per transform		Time per transform (μs)			Efficiency of GPFA (%)
N	CFFT	GPFA	CFFT	PFA	GPFA	
$120 = 2^3 \cdot 3 \cdot 5$	2382/1276	2028/508	19.1	14.1	14.2	97.1
$144 = 2^4 \cdot 3^2$	2834/1444	2594/964	22.5	18.0	18.0	98.0
$180 = 2^2 \cdot 3^2 \cdot 5$	3992/2272	3472/1232	31.1	24.0	24.0	98.2
$240 = 2^4 \cdot 3 \cdot 5$	5362/2788	4686/1436	41.5	32.3	32.3	98.5
$360 = 2^3 \cdot 3^2 \cdot 5$	9062/5260	7844/2644	71.5	54.1	53.9	98.9
$720 = 2^4 \cdot 3^2 \cdot 5$	19922/11236	17578/6584	153.6	120.6	120.2	99.4

TABLE 4
Timing comparisons for self-sorting transforms of length N.

		Time per transform (μs)			Efficiency of GPFA (%)
N	Real operations (+/*) per transform	CFFT	SSIP	GPFA	
$125 = 5^3$	2752/1604	22.2	18.9	18.9	98.8
$243 = 3^5$	5996/3892	47.3	41.4	41.1	99.2
$256 = 2^8$	5122/2052	39.9	35.1	35.3	98.7
$625 = 5^4$	18752/11504	150.1	127.9	127.9	99.7
$729 = 3^6$	21872/14584	171.3	150.4	149.2	99.7
$1024 = 2^{10}$	26114/11268	201.7	178.4	178.9	99.3
$2187 = 3^7$	77276/52492	602.3	530.6	526.6	99.8
$3125 = 5^5$	118752/75004	952.9	809.1	809.1	99.8
$4096 = 2^{12}$	126978/57348	978.5	866.4	868.4	99.4

TABLE 5
Timing comparisons for self-sorting transforms of length N.

N	Real operations (+/*) per transform		Time per transform (μs)		Efficiency of GPFA (%)
	CFFT	GPFA	CFFT	GPFA	
$216 = 2^3 \cdot 3^3$	4862/2812	4444/1868	39.2	30.6	98.7
$300 = 2^2 \cdot 3 \cdot 5^2$	7452/4264	6624/2608	58.3	45.5	99.0
$400 = 2^4 \cdot 5^2$	10002/5284	9282/3844	78.0	63.6	99.3
$600 = 2^3 \cdot 3 \cdot 5^2$	16702/9724	14748/5516	132.3	100.9	99.4
$900 = 2^2 \cdot 3^2 \cdot 5^2$	27152/16384	24272/10624	211.1	165.7	99.6
$1200 = 2^4 \cdot 3 \cdot 5^2$	36402/20644	32646/13132	282.6	222.9	99.6
$1500 = 2^2 \cdot 3 \cdot 5^3$	49252/29704	45024/21248	386.8	307.1	99.7
$1800 = 2^3 \cdot 3^2 \cdot 5^2$	59702/36364	53044/22148	470.5	362.0	99.7
$2400 = 2^5 \cdot 3 \cdot 5^2$	80002/46084	71742/29564	629.1	491.6	99.2
$3000 = 2^3 \cdot 3 \cdot 5^3$	107502/65404	97548/43996	825.5	665.6	99.7
$3200 = 2^7 \cdot 5^2$	107202/58244	100306/42852	842.2	684.0	99.7
$3600 = 2^4 \cdot 3^2 \cdot 5^2$	128402/76324	115538/50596	994.8	787.7	99.7

routine" used as a standard of comparison in [18]; this was originally coded for the Cray-1 and implements the self-sorting form of the Cooley-Tukey algorithm by alternating between the original data array and a work array of the same size. PFA refers to the implementation of the prime factor algorithm described in [15]. SSIP refers to the self-sorting in-place routines presented in [18], and GPFA is the new algorithm. Operation counts given in the tables were computed via (3.5)-(3.8), with appropriate modifications when p is odd (as in [18], the $N_i = 2^p$ part of the computation is done using a radix-4 algorithm with an extra radix-2 or radix-8 section called once if p is odd).

Table 3 presents timing comparisons for values of N suitable for the PFA. Operation counts for the PFA routine are in principle the same as those for GPFA. In practice they are slightly smaller in some cases, since the special DFT modules used in PFA for $N_i = 9$ and $N_i = 16$ include some economies which are not available to the general-purpose radix-3 and radix-4 DFT modules in GPFA. As shown in Table 3, the times for GPFA are almost exactly the same as those for PFA, and represent savings of 20-25 percent over CFFT. The last column of Table 3 gives a measure of the efficiency of GPFA, defined as in [18] as

$$\frac{\text{Minimum possible time}}{\text{Measured time}} \times 100 \text{ percent.}$$

The minimum possible time is computed on the basis of the number of real additions in the algorithm, and the efficiency is equivalent to the percentage of CPU time during which the floating-point addition unit is active. As demonstrated in Table 3, optimum use of the hardware is very nearly achieved. In particular, the rather complicated indexing is successfully hidden behind the floating-point arithmetic.

Table 4 presents timings for values of N suitable for the self-sorting in-place routines described in [18], i.e., of the form $2^p, 3^q$, or 5^r. The operation counts for the three routines are the same. The times for GPFA are almost exactly the same as those for SSIP (despite the more complicated indexing, which is redundant in this case as the Ruritanian map is now one-dimensional), representing savings of 10-15 percent over CFFT.

Table 5 presents timings for values of N which contain a mixture of factors but are unsuitable for the PFA because at least one of the mutually prime factors is too large. The times for GPFA represent savings of 18–24 percent over CFFT, demonstrating that the two major advantages of the prime factor algorithm (self-sorting in-place capability and reduced operation count) have been extended to general values of $N=2^p3^q5^r$.

The results presented in this section show that in GPFA we have a self-sorting in-place FFT algorithm which works for any $N=2^p3^q5^r$. If the transform length N is suitable for the PFA, then GPFA is essentially equivalent to it and runs just as fast. If $N=2^p, 3^q$, or 5^r, then GPFA is equivalent to the self-sorting in-place algorithms in [18], and runs just as fast. If N contains a mixture of factors but the PFA is not applicable, then GPFA has a lower operation count than the conventional FFT, and runs faster.

5. A further refinement. For suitable values of the transform length N, a refinement is possible which leads to a further reduction in the operation count. We illustrate the procedure by way of an example, for $N=3600=3^2\cdot 4^2\cdot 5^2$. From (2.10), we have

$$W_{3600}=R^{-1}(W_{25}^{[9]}\times W_{16}\times W_9^{[4]})R. \tag{5.1}$$

The heart of the algorithm is a three-dimensional transform; the Ruritanian mapping and the rotations are not germane to the present discussion, and the same procedure could be used with any multidimensional transform.

In § 3, (5.1) was implemented by performing transforms of length 9 along the first dimension, then transforms of length 16 along the second dimension, and finally transforms of length 25 along the third dimension. The transform of length 9 is a two-stage procedure using a radix-3 algorithm; it can be written

$$W_9^{[4]}=U_9(D_9T_9). \tag{5.2}$$

The first stage (D_9T_9) consists of three radix-3 "butterflies" (T_9) followed by multiplication by a diagonal matrix (D_9) of twiddle factors. The second stage (U_9) consists of another three radix-3 butterflies (since we are using a self-sorting in-place algorithm here, the butterflies in U_9 are coupled and some results are interchanged [18], but again this does not affect the argument). Similarly, we can write the transforms of length 16 and 25 as two-stage procedures using radix-4 and radix-5 algorithms, respectively:

$$W_{16}=U_{16}(D_{16}T_{16}), \tag{5.3}$$

$$W_{25}^{[9]}=U_{25}(D_{25}T_{25}), \tag{5.4}$$

where D_{16} and D_{25} are diagonal matrices of twiddle factors.

Substituting (5.2)–(5.4) into (5.1),

$$W_{3600}=R^{-1}((U_{25}D_{25}T_{25})\times(U_{16}D_{16}T_{16})\times(U_9D_9T_9))R. \tag{5.5}$$

Using the algebra of Kronecker (tensor) products, (5.5) becomes

$$W_{3600}=R^{-1}((U_{25}\times U_{16}\times U_9)(D_{25}\times D_{16}\times D_9)(T_{25}\times T_{16}\times T_9))R. \tag{5.6}$$

The interpretation of (5.6) is as follows. We can perform the first stage (T_9) of the radix-3 algorithm along the first dimension, without the twiddle factors, and follow it immediately by the first stage (T_{16}) of the radix-4 algorithm along the second dimension, and the first stage (T_{25}) of the radix-5 algorithm along the third dimension. Next, we apply the "delayed" twiddle factors, contained in the matrix $(D_{25}\times D_{16}\times D_9)$.

Finally, we perform the second stages (U_9, U_{16}, U_{25}) of the radix-3, radix-4, and radix-5 algorithms along the three dimensions in turn. The important point about this reordering of the calculation is that ($D_{25} \times D_{16} \times D_9$) is a single diagonal matrix (of order N) of twiddle factors; thus, the twiddle factors in the original algorithm (5.1) have been nested together. Each of these combined twiddle factors can be found somewhere in the precomputed list of length N as used for the Cooley-Tukey algorithm (unfortunately the reduction to a table of length $2^p + 3^q + 5^r$ is now lost). This nesting of the twiddle factors is analogous to that proposed for two-dimensional transforms in [13].

The number of twiddle factors in ($D_{25} \times D_{16} \times D_9$) that have the value 1 is given by the product of the numbers of 1's in the three matrices D_{25}, D_{16}, D_9, respectively. Assuming we can still pick these out and avoid redundant twiddle factor multiplications, the operation count for this refined algorithm (GPFA+), for $N = 3600$, is given in Table 2. Compared with GPFA, the new algorithm saves a further 5 percent of the additions and 20 percent of the multiplications.

For $N = 2^p 3^q 5^r$, nesting the twiddle factors in this fashion can be done whenever at least two of p, q, r are greater than one (so the Ruritanian map is at least two-dimensional, and the transforms in at least two of the dimensions consist of at least two stages with intervening twiddle factors). The practical aspects of indexing the refined algorithm in the general case have not yet been addressed.

6. Summary and discussion. In this paper we have developed a new FFT algorithm which works for any transform length of the form $N = 2^p 3^q 5^r$, and is always self-sorting and in-place. It includes the prime factor algorithm [14] and the self-sorting in-place version of the fixed-radix Cooley-Tukey algorithm [18] as special cases, and is always at least as fast as previously available algorithms.

A refinement, applicable for certain values of N, has already been proposed in § 5; it is natural to ask whether further refinements are possible. Promising directions include replacing the radix-2 part of the procedure by the split-radix algorithm [4], [11], and the radix-3 part by the algorithm of Suzuki, Sone, and Kido [12]; but in both cases it remains to be shown whether these algorithms can be made self-sorting and in-place, and generalized to include rotations. These refinements would reduce the multiplication count to a greater extent than the addition count—which is unfortunate from the viewpoint of implementation on Cray-like machines, where the multiplications are effectively already free of charge.

Another topic worth pursuing is the analogous generalization to any $N = 2^p 3^q 5^r$ of the self-sorting in-place real/half-complex prime factor FFT [17] and the corresponding fast sine and cosine transform algorithms [9].

Acknowledgments. The author wishes to thank Dr. Deborah Salmond of Cray Research Inc. for her help in running the timing experiments on the Cray Y-MP.

REFERENCES

[1] C. S. BURRUS, *Index mappings for multidimensional formulation of the* DFT *and convolution*, IEEE Trans. Acoust. Speech Signal Process., 25 (1977), pp. 239-242.

[2] C. S. BURRUS AND P. W. ESCHENBACHER, *An in-place, in-order prime factor* FFT *algorithm*, IEEE Trans. Acoust. Speech Signal Process., 29 (1981), pp. 806-817.

[3] 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] P. DUHAMEL, *Implementation of "split-radix"* FFT *algorithms for complex, real and real-symmetric data*, IEEE Trans. Acoust. Speech Signal Process., 34 (1986), pp. 285-295.

[5] I. J. GOOD, *The interaction algorithm and practical Fourier analysis*, J. Roy. Statist. Soc. Ser. B., 20 (1958), pp. 361-372.

[6] I. J. Good, *The relationship between two Fast Fourier Transforms*, IEEE Trans. Comput., 20 (1971), pp. 310–317.

[7] H. W. Johnson and C. S. Burrus, *An in-place in-order radix-*2 FFT, IEEE ICASSP-84 (1984), pp. 28A.2.1–28A.2.4.

[8] D. E. Knuth, *The Art of Computer Programming*: *Vol.* 2, *Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 1969.

[9] J. S. Otto, *Symmetric prime factor Fast Fourier Transform algorithms*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 419–431.

[10] J. H. Rothweiler, *Implementation of the in-order prime factor transform for various sizes*, IEEE Trans. Acoust. Speech Signal Process., 30 (1982), pp. 105–107.

[11] H. V. Sorensen, M. T. Heideman, and C. S. Burrus, *On computing the split-radix* FFT, IEEE Trans. Acoust. Speech Signal Process., 34 (1986), pp. 152–156.

[12] Y. Suzuki, T. Sone, and K. Kido, *A new* FFT *algorithm of radix* 3, 6 *and* 12, IEEE Trans. Acoust. Speech Signal Process., 34 (1986), pp. 380–383.

[13] C. Temperton, *Self-sorting mixed-radix Fast Fourier Transforms*, J. Comput. Phys., 52 (1983), pp. 1–23.

[14] ———, *Implementation of a self-sorting in-place prime factor* FFT *algorithm*, J. Comput. Phys., 58 (1985), pp. 283–299.

[15] ———, *Implementation of a prime factor* FFT *algorithm on Cray-*1, Parallel Computing, 6 (1988), pp. 99–108.

[16] ———, *A new set of minimum-add small-n rotated* DFT *modules*, J. Comput. Phys., 75 (1988), pp. 190–198.

[17] ———, *A self-sorting in-place prime factor real/half-complex* FFT *algorithm*, J. Comput. Phys., 75 (1988), pp. 199–216.

[18] ———, *Self-sorting in-place Fast Fourier Transforms*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 808–823.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 3, pp. 687-693, May 1992

A METHOD OF SMOOTH BIVARIATE INTERPOLATION FOR DATA GIVEN ON A GENERALIZED CURVILINEAR GRID*

DAVID W. ZINGG† AND MAURICE YARROW‡

Abstract. A method of locally bicubic interpolation is presented for data given at the nodes of a two-dimensional generalized curvilinear grid. The physical domain is transformed to a computational domain in which the grid is uniform and rectangular by a generalized curvilinear coordinate transformation. The metrics of the transformation are obtained by finite differences in the computational domain. Metric derivatives are determined by repeated application of the chain rule for partial differentiation. Given the metrics and the metric derivatives, the partial derivatives required to determine a locally bicubic interpolant can be estimated at each data point using finite differences in the computational domain. A bilinear transformation is used to analytically transform the individual quadrilateral cells in the physical domain into unit squares, thus allowing the use of simple formulas for bicubic interpolation.

Key words. interpolation, generalized curvilinear coordinates

AMS(MOS) subject classification. 65D05

1. Introduction. When function values are known at rectangular grid points in a plane, smooth bivariate interpolation can be efficiently performed using a method such as that presented by Akima [1]. However, when function values are given at irregularly distributed points in a plane, considerably more time-consuming methods are required. For example, the method developed by Akima [2] involves the following procedures: (1) triangulation, (2) determination of suitable nearest neighbors for each data point, (3) determination of the triangle in which each output point lies, (4) estimation of partial derivatives at each data point, and (5) calculation of the value of the interpolant at each output point. When the data points are given at rectangular grid points, the time-consuming procedures for triangulation and nearest-neighbor determination are not required. Furthermore, the procedures for determining the rectangle in which the output lies and for estimating partial derivatives are simplified.

In this paper, we develop a method of smooth bivariate interpolation for data given at the nodes of a generalized curvilinear grid. This problem is intermediate in complexity between the two problems above. Procedures for triangulation and nearest neighbor determination are not required, since the grid connectivity is implied by the data structure. In the present work, we assume that the cell in which a given output point lies is known. Procedures are described for estimation of partial derivatives and calculation of the value of the interpolant at the output points.

Generalized curvilinear grids are in common use in computational fluid dynamics. The grids are usually numerically generated using algebraic or partial differential equation techniques. Thompson, Warsi, and Mastin [7] give an excellent review of numerical grid generation techniques. Curvilinear grids are structured in that the grid nodes are aligned along curvilinear coordinate lines. Consequently, the grid can be mapped to an equispaced rectangular grid by a generalized curvilinear coordinate transformation. This facilitates the application of finite-difference methods for estimating partial derivatives. The physical coordinates are stored in two-dimensional arrays such that the curvilinear coordinates at each node are given by the array indices. Hence the grid connectivity is implied by the array indices.

* Received by the editors July 17, 1989; accepted for publication February 22, 1991.

† Institute of Aerospace Studies, University of Toronto, Downsview, Ontario M3H 5T6, Canada.

‡ Sterling Software, NASA Ames Research Center, Moffett Field, California 94035.

There are numerous applications for interpolation on curvilinear grids, including transferring data from one grid to another in embedded or overlapping grid schemes and graphical output. While linear interpolants are sometimes used [4], a smooth interpolant may be required for some applications. Yarrow and Mehta [8] found that a cubic interpolant increases the accuracy of the coupling between overlapping grids. They used an iterative method for interpolation on curvilinear grids. The method presented here avoids iteration and associated difficulties with convergence. The bicubic interpolant does not ensure monotonicity. Bilinear interpolation can be used in regions where this is an important consideration.

2. Generalized curvilinear coordinate transformation. The generalized curvilinear coordinate transformation maps a nonuniform, nonrectangular grid in the physical domain to a uniform, rectangular grid in the computational domain [3], as shown in Fig. 1. The physical coordinates (x, y) are mapped to the curvilinear coordinates (ξ, η) by the following general transformation:

$$\xi=\xi(x, y), \qquad \eta=\eta(x, y), \tag{1}$$

where we consider two space dimensions only. Note that the physical coordinates of the grid nodes are normally determined by a numerical grid-generation procedure. Consequently, no analytical mapping of the above form is defined. The mapping is defined at the grid nodes by assigning curvilinear coordinate values to each grid node. The mapping is chosen such that the resulting grid in the computational domain is uniform, rectangular, and of unit length, i.e.,

$$\Delta\xi=\Delta\eta=1. \tag{2}$$

Hence finite-difference representations of ∂_ξ and ∂_η are easily formulated. Each physical point is mapped to one point in the computational domain and vice versa except at topological singularities or cuts, where one physical point may be mapped to many computational points.

The chain rule for partial differentiation gives, in matrix form,

$$\begin{bmatrix}\partial_x\\ \partial_y\end{bmatrix}=\begin{bmatrix}\xi_x & \eta_x\\ \xi_y & \eta_y\end{bmatrix}\begin{bmatrix}\partial_\xi\\ \partial_\eta\end{bmatrix}. \tag{3}$$

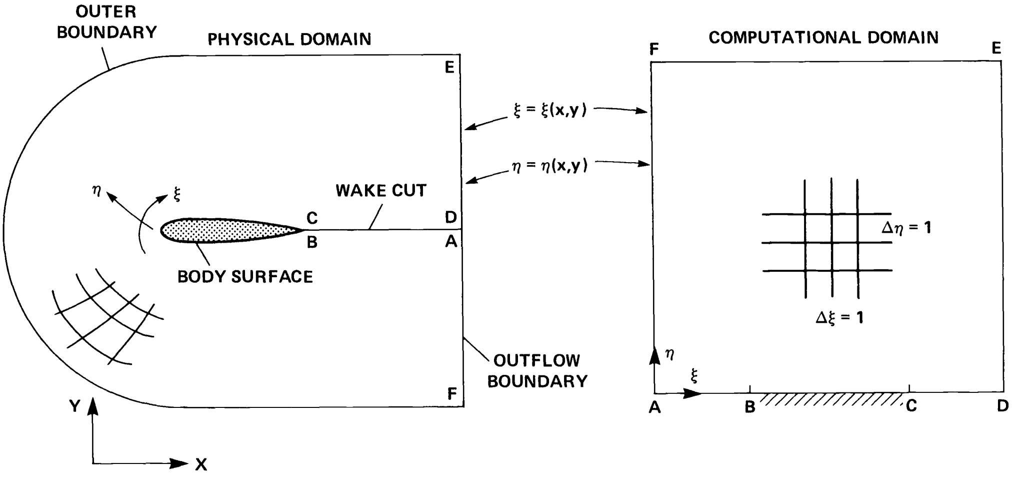


FIG. 1

Therefore, given the values of the elements of the above matrix, which are known as the metrics of the transformation, at each grid node, estimates of the partial derivatives with respect to the physical coordinates can be obtained from the simple finite-difference expressions for ∂_ξ and ∂_η. However, since the coordinate transformation is generally not known analytically, the metrics of the transformation must be determined using finite differences. Reversing the roles of the coordinate systems in (3), the chain rule also gives

$$(4)\qquad \begin{bmatrix} \partial_\xi \\ \partial_\eta \end{bmatrix} = \begin{bmatrix} x_\xi & y_\xi \\ x_\eta & y_\eta \end{bmatrix} \begin{bmatrix} \partial_x \\ \partial_y \end{bmatrix}.$$

Therefore, we must have

$$(5)\qquad \begin{bmatrix} \xi_x & \eta_x \\ \xi_y & \eta_y \end{bmatrix} = \begin{bmatrix} x_\xi & y_\xi \\ x_\eta & y_\eta \end{bmatrix}^{-1}$$

$$(6)\qquad = J \begin{bmatrix} y_\eta & -y_\xi \\ -x_\eta & x_\xi \end{bmatrix},$$

where

$$J^{-1} = x_\xi y_\eta - x_\eta y_\xi.$$

Equation (6) is the matrix form of the metric relations. All of the terms involving ∂_ξ and ∂_η are evaluated as finite differences. In the interior of the domain, standard three-point centered difference expressions are used. At boundaries, one-sided expressions must be used.

3. Interpolation method. Our objective is to smoothly interpolate a function $f(x, y)$ using values given at the nodes of a generalized curvilinear grid. The method proceeds in three steps. First, the required partial derivative information in the physical domain is determined using finite-difference approximations to the partial derivatives in the computational domain. The individual cells are then analytically transformed into unit squares by a bilinear transformation. Finally, a bicubic spline interpolant is determined within each unit square.

For locally bicubic interpolation, we require f_x, f_y, f_{xx}, f_{xy}, f_{yy}, as will be shown below. Using finite differences in computational space, we can determine f_ξ, f_η, $f_{\xi\xi}$, $f_{\xi\eta}$, $f_{\eta\eta}$. Applying the chain rule to (3) yields

$$(7)\qquad \begin{aligned} \partial_{xx} &= \xi_{xx}\partial_\xi + \eta_{xx}\partial_\eta + \xi_x^2\partial_{\xi\xi} + 2\xi_x\eta_x\partial_{\xi\eta} + \eta_x^2\partial_{\eta\eta}, \\ \partial_{xy} &= \xi_{xy}\partial_\xi + \eta_{xy}\partial_\eta + \xi_x\xi_y\partial_{\xi\xi} + (\xi_x\eta_y + \eta_x\xi_y)\partial_{\xi\eta} + \eta_x\eta_y\partial_{\eta\eta}, \\ \partial_{yy} &= \xi_{yy}\partial_\xi + \eta_{yy}\partial_\eta + \xi_y^2\partial_{\xi\xi} + 2\xi_y\eta_y\partial_{\xi\eta} + \eta_y^2\partial_{\eta\eta}. \end{aligned}$$

Equations (3) and (7) may be assembled as

$$(8)\qquad \boldsymbol{\partial}_{xy} = \boldsymbol{B}\boldsymbol{\partial}_{\xi\eta},$$

where

$$B = \begin{bmatrix} \xi_x & \eta_x & 0 & 0 & 0 \\ \xi_y & \eta_y & 0 & 0 & 0 \\ \xi_{xx} & \eta_{xx} & \xi_x^2 & 2\xi_x\eta_x & \eta_x^2 \\ \xi_{xy} & \eta_{xy} & \xi_x\xi_y & \xi_x\eta_y + \eta_x\xi_y & \eta_x\eta_y \\ \xi_{yy} & \eta_{yy} & \xi_y^2 & 2\xi_y\eta_y & \eta_y \end{bmatrix},$$

$$\boldsymbol{\partial}_{xy}=\begin{bmatrix}\partial_x\\ \partial_y\\ \partial_{xx}\\ \partial_{xy}\\ \partial_{yy}\end{bmatrix}, \quad\text{and}\quad \boldsymbol{\partial}_{\xi\eta}=\begin{bmatrix}\partial_\xi\\ \partial_\eta\\ \partial_{\xi\xi}\\ \partial_{\xi\eta}\\ \partial_{\eta\eta}\end{bmatrix}.$$

Similarly,

(9) $$\boldsymbol{\partial}_{\xi\eta}=C\boldsymbol{\partial}_{xy},$$

where

$$C=\begin{bmatrix} x_\xi & y_\xi & 0 & 0 & 0\\ x_\eta & y_\eta & 0 & 0 & 0\\ x_{\xi\xi} & y_{\xi\xi} & x_\xi^2 & 2x_\xi y_\xi & y_\xi^2\\ x_{\xi\eta} & y_{\xi\eta} & x_\xi x_\eta & x_\xi y_\eta+y_\xi x_\eta & y_\xi y_\eta\\ x_{\eta\eta} & y_{\eta\eta} & x_\eta^2 & 2x_\eta y_\eta & y_\eta^2\end{bmatrix}.$$

Therefore, $B=C^{-1}$.

In block matrix form, B can be written as

(10) $$B=\begin{bmatrix}B_1 & 0\\ B_2 & B_3\end{bmatrix},$$

where

$$B_1=\begin{bmatrix}\xi_x & \eta_x\\ \xi_y & \eta_y\end{bmatrix},\qquad B_2=\begin{bmatrix}\xi_{xx} & \eta_{xx}\\ \xi_{xy} & \eta_{xy}\\ \xi_{yy} & \eta_{yy}\end{bmatrix},$$

and

$$B_3=\begin{bmatrix}\xi_x^2 & 2\xi_x\eta_x & \eta_x^2\\ \xi_x\xi_y & \xi_x\eta_y+\eta_x\xi_y & \eta_x\eta_y\\ \xi_y^2 & 2\xi_y\eta_y & \eta_y^2\end{bmatrix}.$$

Similarly,

(11) $$C=\begin{bmatrix}C_1 & 0\\ C_2 & C_3\end{bmatrix},$$

where

$$C_1=\begin{bmatrix}x_\xi & y_\xi\\ x_\eta & y_\eta\end{bmatrix},\qquad C_2=\begin{bmatrix}x_{\xi\xi} & y_{\xi\xi}\\ x_{\xi\eta} & y_{\xi\eta}\\ x_{\eta\eta} & y_{\eta\eta}\end{bmatrix},$$

and

$$C_3=\begin{bmatrix}x_\xi^2 & 2x_\xi y_\xi & y_\xi^2\\ x_\xi x_\eta & x_\xi y_\eta+y_\xi x_\eta & y_\xi y_\eta\\ x_\eta^2 & 2x_\eta y_\eta & y_\eta^2\end{bmatrix}.$$

Now $B=C^{-1}$ gives

(12) $B_1=C_1^{-1}$ (the metric relations),

(13) $B_3=C_3^{-1}$,

and

$$B_2 = -C_3^{-1} C_2 C_1^{-1}. \tag{14}$$

Substituting (12) and (13) into equation (14) gives

$$B_2 = -B_3 C_2 B_1. \tag{15}$$

Finally, employing the metric relations gives

$$\begin{aligned}
\xi_{xx} &= -\frac{1}{J^3}[y_\eta^3 x_{\xi\xi} - 2y_\eta^2 y_\xi x_{\xi\eta} + y_\eta y_\xi^2 x_{\eta\eta} - x_\eta y_\eta^2 y_{\xi\xi} \\
&\qquad +2x_\eta y_\eta y_\xi y_{\xi\eta} - x_\xi y_\xi^2 y_{\eta\eta}], \\
\eta_{xx} &= -\frac{1}{J^3}[-y_\xi y_\eta^2 x_{\xi\xi} + 2y_\xi^2 y_\eta - y_\xi^3 x_{\eta\eta} + x_\xi y_\eta^2 y_{\xi\xi} \\
&\qquad -2x_\xi y_\eta y_\xi y_{\xi\eta} + x_\xi y_\xi^2 y_{\eta\eta}], \\
\xi_{xy} &= -\frac{1}{J^3}[-y_\eta^2 x_\eta x_{\xi\xi} + y_\eta(y_\eta x_\xi + x_\eta y_\xi) x_{\xi\eta} - y_\eta y_\xi x_\xi x_{\eta\eta} + x_\eta^2 y_\eta y_{\xi\xi} \\
&\qquad -x_\eta(y_\eta x_\xi + x_\eta y_\xi) y_{\xi\eta} + x_\eta y_\xi x_\xi y_{\eta\eta}], \\
\eta_{xy} &= -\frac{1}{J^3}[y_\xi y_\eta x_\eta x_{\xi\xi} - y_\xi(y_\eta x_\xi + x_\eta y_\xi) x_{\xi\eta} + y_\xi^2 x_\xi x_{\eta\eta} - x_\xi y_\eta x_\eta y_{\xi\xi} \\
&\qquad +x_\xi(y_\eta x_\xi + x_\eta y_\xi) y_{\xi\eta} - x_\xi^3 y_\xi y_{\eta\eta}], \\
\xi_{yy} &= -\frac{1}{J^3}[y_\eta x_\eta^2 x_{\xi\xi} - 2y_\eta x_\eta x_\xi x_{\xi\eta} + y_\eta x_\xi^2 x_{\eta\eta} - x_\eta^3 y_{\xi\xi} \\
&\qquad +2x_\eta^2 x_\xi y_{\xi\eta} - x_\eta x_\xi^2 y_{\eta\eta}], \\
\eta_{yy} &= -\frac{1}{J^3}[-y_\xi x_\eta^2 x_{\xi\xi} + 2y_\xi x_\eta x_\xi x_{\xi\eta} - y_\xi x_\xi^2 x_{\eta\eta} + x_\xi x_\eta^2 y_{\xi\xi} \\
&\qquad -2x_\eta x_\xi^2 y_{\xi\eta} + x_\xi^3 y_{\eta\eta}].
\end{aligned} \tag{16}$$

Therefore, all of the elements of matrix B can be expressed in terms of the elements of matrix C. As a result, $\partial_{xy} f$ can be determined given $\partial_{\xi\eta} f$. The elements of C and $\partial_{\xi\eta} f$ are approximated using three-point centered finite-difference formulas. At boundaries, one-sided formulas must be used.

Given f_x, f_y, f_{xx}, f_{xy}, f_{yy} at the corners of a given cell, the problem remains to interpolate f on an arbitrary quadrilateral. We now consider the new two-dimensional space given by (p, q) shown in Fig. 2 below. The quadrilateral in physical space is

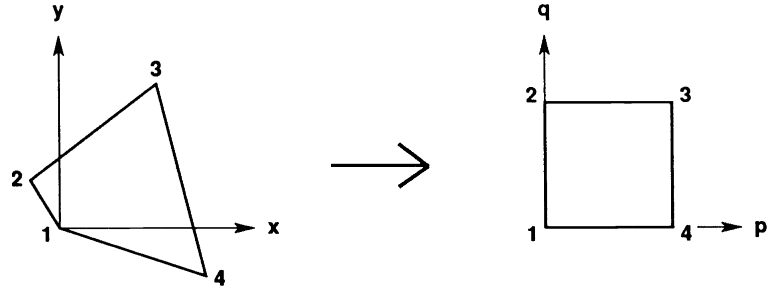


FIG. 2. *Two-dimensional space given by* (p, q).

related to a unit square in (p, q) space by a bilinear mapping, as follows. (Without loss of generality, we will assume the lower left-hand corner to be at $(0, 0)$.)

$$x = ap + bq + cpq, \tag{17a}$$

$$y = dp + eq + fpq. \tag{17b}$$

Note that a different mapping is used for each quadrilateral, or cell, in the grid. This contrasts with the generalized curvilinear coordinate transformation, in which a single smooth numerically generated mapping is applied to the entire grid. Consequently, analytical mappings can be utilized.

The coefficients of the bilinear mapping are given by

$$\begin{aligned} a &= x_4, \quad & b &= x_2, \quad & c &= x_3 - x_4 - x_2, \\ d &= y_4, \quad & e &= y_2, \quad & f &= y_3 - y_4 - y_2. \end{aligned} \tag{18}$$

The required derivatives in (p, q) space are given (as in (8)) by

$$\begin{aligned} \partial_p &= x_p \partial_x + y_p \partial_y, \\ \partial_q &= x_q \partial_x + y_q \partial_y, \\ \partial_{pq} &= x_{pq} \partial_x + y_{pq} \partial_y + x_p x_q \partial_{xx} + (x_p y_q + x_q y_p) \partial_{xy} + y_p y_q \partial_{yy}, \end{aligned} \tag{19}$$

where

$$x_p = a + cq, \quad y_p = d + fq, \quad x_q = b + cp, \quad y_q = e + fp, \quad x_{pq} = c, \quad y_{pq} = f,$$

and we have used $x_{pp} = x_{qq} = y_{pp} = y_{qq} = 0$.

Given a location in physical space (x_0, y_0), we must find the corresponding (p_0, q_0). From (17b), we have

$$q_0 = \frac{y_0 - dp_0}{e + fp_0}. \tag{20}$$

Substituting this into (17a) as follows:

$$x_0 = ap_0 + (b + cp_0)\left(\frac{y_0 - dp_0}{e + fp_0}\right) \tag{21}$$

gives the following quadratic for p_0:

$$p_0^2(cd - af) + p_0(-cy_0 + bd + x_0 f - ae) + (-y_0 b + ex_0) = 0. \tag{22}$$

Solution of (22) gives two values of p_0. The corresponding values of q_0 are determined from (20). One location (p_0, q_0) will not lie within the unit square and hence can be discarded.

Using f_x, f_y, f_{xx}, f_{xy}, f_{yy} as calculated from (8), we can determine f_p, f_q, and f_{pq} from (19) at each corner of the unit square in (p, q) space. We then employ the standard formula for bicubic interpolation within a unit square (see Appendix). The bicubic is finally evaluated at (p_0, q_0), as calculated from (22) and (20), respectively.

4. Conclusions. A method has been presented for smooth bivariate interpolation on a two-dimensional generalized curvilinear grid. Such grids are commonly used in computational fluid dynamics. Applications of the interpolation method include graphical output and problems in which data must be transferred from one grid to another. The method avoids any iterative procedures. Consequently, no difficulties with stability or convergence can occur in the application of the method.

Appendix: Locally bicubic interpolation on a unit square. In this appendix, we give a piecewise bicubic formulation for interpolation within a unit square. De Boor [5] presents a technique for determining the required partial derivatives at the corners of each cell in a rectangular grid such that C^2 continuity is obtained for the domain. In the formulation used here, the required partial derivatives at the corners of the unit square are given by centered difference formulas, leading to C^1 continuity, analogous to the univariate cubic Bessel spline given by de Boor [6]. This formulation has the advantage of producing a local interpolant.

The bicubic interpolant on a unit (p, q) square can be written as

$$f(p,q) \simeq \sum_{m,n=0}^{3} \gamma_{mn} p^m q^n, \qquad 0 \le p, q \le 1, \tag{A-1}$$

where

$$[\gamma_{mn}] = AKA^t,$$

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3 & -2 & 3 & -1 \\ 2 & 1 & -2 & 1 \end{bmatrix},$$

and

$$K = \begin{bmatrix} f(0,0) & f_q(0,0) & f(0,1) & f_q(0,1) \\ f_p(0,0) & f_{pq}(0,0) & f_p(0,1) & f_{pq}(0,1) \\ f(1,0) & f_q(1,0) & f(1,1) & f_q(1,1) \\ f_p(1,0) & f_{pq}(1,0) & f_p(1,1) & f_{pq}(1,1) \end{bmatrix}.$$

REFERENCES

[1] H. Akima, *A method of bivariate interpolation and smooth surface fitting based on local procedures*, Comm. ACM, 17 (1974), pp. 18-20.

[2] ———, *A method of bivariate interpolation and smooth surface fitting for irregularly distributed data points*, ACM Trans. Math. Software, 4 (1978), pp. 148-159.

[3] D. A. Anderson, J. C. Tannehill, and R. H. Pletcher, *Computational Fluid Mechanics and Heat Transfer*, Hemisphere, New York, 1984.

[4] J. A. Benek, P. G. Buning, and J. L. Steger, *A 3-d chimera grid embedding technique*, AIAA 7th Computational Fluid Dynamics Conference, Cincinnati, OH, AIAA paper 85-1523, June 1985.

[5] C. de Boor, *Bicubic spline interpolation*, J. Math. Phys., Vol. XLI, 41 (1962), pp. 212-218.

[6] ———, *A Practical Guide to Splines*, Springer-Verlag, New York, 1978.

[7] J. F. Thompson, Z. U. A. Warsi, and C. W. Mastin, *Numerical Grid Generation*, North-Holland, New York, 1985.

[8] M. Yarrow and U. B. Mehta, *Multiprocessing on supercomputers for computational aerodynamics*, NASA Technical Memorandum 102806, May 1990.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 3, pp. 694-722, May 1992

005

ITERATIVE SOLUTION OF LINEAR SYSTEMS ARISING FROM THE BOUNDARY INTEGRAL METHOD*

KENDALL E. ATKINSON† AND IVAN G. GRAHAM‡

Abstract. In this paper the behavior of some standard two-grid iterative schemes for the solution of linear systems arising from discretizations of second-kind boundary integral equations of potential theory is studied. When the boundary of the domain has corners, these schemes converge slowly, and in some cases even diverge. New iterative schemes are derived which always converge, and estimates for the accuracy of the solution to which they converge are given. Extensive numerical experiments are reported and discussed.

Key words. boundary integral equation, linear systems, iteration methods

AMS(MOS) subject classifications. primary 65R20; secondary 65N99, 65F10, 45L10, 35J05

1. Introduction. When the interior Dirichlet problem for Laplace's equation is solved by the indirect approach using a double layer potential representation, a second-kind integral equation results. If the boundary of the domain is smooth, this integral equation has an operator with a smooth kernel. The classical Nyström method (where the integral operator is approximated by a quadrature rule, and the resulting approximate equation collocated at the quadrature points) may be used to solve this equation. This Nyström method is fully discrete, and in fact can be shown to be equivalent to either a collocation or Galerkin method, with the necessary integrals done by an appropriate quadrature rule. Moreover, two-grid (or more generally multigrid) schemes for the iterative solution of the resulting linear equations always converge, providing the coarse grid is sufficiently fine.

When the boundary has corners, the situation is much less pleasant. However it has recently been shown in Graham and Chandler (1988), henceforth referred to as GC, that a (slightly modified) Nyström method (based on a composite quadrature rule) may still be applied to solve the integral equation, and that optimal uniform convergence is obtained when the mesh is graded near each corner.

This paper examines the performance of two-grid iterative schemes for the solution of the linear systems arising from the Nyström method of GC. We show that unless proper care is taken these iterative schemes may perform very poorly. Then we go on to propose and study three improved iterative schemes. In the process we extend the analysis of GC to a piecewise C^∞ boundary with any finite number of noncuspoidal corners. To simplify notation and to make the paper easier to read, however, we will limit our presentation here to the single-corner case.

In this section we introduce the integral equation, the Nyström method, and the standard two-grid scheme. Let Ω be a bounded simply connected planar domain, and let Γ denote its boundary. Introduce the *double layer potential*

$$Wu(\mathbf{x}) = \int_\Gamma G'(\mathbf{x}, \xi) u(\xi)\, d\Gamma(\xi), \qquad \mathbf{x} \in \Gamma. \tag{1.1}$$

* Received by the editors January 16, 1990; accepted for publication February 28, 1991. This work was supported in part by the University of New South Wales Australian Research Council program grant "Numerical Analysis for Integrals, Integral Equations, and Boundary Value Problems" and by the University of Queensland, Queensland, Australia. Most of this work was done while the authors were visiting the University of Queensland in 1988.

† Department of Mathematics, University of Iowa, Iowa City, Iowa 52242.

‡ School of Mathematical Sciences, University of Bath, Bath BA2 7AY, United Kingdom.

In this formula,

$$G(\mathbf{x}, \xi) = -\frac{1}{2\pi} \log |\mathbf{x} - \xi|,$$

$$G'(\mathbf{x}, \xi) = \frac{\partial}{\partial \mathbf{n}(\xi)} G(\mathbf{x}, \xi), \quad \mathbf{x} \in \mathbb{R}^2, \quad \xi \in \Gamma,$$

with $\partial/\partial\mathbf{n}(\xi)$ denoting the outward normal derivative at $\xi \in \Gamma$. The function u is called a density function.

We are interested in solving the integral equation

$$(1.2) \qquad u(\mathbf{x}) - 2Wu(\mathbf{x}) + \chi(\mathbf{x})u(\mathbf{x}) = -2g(\mathbf{x}), \qquad \mathbf{x} \in \Gamma.$$

In it, $\chi(\mathbf{x}) \in (-1, 1)$ and $[1 + \chi(\mathbf{x})]\pi$ is the exterior angle between the tangents to Γ at $\mathbf{y}$ as $\mathbf{y} \to \pm\mathbf{x}$. (Thus $\chi(\mathbf{x}) = 0$ except at corners of Γ.) This integral equation arises in solving Laplace's equation, in several ways.

For example, if u satisfies (1.2), then the function Wu of (1.1) solves the interior Dirichlet problem for Laplace's equation, with the given Dirichlet data $g(\mathbf{x})$. Alternatively, (1.2) arises in solving the exterior Neumann problem: find U satisfying

$$(1.3) \qquad \begin{aligned} &\Delta U(\mathbf{x}) = 0, \qquad \mathbf{x} \in \Omega_e = \mathbb{R}^2 \backslash \bar{\Omega}, \\ &\frac{\partial U(\mathbf{x})}{\partial \mathbf{n}} = h(\mathbf{x}), \qquad \mathbf{x} \in \Gamma, \\ &U(\mathbf{x}) = O(\log |\mathbf{x}|) \quad \text{as } |\mathbf{x}| \to \infty. \end{aligned}$$

Green's representation formula

$$(1.4) \qquad U(\mathbf{x}) = \int_\Gamma [u(\xi)G'(\mathbf{x}, \xi) - h(\xi)G(\mathbf{x}, \xi)] \, d\Gamma(\xi), \qquad \mathbf{x} \in \Omega_e$$

gives a solution to (1.3), provided $u(\mathbf{x})$ satisfies (1.2) with

$$g(\mathbf{x}) = \int_\Gamma h(\xi)G(\mathbf{x}, \xi) \, d\Gamma(\xi).$$

Let Γ be parameterized by $\mathbf{x}(s)$, $s \in [-1, 1]$, and for notational convenience extend $\mathbf{x}(s)$ to a 2-periodic function on all of $\mathbb{R}$. Assume $x \in C^\infty[-1, 1]\backslash\{0\}$, but allow Γ to have a single corner at $\mathbf{0} = \mathbf{x}_0 = \mathbf{x}(0)$. We shall assume throughout that s is proportional to the arc length along Γ, so that $|\mathbf{x}'(s)| = |\Gamma|/2$, where $|\Gamma|$ = length of Γ. It is straightforward to extend the results to any parameterization $\mathbf{x} \in C^\infty([-1, 1]\backslash\{0\})$ satisfying $|\mathbf{x}'(s)| > 0$, $s \in [-1, 1]\backslash\{0\}$.

For any integer $n \geqq 1$ and real number $q \geqq 1$, introduce the following graded mesh for $[-1, 1]$ and Γ:

$$(1.5) \qquad \begin{aligned} &s_j^{(n)} = \left(\frac{j}{n}\right)^q, \quad s_{2n-j}^{(n)} = -s_j^{(n)}, \quad j = 0, 1, \cdots, n, \\ &\mathbf{x}_j^{(n)} = \mathbf{x}(s_j^{(n)}), \qquad j = 0, 1, \cdots, 2n. \end{aligned}$$

Extend these points periodically with

$$s_{j+2n}^{(n)} = s_j^{(n)}, \quad x_{j+2n}^{(n)} = x_j^{(n)}, \quad -\infty < j < \infty.$$

Let

$$(1.6) \qquad \int_0^1 \varphi(\eta) \, d\eta \doteq \sum_{j=1}^r \omega_j \varphi(\eta_j)$$

be an interpolatory quadrature rule of order R, where $r \leqq R \leqq 2r$, with weights $\{\omega_j\}$ and nodes

$$0 \leqq \eta_1 < \eta_2 < \cdots < \eta_r \leqq 1.$$

Throughout we assume the weights $\{\omega_j\}$ are positive. This is the case with almost all practical interpolatory rules.

The Nyström approximation W_n to the operator W of (1.2) is obtained by applying (1.6) to each of the subintervals of Γ with breakpoints (1.5), that is,

$$W_n\phi(\mathbf{x}) = \sum_{i=1}^{2n} \sum_{j=1}^{r} \omega_{ij} G'(\mathbf{x}, \mathbf{x}_{ij}^{(n)}) \phi(\mathbf{x}_{ij}^{(n)}), \tag{1.7}$$

where the *quadrature points* on Γ are

$$\mathbf{x}_{ij}^{(n)} = \mathbf{x}(s_{ij}^{(n)}) \quad \text{with } s_{ij}^{(n)} = (1-\eta_j)s_{i-1}^{(n)} + \eta_j s_i^{(n)},$$

and the weights are

$$\omega_{ij} = \omega_j (s_i^{(n)} - s_{i-1}^{(n)}) \frac{|\Gamma|}{2}.$$

Rather than approximating W by W_n directly in (1.2), we first rearrange (1.2) slightly. As we shall see later, this rearrangement ensures a uniformly convergent approximate solution. Let

$$\tilde{u}(\mathbf{x}) = u(\mathbf{x}) - u(\mathbf{0}), \qquad \mathbf{x} \in \Gamma. \tag{1.8}$$

Substituting into (1.2) gives

$$(1+\chi)\tilde{u} - 2W\tilde{u} + ((1+\chi) - 2W1)u(\mathbf{0}) = -2g, \tag{1.9}$$

where 1 denotes the unit functions. Now it is well known (e.g., Jaswon and Symm (1977)) that $2W1 = -(1-\chi)$ on Γ, so (1.9) reduces to

$$(1+\chi)\tilde{u} - 2W\tilde{u} + 2u(\mathbf{0}) = -2g$$

or equivalently (since $\tilde{u}(\mathbf{0}) = 0$, and $\chi(\mathbf{x}) = 0$ when $\mathbf{x} \neq \mathbf{0}$)

$$\tilde{u} - 2W\tilde{u} + 2u(\mathbf{0}) = -2g. \tag{1.10}$$

Our numerical method for (1.10) (or equivalently (1.2)) is to seek u_n such that $u_n(\mathbf{0})$ and $\tilde{u}_n := u_n - u_n(\mathbf{0})$ satisfy

$$\tilde{u}_n - 2W_n\tilde{u}_n + 2u_n(\mathbf{0}) = -2g. \tag{1.11}$$

Observe that $\tilde{u}_n(\mathbf{0}) = 0$, so even if $\mathbf{0}$ is a quadrature point, the sum $W_n\tilde{u}_n$ contains no term in $G'(\mathbf{x}, 0)$ and hence is continuous on Γ. Thus, collocation at the points $\{x_{ij}^{(n)}\} \cup \{\mathbf{0}\}$ reduces (1.11) to a linear system for the unknowns $\{\tilde{u}_n(\mathbf{x}_{ij}^{(n)})\} \cup \{u_n(\mathbf{0})\}$. Once this is solved, (1.11) gives a formula for $\tilde{u}_n(\mathbf{x})$ at any $\mathbf{x} \in \Gamma$; this is the *Nyström interpolation formula.* Having found $\tilde{u}_n$ and $u_n(\mathbf{0})$, the approximation u_n to the exact solution u of (1.2) is then recovered by $u_n = \tilde{u}_n + u_n(\mathbf{0})$.

We analyze (1.11) in § 2. To prove stability, we need the following slight modification of the method. For some integer $i^* \geqq 0$ define W_n not by (1.7), but rather by

$$W_n\phi(\mathbf{x}) = \sum_{i=i^*+1}^{2n-i^*} \sum_{j=1}^{r} \omega_{ij} G'(\mathbf{x}, \mathbf{x}_{ij}^{(n)}) \phi(\mathbf{x}_{ij}^{(n)}).$$

Thus if $i^*=0$, W_n is unchanged. Then we shall show in § 2 that there is an $i^* \geqq 0$ and a $q>1$ such that (1.11) defines u_n uniquely for n sufficiently large, and u_n converges optimally, i.e.,

$$\|u-u_n\|_\infty = O\left(\frac{1}{n^R}\right) \quad \text{as } n \to \infty. \tag{1.12}$$

The introduction of i^* is an artifact of the stability proof of CG, and has no effect on the practical direct solution of (1.11). In fact, (1.12) has been observed to hold with $i^*=0$ in all practical computations. However, as we shall see in § 3, i^* turns out to be useful in a different context: as a parameter for the acceleration of the convergence of iterative schemes for solving (1.11).

To analyze (1.11) and to discuss its iterative solution, it is convenient to rearrange (1.10) into a 2×2 system for the unknown $(\tilde{u}, u(\mathbf{0}))$, and to think of (1.11) as a corresponding approximate system for $(\tilde{u}_n, u_n(\mathbf{0}))$. To do this, evaluate (1.10) at $\mathbf{0}$, and subtract the result from (1.10) to obtain the system (which is equivalent to (1.2))

$$(I-\mathscr{L})\begin{bmatrix}\tilde{u}\\ u(\mathbf{0})\end{bmatrix} = \begin{bmatrix}-2\tilde{g}\\ -2g(\mathbf{0})\end{bmatrix}, \tag{1.13}$$

where $\tilde{g}=g-g(\mathbf{0})$ and $\mathscr{L}$ is the operator

$$\mathscr{L}\begin{bmatrix}\tilde{u}\\ u(\mathbf{0})\end{bmatrix} = \begin{bmatrix}2W\tilde{u}-2(W\tilde{u})(\mathbf{0})\\ 2(W\tilde{u})(\mathbf{0})-u(\mathbf{0})\end{bmatrix}.$$

Carry out the same process on (1.11) to obtain

$$(I-\mathscr{L}_n)\begin{bmatrix}\tilde{u}_n\\ u_n(\mathbf{0})\end{bmatrix} = \begin{bmatrix}-2\tilde{g}\\ -2g(\mathbf{0})\end{bmatrix}, \tag{1.14}$$

where $\mathscr{L}_n$ is defined by replacing W by W_n in $\mathscr{L}$.

System (1.14) is in an appropriate form for solution by two-grid (or more generally multigrid) iterative solution procedures. In this paper we discuss some two-grid schemes which are closely related to a method studied in Atkinson (1973). As a motivation, we first describe this latter method.

We first write (1.13) and (1.14) (with n replaced by m) as

$$(I-\mathscr{L})v=z, \tag{1.15}$$

$$(I-\mathscr{L}_m)v_m=z, \tag{1.16}$$

with $v=(\tilde{u}, u(\mathbf{0}))^T$, etc. In § 2 we cast these equations in a Banach space $\mathscr{B}$, with norm $\|\cdot\|$ designed so that $\|v\|=\|(\tilde{u}, u(\mathbf{0}))^T\|=\|u\|_\infty$. Suppose that m is the discretization level of the system we wish to solve iteratively. Assume that we can solve exactly an equation of smaller level n, $(I-\mathscr{L}_n)v_n=y$, for any right-hand side y. Let $v_m^{(0)}$ be an initial guess for the solution v_m of (1.16). Calculate the *residual* or *defect*

$$d_m = z-(1-\mathscr{L}_m)v_m^{(0)}.$$

Then the error $e_m=v_m-v_m^{(0)}$ satisfies

$$(I-\mathscr{L}_m)e_m=d_m.$$

Write this as $e_m=d_m+\delta_m$, with $\delta_m=\mathscr{L}_m d_m$. Then δ_m satisfies the equation

$$(I-\mathscr{L}_m)\delta_m=\mathscr{L}_m d_m.$$

Approximate this with the equation

$$(I-\mathscr{L}_n)\delta_n=\mathscr{L}_m d_m,$$

which is solved directly for δ_n. The improved approximation to v_m is then given by

$$v_m^{(1)}=v_m^{(0)}+d_m+\delta_n.$$

Continuing iteratively, we obtain the following algorithm, which is easily recognizable as a simple multigrid scheme (using only two grids).

ALGORITHM 1.

(1.17a)	Input:	$v_m^{(j)}$
(1.17b)	Compute residual:	$d_m^{(j)}=z-(I-\mathscr{L}_m)v_m^{(j)}$
(1.17c)	Smoothing step:	Find $\mathscr{L}_m d_m^{(j)}$
(1.17d)	Coarse-grid correction:	$\delta_n^{(j)}=(I-\mathscr{L}_n)^{-1}\mathscr{L}_m d_m^{(j)}$
(1.17e)	Output:	$v_m^{(j+1)}=v_m^{(j)}+d_m^{(j)}+\delta_n^{(j)}.$

This scheme can be put into the form of the two-grid algorithm of Hackbusch (1985, p. 309). Some simple manipulations on (1.17a–e) lead to the well-known error expression

$$(v_m-v_m^{(j+1)})=(I-\mathscr{L}_n)^{-1}(\mathscr{L}_m-\mathscr{L}_n)\mathscr{L}_m(v_m-v_m^{(j)}). \tag{1.18}$$

Suppose Γ is smooth. Write the direct Nyström approximation of (1.2) as

$$(I-\mathscr{L}_m)u_m=-2g, \tag{1.19}$$

and apply Algorithm 1 to solve (1.19). Then (1.18) can be used to show

$$\|u_m-u_m^{(j+1)}\|_\infty \leqq C_{m,n}\|u_m-u_m^{(j)}\|_\infty \tag{1.20}$$

with $C_{m,n}<1$, for all $m>n$, provided n is sufficiently large (Atkinson (1973)). However, such estimates cannot be obtained for (1.17a–e) applied to (1.16). Indeed, in § 3 we give an example where the presence of a sharp corner diminishes the smoothing property of the operator $\mathscr{L}_m$ to such an extent that (1.17a–e) diverge. This paper describes some new two-grid methods for (1.16), which converge even in the presence of corners. This convergence will not be so interesting unless it is achieved without too much degradation in the accuracy of v_m, and without substantial escalation of the operation count of the iteration. Thus, we distinguish three properties which we would like an "ideal" iterative scheme to possess:

(P1) Convergence in the presence of any corners.

(P2) Convergence to solutions $v_m=(\tilde{u}_m, u_m(\mathbf{0}))^T$ which satisfy the (optimal) order of accuracy estimate (1.12).

(P3) Convergence achieved with an operation count comparable to that of the basic scheme (1.17).

As we shall see in §§ 3 and 4, there is a certain amount of "trade-off" between these three properties.

There is a large literature on the multigrid (or multilevel) solution of equations such as (1.2), particularly when Γ is smooth. For extensive surveys, see Hackbusch (1985) and references therein, and also Mandel (1985). A fast Fourier method is given in Reichel (1988). There are also several papers discussing the case of nonsmooth Γ, but using methods different from those proposed in the present paper. In particular, we mention Hebeker (1986), (1988) and Schippers (1982), (1985), (1987). We compare these with our own work later in the paper, but we remark here that previously most discretizations of (1.2) (when Γ has corners) were obtained by Galerkin or collocation schemes. These latter schemes generally necessitate further (numerical) integration. With an appropriate choice of quadrature rules, these can be shown to be equivalent to Nyström schemes (see Atkinson and Bogomolony (1987)). Thus this present work covers many common discretization strategies for (1.2), with the extra novelty here being that the preliminary rearrangement (1.10) is being used to improve the accuracy of the integration around the corner.

At this stage it seems important to justify why our main interest here is in two-grid schemes rather than in the more general multigrid method. Recall that we are interested, in the long run, in good approximations to the solution u of (1.2), or in quantities derived from u by postprocessing (potentials in the interior, stress intensity factors, etc.). It is not known a priori what value of m is needed in (1.16) to obtain these quantities within a prescribed tolerance. So in practice the problem would first be solved for a (moderate) value (or values) of m, the error would be estimated by some a posteriori technique (extrapolation or something more sophisticated), and then m would be increased if necessary.

The two-grid iteration (1.17a–e) fits in very nicely with this philosophy. A possible implementation is to first solve (1.16) directly for some moderate sized $m = n = n_0$, say, and then apply (1.17a–e) recursively with $m := 2n$ each time. This yields solutions on a sequence of *increasing* levels $\{2^j n_0 : j = 0, 1, 2, \cdots\}$, with termination when a prescribed accuracy is achieved.

The multigrid technique (e.g., Hackbusch (1985, § 16.5)), on the other hand, would solve (1.16) at level m by recursively applying the two-grid scheme on a nested sequence of *decreasing* levels. This yields a more complicated code, but achieves a better rate of convergence. Specifically it may be shown that when applied to (1.19), the multigrid method yields a sequence $u_m^{(j)}$ with property

$$(1.21) \qquad \|u_m - u_m^{(j+1)}\|_\infty \leqq C'_m \|u_m - u_m^{(j)}\|_\infty,$$

where, in contrast with (1.20), $C'_m \to 0$ as $m \to \infty$. Again this result does not apply for the iterative solution of (1.16), but we would argue that, even in the case of smooth Γ, the power of the full multigrid scheme may be unnecessary. To see this, consider (1.17a–e) applied to (1.19) with $u_m^{(0)} = u_{m/2}$, and suppose estimate (1.12) is sharp, i.e., for suitable constants C_1 and C_2,

$$C_1(1/m^R) \leqq \|u - u_m\|_\infty \leqq C_2(1/m^R),$$

and

$$C_1 2^R(1/m^R) \leqq \|u - u_{m/2}\|_\infty \leqq C_2 2^R(1/m^R).$$

Then

$$\begin{aligned}\|u - u_m\|_\infty &\geqq C_1 C_2^{-1} 2^{-R} \|u - u_{m/2}\|_\infty \\ &\geqq C_1 C_2^{-1} 2^{-R} (\|u_m - u_{m/2}\|_\infty - \|u - u_m\|_\infty),\end{aligned}$$

so that

$$(1 + C_1 C_2^{-1} 2^{-R}) \|u - u_m\|_\infty \geqq C_1 C_2^{-1} 2^{-R} \|u_m - u_{m/2}\|_\infty.$$

But, for sufficiently large values of n, (1.20) implies

$$\begin{aligned}\|u_m - u_m^{(j)}\|_\infty &\leqq (C_{m,n})^j \|u_m - u_m^{(0)}\|_\infty \\ &= (C_{m,n})^j \|u_m - u_{m/2}\|_\infty \\ &\leqq (C_{m,n})^j (C_1^{-1} C_2 2^R + 1) \|u - u_m\|_\infty.\end{aligned}$$

Clearly the iteration should be stopped when $\|u_m - u_m^{(j)}\|_\infty$ is less significant than $\|u-u_m\|_\infty$, i.e., when

$$(1.22) \qquad (C_{m,n})^j < (C_1^{-1}C_2 2^R + 1)^{-1}.$$

Suppose (as seems reasonable for smooth implementations) $C_{m,m/2} = 0.1$, $C_2 C_1^{-1} = 10$ (conservatively large), and the trapezoidal rule ($R = 2$) is used. Then (1.22) requires only $j = 2$. In this context, the significantly greater programming complexity needed to achieve (1.21) seems unnecessary, and so we concentrate our attention in this paper on two-grid methods.

Finally we remark that in (1.17d) it is implicitly assumed that Nyström interpolation is used to transform back and forth between coarse and fine grids. This is not necessary; prolongation and restriction operators may be used, and we shall discuss these in § 4. We remark that Hebeker (1986), (1988) has shown how a judicious choice of such prolongations and restrictions can lead to an accelerated two-grid scheme, satisfying (1.21) with $C'_m \to 0$ as $m \to \infty$.

2. Theory of the Nyström method. The discussion here mirrors that of GC, where the Nyström solution of (1.2) when Γ is polygonal was analyzed. Here we have only one corner, but extra work is necessary to deal with the curved boundary. Our exposition could be extended in a straightforward (but tedious) manner to a piecewise C^∞ boundary with a finite number of corners.

For the rest of the paper, D denotes differentiation, and C denotes a generic constant. Let $C(\Gamma)$ denote the space of continuous real-valued functions on Γ with the uniform norm, and let $C_0(\Gamma)$ denote the subspace $\{v \in C(\Gamma)\colon v(\mathbf{0}) = 0\}$, normed again with $\|\cdot\|_\infty$. We also use $\|\cdot\|_\infty$ to denote the norm of any bounded linear operator on $C(\Gamma)$.

The first step in the analysis is to introduce an operator $\mathcal{K}$ which contains a simpler form of the nonsmooth part of the operator $2W$ of (1.2). To this end, define the function $\mathbf{y}:[-1,1]\to\mathbb{R}^2$ by

$$(2.1) \qquad y(s) = \begin{cases} \left(-\dfrac{s|\Gamma|}{2}\cos(1-\chi_0)\pi,\ -\dfrac{s|\Gamma|}{2}\sin(1-\chi_0)\pi\right), & s\in[-1,0], \\ \left(\dfrac{s|\Gamma|}{2}, 0\right), & s \in [0,1]. \end{cases}$$

Thus $\mathbf{y}(s)$ is a parameterization of the open "wedge" $\tilde{\Gamma}$ consisting of two straight lines, each of length $|\Gamma|/2$, meeting at an exterior angle $(1+\chi_0)\pi$ at $\mathbf{0}$. Then for $v:\Gamma\to\mathbb{R}$, define $\mathcal{K}v:\Gamma\to\mathbb{R}$ by

$$(2.2) \qquad \mathcal{K}v(\mathbf{x}(s)) = 2\int_{\tilde{\Gamma}} G'(\mathbf{y}(s),\mathbf{y}(\sigma))v(\mathbf{x}(\sigma))\,d\Gamma(\mathbf{x}(\sigma)), \qquad s\in[-1,1].$$

Since the relationship between $\mathbf{x}(s)$ and $\mathbf{y}(s)$ is bijective, $\mathcal{K}v:\Gamma\to R$ is a well-defined function. The kernel $G'(\mathbf{y}(s),\mathbf{y}(\sigma))$ may be computed easily (e.g., see Atkinson and deHoog (1984)):

$$(2.3) \qquad \mathcal{K}v(\mathbf{x}(s)) = \begin{cases} \displaystyle\int_0^1 K\left(\frac{s}{\sigma}\right)v(\mathbf{x}(\sigma))\frac{d\sigma}{\sigma}, & s\in[-1,0], \\ \displaystyle -\int_{-1}^0 K\left(\frac{s}{\sigma}\right)v(\mathbf{x}(\sigma))\frac{d\sigma}{\sigma}, & s\in[0,1], \end{cases}$$

where

$$K(\sigma)=\frac{\sin(\chi_0\pi)}{\pi}\left[\frac{\sigma}{1-2\sigma\cos(\chi_0\pi)+\sigma^2}\right].$$

The properties of $\mathcal{K}$ are well understood. In particular,

$$\lim_{s\to 0\pm}\mathcal{K}v(\mathbf{x}(s))=-\chi_0 v(\mathbf{0}). \tag{2.4}$$

Using this limit to define $\mathcal{K}v(\mathbf{0})$, $\mathcal{K}$ is a bounded linear operator on $C(\Gamma)$ to $C(\Gamma)$, with

$$\|\mathcal{K}\|_\infty=|\chi_0|<1. \tag{2.5}$$

By (2.4), $\mathcal{K}$ is also bounded on $C_0(\Gamma)$ with a norm as in (2.5). It was shown by Atkinson and de Hoog (1984) that $\mathcal{N}:=2W-\mathcal{K}$ is compact on $C(\Gamma)$.

We study systems (1.13), (1.14) in the Banach space $\mathcal{B}=C_0(\Gamma)\times\mathbb{R}$, with the norm

$$\left\|\begin{bmatrix} v\\ \nu\end{bmatrix}\right\|=\|v+\nu\|_\infty.$$

Recall the definition of $\mathcal{L}$ following (1.13). Using the decomposition $2W=\mathcal{K}+\mathcal{N}$ and (2.4), we have $\mathcal{L}=\mathcal{S}+\mathcal{M}$, where

$$\mathcal{S}\begin{bmatrix}\tilde u\\ u(\mathbf{0})\end{bmatrix}=\begin{bmatrix}\mathcal{K}\tilde u\\ 0\end{bmatrix},$$

$$\mathcal{M}\begin{bmatrix}\tilde u\\ u(\mathbf{0})\end{bmatrix}=\begin{bmatrix}\mathcal{N}\tilde u-(\mathcal{N}\tilde u)(\mathbf{0})\\ (\mathcal{N}\tilde u)(\mathbf{0})-u(\mathbf{0})\end{bmatrix}.$$

The operator $\mathcal{M}$ is a finite rank-bounded perturbation of the compact operator

$$\begin{bmatrix}\tilde u\\ u(\mathbf{0})\end{bmatrix}\to\begin{bmatrix}\mathcal{N}\tilde u\\ 0\end{bmatrix},$$

and thus $\mathcal{M}$ is compact from $\mathcal{B}$ into $\mathcal{B}$. The operator $\mathcal{S}$ is a contraction, with norm as in (2.5); thus

$$\|(I-\mathcal{S})^{-1}\|\leqq\frac{1}{1-|\chi_0|}. \tag{2.6}$$

Combining these results, the Fredholm Alternative Theorem can be applied to (1.13). Using the equivalence of (1.13) and (1.2), we can show that the only solution of the homogeneous form of (1.13) is the zero solution. That then implies the existence and boundedness of the inverse of $(I-\mathcal{L})$ from $\mathcal{B}$ onto $\mathcal{B}$.

Now we discuss the stability and convergence of a (slightly modified) version of method (1.11), obtaining the result (1.12) described in § 1. Define

$$K(s,\sigma)=\begin{Bmatrix}0 & \sigma\in[-1,0]\\ K(s/\sigma)1/\sigma & \sigma\in[0,1]\end{Bmatrix},\qquad s\in[-1,0],$$

$$K(s,\sigma)=\begin{Bmatrix}-K(s/\sigma)1/\sigma & \sigma\in[-1,0]\\ 0 & \sigma\in[0,1]\end{Bmatrix},\qquad s\in[0,1],$$

so that

$$\mathcal{K}v(\mathbf{x}(\mathrm{s}))=\int_{-1}^{1}K(s,\sigma)v(\mathbf{x}(\sigma))\,d\sigma, \tag{2.7}$$

and introduce $N(s,\sigma)$ as the kernel function of $\mathcal{N}$, i.e.,

$$\mathcal{N}v(\mathbf{x}(s))=\int_{-1}^{1} N(s,\sigma)v(\mathbf{x}(\sigma))\,d\sigma. \tag{2.8}$$

To describe the modified Nyström method, choose $\beta\in[0,1]$, and define the truncated operator $\mathcal{K}_\beta$ by replacing the domain of integration $[-1,1]$ in (2.7) by $[-1,\beta]\cup[\beta,1]$. Define $\mathcal{N}_\beta$ from $\mathcal{N}$ analogously and use these to define $\mathcal{S}_\beta$, $\mathcal{M}_\beta$, and $\mathcal{L}_\beta$ in an obvious way. When $\beta=s_{i^*}^{(n)}$ for some $0\leqq i^*\leqq n$ (i.e., $\pm\beta$ are mesh points in $[-1,1]$), we define $\mathcal{K}_{i^*,n}$, $\mathcal{N}_{i^*,n}$ to be the Nyström approximations to $\mathcal{K}_{s(i^*,n)}$, $\mathcal{N}_{s(i^*,n)}$. (For typesetting purposes, we substitute $s(i^*,n)$ for $s_{i^*}^{(n)}$.) Replacing $\mathcal{K},\mathcal{N}$ by $\mathcal{K}_{s(i^*,n)}$, $\mathcal{N}_{s(i^*,n)}$ or $\mathcal{K}_{i^*,n}$, $\mathcal{N}_{i^*,n}$ in an obvious way, we define operators $\mathcal{S}_{s(i^*,n)}$, $\mathcal{M}_{s(i^*,n)}$, $\mathcal{L}_{s(i^*,n)}$, $\mathcal{S}_{i^*,n}$, $\mathcal{M}_{i^*,n}$, and $\mathcal{L}_{i^*,n}$ on $\mathcal{B}$. The *modified Nyström method* then consists of choosing some $i^*\geqq 0$ and solving

$$(I-\mathcal{L}_{i^*,n})\begin{bmatrix}\tilde{u}_n\\ u_n(\mathbf{0})\end{bmatrix}=\begin{bmatrix}-2\tilde{g}\\ -2g(\mathbf{0})\end{bmatrix} \tag{2.9}$$

for $n\geqq i^*$. When $i^*=0$, (2.9) coincides with (1.14).

We shall analyze (2.9) by appealing to some of the results of GC, and adding some perturbation arguments to cope with the curved boundary Γ. The following lemma summarizes some required results which follow trivially from Theorems 3 and 4 of Graham and Chandler.

LEMMA 1. (i) *For each $\varepsilon>0$ there exists i^*, such that*

$$\|(\mathcal{K}_{s(i^*,n)}-\mathcal{K}_{i^*,n})\mathcal{K}_{i^*,m}\|_\infty<\varepsilon \quad \textit{for all } m,n\geqq i^*.$$

(ii) *If u satisfies the estimate*

$$\sup\{|s|^{k-\alpha}|D_s^R u(\mathbf{x}(s))|: s\in[-1,1]\backslash\{0\}\}\leqq C_k, \tag{2.10}$$

with $1>\alpha>0$, for all $k\geqq 0$, for some constants C_k, then for each fixed $i^*\geqq 0$, we have

$$\|(\mathcal{K}-\mathcal{K}_{i^*,n})\tilde{u}\|_\infty=O\left(\frac{1}{n^R}\right) \quad \textit{as } n\to\infty, \tag{2.11}$$

provided $q>R/\alpha$. (Recall that R is the order of the quadrature rule (1.6).)

A careful examination of GC shows that the proof of Lemma 1 depends on the operator $\mathcal{K}$ only through the estimates

$$|s^kD^k(\mathcal{K}_{i^*,n}v)(\mathbf{x}(s))|\leqq A_k\|v\|_\infty, \qquad s\in[-1,1]\backslash\{0\}, \tag{2.12}$$

$$|\sigma^{k+1}D_\sigma^k(K(s,\sigma))|\leqq B_k, \qquad s,\sigma\in[-1,1]\backslash\{0\}, \tag{2.13}$$

for all $k\geqq 0$, which are easy generalizations of Graham and Chandler's Lemma 2(ii), (iii).

We shall prove the following technical generalization of Lemma 1.

LEMMA 2. (i) *For each $\varepsilon>0$ there exists i^* such that*

$$\|(\mathcal{L}_{s(i^*,n)}-\mathcal{L}_{i^*,n})\mathcal{L}_{i^*,m}\|<\varepsilon \quad \textit{for all } m,n\geqq i^*.$$

(ii) *For each $i^*\geqq 0$ fixed independently of n,*

$$\left\|(\mathcal{L}-\mathcal{L}_{i^*,n})\begin{bmatrix}\tilde{u}\\ u(\mathbf{0})\end{bmatrix}\right\|_\infty=O\left(\frac{1}{n^R}\right) \quad \textit{as } n\to\infty,$$

provided $q>R(1+|\chi_0|)$, where $u=\tilde{u}+u(\mathbf{0})$ is the exact solution of (1.2).

Proof. In the Appendix we demonstrate that (2.12) holds with $\mathcal{K}_{i^*,n}$ replaced by $\mathcal{N}_{i^*,n}$ and (2.13) holds with K replaced by N (where N is defined by (2.8)). Then, by

Lemma 1(i), each of the quantities $\|(\mathcal{K}_{s(i^*,n)}-\mathcal{K}_{i^*,n})\mathcal{N}_{i^*,m}\|$, $\|(\mathcal{N}_{s(i^*,n)}-N_{i^*,n})\mathcal{K}_{i^*,m}\|$, and $\|(\mathcal{N}_{s(i^*,n)}-\mathcal{N}_{i^*,n})\mathcal{N}_{i^*,m}\|$ may be made arbitrarily small, for all $m, n \geqq i^*$, by an appropriate choice of i^*. Writing

$$(\mathcal{L}_{s(i^*,n)}-\mathcal{L}_{i^*,n})\mathcal{L}_{i^*,m}=(\mathcal{S}_{s(i^*,n)}-\mathcal{S}_{i^*,n})\mathcal{S}_{i^*,m}+(\mathcal{S}_{s(i^*,n)}-\mathcal{S}_{i^*,n})\mathcal{M}_{i^*,m}$$
$$+(\mathcal{M}_{s(i^*,n)}-\mathcal{M}_{i^*,n})\mathcal{S}_{i^*,m}+(\mathcal{M}_{s(i^*,n)}-\mathcal{M}_{i^*,n})\mathcal{M}_{i^*,m},$$

and using the above facts yields the proof of Lemma 2(i).

For Lemma 2(ii) we first observe that for the solution u of (1.2), the condition (2.10) is true for all $0<\alpha<(1+|\chi_0|)^{-1}$. This is shown by Costabel and Stephan (1983), who applied a perturbation argument to the usual regularity analysis of (1.2) for polygonal boundaries. Hence (2.11) holds provided $q>R(1+|\chi_0|)$. By the Appendix, the result (2.11) (with $\mathcal{K}$ replaced by $\mathcal{N}$) also holds for the same range of q. Writing

$$(\mathcal{L}-\mathcal{L}_{i^*,n})\begin{bmatrix}\tilde{u}\\ u(\mathbf{0})\end{bmatrix}=(\mathcal{S}-\mathcal{S}_{i^*,n})\begin{bmatrix}\tilde{u}\\ u(\mathbf{0})\end{bmatrix}+(\mathcal{M}-\mathcal{M}_{i^*,n})\begin{bmatrix}\tilde{u}\\ u(\mathbf{0})\end{bmatrix},$$

and using the above facts yields the proof of Lemma 2(ii). □

The following theorem proves the optimal convergence estimate (1.12) provided the mesh grading exponent q is sufficiently large.

THEOREM 3. *There exists i^* fixed independently of n such that for all sufficiently large $n \geqq i^*$, $I-\mathcal{L}_{i^*,n}$ has a uniformly bounded inverse on $\mathcal{B}$. In this case* (2.9) *defines* $(\tilde{u}_n, u_n(\mathbf{0}))^T$ *uniquely in $\mathcal{B}$ and*

$$\|u-u_n\|_\infty=\|(\tilde{u}, u(\mathbf{0}))^T-(\tilde{u}_n, u_n(\mathbf{0}))^T\|=O\left(\frac{1}{n^R}\right) \quad \text{as } n\to\infty,$$

provided $q>R(1+|\chi_0|)$.

Proof. Our first step is to show that

$$\|\mathcal{N}-\mathcal{N}_\beta\|_\infty\to 0 \quad \text{as } \beta\to 0. \tag{2.14}$$

To prove this, let $v\in C(\Gamma)$ with $\|v\|_\infty \leqq 1$. In Atkinson and deHoog (1984), it is shown that N (the kernel of $\mathcal{N}$) is bounded on $[-1,1]\times[-1,1]$. Thus

$$|(\mathcal{N}-\mathcal{N}_\beta)v(\mathbf{x}(s))|\leqq \int_{-\beta}^{\beta}|N(s,\sigma)|\,d\sigma\to 0 \quad \text{as } \beta\to 0,$$

uniformly in s, v, which proves (2.14). It follows directly from (2.14) that

$$\|\mathcal{M}-\mathcal{M}_\beta\|\to 0 \quad \text{as } \beta\to 0. \tag{2.15}$$

We next show that

$$\mathcal{S}_\beta\to\mathcal{S} \quad \text{as } \beta\to 0, \quad \text{pointwise on } \mathcal{B}. \tag{2.16}$$

Clearly (2.16) is true provided $\mathcal{K}_\beta\to\mathcal{K}$, pointwise on $C_0(\Gamma)$ as $\beta\to 0$. Thus let $v\in C_0(\Gamma)$. Then, using (2.5),

$$\begin{aligned}|(\mathcal{K}_\beta-\mathcal{K})v(\mathbf{x}(s))|&=\left|\int_{-\beta}^{\beta}K(s,\sigma)v(\mathbf{x}(\sigma))\right|d\sigma\\ &\leqq|\chi_0|\sup\{|v(\mathbf{x}(\sigma))|:\sigma\in[-\beta,\beta]\}\\ &\to|\chi_0|v(\mathbf{x}(0))=0 \quad \text{as } \beta\to 0,\end{aligned}$$

and (2.16) follows.

Now observe that $\|\mathcal{S}_\beta\| = \|\mathcal{K}_\beta\|_\infty \leqq \|\mathcal{K}\|_\infty = |\chi_0| < 1$, so that $(I - \mathcal{S}_\beta)^{-1}$ exists on $\mathcal{B}$ and is bounded as in (2.6), independently of β. Our next step is to show

$$\|(I - \mathcal{S}_\beta)^{-1}\mathcal{M}_\beta - (I - \mathcal{S})^{-1}\mathcal{M}\| \to 0 \quad \text{as } \beta \to 0. \tag{2.17}$$

To obtain (2.17), simply write

$$(I - \mathcal{S}_\beta)^{-1}\mathcal{M}_\beta - (I - \mathcal{S})^{-1}\mathcal{M} = (I - \mathcal{S}_\beta)^{-1}(\mathcal{S}_\beta - \mathcal{S})(I - \mathcal{S})^{-1}\mathcal{M} + (I - \mathcal{S}_\beta)^{-1}(\mathcal{M}_\beta - \mathcal{M}).$$

Since $\mathcal{M}$ is compact on $\mathcal{B}$, the first term goes to 0 in norm as $\beta \to 0$, by (2.16). Also by (2.15), the second term goes to 0 in norm, proving (2.17).

Now recall that $I - \mathcal{L}$ and $I - \mathcal{S}$ are invertible on $\mathcal{B}$, so that (2.17) ensures $I - (I - \mathcal{S}_\beta)^{-1}\mathcal{M}_\beta$ is invertible for sufficiently small β, with uniformly bounded inverse. The identity $(I - \mathcal{L}_\beta)^{-1} = (I - (I - \mathcal{S}_\beta)^{-1}\mathcal{M}_\beta)^{-1}(I - \mathcal{S}_\beta)^{-1}$ then ensures the existence of constants $\bar{\beta} > 0$, and $\bar{C} > 0$ such that,

$$\|(I - \mathcal{L}_\beta)^{-1}\| \leqq \bar{C} \quad \text{for all } 0 \leqq \beta \leqq \bar{\beta}. \tag{2.18}$$

Now by Lemma 2(i), we can choose i^* so that

$$\|(\mathcal{L}_{s(i^*,n)} - \mathcal{L}_{i^*,n})\mathcal{L}_{i^*,n}\| \leqq \bar{C}^{-1} \leqq \|(I - \mathcal{L}_{s(i^*,n)})^{-1}\|^{-1},$$

for all n sufficiently large. This shows (Atkinson (1976, p. 15)) that

$$\mathcal{R}_n := (I - \mathcal{L}_{s(i^*,n)}) + (\mathcal{L}_{s(i^*,n)} - \mathcal{L}_{i^*,n})\mathcal{L}_{i^*,n}$$

has a uniformly bounded inverse for all sufficiently large $n \geqq i^*$. Then

$$(I - \mathcal{L}_{i^*,n})^{-1} = \mathcal{R}_n^{-1}(I - \mathcal{L}_{s(i^*,n)} + \mathcal{L}_{i^*,n}).$$

Since $\|\mathcal{L}_{s(i^*,n)}\| \leqq \|\mathcal{L}\|$, and (by (2.12) and (A.1) of the Appendix),

$$\|\mathcal{L}_{i^*,n}\| \text{ is bounded independently of } i^*, n, \tag{2.19}$$

it follows that for some $i^* \geqq 0$, $(I - \mathcal{L}_{i^*,n})^{-1}$ exists and is uniformly bounded, for all sufficiently large $n \geqq i^*$. For such i^* and n, (2.9) defines $(\tilde{u}_n, u_n(\mathbf{0}))^T$ uniquely, and Lemma 2(ii) yields

$$\begin{aligned}\|u_n - u\|_\infty &= \|(\tilde{u}_n, u_n(\mathbf{0}))^T - (\tilde{u}, u(\mathbf{0}))^T\| \\ &= \left\|(I - \mathcal{L}_{i^*,n})^{-1}(\mathcal{L}_{i^*,n} - \mathcal{L})\begin{bmatrix}\tilde{u} \\ u(\mathbf{0})\end{bmatrix}\right\| \\ &= O\left(\frac{1}{n^R}\right),\end{aligned}$$

provided $q > R(1 + |\chi_0|)$, as required. □

2.1. Numerical Experiment 1. The above results generalize to regions with any number of corners. As an illustration, we solve the interior Dirichlet problem

$$\Delta U = 0 \quad \text{in } \Omega, \qquad U = g \quad \text{on } \Gamma = \partial\Omega, \tag{2.20}$$

where Ω is the "pie-shaped" region composed of a segment of the unit disk, as depicted in Fig. 1. The exterior angles at corners P_1, P_2 are always $3\pi/2$, while the exterior angle at corner P_3 is $(1+\chi)\pi$ with $\chi < 0$, χ variable. To try the method out on a realistic singularity, we solve (2.20) with the known solution $U(\mathbf{x}) = r^\alpha \cos(\alpha\tilde{\theta})$, where $-\pi < \tilde{\theta} \leqq \pi$ is the polar angle of $\mathbf{x}$ after it has been rotated through $\pi/2$ anticlockwise about the origin, and $\alpha = 1/(1-\chi)$. This U is then harmonic in Ω and has the worst singularity which can be expected at the origin for a solution of (2.20) with smooth g. As described in § 1, (2.20) can be solved by solving (1.2) on Γ.

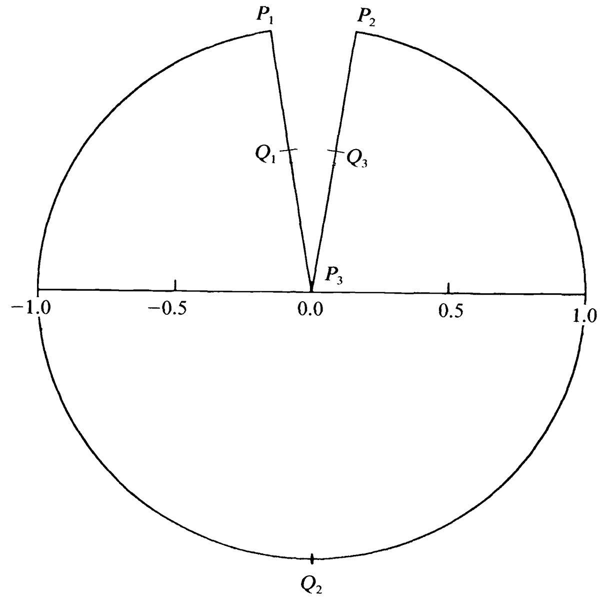


FIG. 1. *Pie-shaped domain,* $\chi = -0.9$.

Let Q_1, Q_2, Q_3 be the midpoints of P_3P_1, P_1P_2, and P_2P_3, respectively, and let Γ_l for $l = 1, 2, 3$ denote the portion of Γ joining Q_l to Q_{l+1} (with $Q_4 = Q_1$). Then the numerical method for (1.2) is to first rearrange it as

$$(1+\chi)u - 2\sum_{l=1}^{3} W\tilde{u}^l - 2\sum_{l=1}^{3} u(P_l)W1^l = -2g,$$

where $\tilde{u}^l$ denotes $u - u(P_l)$ restricted to Γ_l and 1^l denotes the constant function 1 on Γ_l. Discretize this by replacing W in the first sum on the left-hand side by W_n, and collocate at the quadrature points and the corner points. This determines the numerical solution at the collocation points. The solution U_n to (2.20) is then recovered by the approximate potential

$$U_n(x) = \sum_{l=1}^{3} W_n\tilde{u}_n^l(x) + \sum_{l=1}^{3} u_n(P_l)W1^l(x).$$

For more detail see Graham and Chandler (1988), or Atkinson and Graham (1988).

In the experiments reported here, we have chosen in (1.6) the two-point Gauss-Legendre rule (so that $R = 4$). For each n there were n subintervals on each of P_3Q_1, Q_1P_1, P_2Q_3, and Q_3P_3, and $2n$ subintervals on each of P_1Q_2 and Q_2P_2. We have graded the mesh near each of the corners P_l with a grading exponent q_l. Thus, for example, the mesh points on each side of P_3 are a distance $(i/2n)^{q_3}$ from P_3 for $i = 0, \cdots, n$. In Fig. 2 we plot the error $|U(\mathbf{x}) - U_n(\mathbf{x})|$, for $\mathbf{x} = r(\cos(-\chi\pi/4), \sin(-\chi\pi/4))$, $r \in (0, 1)$, with $\mathbf{q} = (q_1, q_2, q_3) = (3, 3, 4)$, $\chi = -0.9$, $i^* = 0$, and various n. The error decreases like $O(n^{-4})$ and becomes more evenly distributed as n increases. Figure 3 shows the effect of varying the grading exponents $\mathbf{q}$. Here we

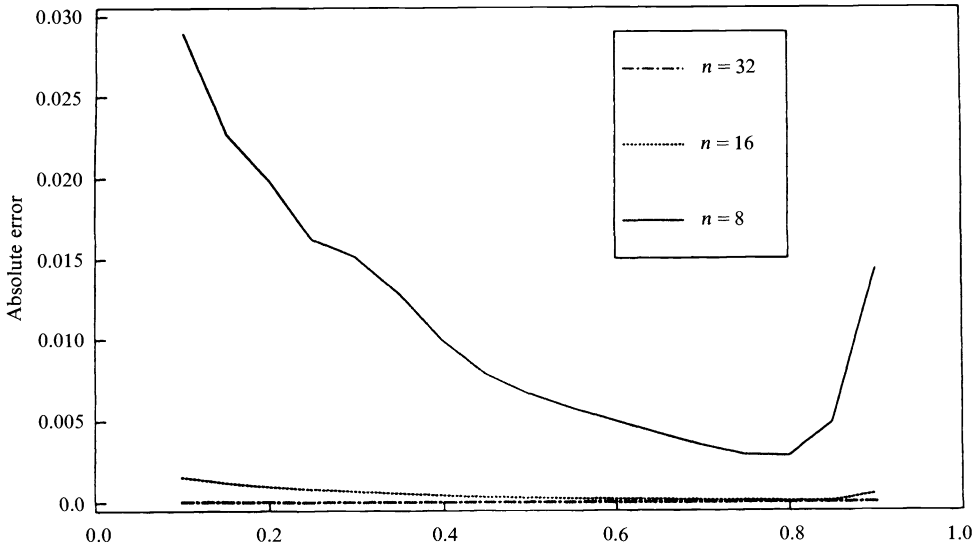


FIG. 2. $\chi=-0.9$, $q=(3,3,4)$, $i^*=0$.

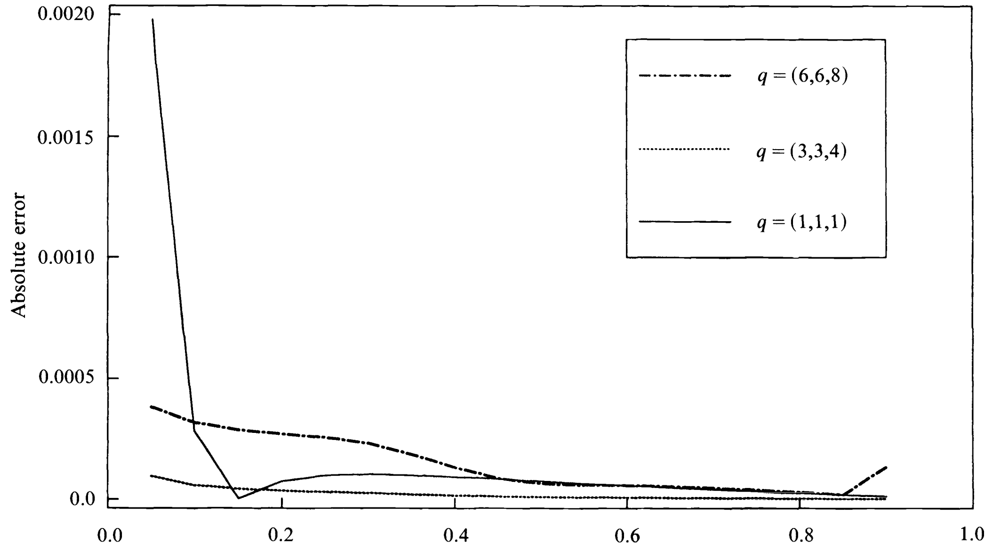


FIG. 3. $\chi=-0.9$, $n=32$, $i^*=0$.

plot the same error with $\chi=-0.9$, $n=32$, and $i^*=0$, but for various $\mathbf{q}$. Note that the analogue of Theorem 3 for the three-cornered case would imply that $\mathbf{q}>(6,6,7.6)$ is needed for optimal convergence of the boundary distribution u. However, U is calculated from u by integration against a smooth function, so we might expect that a smaller $\mathbf{q}$ would achieve optimal results for U. This is borne out in Fig. 3, where $\mathbf{q}=(3,3,4)$ is seen to be optimal. Finally note that in all these experiments, $i^*=0$ is sufficient for stability. Nevertheless, small $i^*>0$ does not affect the accuracy very much. In Table 1 we show the value of the boundary distribution u_n at $\boldsymbol{B}_3$ for $n=32$, $\mathbf{q}=(3,3,4)$, and various i^*.

3. Basic iteration schemes. In this section we discuss the convergence of appropriately modified versions of (1.17a–e).

Modification 1. The modification which is the simplest to analyze is obtained by setting

$$\mathscr{L}_m = \mathscr{L}_{i^*(m),m}, \tag{3.1}$$

$$\mathscr{L}_n = \mathscr{L}_{i^*(n),n} \tag{3.2}$$

in (1.17a–e), where $i^*(n)$, $i^*(m)$ are chosen so that

$$\beta := s(i^*(n),n) = s(i^*(m),m),$$

that is, the cutoff points of the coarse and fine meshes coincide and are independent of m, n. This can easily be arranged, for example, by choosing $m=kn$, $i^*(n)=n/r$, $i^*(m)=kn/r$, where r is any divisor of n.

Then $\{\mathscr{L}_m\}$, $\{\mathscr{L}_n\}$ given by (3.1) and (3.2) are collectively compact approximations to $\mathscr{L}_\beta$. For sufficiently small β, $(I-\mathscr{L}_\beta)^{-1}$ satisfies (2.18) and the standard results of Atkinson (1976) then show that (1.17a–e) converge linearly for all $m>n$, provided n is sufficiently large. This algorithm clearly has properties (P1) and (P3) described in § 1. However (P2) is lost. This is because the limit of $v_m^{(j)}$ as $j\to\infty$ is the numerical solution of (1.15), with $\mathscr{L}$ replaced by $\mathscr{L}_\beta$, where $\beta>0$ is independent of m. (However, the error induced by small β may not be very significant.) To preserve (P2), we need to do something more sophisticated.

Modification 2. To best describe the modification, assume

$$m = kn.$$

Then for any i^*, in (1.17a–e), choose the fine-grid operator

$$\mathscr{L}_m = \mathscr{L}_{i^*,m}. \tag{3.3}$$

If we chose, as the coarse-grid operator, $\mathscr{L}_n=\mathscr{L}_{i^*,n}$, then it is not possible to use (1.18) to show that (1.17a–e) converge for all $m \geqq n$.

This is because the distance between the cutoff points of the coarse and fine meshes $|s(i^*,n)-s(i^*,m)| \not\to 0$ as $m\to\infty$, for fixed n. To overcome this problem, we allow the coarse mesh to be augmented with the subintervals of the fine mesh contained within the troublesome region

$$[-s(i^*,n), -s(i^*,m)] \cup [s(i^*,m), s(i^*,n)], \tag{3.4}$$

that is, we choose as our coarse-grid operator in (1.17a–e)

$$\mathscr{L}_n = \tilde{\mathscr{L}}_{i^*,n}, \tag{3.5}$$

where $\tilde{\mathscr{L}}_{i^*,n}$ is the quadrature approximation to $\mathscr{L}_{s(i^*,m)}$ obtained by using the m level mesh in the region (3.4) and the n level mesh in the region $[-1, -s(i^*,n)] \cup [s(i^*,n), 1]$.

TABLE 1
Numerical values at corner P_3.

i^*	0	2	4
$u_n(P_3)$	14.025602	14.025575	14.025243

This means that the dimension of the n level system is now $O(n+m/n)$ which is increasing with m, so property (P3) is lost. However, (P2) is gained and (P1) remains, as we shall see in the following result which uses (1.18) to estimate the convergence of (1.17a–e).

THEOREM 4. *Suppose* $\mathscr{L}_m$, $\mathscr{L}_n$ *are given by* (3.3), (3.5), *and define*

$$C_{m,n,i^*} = \|(I-\mathscr{L}_n)^{-1}(\mathscr{L}_m-\mathscr{L}_n)\mathscr{L}_m\|. \tag{3.6}$$

Then there exists i^* *fixed independently of* n *such that for all* n *sufficiently large and all* $m=kn$, $k \geqq 1$ *we have* $C_{m,n,i^*}<1$.

Proof. The details are technical but not difficult, so we shall only sketch the proof. First we observe

$$\begin{aligned}\mathscr{L}_m-\mathscr{L}_n &= \mathscr{L}_{i^*,m}-\tilde{\mathscr{L}}_{i^*,n}\\ &= (\mathscr{L}_{i^*,m}-\mathscr{L}_{s(i^*,m)})+(\mathscr{L}_{s(i^*,n)}-\mathscr{L}_{i^*,n})+(\mathscr{L}_{s(i^*,m)}-\mathscr{L}_{s(i^*,n)}-\tilde{\mathscr{L}}_{i^*,n}+\mathscr{L}_{i^*,n}).\end{aligned}$$

The operator in the final term is really just $\mathscr{L}_{s(i^*,m)}-\mathscr{L}_{i^*,m}$ restricted to functions which are zero in $[-1,1]\backslash[-s(i^*,n), s(i^*,n)]$. Thus, by Lemma 2 (or a slight variation of it), we can show that for each $\varepsilon>0$, there exists i^* independent of n, such that for all $n\geqq i^*$ and $m=kn$, $k\geqq 1$,

$$\|(\mathscr{L}_m-\mathscr{L}_n)\mathscr{L}_m\|<\varepsilon, \tag{3.7}$$

$$\|(\mathscr{L}_m-\mathscr{L}_n)\mathscr{L}_n\|<\varepsilon. \tag{3.8}$$

Now recall the identity

$$(I-\mathscr{L}_n)^{-1}=\mathscr{R}^{-1}(I-\mathscr{L}_m+\mathscr{L}_n),$$

where $\mathscr{R}=(I-\mathscr{L}_m)+(\mathscr{L}_m-\mathscr{L}_n)\mathscr{L}_n$. By Theorem 3 and (3.8), $\mathscr{R}^{-1}$ has a uniformly bounded inverse for appropriate fixed i^*, and for all n sufficiently large and $m=kn$. Under these conditions, $\|(I-\mathscr{L}_n)^{-1}\|$ is uniformly bounded; combined with (3.7), we obtain the required result. □

Remark. Theorem 4 corrects an incorrect sentence in the last paragraph of § 2 of Atkinson and Graham (1988).

DISCUSSION. A strict reading of Modification 2 will lead to a modification which is unpleasantly complicated. To speed up the convergence, not only would it be necessary to increase i^*, but also one would be expected to add m level points to the n level grid. It was observed in Atkinson and Graham (1988) that the following simpler algorithm worked well in several cases where a direct application of Algorithm 1 did not.

ALGORITHM 2.

(3.9a) Perform steps: (1.17a)–(1.17e)

(3.9b) Compute: $e_m^{(j+1)}=\|v_m^{(j+1)}-v_m^{(j)}\|$

(3.9c) If $e_m^{(j+1)}<$required tolerance: stop.

(3.9d) Compute: $\Delta_m^{(j+1)}=e_m^{(j+1)}/e_m^{(j)}$

(3.9e) If $\Delta_m^{(j+1)}>1$: Increase i^*, and go to (3.9a).

In the following experiment we study the performance of (3.9).

3.1. Numerical Experiment 2. We discretized (1.2) on the boundary Γ of the pie-shaped region, exactly as described in Numerical Experiment 1. For a given i^*, we removed from W_n contributions from i^* subintervals on each side of each of the three corners of Γ. We show in Table 2 how the convergence of (3.9a–e) is affected by the choice of χ, $\mathbf{q}$, n, m, and i^*. Remarkable points are the following: The iteration works well unmodified (i.e., with $i^*=0$) when all the angles are right angles, a frequently occurring practical situation. As $\chi \to -1$ the iteration slows down and diverges for $\chi \leqq -0.75$, unless a modification is made. The modification i^* needed to restore convergence as $\chi \to -0.9$ increases to the extent that the coarse-grid operator on the "wedge" portion of Γ must be almost removed entirely. There is strong evidence to suggest that the convergence rate remains fixed for a given i^* and $m=2n$, for a range of values of n.

If m were allowed to increase indefinitely compared to n, the distance between the cutoff points $s(i^*, m)$ and $s(i^*, n)$ would grow and may pollute the estimate of C_{m,n,i^*} given in Theorem 4. This could be alleviated by cautiously adding m level points to the coarse grid. (This would mean that extra rows and columns would be added to the coarse-grid matrix, a process which could be done efficiently using block elimination techniques (e.g., Chan (1984)).) We see from Numerical Experiment 2 that if $m \approx 2n$ (again a practically relevant choice—see § 1), then such cautious updating may not be necessary.

In the next section we describe another technique for obtaining a two-grid algorithm with arbitrarily fast convergence for (1.16). This technique also requires that the dimension of the coarse level should be allowed to increase with m, but in a different and more deliberate way than that suggested by Modification 2.

4. An alternative iteration scheme. Recall the system (1.13). We define an iteration method for the first equation in (1.14), finding $\tilde{u}_m$; then the second equation in (1.14) is used to calculate $u_m(\mathbf{0})$. Begin by defining

$$\begin{aligned}(V\tilde{v})(\mathbf{x}) &= 2[(W\tilde{v})(\mathbf{x})-(W\tilde{v})(\mathbf{0})]\\ &= 2\int_\Gamma [G'(\mathbf{x},\xi)-G'(0,\xi)]\tilde{v}(\xi)\,d\Gamma, \qquad \mathbf{x}\in\Gamma,\\ (V_m\tilde{v})(\mathbf{x}) &= 2[(W_m\tilde{v})(\mathbf{x})-(W_m\tilde{v})(\mathbf{0})]\end{aligned}$$

TABLE 2
Iteration results for (3.9).

χ	$\mathbf{q}$	n	m	i^*	n level system	m level system	$\lim_{j\to\infty} \Delta_m^{(j)}$
−0.5	3 3 3	4	8	0	67	131	0.161
				1	55	119	0.104
				2	43	107	0.096
−0.75	3 3 3.5	4	8	0	67	131	1.817
				1	55	119	0.467
				2	43	107	0.335
−0.9	3 3 4	4	8	0	67	131	5.661
				1	55	119	3.285
				2	43	107	1.541
				3	31	95	0.752
				4	19	83	0.639
−0.9	3 3 4	8	16	0	131	259	5.661
				2	107	235	1.487
				4	83	211	0.675
				6	59	187	0.641

for all $\tilde{v} \in C_0(\Gamma)$. We iteratively solve

$$\tilde{u}_m - V_m \tilde{u}_m = -2\tilde{g}, \tag{4.1}$$

which is the Nyström discretization of

$$\tilde{u} - V\tilde{u} = -2\tilde{g}. \tag{4.2}$$

By a simple extension of the proof of Theorem 3, we have the existence and uniform boundedness of $(I - V_m)^{-1}$ on $C_0(\Gamma)$. This assumes that the quantity i^* of Theorem 3 is chosen sufficiently large, which we assume throughout this section.

Consider applying the iteration method (1.17a–e) to (4.1), with the role of $\mathcal{L}_m$ replaced by V_m and $v_m^{(j)} = u_m^{(j)}$. The resulting iteration will often converge, but at a slow rate. The problem lies in the lack of smoothing by the operator V; the function $(V\tilde{v})(\mathbf{x})$ need be no smoother than $\tilde{v}(\mathbf{x})$ for $\mathbf{x}$ near $\mathbf{0}$. To deal with this, we break Γ into two parts: $\Gamma = \Gamma_1 \cup \Gamma_2$. The portion Γ_1 will contain the corner $\mathbf{0}$, and the remaining boundary Γ_2 will be bounded away from $\mathbf{0}$. The discretization of the equation (4.2) on Γ_1 is solved exactly, and then the iteration method of (1.17a–e) is applied to the corner-modified version of (4.1).

The idea of inverting directly that portion of the operator in a small neighborhood of the corner, in order to improve the performance of iteration methods, has been used previously by other authors. It is used in Schippers (1985) to solve a discretization based on Galerkin's method with piecewise constant functions on a uniform grid. Hebeker (1988) uses it to solve boundary integral equations from fluid mechanics for discretizations based on the collocation method with bilinear trial functions. The idea also existed in the work of Radon (1919) who used it in developing a theory for (1.2) generalizing the work of Fredholm. Also, Bruhn and Wendland (1967) used it in the analysis of numerical methods for (1.2).

Let $0 < \delta < 1$, and restrict the values of m to those integers for which

$$\delta = \left(\frac{j^*(m)}{m}\right)^q \tag{4.3}$$

for some $1 \leqq j^*(m) < m$. Define Γ_1 to be the portion of Γ joining $\mathbf{x}(-\delta)$ and $\mathbf{x}(\delta)$, containing the corner $\mathbf{0}$; let Γ_2 be the closure of the remainder of Γ. For $\tilde{v} \in C_0(\Gamma)$, let

$$v^{(1)} = \tilde{v}|_{\Gamma_1}, \qquad v^{(2)} = \tilde{v}|_{\Gamma_2}.$$

Let $\mathscr{X} = C_0(\Gamma_1) \oplus C(\Gamma_2)$. Define $\mathscr{V}: \mathscr{X} \to \mathscr{X}$ by

$$\mathscr{V}\begin{bmatrix} v^{(1)} \\ v^{(2)} \end{bmatrix} = \begin{bmatrix} V^{(1,1)} & V^{(1,2)} \\ V^{(2,1)} & V^{(2,2)} \end{bmatrix} \begin{bmatrix} v^{(1)} \\ v^{(2)} \end{bmatrix}, \tag{4.4}$$

$$V^{(i,j)} v^{(j)}(\mathbf{x}) = 2 \int_{\Gamma_j} [G'(\mathbf{x}, \xi) - G'(\mathbf{0}, \xi)] v^{(j)}(\xi)\, d\Gamma, \qquad \mathbf{x} \in \Gamma_i \tag{4.5}$$

for $i, j = 1, 2$, with $v^{(1)} \in C_0(\Gamma_1)$ and $v^{(2)} \in C(\Gamma_2)$. It is straightforward to extend the unique solvability of (4.2) on $C_0(\Gamma)$ to that of

$$(I - \mathscr{V})\mathbf{u} = \mathbf{f}, \qquad \mathbf{u}, \mathbf{f} \in \mathscr{X}. \tag{4.6}$$

(Note that $C_0(\Gamma)$ is isomorphic to a closed subspace of $\mathscr{X}$ of codimension two.) For notational simplicity, we have let $\tilde{f} = -2\tilde{g}$ in going from (4.2) to (4.6).

Extend the above reformulation to the discretized equation (4.1):

$$\mathcal{V}_m\begin{bmatrix} v^{(1)} \\ v^{(2)} \end{bmatrix}=\begin{bmatrix} V_m^{(1,1)} & V_m^{(1,2)} \\ V_m^{(2,1)} & V_m^{(2,2)} \end{bmatrix}\begin{bmatrix} v^{(1)} \\ v^{(2)} \end{bmatrix}, \qquad \mathbf{v}\in\mathscr{X}, \tag{4.7}$$

$$(I-\mathcal{V}_m)\mathbf{u}_m=\mathbf{f}, \qquad \mathbf{u}_m, \mathbf{f}\in\mathscr{X}.$$

The operators $V^{(i,j)}, (i,j)\neq(1,1)$, are compact; their approximations $\{v_m^{(i,j)}\}, (i,j)\neq(1,1)$, are collectively compact and pointwise convergent. This follows from (a) the continuity of the kernel function $[G'(\mathbf{x},\xi)-G'(\mathbf{0},\xi)]$ for $\mathbf{x}\in\Gamma_i$, $\xi\in\Gamma_j$, $(i,j)\neq(1,1)$, and (b) the convergence for all continuous integrands of the integration scheme used in (1.7).

The equation

$$[I-V^{(1,1)}]u^{(1)}=f^{(1)}, \qquad f^{(1)}\in C_0(\Gamma_1)$$

is defined on a "wedge boundary," and it has a theory completely analogous to that of $(I-\mathscr{K})\tilde{u}=f$. The use of a wedge boundary is investigated in Atkinson and deHoog (1984); the arguments leading to Theorem 3 are easily extended to the wedge equation with operator $I-V^{(1,1)}$. The above equation is uniquely solvable for all $f^{(1)}\in C_0(\Gamma_1)$. For its discretization, we have that $[I-V_m^{(1,1)}]^{-1}$ exists and is uniformly bounded on $C_0(\Gamma_1)$ for all sufficiently large m.

Apply these results to equations (4.6) and (4.7) to obtain the equivalent equations

$$\begin{bmatrix} I & -[I-V^{(1,1)}]^{-1}V^{(1,2)} \\ -V^{(2,1)} & I-V^{(2,2)} \end{bmatrix}\begin{bmatrix} u^{(1)} \\ u^{(2)} \end{bmatrix}=\begin{bmatrix} [I-V^{(1,1)}]^{-1}f^{(1)} \\ f^{(2)} \end{bmatrix}, \tag{4.8}$$

$$\begin{bmatrix} I & -[I-V_m^{(1,1)}]^{-1}V_m^{(1,2)} \\ -V_m^{(2,1)} & I-V_m^{(2,2)} \end{bmatrix}\begin{bmatrix} u_m^{(1)} \\ u_m^{(2)} \end{bmatrix}=\begin{bmatrix} [I-V_m^{(1,1)}]^{-1}f^{(1)} \\ f^{(2)} \end{bmatrix}. \tag{4.9}$$

Denote these symbolically by

$$(I-\mathscr{L})\mathbf{u}=\mathbf{z}, \tag{4.10}$$

$$(I-\mathscr{L}_m)\mathbf{u}_m=\mathbf{z}_m, \tag{4.11}$$

respectively. (Note: This $\mathscr{L}$ is not the same as that used in (1.13) of § 1 nor in §§ 2 and 3.) Using the uniform boundedness of $[I-V_m^{(1,1)}]^{-1}$, together with Theorem 3 for the original equation (4.1), we have that $(I-\mathscr{L}_m)^{-1}$ is uniformly bounded for all sufficiently large m.

Apply the iteration method (1.17a-e) to (4.9). The steps for solving (4.1) are as follows: (1) Calculate the matrix associated with $I-\mathscr{L}_n$ and its LU factorization, for some $n<m$. (2) Calculate the matrix associated with $I-V_m^{(1,1)}$ and its LU factorization; modify the right-hand side to form $z_m^{(1)}=[I-V_m^{(1,1)}]^{-1}f^{(1)}$. (3) Given an initial guess $\mathbf{u}_m^{(0)}$ for the solution to (4.9), perform the following iteration:

$$\begin{gathered} \mathbf{d}_m^{(j)}=\mathbf{z}_m-(I-\mathscr{L}_m)\mathbf{u}_m^{(j)}, \\ \delta_n^{(j)}=(I-\mathscr{L}_n)^{-1}\mathscr{L}_m\mathbf{d}_m^{(j)}, \\ \mathbf{u}_m^{(j+1)}=\mathbf{u}_m^{(j)}+\mathbf{d}_m^{(j)}+\delta_n^{(j)} \end{gathered} \tag{4.12}$$

for $j=0,1,\cdots$. The calculation of $\mathbf{d}_m^{(j)}$ and $\mathscr{L}_m\mathbf{d}_m^{(j)}$ uses the LU factorization of $I-V_m^{(1,1)}$ obtained earlier.

THEOREM 5. *Assume the existence and uniform boundedness of* $[I-\mathscr{L}_m]^{-1}$ *for all sufficiently large* m, *say,* $m \geqq N$. *Then the iteration method* (4.12) *will converge if* n *is chosen sufficiently large, uniformly for* $m>n$. *Moreover,*

$$\|\mathbf{u}_m-\mathbf{u}_m^{(j+1)}\| \leqq c\lambda_n \|\mathbf{u}_m-\mathbf{u}_m^{(j)}\|, \qquad j \geqq 0, \tag{4.13}$$

where λ_n *is defined below in* (4.15) *and*

$$c=\operatorname*{Sup}_{m \geqq N} \|[I-\mathscr{L}_m]^{-1}\|.$$

Proof. As in (1.18), we have the identity

$$\begin{aligned} \mathbf{u}_m-\mathbf{u}_m^{(j+1)} &= M_{n,m}[\mathbf{u}_m-\mathbf{u}_m^{(j)}], \qquad j \geqq 0, \\ M_{n,m} &= [I-\mathscr{L}_m]^{-1}[\mathscr{L}_m-\mathscr{L}_n]\mathscr{L}_m. \end{aligned} \tag{4.14}$$

To obtain convergence of $\{\mathbf{u}_m^{(j)} \mid j \geqq 0\}$, we show

$$\lambda_n \equiv \operatorname*{Sup}_{m>n} \|[\mathscr{L}_m-\mathscr{L}_n]\mathscr{L}_m\| \to 0 \tag{4.15}$$

as $n \to \infty$.

We begin by showing that the set $\{\mathscr{L}_m\}$ is collectively compact on the Banach space $\mathscr{X}$. To prove this is relatively straightforward. First, the families of operators $\{V_m^{(i,j)} \mid m \geqq 1\}$ are collectively compact for $(i,j) \neq (1,1)$. Second, the operators $[I-V_m^{(1,1)}]^{-1}$ are uniformly bounded on $C_0(\Gamma_1)$, for all large m. If these results are combined with the definition of $\mathscr{L}_m$ given in (4.9), then the desired collective compactness of $\{\mathscr{L}_m\}$ follows.

Next we show that $[\mathscr{L}_m-\mathscr{L}_n]\mathbf{v} \to 0$ as $m, n \to \infty$, for all $\mathbf{v} \in \mathscr{X}$. From the definition in (4.9),

$$\mathscr{L}_m-\mathscr{L}_n=\begin{bmatrix} 0 & [I-V_m^{(1,1)}]^{-1}V^{(1,2)}-[I-V_n^{(1,1)}]^{-1}V_n^{(1,2)} \\ -V_m^{(2,1)}+V_n^{(2,1)} & -V_m^{(2,2)}+V_n^{(2,2)} \end{bmatrix}.$$

The uniform convergence of

$$[-V_m^{(2,j)}+V_n^{(2,j)}]v^{(j)} \to 0 \quad \text{as } n, m \to \infty, \qquad j=1,2, \tag{4.16}$$

is straightforward based on the convergence of the quadrature method in (1.7) for all continuous integrands. For the remaining element, in the $(1,2)$ position of $\mathscr{L}_m-\mathscr{L}_n$, write

$$\begin{aligned} &\{[I-V_m^{(1,1)}]^{-1}V_m^{(1,2)}-[I-V_n^{(1,1)}]^{-1}V_n^{(1,2)}\}v^{(2)} \\ &\quad=\{[I-V_m^{(1,1)}]^{-1}V_m^{(1,2)}-[I-V^{(1,1)}]^{-1}V^{(1,2)}\}v^{(2)} \\ &\qquad+\{[I-V^{(1,1)}]^{-1}V^{(1,2)}-[I-V_n^{(1,1)}]^{-1}V_n^{(1,2)}\}v^{(2)}. \end{aligned} \tag{4.17}$$

For either of these expressions, say the one depending on m, we further decompose it as

$$\begin{aligned} &[I-V_m^{(1,1)}]^{-1}[V_m^{(1,1)}-V^{(1,1)}][I-V^{(1,1)}]^{-1}V^{(1,2)}v^{(2)} \\ &\quad+[I-V_m^{(1,1)}]^{-1}\{V_m^{(1,2)}-V^{(1,2)}\}v^{(2)}. \end{aligned} \tag{4.18}$$

For the first term, note that $V^{(1,2)}v^{(2)}$ is in $C_0(\Gamma_1)$; consequently, the term $[I-V^{(1,1)}]^{-1}V^{(1,2)}v^{(2)}$ can be shown to satisfy the estimate (2.10) near the corner. Then following Lemma 1(ii), the first term in (4.18) converges to zero as $n \to \infty$. The second term converges to zero since $V_m^{(1,2)}$, converges pointwise to the smoothing compact operator $V^{(1,2)}$. Combining with (4.16), we have $[\mathscr{L}_m-\mathscr{L}_n]\mathbf{v} \to 0$ as $m, n \to \infty$, for all $\mathbf{v} \in \mathscr{X}$.

To show (4.15) follows by standard arguments from the theory of collectively compact operator approximations. For example, see Atkinson (1976, p. 97). Combining this with (4.14) shows (4.13). □

We have actually implemented a variation on the above iteration, using the framework of prolongation and restriction operators to define the passage between functions defined on the coarse grid and on the fine grid. Collocate (4.7) at all nonzero points $\mathbf{x}_{ij}$. This yields the linear system

$$(4.19) \qquad (I-\hat{\mathscr{V}}_m)\hat{\mathbf{u}}_m=\hat{\mathbf{f}}, \qquad \hat{\mathbf{u}}_m, \hat{\mathbf{f}}\in\mathbb{R}^D$$

with $D=D_m$ the number of nonzero quadrature points. The matrix equation is

$$(4.20) \qquad \begin{bmatrix} I-\hat{V}_m^{(1,1)} & -\hat{V}_m^{(1,2)} \\ -\hat{V}_m^{(2,1)} & I-\hat{V}_m^{(2,2)} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{u}}^{(1)} \\ \hat{\mathbf{u}}^{(2)} \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{f}}^{(1)} \\ \hat{\mathbf{f}}^{(2)} \end{bmatrix}.$$

Here

$$\hat{\mathbf{f}}=\begin{bmatrix} \hat{\mathbf{f}}^{(1)} \\ \hat{\mathbf{f}}^{(1)} \end{bmatrix}$$

denotes the values of f at the quadrature points. Similarly, $\hat{\mathscr{V}}_m$ is the matrix based on regarding $\mathscr{V}_m$ as an operator from $\mathbb{R}^D$ to $\mathbb{R}^D$. Let $\hat{\mathbf{f}}^{(1)}$ denote the restriction to $x_{ij}\in\Gamma_1$ of the function $\mathbf{f}^{(1)}$, and similarly for $\hat{\mathbf{f}}^{(2)}$. Let $D^{(1)}$ and $D^{(2)}$ denote the respective orders of the vectors $\hat{\mathbf{f}}^{(1)}$ and $\hat{\mathbf{f}}^{(2)}$. If in (1.6) the points η_j all lie in $0<\eta<1$, then $D^{(1)}=2j^*(m)r$, $D^{(2)}=2(m-j^*(m))r$, and $D=D^{(1)}+D^{(2)}$.

Define the *prolongation* operator $P_{nm}:\mathbb{R}^{D_n}\to\mathbb{R}^{D_m}$ by using piecewise polynomial interpolation of degree $2r-1$. For $\mathbf{v}\in\mathbb{R}^{D_n}$, let $P_{nm}\mathbf{v}$ be the values at $\{\mathbf{x}_{ij}^{(m)}\}$ of the piecewise polynomial interpolate of degree $2r-1$, interpolating the given values in $\mathbf{v}$ at the abscissae $\{\mathbf{x}_{ij}^{(n)}\}$. Define the *restriction* operator $R_{mn}:\mathbb{R}^{D_m}\to\mathbb{R}^{D_n}$ similarly. For $\mathbf{v}\in\mathbb{R}^{D_m}$, let $R_{mn}\mathbf{v}$ be the values at $\{\mathbf{x}_{ij}^{(n)}\}$ of the piecewise polynomial interpolate of degree $2r-1$, interpolating the given values in $\mathbf{v}$ at the abscissae $\{\mathbf{x}_{ij}^{(m)}\}$. In the case $\{\mathbf{x}_{ij}^{(n)}\}\subset\{\mathbf{x}_{ij}^{(m)}\}$, $R_{mn}\mathbf{v}$ is simply the restriction of $\mathbf{v}$ to those components corresponding to nodes $\mathbf{x}_{ij}^{(n)}$.

In analogy with (4.9), transform (4.20) to the equivalent equation

$$(4.21) \qquad \begin{bmatrix} I & -[I-\hat{V}_m^{(1,1)}]^{-1}\hat{V}_m^{(1,2)} \\ -\hat{V}_m^{(2,1)} & I-\hat{V}_m^{(2,2)} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{u}}_m^{(1)} \\ \hat{\mathbf{u}}_m^{(2)} \end{bmatrix} = \begin{bmatrix} [I-\hat{V}_m^{(1,1)}]^{-1}\hat{\mathbf{f}}^{(1)} \\ \hat{\mathbf{f}}^{(2)} \end{bmatrix}$$

which contains the exact solution of the discrete equation in a neighborhood of the corner. Denote this equation by

$$(4.22) \qquad (I-\hat{\mathscr{L}}_m)\hat{\mathbf{u}}_m=\hat{\mathbf{z}}_m.$$

We define the iteration in analogy with (4.12), but we use the prolongation-restriction operator framework to move between the coarse and fine grids, rather than the Nyström interpolation used in (4.12). Define

$$(4.23) \qquad \begin{aligned} \hat{\mathbf{d}}_m^{(j)} &= \hat{\mathbf{z}}_m-(I-\hat{\mathscr{L}}_m)\hat{\mathbf{u}}_m^{(j)}, \\ \hat{\delta}_n^{(j)} &= P_{nm}(I-\hat{\mathscr{L}}_n)^{-1}R_{mn}\hat{\mathscr{L}}_m\hat{\mathbf{d}}_m^{(j)}, \\ \hat{\mathbf{u}}_m^{(j+1)} &= \hat{\mathbf{u}}_m^{(j)}+\hat{\mathbf{d}}_m^{(j)}+\hat{\delta}_n^{(j)} \end{aligned}$$

for $j=0, 1, \cdots$. This requires the exact solution of the linear systems with coefficient matrices $I-\hat{\mathscr{L}}_n$ and $I-\hat{V}_m^{(1,1)}$, of orders D_n and $D_m^{(1)}$, respectively.

For convergence, we use

$$\hat{\mathbf{u}}_m - \hat{\mathbf{u}}_m^{(j+1)} = \hat{M}_{nm}[\hat{\mathbf{u}}_m - \hat{\mathbf{u}}_m^{(j)}], \qquad j \geqq 0,$$
$$\hat{M}_{nm} = [I - P_{nm}(I - \hat{\mathscr{L}}_n)^{-1} R_{mn}(I - \hat{\mathscr{L}}_m)]\hat{\mathscr{L}}_m.$$

Then by an argument similar to that of Theorem 5,

$$\operatorname*{Sup}_{m>n} \|\hat{M}_{nm}\| \to 0 \quad \text{as } n \to \infty.$$

The matrix norm is the row norm, the operator norm induced by $\|\cdot\|_\infty$ on $\mathbb{R}^D$.

4.1. Numerical Experiment 3. Let Γ be the curve

$$\mathbf{x}(s) = \sin\left(\frac{\pi s}{\theta}\right)(\cos s, \sin s), \qquad 0 \leqq s \leqq \theta,$$

with $0 < \theta < \pi$ given. Then Γ has an interior angle of θ at the corner $\mathbf{0}$, and $\chi_0 = 1 - (\theta/\pi)$. (We refer to Γ as having a "teardrop" shape.) As earlier in Numerical Experiment 1, consider solving the interior Dirichlet problem

$$\begin{aligned} &\Delta U(\mathbf{x}) = 0, \qquad \mathbf{x} \in D, \\ &U(\mathbf{x}) = r^\beta \cos(\beta s), \qquad \mathbf{x} \in \Gamma = \partial D \end{aligned} \tag{4.24}$$

with $\mathbf{x} = (r \cos s,\ r \sin s)$ and $\beta = 2/\pi$. This is chosen to exhibit the behavior of the method of GC and the iteration method for a fairly ill behaved problem.

As before, we use the double layer representation

$$U(\mathbf{x}) = \int_\Gamma G'(\mathbf{x}, \xi) u(\xi)\, d\Gamma, \qquad \mathbf{x} \in D$$

with u found by solving (1.2). Having found $u_m \doteq u$, we evaluate the resulting potential $U_m \doteq U$ in the equivalent form

$$U_m(\mathbf{x}) = \int_\Gamma G'(\mathbf{x}, \xi)\tilde{u}_m(\xi)\, d\Gamma - u_m(\mathbf{0}). \tag{4.25}$$

Because the region D is convex, we can show the potential U_m satisfies the error bound

$$|U(\mathbf{x}) - U_m(\mathbf{x})| \leqq \|u - u_m\|_\infty, \qquad \mathbf{x} \in D. \tag{4.26}$$

The quantity $\|u - u_m\|_\infty$ can usually be estimated while solving for u_m, for example, by using Richardson's error estimation formula.

To solve the integral equation (1.2), we use the two-point Gauss-Legendre formula in (1.6), with $r = 2$. Then the number of integration nodes $\{\mathbf{x}_{ij}^{(n)}\}$ is $4n$, and we use these nodes in defining the numerical integral operator W_n in (1.7). We also take $i^* = 0$ in all cases, because empirically this has been completely satisfactory for all angles considered. (There are integration formulas (1.7) for which there are angles θ where $i^* > 0$ is necessary.)

To evaluate $U_m(\mathbf{x})$, we must use an additional numerical integration. Let $U_{m,r}(\mathbf{x})$ denote the numerical evaluation of $U_m(\mathbf{x})$ using an r-point Gauss-Legendre quadrature formula on each $[\mathbf{x}_{j-1}, \mathbf{x}_j]$ of Γ (or, more precisely, on each subinterval $[s_{j-1}, s_j]$ of $[0, 1]$). When $r = 2$, we use the density $\tilde{u}_m(\mathbf{x})$ at the integration nodes $\{\mathbf{x}_{ij}^{(m)}\}$, and these were the values of $\tilde{u}_m$ obtained in solving (4.1). For $r > 2$, we obtain the needed new integration node points by using Nyström interpolation, namely,

$$\tilde{u}_m(\mathbf{x}) = -2g(\mathbf{x}) + V_m\tilde{u}_m(\mathbf{x}). \tag{4.27}$$

The total error in approximating $U(\mathbf{x})$ is given by

$$(4.28)\qquad \begin{aligned} |U(\mathbf{x})-U_{m,r}(\mathbf{x})| &\leqq |U(\mathbf{x})-U_m(\mathbf{x})|+|U_m(\mathbf{x})-U_{m,r}(\mathbf{x})| \\ &\leqq \|u-u_m\|_\infty+|U_m(\mathbf{x})-U_{m,r}(\mathbf{x})|. \end{aligned}$$

As $r\to\infty$, $U_{m,r}(\mathbf{x})\to U_m(\mathbf{x})$. But for $\mathbf{x}$ near to Γ, the integrand $G'(\mathbf{x},\xi)$ in (4.25) is very peaked, and thus r will probably need to be quite large to make $|U_m(\mathbf{x})-U_{m,r}(\mathbf{x})|$ less significant than $\|u-u_m\|_\infty$.

Using the convergence theory from GC and that given earlier in Theorem 3, we have that

$$\|u-u_m\|_\infty=O(m^{-4}),$$

provided

$$(4.29)\qquad q>\frac{4}{\alpha},\qquad \alpha=\operatorname{Min}\{\beta,1/(1+\chi_0)\}.$$

As was noted earlier in Numerical Experiment 1, we obtain the order of convergence $O(m^{-4})$ with smaller values of q. We give results for several values of q. Rather than study the speed of convergence of $\{u_m\}$, we use points chosen along the axis of symmetry of D; in particular, take

$$\mathbf{x}=\mathbf{z}_j=\gamma_j\mathbf{x}(\theta/2),$$
$$\gamma_j=0.001,0.01,0.1,0.5,0.75,0.9,0.99$$

for $j=1,\cdots,7$. $\mathbf{z}_1$ is very close to the corner Γ at $\mathbf{0}$, and $\mathbf{z}_7$ is close to the part of Γ furthest from $\mathbf{0}$.

Tables 3, 4, and 5 show the errors in $U_{m,2}$ for $\theta=\pi/5$, for increasing values of m. (Recall that the number of equations being solved is $4m$.) Condition (4.29) says that

TABLE 3
Error in $U_{m,2}(\mathbf{z}_j)$: $q=2$.

j	E_8	E_{16}	E_{32}	E_4/E_8	E_8/E_{16}	E_{16}/E_{32}
1	9.27E−3	−4.71E−3	1.35E−3	1.2	−2.0	−3.5
2	2.20E−2	−1.31E−2	2.46E−3	1.1	−1.7	−5.3
3	−4.85E−3	−2.28E−3	−5.95E−5	−5.9	2.1	38.4
4	8.97E−3	4.44E−5	4.06E−7	−3.9	202	109
5	−9.62E−3	−2.42E−4	−7.03E−8	2.1	39.8	3440
6	−9.13E−3	−3.61E−5	−2.58E−8	−10.6	255	1400
7	5.39E−2	−1.68E−2	−5.51E−3	3.8	−3.2	3.1

TABLE 4
Error in $U_{m,2}(\mathbf{z}_j)$: $q=4$.

j	E_8	E_{16}	E_{32}	E_4/E_8	E_8/E_{16}	E_{16}/E_{32}
1	2.43E−3	1.94E−3	−3.18E−4	−2.0	1.2	−6.1
2	−1.36E−2	2.90E−3	−2.64E−4	−2.1	−4.7	−11.0
3	−1.85E−3	−2.36E−3	−6.01E−5	−23.9	0.8	39.2
4	−2.92E−2	−2.27E−3	3.67E−5	−2.8	12.9	−61.9
5	−1.89E−2	−9.64E−4	8.46E−5	−6.5	19.6	−11.4
6	5.59E−2	−4.77E−3	7.52E−5	3.4	−11.7	−63.5
7	1.94E−1	4.79E−2	−1.71E−2	1.6	4.0	−2.8

TABLE 5
Error in $U_{m,10}(\mathbf{z}_j)$: $q=2$.

j	E_8	E_{16}	E_{32}	E_4/E_8	E_8/E_{16}	E_{16}/E_{32}
1	9.61E−4	−6.04−4	−6.47E−4	−30.7	−1.6	0.9
2	−4.28E−3	6.84E−5	−5.15E−5	−0.7	−62.7	−1.3
3	−4.62E−4	−4.34E−5	−3.44E−6	9.0	10.7	12.6
4	−1.36E−4	−6.79E−6	−6.42E−7	17.9	20.0	10.6
5	−8.23E−5	−4.01E−6	−4.00E−7	10.4	20.5	10.0
6	−7.77E−5	−3.34E−6	−3.31E−7	60.0	23.2	10.1
7	−1.50E−5	−6.35E−6	−5.40E−7	−115	2.4	11.8

$q>7.2$ will yield $O(m^{-4})$ convergence in approximating $U(\mathbf{x})$ with $U_{m,2}(\mathbf{x})$. Empirically, much smaller values of q are sufficient. In the tables,

$$E_m \equiv U(\mathbf{z}_j) - U_{m,r}(\mathbf{z}_j).$$

All of the numerical calculations of this section were carried out on a microcomputer with an 80286/287 microprocessor and a standard sized memory of 640K bytes.

To show the effect of using a more accurate integration in evaluating the potential U_m of (4.25), we calculate $U_{m,10}(\mathbf{x})$ for $q=2$. The results are shown in Table 5. For points $\mathbf{z}_j$ away from the boundary Γ, the error $|U_m(\mathbf{z}_j) - U_{m,10}(\mathbf{z}_j)|$ is less than $|U(\mathbf{z}_j) - U_m(\mathbf{z}_j)|$; thus the ratios in Table 5 are indicative of the rate of convergence of $U_m(\mathbf{z}_j)$ to $U(\mathbf{z}_j)$. Empirically, this rate is about $O(m^{-3.3})$. More importantly, considerable additional accuracy in the approximation of $U(\mathbf{x})$ is obtained by simply using a more accurate integration formula. Comparing Tables 3 and 5, we have obtained much more accuracy at most points in D without needing to solve a larger linear system of equations. The results in Table 5 also show that a larger grading parameter q is needed to increase the accuracy of the approximate solution $U_m(\mathbf{x})$ for $\mathbf{x}$ near $\mathbf{0}$. In fact, $q=4$ gives much better results around $\mathbf{0}$, while slightly increasing the error around $\mathbf{z}_7$ at the portion of Γ opposite the corner $\mathbf{0}$.

We study the iteration method (4.23) for solving the linear system (4.19) arising in solving (4.24). Results have been obtained for a variety of values of q, (n, m), θ, and Dirichlet data g. The results given here illustrate that for fixed cutoff parameter δ, the iteration method converges geometrically and

$$\operatorname*{Limit}_{m,n\to\infty} \|\hat{M}_{n,m}\| = 0.$$

In Tables 6 and 7, we give the empirical rate of convergence

$$\Delta_{n,m} \equiv \operatorname*{Limit}_{j\to\infty} \frac{\|\hat{\mathbf{u}}_m^{(j)} - \hat{\mathbf{u}}_m^{(j-1)}\|_\infty}{\|\hat{\mathbf{u}}_m^{(j-1)} - \hat{\mathbf{u}}_m^{(j-2)}\|_\infty}, \tag{4.30}$$

if it exists. For cases where this limit was not evident after $\hat{\mathbf{u}}_m$ was obtained to machine precision (about 16 decimal digits), we give the geometric mean of the last several ratios in (4.30), indicating such entries in the table by *. For the corner portion Γ_1 of Γ, we give $j^*(m)$; also given is $2\delta/\theta$, the fraction of the parameterization interval $[0, \theta]$ that is associated with Γ_1. The order of the linear systems associated with $n, m, j^*(m)$ are, respectively, $4n$, $4m$, and $4j^*(m)$.

As shown in Tables 6 and 7, the rate of convergence can be improved by increasing n while keeping δ fixed (e.g., in Table 7, see $(n, m, j^*(m)) = (8, 32, 4)$ and $(16, 32, 4)$ for $q=3$). The results in both tables illustrate that the size of δ can be quite small

TABLE 6
Rates of convergence $\Delta_{n,m}$: $\theta = \pi/2$.

q	n	m	$j^*(m)$	$2\delta/\theta$	$\Delta_{n,m}$
2	4	8	2	1/16	0.0080
		16	4	1/16	0.0289
		32	8	1/16	0.0406
		64	16	1/16	0.0042*
		128	32	1/16	0.0040*
2	8	16	4	1/64	0.0079
		32	8	1/64	0.0276
		64	16	1/64	0.0396
		128	32	1/64	0.0039*
2	8	16	2	1/16	0.0145
		32	4	1/16	0.0161
		64	8	1/16	0.0010*
		128	16	1/16	0.0010*
3	8	16	2	1/512	0.014*
		32	4	1/512	0.026*
		64	8	1/512	0.031
		128	16	1/512	0.011*
3	16	32	2	1/4096	0.016
		64	4	1/4096	0.026
		128	8	1/4096	0.031

* Geometric mean of the last several ratios in (4.30).

TABLE 7
Rates of convergence $\Delta_{n,m}$: $\theta = \pi/5$.

q	n	m	$j^*(m)$	$2\delta/\theta$	$\Delta_{n,m}$
2	4	8	2	1/16	0.219
		16	4	1/16	0.161
		32	8	1/16	0.091*
		64	16	1/16	0.092*
		128	32	1/16	0.093*
2	8	16	2	1/64	0.199
		32	4	1/64	0.171
		64	8	1/64	0.091
		128	16	1/64	0.112
2	16	32	2	1/256	0.195
		64	4	1/256	0.174
		128	8	1/256	0.091*
3	8	16	2	1/512	0.460
		32	4	1/512	0.329
		64	8	1/512	0.340
		128	16	1/512	0.091
3	16	32	4	1/512	0.068
		64	8	1/512	0.061*
		128	16	1/512	0.049

* Geometric mean of the last several ratios in (4.30).

while still obtaining an acceptable rate of convergence $\Delta_{n,m}$. This is important from a practical view, to keep down the order $4j^*(m)$ of the "corner matrix" $I-\hat{V}_m^{(1,1}$, since linear systems with this matrix must be solved exactly, twice in every iteration of (4.23). The rates for $\theta=\pi/5$ are worse than for $\theta=\pi/2$, but are still generally acceptable.

4.2. Storage considerations. In carrying out the iteration (4.23), many evaluations of the kernel of the double layer integral operator W are needed. For example, with $m=128$, the linear system (4.1) being solved is of order 512. Thus there are 512^2 kernel evaluations needed to define the matrix of coefficients. These kernel evaluations should be calculated during only the first loop in (4.23) and then re-used in subsequent loops. With many machines, there is insufficient main memory to store these coefficients, and thus they must be stored on disk.

To avoid having a complicated algorithm to remember the origin of the various quantities that are loop independent, we have created two simple Fortran subroutines STORE and LOAD to use in storing these quantities onto a disk and then retrieving them. A buffer is set up in STORE. In the main iteration program, each time in loop 1 that a quantity is calculated that is loop independent, we call STORE to save it. In subsequent loops, we use LOAD to retrieve these quantities in the same order that they were stored, again using a buffer. In the subroutine STORE, as soon as the buffer fills, it is dumped in "free format" to a disk and the buffer is initialized to being empty again. The main program need not have knowledge of the size of the buffer nor of when the subroutines STORE and LOAD are using the disk. We have found that the cost in time of the first loop is from 10 to 20 times that of subsequent loops, and the savings after the first loop are due entirely to the storing of the quantities that are loop independent.

Appendix. We sketch proofs of the inequalities (2.12) and (2.13), with $\mathscr{K}$ replaced by $\mathscr{N}$. Recall that the double layer integral operator $2W$ can be decomposed as

$$2W=\mathscr{K}+\mathscr{N},$$

where $\mathscr{K}$ is the double layer operator on an open wedge and N is a compact remainder. The operator $\mathscr{K}$ is defined in (2.2) and in the following. Let $K(s,\sigma)$ and $N(s,\sigma)$ denote the kernel functions of $\mathscr{K}$ and $\mathscr{N}$, respectively. We must show that for $k\geqq 0$,

$$|s^kD_s^k(\mathscr{N}_{i^*,n}v)(\mathbf{x}(s))|\leqq A_k'\|v\|_\infty,\qquad s\in[-1,1]\backslash\{0\},\tag{A.1}$$

$$|\sigma^{k+1}D_\sigma^k(N(s,\sigma))|\leqq B_k',\qquad s,\sigma\in[-1,1]\backslash\{0\},\tag{A.2}$$

for some constants A_k', B_k'.

We first note that if $-1\leqq s$, $\sigma<0$ (or $0<s$, $\sigma\leqq 1$), then the function $K(s,\sigma)\equiv 0$ and the function $N(s,\sigma)$ are the standard double layer potential functions one obtains for a smooth boundary. Thus the above inequalities are satisfied easily. From here on, we are interested in the case $-1\leqq s<0<\sigma\leqq 1$ (and the analogous case $-1\leqq\sigma<0<s\leqq 1$). We assume $-2W$ is defined on an open curved wedge Γ of the type pictured in Fig. 4. For simplicity, we assume Γ has a C^∞ parameterization on both arms of the wedge, with the origin $\mathbf{0}$ the sole point at which the tangent does not exist. (For the case of Γ a closed contour, the operator $-2W$ can be decomposed as two operators, as in § 4, with one operator the double layer integral operator on an open wedge and the second operator a smoothing compact integral operator.) Let Γ be as pictured in Fig. 4, with one arm tangent to the positive x-axis, the vertex of Γ at the origin, and the second arm making an angle of $\theta=(1-\chi)\pi$ with the positive x-axis. We let Γ' denote the wedge with linear arms, tangent to Γ; the lengths of the arms of Γ' are to

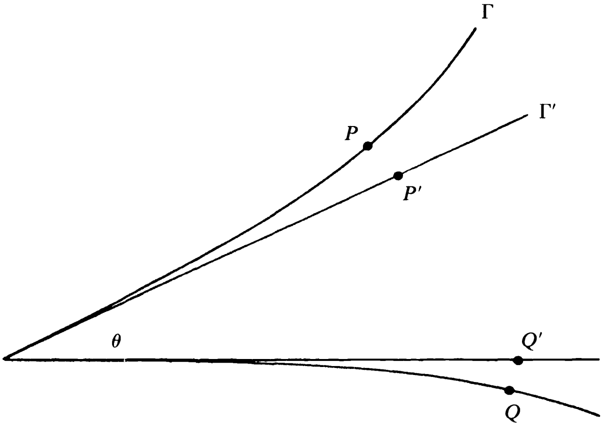


FIG. 4. *Corner region with tangent wedge.*

be the same as the corresponding arms of Γ. We are interested in the double layer integral operator defined on Γ with the kernel function

$$V(P, Q) = \frac{1}{\pi}\mathbf{n}_Q \cdot \frac{Q-P}{|Q-P|^2}. \tag{A.3}$$

In line with the earlier restriction $-1 \leqq s < 0 < \sigma \leqq 1$, the points P and Q are to be on different arms of the wedge Γ, as pictured in Fig. 4.

To obtain $N(s, \sigma)$, we need the parametric representation of Γ in terms of arc length. The lower and upper arms of the tangent wedge Γ′ are given by

$$Q' = (\sigma, 0), \qquad 0 \leqq \sigma \leqq 1,$$
$$P' = (s \cos\theta, s \sin\theta), \qquad 0 \leqq s \leqq 1.$$

Here σ and s denote arc length along Γ′, measured from (0, 0); for convenience, we let σ and s both vary over [0, 1]. For the parameterization of Γ, the use of arc length σ and s leads to

$$\begin{aligned} Q &= (\sigma + \sigma^3 c(\sigma), \sigma^2 b(\sigma)), && 0 \leqq \sigma \leqq 1, \\ P &= (s\cos\theta + s^2 a(s), s\sin\theta + s^2 d(s)), && 0 \leqq s \leqq 1 \end{aligned} \tag{A.4}$$

with $a, b, c, d \in C^\infty[0, 1]$. The arc length parameterization also imposes other consistency restrictions on these functions, for example,

$$6c(0) + 4b(0)^2 = 0.$$

But such conditions are not needed in our derivations. The normal to Γ at Q is given by

$$\mathbf{n}_Q = (2\sigma b + \sigma^2 b', -1 - 3t^2 c - t^3 c').$$

We can now give an explicit formula for the function $V(s, \sigma) \equiv V(P, Q)$ of (A.3). The formula is algebraically complicated, and for that reason we consider here only the right angle case $\theta = \pi/2$. The techniques used and results obtained will generalize to the case of a general angle θ, $0 < \theta < 2\pi$.

With $\theta = \pi/2$, the kernel function V of (A.3) becomes

$$V(s, \sigma) = \frac{1}{\pi}\frac{(2b\sigma + \sigma^2 b')(\sigma + \sigma^3 c - s^2 a) - (1 + 3c\sigma^2 + \sigma^3 c')(\sigma^2 b - s - s^2 d)}{[\sigma + \sigma^3 c - s^2 a]^2 + [s + s^2 d - \sigma^2 b]^2}. \tag{A.5}$$

Note that from the definition (A.3), the denominator of (A.5) is zero if and only if $P=Q=0$, or equivalently, $s=\sigma=0$. Also, from (2.3),

$$K(s,\sigma)=\frac{1}{\pi}\frac{s}{s^2+\sigma^2}.$$

Subtracting $N=V-K$, we obtain

$$\text{(A.6)} \qquad N(s,\sigma)=\frac{\alpha(s,\sigma)}{\beta(s,\sigma)}=\frac{\alpha(s,\sigma)}{|P'-Q'|^2|P-Q|^2}.$$

The denominator

$$\beta(s,\sigma)=\{[\sigma+\sigma^3c-s^2a]^2+[s+s^2d-\sigma^2b]^2\}\{s^2+\sigma^2\}$$

is a "polynomial" in s,σ of degree 4 exactly, and it is zero if and only if $P=Q=0$. The numerator $\alpha(s,\sigma)$ is a "polynomial" in s,σ and all of its terms are of degree $\geqq 4$. Both of these polynomials have coefficients that contain the C^∞ functions a, b, c, d, c', d', but their presence will not affect the validity of the following arguments, since we will be assuming s and σ are sufficiently close to zero. Then, for example,

$$\beta \doteq [s^2+\sigma^2]^2.$$

Because the numerator and denominator are C^∞ functions for $0<s,\sigma\leqq 1$, and because the denominator is zero only when $s=\sigma=0$, we can check the boundedness of $N(s,\sigma)$ and its derivatives by considering only a neighborhood in (s,σ) about $(0,0)$. For example, $N(s,\sigma)$ is easily shown to be bounded for (s,σ) near $(0,0)$ by using Taylor's theorem and the inequalities

$$\text{(A.7)} \qquad \frac{s^2}{s^2+\sigma^2}\leqq 1, \quad \frac{\sigma^2}{s^2+\sigma^2}\leqq 1, \quad \frac{s\sigma}{s^2+\sigma^2}\leqq\frac{1}{2}.$$

We first show (A.2). Introduce

$$N^{(k)}(s,\sigma)=\sigma D_\sigma N^{(k-1)}(s,\sigma), \qquad k\geqq 1,$$
$$N^{(0)}(s,\sigma)=\sigma N(s,\sigma).$$

The proof of (A.2) is equivalent to showing

$$\text{(A.8)} \qquad |N^{(k)}(s,\sigma)|\leqq c_k, \quad 0<s,\sigma\leqq 1, \quad k\geqq 0.$$

The proof for $k=0$ was described in the preceding paragraph; we can write

$$N^{(0)}(s,\sigma)=\frac{\sigma\alpha(s,\sigma)}{\beta(s,\sigma)}\equiv\frac{\gamma(s,\sigma)}{\beta(s,\sigma)}.$$

All terms in the numerator γ have degree $\geqq 5$. For $k=1$,

$$N^{(1)}(s,\sigma)=\sigma D_\sigma(\gamma/\beta)=\frac{\sigma[\gamma'\beta-\gamma\beta']}{\beta^2}.$$

This fraction has the same form as previously, but now the denominator is β^2 rather than β. All terms in the numerator have degree one greater (or more) than that of the denominator. As a consequence, we can again easily show $N^{(1)}(s,\sigma)$ is uniformly bounded for (s,σ) near $(0,0)$. This form of argument can be continued inductively to show (A.8) for all k.

For showing (A.1), we first refer the reader to the corresponding proof of (2.12) in (GC, Lemma 2(ii)). Using that proof to show (A.1), it is sufficient to show

$$\text{(A.9)} \qquad |s^k D_s^k N(s,\sigma)| \leqq c_k, \qquad 0 < s, \sigma \leqq 1, k \geqq 0.$$

Equivalently, show

$$\text{(A.10)} \qquad |N_k(s,\sigma)| \leqq d_k, \qquad 0 < s, \sigma \leqq 1, \quad k \geqq 0,$$

where

$$N_0(s,\sigma) = N(s,\sigma), \qquad N_k(s,\sigma) = sD_s N_{k-1}(s,\sigma), \quad k \geqq 1.$$

The proof of (A.10) when $k = 0$ has been given previously. The cases $k \geqq 1$ are essentially the same as for (A.8). The only major difference is that all terms in the numerator will have degree greater than or equal to the degree of the denominator, but this is still sufficient for the proof of boundedness. This completes the proof.

Acknowledgments. We wish to thank everyone at the Department of Mathematics, University of Queensland, Queensland, Australia, for their hospitality during our visit, especially Graeme Chandler.

REFERENCES

K. E. Atkinson (1973), *Iterative variants of the Nyström method for the numerical solution of integral equations*, Numer. Math., 22, pp. 17-31.

——— (1976), *A Survey of Numerical Methods for the Solution of Fredholm Integral Equations of the Second Kind*, Society for Industrial and Applied Mathematics, Philadelphia, PA.

K. E. Atkinson and A. Bogomolony (1987), *The discrete Galerkin method for integral equations*, Math. Comp., 48, pp. 596-616.

K. E. Atkinson and F. R. deHoog (1984), *The numerical solution of Laplace's equation on a wedge*, IMA J. Numer. Anal., 4, pp. 19-41.

K. E. Atkinson and I. G. Graham (1988), *An iterative variant of the Nyström method for boundary integral equations on nonsmooth boundaries*, in The Mathematics of Finite Elements and Applications, J. R. Whiteman, ed., Academic Press, London, pp. 297-304.

G. Bruhn and W. Wendland (1967), *Über die näherungsweise Lösung von linearen Funktionalgleichungen*, in Funktionalanalysis, Approximations-theorie, Numerische Mathematik, L. Collatz, G. Meinardus, and H. Unger, eds., Birkhauser-Verlag, Basel, pp. 136-164.

T. F. Chan (1984), *Deflation techniques and block elimination algorithms for solving bordered singular systems*, SIAM J. Sci. Statist. Comput., 5, pp. 121-134.

M. Costabel and E. Stephan (1983), *Curvature terms in the asymptotic expansions for solutions of boundary integral equations on curved polygons*, J. Integral Equations, 5, pp. 353-371.

I. G. Graham and G. A. Chandler (1988), *High order methods for linear functionals of solutions of second kind integral equations*, SIAM J. Numer. Anal., 25, pp. 1118-1137.

F.-K. Hebeker (1986), *Efficient boundary element methods for three-dimensional exterior viscous flows*, Numer. Meth. Partial Differential Equations, 2, pp. 273-297.

——— (1988), *On the numerical treatment of viscous flows against bodies with corners and edges by boundary element and multigrid methods*, Numer. Math., 52, pp. 81-99.

W. Hackbusch (1985), *Multigrid Methods and Applications*, Springer-Verlag, Berlin.

M. Jaswon and G. Symm (1977), *Integral Equation Methods in Potential Theory and Elastostatics*, Academic Press, New York.

J. Mandel (1985), *On multilevel iterative methods for integral equations of the second kind*, Numer. Math., 46, pp. 147-157.

J. Radon (1919), *Über die Randwertaufgaben beim logarithmischen Potential*, Sitzungsberichte der Akademie der Wissenschafter Wien, 128 Abt. IIa, pp. 1123-1167.

L. Reichel (1988), *Fast solution methods for Fredholm integral equations of the second kind*, Report 88/20, IBM Bergen Scientific Centre, Bergen, Norway.

H. SCHIPPERS (1982), *Application of multigrid methods for integral equations to two problems from fluid dynamics*, J. Comput. Phys., 48, pp. 441-461.

——— (1985), *Multigrid methods for boundary integral equations*, Numer. Math., 46, pp. 351-363.

——— (1987), *Multigrid methods in boundary element calculations*, in Boundary Elements IX: Vol. 1—Mathematical and Computational Aspects, C. Brebbia, W. Wendland, and G. Kuhn, eds., Springer-Verlag, Berlin, New York, pp. 475-492.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 3, pp. 723-726, May 1992

006

MODIFICATION OF THE HOUSEHOLDER METHOD BASED ON THE COMPACT WY REPRESENTATION*

CHIARA PUGLISI†

Abstract. This paper presents a modification of the block Householder method based on the compact **WY** representation [R. Schreiber and C. Van Loan, *SIAM J. Sci. Statist. Comput.*, 10 (1989), pp. 52-57]. It is modified in order to introduce more matrix-matrix operations.

Key words. Householder method, compact **WY** representation, matrix-matrix multiplication, Level 3 BLAS

AMS(MOS) subject classification. 65F

1. Introduction. The Householder method for factorizing an $m \times n$ matrix $\mathbf{A}$, $n \leqq m$, consists of building n Householder matrices $\mathbf{H}_i = (\mathbf{I} - (2/\mathbf{u}_i^T\mathbf{u}_i)\mathbf{u}_i\mathbf{u}_i^T)$ so that if

$$\mathbf{Q} = \mathbf{H}_n\mathbf{H}_{n-1} \cdots \mathbf{H}_1, \tag{1.1}$$

and $\mathbf{QA} = \mathbf{R}$, then $\mathbf{R}$ is upper triangular. The Householder matrices are orthogonal and so it follows that $\mathbf{Q}$ is orthogonal. We are interested in "Householder methods" using as many matrix-matrix operations as possible in order to increase the efficiency of the algorithm on vector and parallel machines (see [2]). Some block techniques have been studied by Schreiber and Van Loan [3], all of them using a block representation of the matrix $\mathbf{Q}$. We study one of them, the so-called compact **WY** representation (also used in the new LAPACK [1]) and modify it to introduce more matrix-matrix operations. We describe the **WY** representation in § 2 and our modified version in § 3. We show the results of some numerical experiments in § 4.

2. The compact WY representation. A generic Householder matrix $\mathbf{H}_i = (\mathbf{I} - (2/\mathbf{u}_i^T\mathbf{u}_i)\mathbf{u}_i\mathbf{u}_i^T)$ can also be written as

$$\mathbf{H}_i = \mathbf{I} - \mathbf{w}_i\mathbf{w}_i^T, \tag{2.1}$$

where

$$\|\mathbf{w}_i\|_2 = \sqrt{2}. \tag{2.2}$$

In the compact **WY** representation $\mathbf{Q}$ is written as

$$\mathbf{Q} = \mathbf{I} - \mathbf{YTY}^T, \tag{2.3}$$

where $\mathbf{Y}$ is a rectangular $m \times n$ matrix, and each of its columns is a Householder vector $\mathbf{w}_i$, and $\mathbf{T}$ is a lower unit triangular matrix. For the proof, we refer to Schreiber and Van Loan [3] noticing that they proved (2.3) for $\mathbf{Q}^T$ using Householder matrices of the type $\mathbf{I} - 2\mathbf{v}_i\mathbf{v}_i^T(\|\mathbf{v}_i\|_2 = 1)$.

The algorithm for computing $\mathbf{T}$ is the following:

$$\mathbf{T}_1 = 1, \qquad \mathbf{T}_i = \begin{bmatrix} \mathbf{T}_{i-1} & 0 \\ \mathbf{z}_i & 1 \end{bmatrix}, \tag{2.4}$$

where $\mathbf{z}_i = \mathbf{w}_i^T\mathbf{Y}_{i-1}\mathbf{T}_{i-1}$.

* Received by the editors July 30, 1990; accepted for publication (in revised form) February 8, 1991.

† Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique (CERFACS), 42 Avenue Gustave-Coriolis, 31057 Toulouse Cedex, France (puglisi @ cerfacs. fr).

Suppose $n = rN$; then, using the compact **WY** representation, we have the following block algorithm (these are the basic steps of the algorithm of the new LAPACK).

```
ALGORITHM 1.
for $k = 1, N$
  for $s = 1, r$
    1. $i = (k-1)r + s$
    2. compute $\mathbf{w}_i$
    3. if $s = 1$ then $\mathbf{Y}_i = \mathbf{w}_i$
         else $\mathbf{Y}_i = [\mathbf{Y}_{i-1}\mathbf{w}_i]$
       endif
    4. $\mathbf{A}(i\colon m;\ i+1\colon kr) = (\mathbf{I} - \mathbf{w}_i\mathbf{w}_i^T)\mathbf{A}(i\colon m;\ i+1\colon kr)$
  endfor
  if $k < N$ then
    5. compute $\mathbf{T}_k$
    6. $j = kr+1$; $\mathbf{A}((k-1)r+1\colon m; j\colon n) = (\mathbf{I} - \mathbf{Y}_{kr}\mathbf{T}_k\mathbf{Y}_{kr}^T)\mathbf{A}((k-1)r+1\colon m; j\colon n)$
  endif
endfor
```

$\mathbf{A}(a\colon b;\ c\colon d)$ means the block of $\mathbf{A}$ with rows from a to b and columns from c to d.

Step 5 of the algorithm involves a for-loop over i, $i = (k-1)r + 1, kr$, and each iteration in the loop involves two matrix-vector operations:

$$\mathbf{x}_i^T = \mathbf{w}_i^T\mathbf{Y}_{i-1} \qquad \text{(DGEMV)},$$

$$\mathbf{z}_i = -\mathbf{x}_i^T\mathbf{T}_{i-1} \qquad \text{(DTRMV)}.$$

Step 6 involves three matrix-matrix operations:

$$\mathbf{B} = \mathbf{Y}_{kr}^T\mathbf{A} \qquad \text{(DGEMM)},$$

$$\mathbf{B} = \mathbf{T}_k\mathbf{B} \qquad \text{(DTRMM)},$$

$$\mathbf{A} = \mathbf{A} - \mathbf{Y}_{kr}\mathbf{B} \qquad \text{(DGEMM)}.$$

We now modify the algorithm in order to substitute the for-loop with the matrix-vector operations in step 5 with a matrix-matrix operation.

3. The new algorithm. As we mentioned before, the **Q** matrix in (1.1) is orthogonal, that is,

$$\mathbf{Q}^{-1} = \mathbf{Q}^T.$$

In the case of **Q** in compact **WY** representation, this means that

$$\mathbf{Q}^{-1} = (\mathbf{I} - \mathbf{YTY}^T)^T = \mathbf{I} - \mathbf{YT}^T\mathbf{Y}^T. \tag{3.1}$$

Another way to compute the inverse of (2.3) is to use the Woodbury-Morrison formula, which can be expressed as the following: Given

$$\mathbf{A} = \mathbf{B} + \mathbf{UV}^T,$$

where $\mathbf{A}$, $\mathbf{B}$ are invertible $m \times m$ matrices, and $\mathbf{U}$, $\mathbf{V}$ are $m \times p$ matrices, $p < m$, we can compute its inverse by

$$\mathbf{A}^{-1} = \mathbf{B}^{-1} - \mathbf{B}^{-1}\mathbf{U}(\mathbf{I}_p + \mathbf{V}^T\mathbf{B}^{-1}\mathbf{U})^{-1}\mathbf{V}^T\mathbf{B}^{-1}. \tag{3.2}$$

Substituting in (3.2) $\mathbf{B} = \mathbf{I}$, $\mathbf{U} = -\mathbf{Y}$, and $\mathbf{V}^T = \mathbf{T}\mathbf{Y}^T$, we obtain

$$(\mathbf{I} - \mathbf{Y}\mathbf{T}\mathbf{Y}^T)^{-1} = \mathbf{I} + \mathbf{Y}(\mathbf{I}_p - \mathbf{T}\mathbf{Y}^T\mathbf{Y})^{-1}\mathbf{T}\mathbf{Y}^T. \tag{3.3}$$

Comparing (3.1) and (3.3) gives us

$$\mathbf{I} + \mathbf{Y}(\mathbf{I}_p - \mathbf{T}\mathbf{Y}^T\mathbf{Y})^{-1}\mathbf{T}\mathbf{Y}^T = \mathbf{I} - \mathbf{Y}\mathbf{T}^T\mathbf{Y}^T.$$

Postmultiplying by $\mathbf{Y}$ and premultiplying by $\mathbf{Y}^T$ gives us

$$\mathbf{Y}^T\mathbf{Y}(\mathbf{I}_p - \mathbf{T}\mathbf{Y}^T\mathbf{Y})^{-1}\mathbf{T}\mathbf{Y}^T\mathbf{Y} = -\mathbf{Y}^T\mathbf{Y}\mathbf{T}^T\mathbf{Y}^T\mathbf{Y}.$$

Because $\mathbf{Y}^T\mathbf{Y}$ is invertible,

$$\begin{aligned}(\mathbf{I}_p - \mathbf{T}\mathbf{Y}^T\mathbf{Y})^{-1}\mathbf{T} &= -\mathbf{T}^T,\\ (\mathbf{I}_p - \mathbf{T}\mathbf{Y}^T\mathbf{Y}) &= -\mathbf{T}\mathbf{T}^{-T},\\ \mathbf{I}_p &= \mathbf{T}(\mathbf{Y}^T\mathbf{Y} - \mathbf{T}^{-T}),\\ \mathbf{T}^{-1} + \mathbf{T}^{-T} &= \mathbf{Y}^T\mathbf{Y},\end{aligned}$$

so that

$$\mathbf{T}^{-1}(i, i) = \frac{\mathbf{w}_i^T\mathbf{w}_i}{2}, \quad \mathbf{T}^{-1}(i, j) = \mathbf{w}_i^T\mathbf{w}_j, \quad i > j. \tag{3.4}$$

Note also that $\mathbf{w}_i^T\mathbf{w}_i/2 = 1$, $i = 1, \cdots, n$, from (2.2).

Using (3.4), steps 5 and 6 of Algorithm 1 can be modified as follows.

```
5.
  $\mathbf{T}_k^{-1} = \mathbf{Y}_{kr}^T\mathbf{Y}_{kr}$          (DSYRK)
  for $i = 1, r$
    $\mathbf{T}_k^{-1}(i, i) = 1$
  endfor
6.
  $\mathbf{B} = \mathbf{Y}_{kr}^T\mathbf{A}$,            (DGEMM)
  $\mathbf{B} = (\mathbf{T}_k^{-1})^{-1}\mathbf{B}$,       (DTRSM)
  $\mathbf{A} = \mathbf{A} - \mathbf{Y}_{kr}\mathbf{B}$         (DGEMM)
```

Essentially, both algorithms compute the matrix $\mathbf{T}_k^{-1}$, but in the first one we invert the matrix $\mathbf{T}_k^{-1}$ explicitly and then perform a matrix-matrix multiplication; in the second one we solve several triangular systems with $\mathbf{T}_k^{-1}$ as the coefficient matrix. For factorizing an $m \times n$ matrix with blocks of size r, the modified LAPACK algorithm does $(n-r) \times (2m - \frac{3}{2}n + (n^2/3) + 2r - \frac{10}{3})$ operations more than the LAPACK algorithm.

4. Numerical results. We show in Table 1 the time to factorize three matrices with the same algorithm as in LAPACK and the modified LAPACK algorithm with different block sizes. All the experiments have been done on an Alliant FX/80.

As may be noticed, the modified LAPACK algorithm is better than the "LAPACK" algorithm, but not very much so. This reflects the poor performance of the subroutine DSYRK on the target machine.

TABLE 1
Execution time of the two algorithms with different block sizes. A; 1024 × 512, B: 512 × 512, C: 1024 × 1024.

Size of the blocks		LAPACK	Modified LAPACK
16	Matrix A	12.67	11.88
	Matrix B	6.09	5.48
	Matrix C	31.49	29.20
32	Matrix A	11.32	9.98
	Matrix B	5.12	4.62
	Matrix C	28.04	26.34
64	Matrix A	11.64	10.80
	Matrix B	5.02	4.66
	Matrix C	27.16	25.45
128	Matrix A	15.99	15.13
	Matrix B	5.92	5.34
	Matrix C	34.25	32.81

Finally, it can be shown that from the stability point of view, the new algorithm is as stable as the old one. The propagation of the error in the two algorithms is different because in the old one we compute $\mathbf{T}_k$ inverting in place $\mathbf{T}_k^{-1}$ and we successively update by multiplying by $\mathbf{T}_k$, whereas in the new one $\mathbf{T}_k^{-1}$ is stored and we update by solving lower triangular systems, but the final errors are of the same order.

Acknowledgments. I would like to thank Mario Arioli and Iain Duff for supporting me in writing this communication.

REFERENCES

[1] J. DEMMEL, J. J. DONGARRA, J. DU CROZ, A. GREENBAUM, S. HAMMARLING, AND D. SORENSEN, *Prospectus for the development of a linear algebra library for high-performance computers*, Tech. Memorandum No. 97, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, September 1987.

[2] J. J. DONGARRA, J. DU CROZ, I. S. DUFF, AND S. HAMMARLING, *A set of level* 3 *basic linear algebra subprograms*, ACM Trans. Math. Software, 16 (1988), pp. 1-17.

[3] R. SCHREIBER AND C. VAN LOAN, *A storage-efficient* WY *representation for products of Householder transformations*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 52-57.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 3, pp. 727–741, May 1992

007

SOME APPLICATIONS OF THE RANK REVEALING QR FACTORIZATION*

TONY F. CHAN† AND PER CHRISTIAN HANSEN‡

Abstract. The rank revealing QR factorization of a rectangular matrix can sometimes be used as a reliable and efficient computational alternative to the singular value decomposition for problems that involve rank determination. This is illustrated by showing how the rank revealing QR factorization can be used to compute solutions to rank deficient least squares problems, to perform subset selection, to compute matrix approximations of given rank, and to solve total least squares problems.

Key words. rank revealing QR factorization, numerical rank, rank deficient problems, subset selection, matrix approximation, total least squares

AMS(MOS) subject classifications. 65F25, 65F20

1. Introduction. One of the more intricate problems in numerical linear algebra is to find the numerical rank of a matrix. This computational problem is the heart of many numerical methods, such as subset selection, total least squares, regularization, and matrix approximation. The *singular value decomposition* (SVD) is undoubtedly the most reliable method for computing the numerical rank. A big disadvantage of the SVD is, however, the high computational complexity of the standard SVD algorithm, as compared to a QR factorization, for example, [12, p. 248]. The same is true for SVD algorithms based on Jacobi iteration. SVD algorithms for sparse or structured matrices based on Lanczos iteration are faster, but they have problems with computation of the smallest singular values and, therefore, are not reliable for numerical rank determination.

A number of alternative, less computationally demanding, methods have been proposed. Most of these are based on a QR factorization with column pivoting [4], [18], [19], but the numerical rank computed by these methods is not entirely reliable (see [12, §5.5.7] and [1, §7]), and none of these methods are suited for sparse matrices because of the column pivoting. Similar QR-based methods specially designed for sparse matrices [16], [21], based solely on detecting small elements on the diagonal of the triangular matrix, are not reliable either. Another alternative method, the partial SVD (PSVD) [25], is as reliable as the SVD. However, a complete reduction to bidiagonal form is required, so its complexity is still higher than that of a QR factorization, and it is not well suited to general sparse matrices.

The most promising alternative to the SVD is the *rank revealing* QR *factorization* (RRQR factorization) defined by Chan [6], [7] (we note that similar ideas were proposed independently by Foster [10]). An important existence proof of RRQR factorizations is given in [17]. The RRQR factorization will reveal the numerical rank of any matrix, because it is guaranteed to capture all the small singular values of the matrix by producing reasonably tight upper and lower bounds for these singular

* Received by the editors May 14, 1990; accepted for publication April 10, 1991. This research was supported by NATO Collaborative Research grant 5-2-05/RG900098.

† Department of Mathematics, University of California, 405 Hilgard Ave., Los Angeles, California 90024 (chan@math.ucla.edu). The work of this author was supported by Army Research Office contract DAAL03-88-K-0085, by Department of Energy contract DE-FG-03-87-ER-25037, and by National Science Foundation contract NSF-DMS87-14612.

‡ UNI•C (Danish Computing Center for Research and Education), Building 305, Technical University of Denmark, DK-2800 Lyngby, Denmark (unipch@wuli.uni-c.dk).

values. In addition, the RRQR algorithm produces a set of linearly independent vectors that span a good approximation to the numerical null-space of the matrix. This information is sufficient to solve many problems in numerical linear algebra.

The computational complexity of the RRQR algorithm is only slightly larger than that of the standard QR algorithm, as long as the nullity is small compared to the dimensions of the matrix. A Fortran implementation of the algorithm is now available from ACM TOMS [22]. Moreover, the RRQR factorization of a sparse matrix can be computed efficiently without destroying the sparsity pattern of the matrix, and Bischof and Hansen [1] have demonstrated how to implement the RRQR factorization algorithm with a minimum of column interchanges. Thus, we feel that the time is ripe for using the RRQR factorization in numerical linear algebra.

Chan and Hansen [8] showed how truncated SVD solutions can be computed efficiently by means of the RRQR factorization, and Hansen, Sekhii, and Shibahashi [15] use RRQR factorizations to regularize discrete ill-posed problems. Comon and Golub [9] use RRQR factorizations in conjunction with Lanczos block-bidiagonalization. Bischof and Shroff [2] have shown how to use the null-space information from an RRQR factorization in conjunction with the "signal subspace" approach to parameter estimation in signal processing. In this paper, we illustrate several other important applications of the RRQR factorization in numerical linear algebra.

Stewart [24] has recently proposed a related factorization, namely, a rank revealing complete orthogonal decomposition, which is particularly suited to "subspace tracking" in signal processing. This factorization is computationally more expensive than the RRQR factorization, but *updating of the null-space* from Stewart's factorization is cheaper than updating the null-space from an RRQR factorization.

Our paper is organized as follows. In §2, we summarize the most important properties of the RRQR factorization. Then, we show how the RRQR factorization can be used in rank deficient least squares problems (§3), in subset selection problems (§4), and in matrix approximations (§5). Finally, we demonstrate in §6 how the RRQR factorization can be used to solve total least squares problems with full rank as well as rank deficient coefficient matrices.

We shall use two-norms almost entirely, so we use the abbreviation $\|\cdot\|$ for $\|\cdot\|_2$. The range (column space) of a matrix is denoted by $\mathcal{R}(\cdot)$. Throughout this paper, in addition to our new results, we include a few results already published in other manuscripts. We feel that the present constellation of this material will provide new insight into the applications and practical use of the RRQR factorization.

2. RRQR factorizations.

Throughout the paper, we assume that the matrix A has been properly scaled, for example, such that the uncertainties in its elements are roughly of the same size [23]. The numerical rank, or *ϵ-rank*, of A with respect to the tolerance ϵ is defined by

$$k = k(A,\epsilon) \equiv \min_{\|A-B\|\leq\epsilon} \operatorname{rank}(B). \tag{1}$$

In other words, the ϵ-rank of A is equal to the number of columns in A that are guaranteed to be linearly independent for any perturbation of A with norm less than or equal to the tolerance ϵ. As a guide to choosing this tolerance, it is customary to let ϵ reflect the uncertainties in A [23]. See also [11] and [12, §2.5.4] for more details.

The most reliable way of computing the ϵ-rank of A is via its SVD. Assume for simplicity that $A \in \Re^{m\times n}$ with $m \geq n$. Then the SVD of A is

$$A = U\Sigma V^T = \sum_{i=1}^{n} \mathbf{u}_i\, \sigma_i\, \mathbf{v}_i^T, \tag{2}$$

where $U = [\mathbf{u}_1, \cdots, \mathbf{u}_n]$ and $V = [\mathbf{v}_1, \cdots, \mathbf{v}_n]$ are matrices with orthonormal columns, and $\Sigma = \mathrm{diag}(\sigma_1, \cdots, \sigma_n)$ is an $m \times n$ diagonal matrix whose diagonal entries, the *singular values* of A, are ordered such that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. From the orthonormality of the columns of U and V it follows that $\|A\mathbf{v}_i\| = \sigma_i$, $i = 1, \cdots, n$. It is straightforward to show that if k is the number of singular values strictly greater than ϵ, i.e., $\sigma_k > \epsilon \geq \sigma_{k+1}$, then k is the ϵ-rank of A as defined in (1). For every singular value $\sigma_i < \epsilon$, the corresponding right singular vector $\mathbf{v}_i$ is a numerical null-vector of A in the sense that $\|A\,\mathbf{v}_i\| \leq \epsilon$. It is therefore natural to define the *numerical null-space* of A as the space spanned by the vectors $\mathbf{v}_{k+1}$ through $\mathbf{v}_n$:

$$\mathcal{N}_k(A) \equiv \mathrm{span}\{\mathbf{v}_{k+1}, \cdots, \mathbf{v}_n\}. \tag{3}$$

We can now define a rank revealing QR factorization of A as a special QR factorization $A\Pi = QR$, which is guaranteed to reveal the ϵ-rank k of A in displaying elements in the lower portion of R with magnitude of the order σ_{k+1} or less. An RRQR factorization thus has the form

$$A\Pi = QR = Q\begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}, \tag{4}$$

where Π is a permutation matrix, Q has orthonormal columns, R_{11} is a $k \times k$ matrix with condition number approximately equal to σ_1/σ_k, $\|R_{22}\|$ is of the order σ_{k+1}, and k is the ϵ-rank of A. Such an RRQR factorization of A is not unique, and different RRQR algorithms may produce different factorizations. The key idea in all RRQR algorithms is, however, the same: first compute any QR factorization of A and then construct Π and Q by building up R_{22} one row at a time, starting from the bottom. Assume that a trailing $(n-i) \times (n-i)$ submatrix with small norm has already been generated. In the next step, the RRQR algorithm then proceeds as follows:

1. Compute the smallest singular value δ_i and the corresponding right null-vector $\mathbf{w}^{(i)} \in \Re^i$ of the leading $i \times i$ submatrix $R^{(i)}$ of R, such that

$$\|\mathbf{w}^{(i)}\| = 1, \qquad \|R^{(i)}\mathbf{w}^{(i)}\| = \delta_i \leq \sigma_i. \tag{5}$$

   The inequality $\delta_i \leq \sigma_i$ follows immediately from the interlacing inequality for singular values [12, Cor. 8.3.3].
2. Find the permutation that permutes the largest element in absolute value of $\mathbf{w}^{(i)}$ to the bottom.
3. Apply this permutation to the columns of $R^{(i)}$ and compute a new QR factorization of this matrix.
4. The (i,i)-element of $R^{(i)}$ is now *guaranteed* to be of the order δ_i.

This process continues until $\delta_i > \epsilon$ and then the ϵ-rank k of A, given by (1), is equal to i. The vectors $\mathbf{w}^{(i)}$ are padded with zeros and gathered in a matrix

$$W = \begin{pmatrix} W_1 \\ W_2 \end{pmatrix}$$

in such a way that $W_2 \in \Re^{(n-k)\times(n-k)}$ is upper triangular. The resulting column permutation matrix Π seeks to make W_2, produced by the RRQR algorithm, as well conditioned as possible (a priori upper bounds for $\|W_2^{-1}\|$, which depend on the particular RRQR algorithm, can be found in [1], [8], [10]). It is very important that the submatrix W_2 be well conditioned, for then we are guaranteed to obtain tight bounds for the singular values of A due to the following theorem.

THEOREM 2.1. *Let $R_{22}^{(i)}$ and $W_2^{(i)}$ denote the lower right $(n-i+1)\times(n-i+1)$ submatrices of R_{22} and W_2, respectively. Also, let δ_i denote the smallest singular value of the leading principal $i \times i$ submatrices of R. Then for $i = k+1, \cdots, n$:*

$$\frac{\sigma_i}{\sqrt{n-i+1}\,\|(W_2^{(i)})^{-1}\|} \le \delta_i \le \sigma_i \le \|R_{22}^{(i)}\| \le \sigma_i\sqrt{n-i+1}\,\|(W_2^{(i)})^{-1}\|. \tag{6}$$

Proof. See [7, Cor. 4.1]. □

Theorem 2.1 shows that the quantities δ_i and $\|R_{22}^{(i)}\|$ provide easily computed lower and upper bounds for the singular values σ_i. Moreover, the outermost bounds in (6) show that if $\|(W_2^{(i)})^{-1}\|$ is not large, then δ_i and $\|R_{22}^{(i)}\|$ are guaranteed to be tight bounds for σ_i. Therefore, the ϵ-rank of A will always be revealed from inspection of the upper and lower bounds for σ_i produced by the RRQR algorithm. In addition, Theorem 2.1 guarantees that $\|R_{22}\|$ is indeed of the order σ_{k+1}.

The matrix W produced during the RRQR algorithm is such an integral part of the RRQR that one may almost consider it a part of the factorization, the reason being that $\mathcal{R}(\Pi W)$ is a good approximation to the numerical null-space $\mathcal{N}_k(A)$. In fact, as shown in [8, Thm. 4.1], the larger the gap between σ_k and σ_{k+1}, the smaller the subspace angle between $\mathcal{R}(\Pi W)$ and $\mathcal{N}_k(A)$, i.e., the better $\mathcal{R}(\Pi W)$ approximates the numerical null-space. If a more accurate basis for $\mathcal{N}_k(A)$ is required, the columns of W can always be improved by a few inverse iterations, as shown in [8].

3. Rank deficient least squares problems.

In this section we consider algorithms for solving the linear least squares problem

$$\min \|A\mathbf{x} - \mathbf{b}\|, \qquad A \in \Re^{m\times n}, \tag{7}$$

where the matrix A is very ill conditioned. The usual least squares solution, formally given by $\mathbf{x} = \Pi R^{-1}Q^T\mathbf{b}$, is then of no use because it is extremely sensitive to perturbations of $\mathbf{b}$ and it is usually dominated by highly oscillating contributions from the errors in the right-hand side $\mathbf{b}$. A standard approach to computing a least squares solution that is less sensitive to perturbations is to transform (7) into a nearby problem that is better conditioned. In practice, this process usually corresponds to damping or filtering the contributions to the least squares solution corresponding to the smallest singular values of A [13], [14]. One such approach is the *truncated* SVD (TSVD) method, in which one completely filters out all the small singular values below the ϵ-rank k. The TSVD solution $\mathbf{x}_{TSVD}$ is thus defined as

$$\mathbf{x}_{TSVD} \equiv \sum_{i=1}^{k} \frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}\mathbf{v}_i, \qquad k = k(A,\epsilon). \tag{8}$$

The TSVD solution can always be computed from the complete SVD of A, but this is computationally expensive because a great deal of the information provided by the SVD is not used. In fact, only the ϵ-rank and information about the numerical

TABLE 1
Dominating terms of the computational effort.

	excl. RRQR factorization	incl. RRQR factorization
$\mathbf{x}_{TSVD}$	$(2q(n-k)+1)n^2$	$(2m-\frac{2}{3}n+(\frac{7}{2}+2q)(n-k)+1)n^2$
$\mathbf{x}_{TQR}$	$(2(n-k)+1)n^2$	$(2m-\frac{2}{3}n+\frac{11}{2}(n-k)+1)n^2$
$\mathbf{x}_B$	n^2	$(2m-\frac{2}{3}n+\frac{7}{2}(n-k)+1)n^2$

null-space is required. Therefore, the TSVD solution can be computed efficiently by means of the RRQR factorization of A, as shown in [8].

Instead of using the RRQR to compute $\mathbf{x}_{TSVD}$, one may define a least squares solution in terms of the RRQR factorization itself. Here, we shall analyze and compare this approach to the TSVD method. Let the RRQR factorization of A be given by (4). The standard approach [12, §5.5] is to neglect the submatrix R_{22} (which is guaranteed to have a norm of the order σ_{k+1} due to Theorem 2.1) and to then solve this modified problem. In analogy with the TSVD solution, it is natural to define the *truncated* QR (TQR) solution $\mathbf{x}_{TQR}$ as the minimum two-norm least squares solution to the modified problem. To compute $\mathbf{x}_{TQR}$, which involves the pseudoinverse of

$$\begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix},$$

it is convenient to use a right orthogonal transformation P to annihilate R_{12} by means of R_{11}, i.e., $(R_{11}, R_{12})P = (\hat{R}_{11}, 0)$. Then the TQR solution is given by

$$\mathbf{x}_{TQR} \equiv \Pi P \begin{pmatrix} \hat{R}_{11}^{-1} & 0 \\ 0 & 0 \end{pmatrix} Q^T \mathbf{b}. \tag{9}$$

Alternatively, one can compute the *basic solution* $\mathbf{x}_B$, defined as the solution to another modified least squares problem where both submatrices R_{12} and R_{22} are neglected. The basic solution is given by

$$\mathbf{x}_B \equiv \Pi \begin{pmatrix} R_{11}^{-1} & 0 \\ 0 & 0 \end{pmatrix} Q^T \mathbf{b}. \tag{10}$$

We list in column one of Table 1 the dominating terms of the computational effort required to compute the three solutions $\mathbf{x}_{TSVD}$, $\mathbf{x}_{TQR}$, and $\mathbf{x}_B$, assuming that the RRQR factorization of A has been computed and that $Q^T\mathbf{b}$ is computed simultaneously with the factorization. The computational effort is measured in flops (a flop is either one addition or one multiplication). The quantity q is the number of inverse iterations used to compute accurate singular subspaces, and usually q is less than 4. The computational effort to compute the RRQR factorization itself depends on the column permutations needed during the computations, but it never exceeds $(2m - \frac{2}{3}n + \frac{7}{2}(n-k))n^2$ flops (provided that the Linpack condition estimator is used in each step). This leads to the total amount of computational effort given in column two. We note that more recent condition estimators, such as Incremental Condition Estimation and related algorithms, may be much more efficient in this context; see [1] for details.

In comparison, we can compute the complexity required by a similar technique for computing $\mathbf{x}_{TSVD}$ based on the PSVD algorithm [25] with accumulation of $Q^T\mathbf{b}$

(PSVD is not suited to computation of $\mathbf{x}_{TQR}$ or $\mathbf{x}_B$). The R-bidiagonalization of $A = U_B B V_B^T$ requires $2(m+n)n^2$ flops. Computation of the $n-k$ smallest singular values and the corresponding left and right singular vectors, using on average two "chases" per singular value, requires $24(n-k)n^2$ flops. Backsubstitution with B and orthogonalization with respect to the left and right null spaces requires $O(n)$ flops. Coordinate transformation with V_B requires $2n^2$ flops. Thus the total is $(2m+2n+24(n-k)+2)n^2$ flops. Therefore, the RRQR algorithm is always less computationally demanding than the PSVD-based algorithm.

For our numerical comparison, it is convenient to define the residual vectors corresponding to the three solutions:

$$\mathbf{r}_i = A\mathbf{x}_i - \mathbf{b}, \qquad i = TSVD, TQR, B. \tag{11}$$

Then we have the following results.

THEOREM 3.1. *The* TSVD *and* TQR *solutions are related by*

$$\|\mathbf{x}_{TSVD} - \mathbf{x}_{TQR}\| \le \|R_{22}\|\,\|R_{11}^{-1}\|\left(2\,\|\mathbf{x}_{TSVD}\| + \frac{\|\mathbf{r}_{TSVD}\|}{\sigma_k}\right) \tag{12}$$

and the TQR *and basic solutions are related by*

$$\|\mathbf{x}_{TQR} - \mathbf{x}_B\| \le \frac{1+\sqrt{5}}{2}\,\|R_{11}^{-1}\|^2\,\|R_{12}\|\,\|\mathbf{b}\|. \tag{13}$$

The three residual vectors satisfy

$$\|\mathbf{r}_{TSVD} - \mathbf{r}_{TQR}\| \le \|R_{22}\|\left(\|\mathbf{x}_{TSVD}\| + \frac{\|\mathbf{r}_{TSVD}\|}{\sigma_k}\right) \tag{14}$$

and

$$\|\mathbf{r}_{TQR} - \mathbf{r}_B\| \le \|R_{22}\|\,\|R_{11}^{-1}\|\,\|\mathbf{b}\|. \tag{15}$$

Proof. The TQR solution $\mathbf{x}_{TQR}$ is identical to a truncated SVD solution to the problem

$$\min\left\|Q\begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix}\Pi^T\mathbf{x} - \mathbf{b}\right\| = \min\left\|\left(A - Q\begin{pmatrix} 0 & 0 \\ 0 & R_{22} \end{pmatrix}\Pi^T\right)\mathbf{x} - \mathbf{b}\right\|.$$

Thus, we can consider $\mathbf{x}_{TQR}$ a perturbation of $\mathbf{x}_{TSVD}$, with the perturbed matrix given by

$$\tilde{A} = Q\begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix}\Pi^T$$

(which has rank k) and with the perturbation matrix given by

$$E = Q\begin{pmatrix} 0 & 0 \\ 0 & R_{22} \end{pmatrix}\Pi^T.$$

Note in particular that $\|E\| = \|R_{22}\|$, that $\|\tilde{A}^+\| = \|\hat{R}_{11}^{-1}\| \le \|R_{11}^{-1}\|$, and that the $(k+1)$th singular value of $\tilde{A}$ is zero. In [13, §3] Hansen derived general perturbation bounds for TSVD problems. For this special case, these bounds become somewhat simpler:

$$\|\mathbf{x}_{TSVD} - \mathbf{x}_{TQR}\| \le \|\tilde{A}^{+}\| \, (\|E\| \, \|\mathbf{x}_{TSVD}\| + \sin\theta_k \, \|\mathbf{r}_{TSVD}\|) + \sin\theta_k \, \|\mathbf{x}_{TSVD}\|,$$
$$\|\mathbf{r}_{TSVD} - \mathbf{r}_{TQR}\| \le \|E\| \, \|\mathbf{x}_{TSVD}\| + \sin\theta_k \, \|\mathbf{r}_{TSVD}\|,$$

where $\sin\theta_k \le \|E\|/\sigma_k$. Inserting this bound into the above expressions, and using the fact that $\sigma_k^{-1} \le \delta_k^{-1} = \|R_{11}^{-1}\|$, we obtain (12) and (14). The difference between $\mathbf{x}_{TQR}$ and $\mathbf{x}_B$ satisfies

$$\|\mathbf{x}_{TQR} - \mathbf{x}_B\| \le \left\| \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix}^{+} - \begin{pmatrix} R_{11} & 0 \\ 0 & 0 \end{pmatrix}^{+} \right\| \, \|\mathbf{b}\|.$$

Here, we can consider

$$\begin{pmatrix} 0 & R_{12} \\ 0 & 0 \end{pmatrix}$$

a perturbation of

$$\begin{pmatrix} R_{11} & 0 \\ 0 & 0 \end{pmatrix},$$

and a standard result for perturbed pseudoinverses [3, Thm. 5.3], $\|(A+E)^{+} - A^{+}\| \le \frac{1+\sqrt{5}}{2} \, \|A^{+}\| \, \|(A+E)^{+}\| \, \|E\|$, then yields (13). Finally, to prove (15), we have

$$\mathbf{r}_B - \mathbf{r}_{TQR} = A\,(\mathbf{x}_{TQR} - \mathbf{x}_B) = Q \begin{pmatrix} 0 & 0 \\ R_{22}P_{21}\hat{R}_{11}^{-1} & 0 \end{pmatrix} Q^T\mathbf{b},$$

where P_{21} is the bottom left $(n-k) \times k$ submatrix of the orthogonal matrix P. Taking norms and using $\|\hat{R}_{11}^{-1}\| \le \|R_{11}^{-1}\|$, we obtain (15). □

Due to Theorem 2.1 we are guaranteed that $\|R_{22}\| \le \sigma_{k+1}\sqrt{n-k}\,\|W_2^{-1}\|$ when R is computed via a RRQR factorization. Hence, as long as σ_{k+1} is small, Theorem 3.1 guarantees that TSVD and TQR will produce approximately the same solutions, as well as residuals. These two methods are therefore in many circumstances equally well suited for solving rank deficient least squares problems with well-determined ϵ-rank (unless, of course, the exact TSVD solution is required), and $\mathbf{x}_{TQR}$ is cheaper to compute than $\mathbf{x}_{TSVD}$.

The RRQR also leads to a basic solution $\mathbf{x}_B$ with approximately the same residual vector as both $\mathbf{r}_{TQR}$ and $\mathbf{r}_{TSVD}$, because R_{11} is guaranteed to be well conditioned. The basic solution itself, on the other hand, may be very different from both $\mathbf{x}_{TSVD}$ and $\mathbf{x}_{TQR}$. This is so because the basic solution $\mathbf{x}_B$ has a (possibly large) component in the numerical null-space of A.

To illustrate the different properties of all the above-mentioned solutions, we have carried out a series of experiments using Pro-Matlab [20]. The dimensions were $m = n = 100$, the ratio σ_1/σ_k (where k is the ϵ-rank) was 10^3 for all the matrices, and the right-hand side $\mathbf{b}$ was generated such that the TSVD residual vector satisfies $\|\mathbf{r}_{TSVD}\| = 10^{-3}\|A\,\mathbf{x}_{TSVD}\|$. Typical numerical results are shown in Table 2 for three different values of the ϵ-rank k and three different ratios σ_k/σ_{k+1}. These results clearly illustrate how the similarity of $\mathbf{x}_{TSVD}$ and $\mathbf{x}_{TQR}$, as well as the similarity of all the residual vectors, depends on the size of the gap between σ_k and σ_{k+1}, but not on the ϵ-rank k. Table 2 also illustrates that the solutions $\mathbf{x}_{TQR}$ and $\mathbf{x}_B$ generally are very different.

TABLE 2
Typical numerical results for rank deficient least squares problems with $m = n = 100$, $\sigma_1/\sigma_k = 10^3$.

Legend for each entry below	
$\|\mathbf{x}_{TSVD} - \mathbf{x}_{TQR}\|$	$\|\mathbf{r}_{TSVD} - \mathbf{r}_{TQR}\|/\|\mathbf{b}\|$
$\|\mathbf{x}_{TQR} - \mathbf{x}_B\|$	$\|\mathbf{r}_{TQR} - \mathbf{r}_B\|/\|\mathbf{b}\|$

$\frac{\sigma_k}{\sigma_{k+1}}$	$k = 50$		$k = 75$		$k = 90$	
10^6	$4.51 \cdot 10^{-10}$ 3.53	$1.55 \cdot 10^{-10}$ $1.22 \cdot 10^{-7}$	$6.76 \cdot 10^{-10}$ 1.70	$2.19 \cdot 10^{-10}$ $9.59 \cdot 10^{-8}$	$2.14 \cdot 10^{-10}$ 1.51	$8.04 \cdot 10^{-11}$ $5.05 \cdot 10^{-8}$
10^3	$4.34 \cdot 10^{-7}$ 3.53	$1.34 \cdot 10^{-7}$ $1.04 \cdot 10^{-4}$	$7.31 \cdot 10^{-7}$ 1.70	$2.26 \cdot 10^{-7}$ $9.35 \cdot 10^{-5}$	$1.93 \cdot 10^{-7}$ 1.51	$7.20 \cdot 10^{-8}$ $3.11 \cdot 10^{-5}$
10	$1.90 \cdot 10^{-3}$ 3.52	$3.80 \cdot 10^{-5}$ $9.52 \cdot 10^{-3}$	$2.00 \cdot 10^{-3}$ 1.70	$5.73 \cdot 10^{-5}$ $9.10 \cdot 10^{-3}$	$2.29 \cdot 10^{-5}$ 1.51	$7.03 \cdot 10^{-6}$ $2.62 \cdot 10^{-3}$

4. Subset selection problems. *Subset selection* is the problem of determining the most linearly independent columns of the matrix A. To be precise, if k is the ϵ-rank of A, then the aim is to find a column permutation Π such that the submatrix consisting of the first k columns of $A\,\Pi$ are as well conditioned as possible. The RRQR factorization of A obviously produces such a permutation Π. The basic solution $\mathbf{x}_B$ discussed in the previous section is in fact the least squares solution derived from this strategy by forcing to zero those elements of $\mathbf{x}_B$ that correspond to the linearly dependent columns of A. Such a solution may in some applications be preferred to the TSVD and TQR solutions. Subset selection is also interesting in its own right.

It is therefore interesting to compare the RRQR-based subset selection algorithm with the standard SVD-based algorithm proposed by Golub, Klema, and Stewart [11], [12, §12.2]. Their algorithm constructs a permutation matrix Π_{SVD} such that the bottom right $(n-k) \times (n-k)$ submatrix $\tilde{V}_{22}$ of $\Pi_{SVD}^T V$ is well conditioned, and then the first $n-k$ columns of $A\,\Pi_{SVD}$ are guaranteed to form a well-conditioned matrix. In other words, these columns form a linearly independent set of the columns of A.

The RRQR factorization also produces a permutation Π such that the first $n-k$ columns of $A\Pi$ are linearly independent, but this Π is constructed on the basis of information in the matrix W. The difference between these two methods therefore basically lies in the way that Π is computed. In general, we cannot guarantee that the two algorithms give identical permutations, and this makes a comparison of the two solutions difficult. On the other hand, it is more appropriate for subset selection problems to compare the subspaces spanned by the first $n-k$ columns of $A\,\Pi_{SVD}$ and $A\,\Pi$.

THEOREM 4.1. *Let* $\mathcal{R}(U_k)$ *denote the subspace* $\operatorname{span}\{\mathbf{u}_1, \cdots, \mathbf{u}_k\}$, *and let* B_{SVD} *and* B_{RRQR} *denote the submatrices consisting of the first* $n-k$ *columns of* $A\,\Pi_{SVD}$ *and* $A\,\Pi$, *respectively. Then*

$$\sin\theta(\mathcal{R}(U_k), \mathcal{R}(B_{SVD})) \le \sigma_{k+1}\|\tilde{V}_{22}^{-1}\|\,\sigma_k^{-1}, \tag{16}$$

$$\sin\theta(\mathcal{R}(U_k), \mathcal{R}(B_{RRQR})) \le \sigma_{k+1}\|R_{11}^{-1}\|. \tag{17}$$

Proof. Our proof follows that given by Golub and Van Loan for [12, Thm. 12.2.2]. In their proof, they derive the upper bound $\sin\theta(\mathcal{R}(U_k), \mathcal{R}(B_{SVD})) \le \sigma_{k+1}/\sigma_k(B_{SVD})$, where $\sigma_k(B_{SVD})$ denotes the smallest singular value of B_{SVD}. They also show in [12,

TABLE 3
Typical numerical results for the RRQR *subset selection algorithm. Each entry in the table shows the subspace angles* $\sin\theta(\mathcal{R}(U_k),\mathcal{R}(B_{RRQR}))$ *and* $\sin\theta(\mathcal{R}(B_{SVD}),\mathcal{R}(B_{RRQR}))$. *The dimensions are* $m = n = 100$.

$\frac{\sigma_k}{\sigma_{k+1}}$	$k = 50$		$k = 75$		$k = 90$	
10^6	$1.03\cdot10^{-6}$	$1.26\cdot10^{-6}$	$1.58\cdot10^{-6}$	$1.74\cdot10^{-6}$	$1.30\cdot10^{-6}$	$1.51\cdot10^{-6}$
10^3	$9.70\cdot10^{-4}$	$1.18\cdot10^{-3}$	$1.54\cdot10^{-3}$	$1.66\cdot10^{-3}$	$1.30\cdot10^{-3}$	$1.51\cdot10^{-3}$
10	$9.38\cdot10^{-2}$	$1.14\cdot10^{-1}$	$1.50\cdot10^{-3}$	$1.62\cdot10^{-1}$	$1.30\cdot10^{-7}$	$1.50\cdot10^{-1}$

Thm. 12.2.1] that $\sigma_k(B_{SVD}) \geq \sigma_k\,\|\tilde{V}_{22}^{-1}\|$, which yields (16). Similarly, since

$$B_{RRQR} = Q\begin{pmatrix} R_{11} \\ 0 \end{pmatrix},$$

we obtain that

$$\sin\theta(\mathcal{R}(U_k),\mathcal{R}(B_{RRQR})) \leq \frac{\sigma_{k+1}}{\sigma_k(B_{RRQR})} = \frac{\sigma_{k+1}}{\sigma_k(R_{11})},$$

which is (17). □

The submatrix R_{11} in the RRQR factorization of A is guaranteed to be well conditioned and $\|R_{11}^{-1}\|$ is of the order σ_k^{-1}. Theorem 4.1 therefore ensures that the sine of both subspace angles is of the same order as σ_{k+1}/σ_k. Moreover, if this ratio is small, then Theorem 4.1 ensures that both $\mathcal{R}(B_{SVD})$ and $\mathcal{R}(B_{RRQR})$ will be close to the subspace $\mathcal{R}(U_k)$, and the subspace angle between $\mathcal{R}(B_{SVD})$ and $\mathcal{R}(B_{RRQR})$ is therefore also bound to be small. Hence, if σ_{k+1}/σ_k is small, then SVD and RRQR yield approximately the same subspaces. We illustrate this in Table 3, where we show typical values of $\sin\theta(\mathcal{R}(U_k),\mathcal{R}(B_{RRQR}))$ and $\sin\theta(\mathcal{R}(B_{SVD}),\mathcal{R}(B_{RRQR}))$. We see that both subspace angles are indeed of the same order as the ratio σ_{k+1}/σ_k. The conclusion is that although the SVD and RRQR subset selection algorithms do not necessarily produce the same column permutations and thus the same subset of columns of A, the subspaces spanned by these two sets of columns are still almost identical whenever σ_{k+1}/σ_k is small.

5. Matrix approximation. It is well known that the best rank-k approximation A_k to the matrix A, in any unitarily invariant norm, is the matrix obtained by truncating the SVD expansion in (2) after the first k terms. That is, the matrix A_k given by

$$A_k = \sum_{i=1}^{k} \mathbf{u}_i\,\sigma_i\,\mathbf{v}_i^T, \qquad k \leq n \tag{18}$$

solves the problem $\min_{\mathrm{rank}(X)=k}\|A - X\|$ (this is the Eckart–Young–Mirsky Theorem [12, Thm. 2.5.2]). In particular, $\|A - A_k\| = \sigma_{k+1}$ and $\|A - A_k\|_F = (\sigma_{k+1}^2 + \cdots + \sigma_n^2)^{1/2}$. As we shall demonstrate in the following theorem, neglection of the submatrix R_{22} in a RRQR factorization also yields a good rank-k approximation, provided that W_2 is well conditioned.

THEOREM 5.1. *Let*

$$B_k = Q\begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix}\Pi^T$$

TABLE 4
Typical numerical results for RRQR *matrix approximations. Each entry shows* $\|A - B_k\|$ *and* $\|A_k - B_k\|$ *for matrices with* $m = n = 100$, $\sigma_1 = 1$, *and* $\sigma_{k+1} = \|A - A_k\| = 10^{-7}$.

$\frac{\sigma_k}{\sigma_{k+1}}$	$k = 50$		$k = 75$		$k = 90$	
10^6	$3.05 \cdot 10^{-7}$	$2.90 \cdot 10^{-7}$	$2.55 \cdot 10^{-7}$	$2.48 \cdot 10^{-7}$	$2.63 \cdot 10^{-7}$	$2.51 \cdot 10^{-7}$
10^3	$2.39 \cdot 10^{-7}$	$2.17 \cdot 10^{-7}$	$2.51 \cdot 10^{-7}$	$2.31 \cdot 10^{-7}$	$5.68 \cdot 10^{-7}$	$5.60 \cdot 10^{-7}$
10	$4.20 \cdot 10^{-7}$	$4.08 \cdot 10^{-7}$	$5.02 \cdot 10^{-7}$	$4.92 \cdot 10^{-7}$	$6.98 \cdot 10^{-7}$	$6.91 \cdot 10^{-7}$

denote the matrix obtained from the RRQR *factorization* (4) *by neglecting the submatrix* R_{22}. *Then*

$$\|A - B_k\| \le \sqrt{n-k}\,\|W_2^{-1}\|\,\sigma_{k+1}, \tag{19}$$

$$\|A - B_k\|_F \le \sqrt{n-k}\,\|W_2^{-1}\|\,(\sigma_{k+1}^2 + \cdots + \sigma_n^2)^{1/2}, \tag{20}$$

where A_k *denotes the truncated* SVD *matrix* (18).

Proof. Obviously, $\|A - B_k\| = \|R_{22}\|$ and $\|A - B_k\|_F = \|R_{22}\|_F$, and (19) then follows from Theorem 2.1. To get an upper bound for $\|R_{22}\|_F$, we use the fact that $\|R_{22}\|_F \le \|R_{22}\,W_2\|_F\,\|W_2^{-1}\|_F \le \|A\Pi W\|_F\,\|W_2^{-1}\|_F$. Since each column vector $\mathbf{w}_i$ of W satisfies

$$\|A\Pi\mathbf{w}_i\| = \left\|QR\begin{pmatrix}\mathbf{w}^{(i)}\\ \mathbf{0}\end{pmatrix}\right\| = \|R^{(i)}\mathbf{w}^{(i)}\| = \delta_i \le \sigma_{k+i}$$

(cf. (5)), we obtain $\|A\Pi W\|_F^2 \le \sigma_{k+1}^2 + \cdots + \sigma_n^2$. This yields (20). □

Theorem 5.1 states that matrix approximations derived from RRQR factorizations are almost as good as those derived from truncated SVD approximations. Moreover, it is trivial from Theorem 5.1 that the difference between A_k and B_k satisfies $\|A_k - B_k\| \le (1 + \sqrt{n-k}\,\|W_2^{-1}\|)\,\sigma_{k+1}$. The interesting fact about the RRQR matrix approximations is that the bounds in (19) and (20) do not depend on the gap between σ_k and σ_{k+1}. The algorithm can therefore be applied to any matrix independently of its singular value spectrum and with potential application to digital image compression. We illustrate this in Table 4, which shows typical values of $\|A - B_k\|$ and $\|A_k - B_k\|$ for random matrices with $m = n = 100$, $\sigma_1 = 1$, $\sigma_{k+1} = \|A - A_k\| = 10^{-7}$, and different values of the ϵ-rank k and the ratio σ_k/σ_{k+1}. The table confirms that $\|A - B_k\|$ and $\|A_k - B_k\|$ are both of the order σ_{k+1}, as proved in the above theorem, and are independent of σ_k/σ_{k+1}.

6. Total least squares problems.

Another aspect of matrix approximation arises in connection with total least squares (TLS) problems [12, §12.3]. The key problem here is to find three matrices E, R, and X such that $\|(E, R)\|$ is small and such that $(A + E)X = B + R$, with $A \in \Re^{m\times n}$ and $B \in \Re^{m\times p}$ (the ordinary least squares problem corresponds to setting $E = 0$). Total least squares problems typically arise in applications where both the coefficient matrix A and the right-hand side B are contaminated with errors, in which case one can think of E and R as being residual matrices.

The classical approach to TLS is based on the SVD and is described in [12, §12.3]. Let

$$V = \begin{pmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{pmatrix}$$

be the matrix of right singular vectors in the SVD of the compound matrix (A, B), partitioned such that $V_{11} \in \Re^{n \times r}$, where r is the ϵ-rank of A. If A has full rank $(r = n)$, then the solution that minimizes the Frobenius norm of the compound residual matrix $\|(E, R)\|_F$ is given by $X_{SVD} = -V_{12}V_{22}^{-1}$ [12, Thm. 12.3.1]. When A is rank deficient $(r < n)$, the solution of the minimum Frobenius norm that minimizes $\|(E, R)\|_F$ is given by [27]

$$X_{SVD} = -V_{12}V_{22}^{+}. \tag{21}$$

Here, V_{22}^{+} is the pseudoinverse of the $p \times (n - r + p)$ submatrix V_{22}, and V_{22}^{+} is identical to V_{22}^{-1} when $r = n$. As long as $p \ll n$, V_{22}^{+} can easily be computed stably, e.g., by a QR factorization of V_{22}. Note that there is no guarantee that V_{22} is well conditioned. Also note that the numerical rank of the compound matrix (A, B) is not used in total least squares, only the ϵ-rank of A is required. We return to this aspect at the end of this section.

From (21) it is obvious that the complete SVD of (A, B) is not needed for computing X_{SVD}. Instead, only the right singular vectors corresponding to the smallest $n - r + p$ singular values are required. It is possible to modify the classical SVD algorithm by taking this into account. This is done in the PSVD algorithm [25], [26], developed for TLS problems, which requires $(2m + 2n + 14p + 2)(n + p)^2 + O(n)$ flops to compute X_{SVD} if the bidiagonalization part is preceded by a standard QR factorization, as described in [5].

Here we derive algorithms for TLS based on the RRQR factorization of (A, B). The key idea is to use the RRQR algorithm to compute approximate null-vectors corresponding to the $n-r+p$ smallest singular values of (A, B) (assume for the moment that r is known). We can then use the same approach as that used in [8] to compute TSVD solutions, namely, to use inverse subspace iterations to refine the numerical null-vectors in W. This approach yields the right singular vectors required to compute X_{SVD} by (21), and the accuracy of the solution depends primarily on the number of subspace iterations. Except for a single backsubstitution, which is not required to compute X_{SVD}, the dominating computational effort remains the same as that for computing $\mathbf{x}_{SVD}$; cf. §3. Hence, this approach requires $(2m - \frac{2}{3}n + (2q + 3))(n + p)^2$ flops (where q is the number of inverse iterations required to refine the null vectors), and it is actually less demanding than the PSVD approach, mainly because a reduction to bidiagonal form is avoided.

We shall now analyze an even simpler approach to TLS, based directly on the matrix W without performing any inverse iterations. Whenever the singular values σ_r and σ_{r+1} are well separated, it is shown in [8, Thm. 4.1] that the range of ΠW is a good approximation to $\mathcal{N}_r(A, B)$, in the sense that the subspace angle θ between $\mathcal{R}(\Pi W)$ and $\mathcal{N}_r(A, B)$ is small. Thus, it is natural to obtain an approximation to the last $n - r + p$ null vectors of (A, B) simply by orthonormalization of the columns of ΠW, e.g., by means of the modified Gram–Schmidt process, to obtain

$$\Pi W = \begin{pmatrix} \bar{V}_{12} \\ \bar{V}_{22} \end{pmatrix} \bar{R}, \qquad \bar{V}_{12}^T \bar{V}_{12} + \bar{V}_{22}^T \bar{V}_{22} = I_{n-r+p}, \tag{22}$$

and then define the alternative TLS solution by $X_{RRQR} = -\bar{V}_{12}\bar{V}_{22}^{-1}$ or, in the general case, by

$$X_{RRQR} = -\bar{V}_{12}\bar{V}_{22}^{+}. \tag{23}$$

As is the case in the SVD approach, we cannot guarantee that $\bar{V}_{22}$ is well conditioned. The question is then whether a small subspace angle θ ensures that X_{RRQR} is close to X_{SVD}. In Theorem 6.3 below we give a positive answer to this question for the full rank case $(r = n)$, but first we need the following two lemmas.

LEMMA 6.1. *If*

$$V = \begin{pmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{pmatrix}$$

is orthogonal, then the norm of the Schur complement of V_{22} *satisfies*

$$\|V_{11} - V_{12}V_{22}^{-1}V_{21}\| = \|V_{22}^{-1}\|. \tag{24}$$

Proof. For simplicity, we assume $V_{11} \in \Re^{q\times q}$, $V_{22} \in \Re^{p\times p}$ with $q > p$. Then the CS decomposition [12, Thm. 2.6.1] of V is given by

$$\begin{pmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{pmatrix} = \begin{pmatrix} \hat{U}_1 & 0 \\ 0 & \hat{U}_2 \end{pmatrix} \begin{pmatrix} I_{q-p} & 0 & 0 \\ 0 & C & S \\ 0 & -S & C \end{pmatrix} \begin{pmatrix} \hat{V}_1^T & 0 \\ 0 & \hat{V}_2^T \end{pmatrix},$$

where $C^2 + S^2 = I_p$. It is straightforward to show that $V_{22}^{-1} = \hat{V}_2 C^{-1}\hat{U}_2^T \Rightarrow \|V_{22}^{-1}\| = \|C^{-1}\|$ and, by inserting the CS decomposition, that

$$V_{11} - V_{12}V_{22}^{-1}V_{21} = \hat{U}_1 \begin{pmatrix} I_{q-p} & 0 \\ 0 & C^{-1} \end{pmatrix} \hat{V}_1^T,$$

from which (24) immediately follows. □

LEMMA 6.2. *If* Q, V, *and* $\bar{V}$ *are orthogonal matrices such that*

$$\bar{V} = (\bar{V}_1, \bar{V}_2) = VQ = (V_1, V_2) \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{pmatrix}, \tag{25}$$

then the subspace angle θ *between the subspaces* $\mathcal{R}(V_2)$ *and* $\mathcal{R}(\bar{V}_2)$ *satisfies*

$$\sin\theta = \|Q_{12}\|, \qquad \cos\theta = \|Q_{22}^{-1}\|^{-1}. \tag{26}$$

Proof. We have $\sin\theta = \|V_1^T\bar{V}_2\| = \|V_1^T(V_1Q_{12} + V_2Q_{22})\| = \|Q_{12}\|$. The relation for $\cos\theta$ follows from the CS decomposition of Q. □

THEOREM 6.3. *Let* X_{SVD} *and* X_{RRQR} *be given by* (21) *and* (23), *respectively, and assume that* A *has full rank. If* θ *denotes the subspace angle between* $\mathcal{N}_r(A,B)$ *and* $\mathcal{R}(\Pi W)$, *and if* $\tan\theta < \|V_{22}^{-1}\|^{-1}$, *then*

$$\|X_{SVD} - X_{RRQR}\| \le \tan\theta\, \|V_{22}^{-1}\|^2 \Big(1 + O(\tan\theta)\Big). \tag{27}$$

Proof. Using the notation from (25) with

$$V_2 = \begin{pmatrix} V_{12} \\ V_{22} \end{pmatrix} \quad \text{and} \quad \bar{V}_2 = \begin{pmatrix} \bar{V}_{12} \\ \bar{V}_{22} \end{pmatrix},$$

we obtain $\bar{V}_{12} = V_{11}Q_{12} + V_{12}Q_{22}$ and $\bar{V}_{22} = V_{21}Q_{12} + V_{22}Q_{22} = (V_{21}Q_{12}Q_{22}^{-1}V_{22}^{-1} + I_p)V_{22}Q_{22}$. If we define $\Delta = V_{21}Q_{12}Q_{22}^{-1}V_{22}^{-1}$, then $\bar{V}_{22} = (I_p + \Delta)V_{22}Q_{22}$. From

TABLE 5

Mean and maximum values for an experiment with 100 *matrices with dimensions* $m = 150$, $n = 100$, *and* $p = 5$. *The* ϵ*-rank of* A *is* $r = 90$.

	Mean	Maximum
$\tan\theta$	$1.92 \cdot 10^{-7}$	$1.92 \cdot 10^{-6}$
$\lVert\bar{V}_{22}^{+}\rVert$	1.20	1.30
$\tan\theta\,\lVert V_{22}^{+}\rVert^2/\lVert X_{SVD}\rVert$	$4.07 \cdot 10^{-7}$	$3.95 \cdot 10^{-6}$
$\lVert X_{SVD} - X_{RRQR}\rVert/\lVert X_{SVD}\rVert$	$5.06 \cdot 10^{-8}$	$5.27 \cdot 10^{-7}$

Lemma 6.2 we have $\|\Delta\| \leq \tan\theta\,\|V_{22}^{-1}\|$, and since we have assumed that $\tan\theta < \|V_{22}^{-1}\|^{-1}$ such that $\|\Delta\| < 1$, we have

$$\bar{V}_{22}^{-1} = Q_{22}^{-1}V_{22}^{-1}(I_p + \Delta)^{-1} = Q_{22}^{-1}V_{22}^{-1}(I_p - \Delta + O(\Delta^2)).$$

Thus, we obtain that

$$X_{RRQR} = -\bar{V}_{12}\bar{V}_{22}^{-1} = -V_{11}Q_{12}Q_{22}^{-1}V_{22}^{-1} - V_{12}V_{22}^{-1} + (V_{11}Q_{12}Q_{22}^{-1} + V_{12})V_{22}^{-1}\Delta + O(\Delta^2).$$

Inserting the expression for Δ and using the fact that $X_{SVD} = -V_{12}V_{22}^{-1}$, we then obtain

$$X_{RRQR} - X_{SVD} = -(V_{11} - V_{12}V_{22}^{-1}V_{21})\,Q_{12}Q_{22}^{-1}V_{22}^{-1} + V_{11}Q_{12}Q_{22}^{-1}V_{22}^{-1}\Delta + O(\Delta^2).$$

Taking norms on both sides of the above equation and using Lemmas 6.1 and 6.2, we then obtain (27). □

Theorem 6.3 states that if A has full rank, if the subspace angle θ is small, and if V_{22} is well conditioned, then the total least squares solution X_{RRQR} defined by (23) will be close to the ordinary total least squares solution X_{SVD} given by (21). Although we cannot ensure that the matrix V_{22} is well conditioned, it is our experience that it is very unlikely to be ill conditioned.

Unfortunately, we were not able to prove a similar result for the rank deficient case ($r < n$), the reason being the appearance of the pseudoinverse V_{22}^{+}, which severely complicates the relations. On the other hand, our experiments with Matlab strongly suggest that (27) holds in general: among 100 random matrices with dimensions $m = 150$, $n = 100$, and $p = 5$ and with A's ϵ-rank $r = 90$, the ratio between $\|X_{SVD} - X_{RRQR}\|$ and $\tan\theta\,\|V_{22}^{+}\|^2$ in (27) never exceeded 0.6. The results from this test are summarized in Table 5, where we list the mean and maximum values of $\tan\theta$, $\|\bar{V}_{22}^{+}\|$, $\tan\theta\,\|V_{22}^{+}\|^2/\|X_{SVD}\|$, and $\|X_{SVD} - X_{RRQR}\|/\|X_{SVD}\|$. This confirms the fact that $\bar{V}_{22}$ is in all likelihood a well-conditioned matrix and that the upper bound in (27) is not large. The conclusion is that X_{RRQR} is indeed a good TLS solution.

Besides being faster than both the traditional SVD algorithm and the PSVD algorithm, our approach to TLS based on RRQR has one more important advantage: if the ϵ-rank of A is unknown, then one can compute it during the RRQR factorization of (A, B) with very little overhead. First compute an RRQR factorization of A,

$$A\,\Pi^{(1)} = Q^{(1)}\begin{pmatrix} R^{(1)} \\ 0 \end{pmatrix} = Q^{(1)}\begin{pmatrix} R_{11}^{(1)} & R_{12}^{(1)} \\ 0 & R_{22}^{(1)} \\ 0 & 0 \end{pmatrix}, \tag{28}$$

revealing the ϵ-rank r of A. Then append

$$\begin{pmatrix} R^{(1)} \\ 0 \end{pmatrix}$$

with $(Q^{(1)})^T B$, to form

$$C = \begin{pmatrix} R_{11}^{(1)} & R_{12}^{(1)} & \hat{B}_1 \\ 0 & R_{22}^{(1)} & \hat{B}_2 \\ 0 & 0 & \hat{B}_3 \end{pmatrix}, \quad \begin{pmatrix} \hat{B}_1 \\ \hat{B}_2 \\ \hat{B}_3 \end{pmatrix} = (Q^{(1)})^T B$$

and compute the RRQR factorization of the lower right 2×2 block submatrix of C:

$$\begin{pmatrix} R_{22}^{(1)} & \hat{B}_2 \\ 0 & \hat{B}_3 \end{pmatrix} \Pi^{(2)} = Q^{(2)} \begin{pmatrix} R_{22}^{(2)} & R_{23}^{(2)} \\ 0 & R_{33}^{(2)} \end{pmatrix}. \tag{29}$$

Finally, apply the second set of permutations to $\begin{pmatrix} R_{12}^{(1)} & \hat{B}_2 \end{pmatrix}$ to obtain

$$\begin{pmatrix} R_{12}^{(2)} & R_{13}^{(2)} \end{pmatrix} = \begin{pmatrix} R_{12}^{(1)} & \hat{B}_1 \end{pmatrix} \Pi^{(2)}. \tag{30}$$

Then the resulting triangular factor of (A, B) is given by

$$R = \begin{pmatrix} R_{11}^{(1)} & R_{12}^{(2)} & R_{13}^{(2)} \\ 0 & R_{22}^{(2)} & R_{23}^{(2)} \\ 0 & 0 & R_{33}^{(2)} \end{pmatrix}.$$

During the second factorization (29), one can take advantage of the fact that the first $n - r$ columns are in all likelihood linear combinations within the tolerance ϵ of the remaining columns. Since $R_{11}^{(1)}$ is guaranteed to be well conditioned, the QR factorization resulting from (28)–(30) is close to being an RRQR factorization of (A, B), and the desired RRQR factorization can then be achieved by a few "backward passes" through R as described in [1, §5].

REFERENCES

[1] C. H. Bischof and P. C. Hansen, *Structure-preserving and rank-revealing QR-factorizations*, Report MCS-P100-0989, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 1989; SIAM J. Sci. Statist. Comput., 12 (1991), pp. 1332–1350.

[2] C. H. Bischof and G. M. Shroff, *On updating signal subspaces*, IEEE Trans. Signal Process., 40 (1992), pp. 96–105.

[3] Å. Björck, *Least squares methods*, in Handbook of Numerical Analysis, Vol. I: Finite Difference Methods—Solution of Equations in R^n, P. G. Ciarlet and J. L. Lions, eds., Elsevier, Amsterdam, 1990.

[4] P. A. Businger and G. H. Golub, *Linear least squares solution by Householder transformation*, Numer. Math., 7 (1965), pp. 269–276.

[5] T. F. Chan, *An improved algorithm for computing the singular value decomposition*, ACM Trans. Math. Software, 8 (1982), pp. 72–83.

[6] ———, *Alternative to the* SVD: *Rank revealing* QR-*factorizations*, in Advanced Algorithms and Architectures for Signal Processing, J. M. Speiser, ed., SPIE Proceedings Vol. 696, 1986.

[7] ———, *Rank revealing* QR *factorizations*, Linear Algebra Appl., 88/89 (1987), pp. 67–82.

[8] T. F. Chan and P. C. Hansen, *Computing truncated* SVD *least squares solutions by rank revealing* QR *factorizations*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 519–530.

[9] P. Comon and G. H. Golub, *Tracking a few extreme singular values and vectors in signal processing*, Proc. IEEE, 78 (1990), pp. 1327–1343.

[10] L. Foster, *Rank and null space calculations using matrix decomposition without column interchanges*, Linear Algebra Appl., 74 (1986), pp. 47–71.

[11] G. H. Golub, V. Klema, and G. W. Stewart, *Rank degeneracy and least squares problems*, Report TR-456, Department of Computer Science, University of Maryland, College Park, MD, 1976.

[12] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Second Edition, The Johns Hopkins University Press, Baltimore, MD, 1989.

[13] P. C. HANSEN, *The truncated* SVD *as a method for regularization*, BIT, 27 (1987), pp. 534–553.

[14] ———, *Truncated* SVD *solutions to discrete ill-posed problems with ill-determined numerical rank*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 503–518.

[15] P. C. HANSEN, T. SEKII, AND H. SHIBAHASHI, *The modified truncated* SVD *method for regularization in general form*, SIAM J. Sci. Statist. Comput., 13 (1992), to appear.

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

[17] Y. P. HONG AND C.-T. PAN, *Rank-revealing* QR *factorizations and* SVD, Math. Comput., to appear.

[18] C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[19] T. A. MANTEUFFEL, *An interval analysis approach to rank determination in linear least squares problems*, SIAM J. Sci. Statist. Comput., 2 (1981), pp. 335–348.

[20] C. B. MOLER, J. LITTLE, AND S. BANGERT, *Pro-Matlab User's Guide*, The MathWorks, Sherborn, MA, 1987.

[21] E. NG, *A scheme for handling rank-deficiency in the solution of sparse linear least squares problems*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 1173–1183.

[22] L. REICHEL AND W. B. GRAGG, *Algorithm* 686: *Fortran subroutines for updating the* QR *decomposition*, ACM Trans. Math. Software, 16 (1990), pp. 369–377.

[23] G. W. STEWART, *Rank degeneracy*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 403–413.

[24] ———, *An updating algorithm for subspace tracking*, Report CS-TR 2494, Department of Computer Science, University of Maryland, College Park, MD, 1990; IEEE Trans. Signal Process., to appear.

[25] S. VAN HUFFEL, J. VANDEWALLE, AND A. HAEGEMANS, *An efficient and reliable algorithm for computing the singular subspace of a matrix, associated with its smallest singular values*, J. Comput. Appl. Math., 19 (1987), pp. 313–330.

[26] S. VAN HUFFEL AND J. VANDEWALLE, *The partial total least squares algorithm*, J. Comput. Appl. Math., 21 (1988), pp. 333–341.

[27] M. D. ZOLTOWSKI, *Generalized minimum norm and constrained total least squares with applications to array signal processing*, in Advanced Algorithms and Architectures for Signal Processing III, F. T. Luk, ed., SPIE Vol. 975, 1988, pp. 78–85.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 3, pp. 742–764, May 1992

STABLE PARALLEL ALGORITHMS FOR TWO-POINT BOUNDARY VALUE PROBLEMS*

STEPHEN J. WRIGHT†

Abstract. Some of the most widely used algorithms for two-point boundary value ordinary differential equations, namely, finite-difference and collocation methods and standard multiple shooting, proceed by setting up and solving a structured system of linear equations. It is well known that the linear system can be set up efficiently in parallel; we show here that a structured orthogonal factorization technique can be used to solve this system, and hence the overall problem, in an efficient, parallel, and stable way.

Key words. parallel algorithms, two-point boundary value problems, error analysis, stability

AMS(MOS) subject classifications. 65F05, 65G05, 65L10, 65L20, 65W05

1. Introduction.

Many numerical algorithms for solving the linear two-point boundary value problem (BVP)

$$y' = M(t)y + q(t), \qquad t \in [a,b], \qquad y \in R^n, \tag{1}$$

$$B_a y(a) + B_b y(b) = d \tag{2}$$

have been proposed and studied over the last 30 years. Many of these methods require a structured linear algebraic system (for example, a block-tridiagonal or staircase system) to be solved as a "core" operation, and considerable effort has been devoted to minimizing the amount of computer time and storage required during the factorization of the coefficient matrix of this system. Efficient factorization schemes, based on structured Gaussian elimination, have been implemented and are widely available (see §2 and Varah [19]; Diaz, Fairweather, and Keast [7]; Lentini and Pereyra [13]; and Keller [9]).

During the last 10 years, the question of stability of algorithms for (1), (2) has received a great deal of attention (see, for example, Mattheij [15]). It has been recognized that in a well-conditioned problem (that is, one whose solution is not too sensitive to perturbations in M, q, or the boundary conditions (2)), the fundamental solution space generally contains both exponentially increasing and exponentially decreasing modes. The stability of a numerical method for (1), (2) depends on its ability to at some point perform a "decoupling" of these modes. For the standard multiple shooting and finite-difference algorithms, this decoupling is performed implicitly, during the factorization of the structured linear system, through the use of a pivoting strategy that prevents element growth in the factors. Unfortunately, parallel and vectorizable algorithms that have been proposed for solving the linear system invariably place some restriction on the choice of pivots. This can lead to undesirable element growth in the factors, and such methods are in fact similar to compactification algorithms for (1), (2), which are known to be unstable. In this paper, we use instead a structured orthogonal factorization technique that is stable, has an identical serial complexity to the best-known algorithms, and can be efficiently implemented

* Received by the editors September 26, 1990; accepted for publication (in revised form) February 5, 1991. This research was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, United States Department of Energy, contract W-31-109-Eng-38. A grant of computer time at the North Carolina Supercomputing Center is gratefully acknowledged.

† Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439.

on a wide variety of parallel architectures. A variant of the algorithm vectorizes efficiently, in much the same way as cyclic reduction for block-tridiagonal linear systems. No separability of the boundary conditions is needed. Although we focus on matrices arising from multiple shooting and one-step differencing schemes, the technique can be applied equally well to the more general staircase matrix structures that arise in other numerical schemes, such as collocation.

We assume throughout that n is too small to allow efficient parallel or vector implementation of order-n matrix and vector operations. Instead, parallelism is sought by partitioning the domain $[a, b]$ of the independent variable.

The remainder of the paper is organized as follows: in §2 we review the multiple shooting and finite-difference algorithms, the structured linear systems that result from them, and the efficient Gaussian elimination techniques used to solve them on serial computers. In §3 we briefly review previous work on parallel and vectorizable algorithms. The new algorithm is presented in §4, together with a stability result and analysis of serial and parallel complexity. In §5 we describe parallel implementation of a standard technique for estimating the condition number of the coefficient matrix, which can be used for purposes of a posteriori error analysis. Finally, in §6 we describe the results of implementation of the scheme on shared-memory and vector architectures.

2. Serial algorithms. The "standard" multiple shooting technique for (1), (2) proceeds by defining a mesh

$$a = t_1 < t_2 < \cdots < t_{k+1} = b, \tag{3}$$

and finding a fundamental and particular solution on each interval of the mesh:

$$Y_i' = M(t)Y_i, \qquad Y_i(t) \in R^{n\times n}, \qquad t \in [t_i, t_{i+1}], \qquad Y_i(t_i) = I,$$

$$y_{pi}' = M(t)y_{pi} + q(t), \qquad y_{pi}(t) \in R^n, \qquad t \in [t_i, t_{i+1}], \qquad y_{pi}(t_i) = 0.$$

Then, we try to find $s_i \in R^n$, $i = 1, \cdots, k+1$, such that

$$y(t) = Y_i(t)s_i + y_{pi}(t), \qquad t \in [t_i, t_{i+1}], \quad i = 1, \cdots, k.$$

(Note in particular that $s_i = y(t_i)$, $i = 1, \cdots, k+1$.) By applying the boundary conditions (2), and continuity at the meshpoints, we obtain the following linear system in $s_1, \cdots, s_{k+1}$:

$$\begin{bmatrix} B_a & & & & B_b \\ Y_1(t_2) & -I & & & \\ & Y_2(t_3) & -I & & \\ & & \ddots & \ddots & \\ & & & Y_k(t_{k+1}) & -I \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_{k+1} \end{bmatrix} = \begin{bmatrix} d \\ -y_{p1}(t_2) \\ -y_{p2}(t_3) \\ \vdots \\ -y_{pk}(t_{k+1}) \end{bmatrix}. \tag{4}$$

In the (quite common) case in which the boundary conditions are separated, this system can be rearranged into a block-banded form. If p is the number of initial conditions and q is the number of final conditions ($p + q = n$), we can assume without loss of generality that the boundary data (2) can be partitioned as follows:

$$B_a = \begin{bmatrix} \bar{B}_a \\ 0 \end{bmatrix}, \quad B_b = \begin{bmatrix} 0 \\ \bar{B}_b \end{bmatrix}, \quad d = \begin{bmatrix} d_a \\ d_b \end{bmatrix}.$$

If the rows $p+1, \cdots, n$ of (4) are moved to the bottom, we obtain the form

$$\begin{bmatrix} \bar{B}_a & & & & & \\ Y_1(t_2) & -I & & & & \\ & Y_2(t_3) & -I & & & \\ & & \ddots & \ddots & & \\ & & & Y_k(t_{k+1}) & -I \\ & & & & \bar{B}_b \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_k \\ s_{k+1} \end{bmatrix} = \begin{bmatrix} d_a \\ -y_{p1}(t_2) \\ -y_{p2}(t_3) \\ \vdots \\ -y_{pk}(t_{k+1}) \\ d_b \end{bmatrix}. \tag{5}$$

One-step finite-difference methods proceed by again choosing a mesh of the form (3) and seeking $s_1, \cdots, s_{k+1}$ to approximate $y(t_1), \cdots, y(t_{k+1})$. On the interval $[t_i, t_{i+1}]$, (1) is approximated by the linear relationship

$$\frac{s_{i+1} - s_i}{h_i} = \bar{A}_i s_i + \bar{C}_i s_{i+1} + \bar{f}_i, \tag{6}$$

where $h_i = t_{i+1} - t_i$. Two schemes with a local truncation error of $O(h_i^2)$ are the "box scheme" ($\bar{A}_i = \bar{C}_i = \frac{1}{2}M(t_{i+1/2})$, $\bar{f}_i = q(t_{i+1/2})$) and the trapezoidal rule ($\bar{A}_i = \frac{1}{2}M(t_i)$, $\bar{C}_i = \frac{1}{2}M(t_{i+1})$, $\bar{f}_i = \frac{1}{2}(q(t_i) + q(t_{i+1}))$). By including (2), we obtain a linear system of the form

$$\begin{bmatrix} B_a & & & & B_b \\ A_1 & C_1 & & & \\ & A_2 & C_2 & & \\ & & \ddots & \ddots & \\ & & & A_k & C_k \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_{k+1} \end{bmatrix} = \begin{bmatrix} d \\ f_1 \\ f_2 \\ \vdots \\ f_k \end{bmatrix}, \tag{7}$$

where $A_i = -I - h_i \bar{A}_i$, $C_i = I - h_i \bar{C}_i$, and $f_i = h_i \bar{f}_i$. If the boundary conditions are separated, we obtain

$$\begin{bmatrix} \bar{B}_a & & & & \\ A_1 & C_1 & & & \\ & A_2 & C_2 & & \\ & & \ddots & \ddots & \\ & & & A_k & C_k \\ & & & & \bar{B}_b \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_k \\ s_{k+1} \end{bmatrix} = \begin{bmatrix} d_a \\ f_1 \\ f_2 \\ \vdots \\ f_k \\ d_b \end{bmatrix}. \tag{8}$$

The accuracy of finite-difference schemes is often enhanced by the use of deferred correction techniques (see, for example, Pereyra [18]).

As has been observed, the two algorithms are closely related, in the sense that for a reasonable choice of the approximation (6), $-C_i^{-1}A_i$ should be close to $Y_i(t_{i+1})$. Hence, the conditioning of the matrices in (4) and (7) is quite similar when the h_i are small. If we quantify the conditioning of the problem (1), (2) by a bound κ on its Green's function (see Ascher, Mattheij, and Russell [3, §3.2]), it has been shown by Osborne [16] (and also by Mattheij [14] and Lentini, Osborne, and Russell [12]) that the inverse of A_S from (4) satisfies the bound

$$\|A_S^{-1}\|_\infty \leq k\kappa.$$

Hence if we define γ by

$$\gamma = 1 + \max_{i=1,\cdots,k} \|Y_i(t_{i+1})\|_\infty$$

and assume that B_a and B_b are scaled such that $\|[B_a, B_b]\|_\infty = 1$, then from (4), (5),

$$\text{cond}_\infty(A_S) \leq \gamma k \kappa. \tag{9}$$

For the finite-difference coefficient matrix A_D in (7), a similar analysis applies if we note that $\gamma = 2 + O(\bar{h})$ for small $\bar{h} = \max_{i=1,\cdots,k} h_i$. We use this fact, together with $A_D = A_S + O(\bar{h})$, to derive the bound

$$\text{cond}_\infty(A_D) \leq (2 + O(\bar{h}))k\kappa. \tag{10}$$

The slightly different form of the bounds (9) and (10) is motivated by the fact that k is typically larger in a finite-difference algorithm than in a multiple shooting algorithm. Bounds for the coefficient matrices in (5) and (8) are, of course, identical to their nonseparated counterparts.

As stated earlier, Gaussian elimination algorithms with various forms of pivoting have been previously proposed for solving (4), (5), (7), and (8). The practical stability of such algorithms for general matrices is well known, but it is also well known that the worst-case behavior can be very bad, as a result of possible growth of elements in the factors, which is exponential in the dimension $(k+1)n$ of the system. It has been shown (see, for example, Mattheij [15]) that the presence of an exponential dichotomy in (1), (2) ensures that this worst-case behavior does not arise when elimination algorithms are applied to the matrices in (4), (5), (7), and (8). In the partially separated cases (5), (8), similar results can be proved, without referring to the dichotomy at all, by using the results of Bohte [4]. Bohte shows that for banded linear systems, the bound on element growth in partial pivoting algorithms is exponential only in the bandwidth.

In the simplest case of Gaussian elimination with row partial pivoting (with coefficient matrices and right-hand sides denoted by A_{SP} and f_{SP} in (5) and A_{DP} and f_{DP} in (8)), possible fill-in of kpn elements can occur in the upper triangular factor. Element growth is, however, bounded by 2^{2n-1}, and so we obtain the following theorem.

THEOREM 2.1. *Let $s = (s_1^T, \cdots, s_{k+1}^T)^T$ denote the true solution of the system (8), and suppose that $\hat{s}$ is the approximate solution obtained by Gaussian elimination with row partial pivoting. Then, provided that*

(i) *$\bar{h}$ is chosen small enough that* $\text{cond}_\infty(A_{DP}) \leq 4k\kappa$,

(ii) $8c_1\kappa\mathbf{u} \leq 1$, *where* $c_1 = (1.12)2^{2n}n^3k(k+1)(k+8) = O(k^3n^32^{2n})$ *and* $\mathbf{u}$ *is the unit roundoff error,*

the following relative error bound applies:

$$\frac{\|\hat{s} - s\|_\infty}{\|s\|_\infty} \leq 16c_1\kappa\mathbf{u}. \tag{11}$$

Suppose that the fundamental and particular solutions that are used to construct A_{SP} and f_{SP} in (5) are calculated to a tolerance of τ, that is,

$$\|A_{SP} - \bar{A}_{SP}\|_\infty \leq \tau\|A_{SP}\|_\infty, \qquad \|f_{SP} - \bar{f}_{SP}\|_\infty \leq \tau\|f_{SP}\|_\infty. \tag{12}$$

Let $\tilde{s}$ be the solution obtained by using Gaussian elimination with row partial pivoting. Then provided that

(iii) $\tau k\kappa\gamma \leq \frac{1}{2}$,

(iv) $6c_1\mathbf{u}\kappa\gamma \leq 1$, *we have*

$$\frac{\max_{i=1,\cdots,k+1}\|\tilde{s}_i - y(t_i)\|_\infty}{\max_{i=1,\cdots,k+1}\|y(t_i)\|_\infty} \le 4\gamma\kappa[k\tau + 3c_1\mathbf{u} + c_2\mathbf{u}\tau], \tag{13}$$

where $c_2 = 12c_1k\kappa\gamma$.

Proof. For the first part of the theorem, the proof follows from Theorem 2.7.2 in Golub and Van Loan [8] and §4 of Bohte [4]. The latter assumes that

$$\mathbf{u} \le .009 \qquad \text{and} \qquad (k+1)n\mathbf{u} \le 0.1,$$

which are clearly implied by assumption (ii), since $\kappa \ge 1$, $k \ge 1$, and $n \ge 1$. Bohte then shows that $\hat{s}$ solves the system

$$(A_{DP} + E)\hat{s} = f_{DP},$$

where E satisfies the bound

$$\|E\|_\infty \le \Lambda\mathbf{u}(0.56)(4n-1)n(k+1)(kn+7n+3),$$

and Λ is a bound on the maximal element that arises during the elimination. By simplifying this expression, and noting from the discussion above that

$$\Lambda \le 2^{2n-1}\max_{i,j}|(A_{DP})_{ij}| \le 2^{2n-1}\|A_{DP}\|_\infty,$$

we have that

$$\|E\|_\infty \le (c_1/k)\mathbf{u}\|A_{DP}\|_\infty.$$

The result now follows by setting $r = \frac{1}{2}$ in [8, Thm. 2.7.2].

For the second part, note first that

$$\|\bar{A}_{SP}\|_\infty \le (1+\tau)\|A_{SP}\|_\infty$$

and

$$\|\bar{A}_{SP}^{-1}\|_\infty \le \|[I + A_{SP}^{-1}(\bar{A}_{SP} - A_{SP})]^{-1}\|_\infty\|A_{SP}^{-1}\|_\infty.$$

Provided

$$\|A_{SP}^{-1}(\bar{A}_{SP} - A_{SP})\|_\infty \le 1, \tag{14}$$

we have from [8, Lemma 2.3.3] that

$$\|\bar{A}_{SP}^{-1}\|_\infty \le \frac{\|A_{SP}^{-1}\|_\infty}{1 - \|A_{SP}^{-1}\|_\infty\tau} \le \frac{k\kappa}{1 - \tau k\kappa} \le 2k\kappa.$$

The inequality (14) is implied by (iii). Since k, γ, and κ are all at least 1, assumption (iii) implies that $\tau \le \frac{1}{2}$, and so

$$\operatorname{cond}_\infty(\bar{A}_{SP}) \le 2(1+\tau)k\kappa\gamma \le 3k\kappa\gamma. \tag{15}$$

Now let $\bar{s}$ be the exact solution of $\bar{A}_{SP}\bar{s} = \bar{f}_{SP}$. Direct application of [8, Thm. 2.7.2] with $r = \frac{1}{2}$ shows, again using (iii), that

$$\frac{\max_{i=1,\cdots,k+1}\|\bar{s}_i - y(t_i)\|_\infty}{\max_{i=1,\cdots,k+1}\|y(t_i)\|_\infty} \le 4\operatorname{cond}_\infty(A_{SP})\tau \le 4k\kappa\gamma\tau. \tag{16}$$

Application of Bohte's results shows that the computed solution $\tilde{s}$ satisfies $(\bar{A}_{SP} + E)\tilde{s} = \bar{f}_{SP}$, where

$$\|E\|_\infty \le \frac{c_1}{k}\mathbf{u}\|\bar{A}_{SP}\|_\infty.$$

Assumption (iv) and (15) imply that

$$\frac{c_1}{k}\mathbf{u}\mathrm{cond}_\infty(\bar{A}_{SP}) \le \frac{1}{2},$$

and so

$$\frac{\|\tilde{s}-\bar{s}\|_\infty}{\|\bar{s}\|_\infty} \le \frac{4c_1\mathbf{u}}{k}\mathrm{cond}_\infty(\bar{A}_{SP}) \le 12c_1\mathbf{u}\kappa\gamma. \tag{17}$$

The result follows by combining (16) and (17) in an elementary way. □

A more economical scheme for solving (5) and (8), described by Varah [19] and implemented by Diaz, Fairweather, and Keast [7], is the method of alternate row and column elimination. For the first p stages of the process, we use *column* pivoting and elimination (involving columns 1 through n); this produces no fill-in. For stages $p+1$ through n we use *row* pivoting and elimination for the same reason. Column and row elimination alternate in this way until a factorization of the form

$$A_{DP} = PLBU\Pi \tag{18}$$

is produced, where P and Π are permutation matrices, L and U are lower and upper triangular matrices of multipliers, and B has the form

$$B = \begin{bmatrix}
L & & & & & & & & & \\
X & R & X & X & & & & & & \\
X & & L & & & & & & & \\
 & & X & R & X & X & & & & \\
 & & X & & L & & & & & \\
 & & & & \ddots & \ddots & \ddots & & & \\
 & & & & & & X & 0 & L & \\
 & & & & & & & & X & R
\end{bmatrix}.$$

(L denotes a lower triangular $p \times p$ block, R denotes an upper triangular $(n-p)\times(n-p)$ block, and X denotes a dense block.) It is easy to show that element growth in the B factor is bounded by 2^{n-1} and, hence, that the scheme is very stable. We can prove the following.

THEOREM 2.2. *Let $s = (s_1^T, \cdots, s_{k+1}^T)^T$ denote the true solution of the system (8), and let $\hat{s}$ be the approximate solution calculated by alternate row and column elimination. Suppose further that*

(i) *$\bar{h}$ is chosen small enough that* $\mathrm{cond}_\infty(A_{DP}) \le 4k\kappa$, *and*

(ii) $8c_4\kappa\mathbf{u} \le 1$,

where

$$c_4 = k(k+2)n(5+6n)(1+n^2 2^{n-1}) = O(k^2n^4 2^n).$$

Then

$$\frac{\|\hat{s}-s\|_\infty}{\|s\|_\infty} \le 16c_4\kappa\mathbf{u}.$$

Let $\tilde{s}$ be obtained by multiple shooting with alternate row and column elimination, and suppose that (12) *holds, with*

(iii) $\tau k\kappa\gamma \leq \frac{1}{2}$,

(iv) $6c_4\mathbf{u}\kappa\gamma \leq 1$.

Then

$$\frac{\max_{i=1,\cdots,k+1}\|\tilde{s}_i - y(t_i)\|_\infty}{\max_{i=1,\cdots,k+1}\|y(t_i)\|_\infty} \leq \gamma k\kappa[k\tau + 3c_4\mathbf{u} + c_5\mathbf{u}\tau],$$

where $c_5 = 12c_4k\kappa\gamma$.

Proof. The assumptions of Lemma A.1 are consequences of (ii) and (iv) above. The first part of the result follows directly from Lemma A.2 and [8, Thm. 2.7.2]. The second part is analogous to the second part of Theorem 2.1, with c_4 replacing c_1 and c_5 replacing c_2. □

3. Parallel elimination algorithms. Other parallel and vectorizable algorithms, based on Gaussian elimination, have recently been proposed for (4), (5), (7), and (8). In general, they suffer either from poor stability properties or from limitations in the amount of parallelism that is possible.

Wright and Pereyra [21] describe variants of a block factorization algorithm applied to (7). In the most highly vectorized variant, a factorization of the form

$$\begin{bmatrix} \tilde{A}_1 & & & & \\ & \tilde{A}_2 & & & \\ & & \ddots & & \\ & & & \tilde{A}_k & \\ Z_1 & Z_2 & \cdots & Z_k & \tilde{A}_{k+1} \end{bmatrix} \begin{bmatrix} I & W_1 & & & \\ & I & W_2 & & \\ & & \ddots & \ddots & \\ & & & I & W_k \\ & & & & I \end{bmatrix} = \begin{bmatrix} P_1 & & & & \\ & P_2 & & & \\ & & \ddots & & \\ & & & P_k & \\ & & & & P_{k+1} \end{bmatrix} \begin{bmatrix} A_1 & C_1 & & & \\ & A_2 & C_2 & & \\ & & \ddots & \ddots & \\ & & & A_k & C_k \\ B_a & & & & B_b \end{bmatrix}$$

(where the P_i are permutation matrices) is produced. It is easy to show that for h sufficiently small, this factorization exists. It is equivalent to compactification [3, p. 153] which, because of its similarity to single shooting, is known to be potentially unstable. However, this instability can usually be recognized by the presence of large elements in the Z_i blocks. The strategy described in [21, §5] is to use this factorization where possible, and discard it in favor of a more stable method if the $\|Z_i\|$ are too large. In many applications (such as the one described in [21]) the lack of stability is not a problem.

Paprzycki and Gladwell [17] describe a partitioned elimination algorithm in which (8) is torn into P submatrices, each of which has the same form as the original A_{DP}, and alternate row and column elimination is applied to each piece. This corresponds to partitioning the interval $[a, b]$ and solving a number of sub-BVPs, each of which has p initial and q final conditions. Although the number of initial and final conditions is correct, this alone is not enough to ensure well-conditioning of the sub-BVPs. In a linear algebra sense, well-conditioning of the whole matrix A_{DP} does not guarantee well-conditioning of each of the P submatrices. Ascher and Mattheij [2] develop a "theoretical multiple shooting" framework in which they show how boundary values for the sub-BVPs should be chosen to ensure well-conditioning. Ascher and Chan [1] suggest how to implement this in a parallel environment.

Another possibility, which leads to near-perfect speedup on two processors (but cannot be generalized to a higher order of parallelism) is the "burn at both ends" or "twisted" factorization. Here, some form of pivoted Gaussian elimination is applied simultaneously from both ends of the matrix (either A_{DP} or A_{SP}). When the factorizations meet in the center, a small reduced system is formed and factored. This is analogous to the approach of Lentini [11].

Finally, we mention the approach of Ascher and Chan [1], who form the normal equations for (5) and (8), and factorize the resulting symmetric, positive definite, block-tridiagonal system using cyclic reduction. In exact arithmetic, this scheme produces a triangular factor that is identical to that given by the "cyclic reduction" variant of the algorithm to be described in the next section. The difference lies in the fact that by explicitly forming the normal equations, the condition number of a linear system is squared, an effect that is avoided when the algorithm from §4 is used.

4. Structured orthogonal factorization. We now describe the structured QR factorization algorithm, as applied to the system (7). It can of course be applied equally well to the systems (8), (4), and (5), since it is indifferent to separability of the boundary conditions.

The first step is to partition the system into, say, P pieces of approximately equal size. We choose indices $k_1, k_2, \cdots, k_{P+1}$ to satisfy

$$0 = k_1 < k_2 < \cdots < k_{P+1} = k, \qquad k_{j+1} \geq k_j + 2, \quad j = 1, \cdots, P.$$

(P could, for instance, be the number of processors on the physical machine being used to solve the problem.) Partition j then consists of rows $(k_j+1)n+1, \cdots, (k_{j+1}+1)n$ of (7). Each partition is now processed independently; in effect, the variables s_i for $i \neq k_j + 1$, $j = 0, \cdots, P$, are eliminated from the problem.

We describe this process in detail for the first partition, which, if augmented with its right-hand side, has the form

$$\left[\begin{array}{ccccc|c} A_1 & C_1 & & & & f_1 \\ & A_2 & C_2 & & & f_2 \\ & & \ddots & \ddots & & \vdots \\ & & & A_{k_2} & C_{k_2} & f_{k_2} \end{array}\right]. \tag{19}$$

We first find an orthogonal matrix $Q_1 \in R^{2n \times 2n}$ such that

$$Q_1^T \begin{bmatrix} C_1 \\ A_2 \end{bmatrix} = \begin{bmatrix} R_1 \\ 0 \end{bmatrix},$$

where $R_1 \in R^{n \times n}$ is upper triangular. If we form Q_1 as a product of n Householder transformations, the information needed to reconstruct Q_1 could be stored in the space formerly occupied by the "zeroed" elements of $[C_1^T, A_2^T]^T$, plus an additional n locations. We also need to apply Q_1^T to the other columns of the matrix (19), and the overall effect is

$$\begin{bmatrix} Q_1^T & 0 \\ 0 & I \end{bmatrix} \left[\begin{array}{ccccc|c} A_1 & C_1 & & & & f_1 \\ & A_2 & C_2 & & & f_2 \\ & & \ddots & \ddots & & \vdots \\ & & & A_{k_2} & C_{k_2} & f_{k_2} \end{array}\right] \tag{20}$$

$$= \left[\begin{array}{cccccc|c} G_1 & R_1 & E_1 & & & & g_1 \\ \bar{G}_2 & 0 & \bar{C}_2 & & & & \bar{f}_2 \\ & & A_3 & C_3 & & & f_3 \\ & & & \ddots & \ddots & & \vdots \\ & & & & A_{k_2} & C_{k_2} & f_{k_2} \end{array}\right].$$

The next step is to find an orthogonal $Q_2 \in R^{2n \times 2n}$ such that

$$Q_2^T \begin{bmatrix} \bar{C}_2 \\ A_3 \end{bmatrix} = \begin{bmatrix} R_2 \\ 0 \end{bmatrix},$$

and to apply Q_2^T to rows $n+1, \cdots, 3n$ of the reduced system (20). This process is repeated a total of $k_2 - 1$ times, until finally we obtain a system equivalent to (19), which has the form

$$(21) \qquad \left[\begin{array}{cccccc|c} G_1 & R_1 & E_1 & & & & g_1 \\ G_2 & & R_2 & E_2 & & & g_2 \\ \vdots & & & \ddots & \ddots & & \vdots \\ G_{k_2-1} & & & & R_{k_2-1} & E_{k_2-1} & g_{k_2-1} \\ \tilde{A}_1 & & & & & \tilde{C}_1 & \tilde{f}_1 \end{array}\right].$$

Formally, the reduction process for partition j can be specified as follows:

$$\bar{C}_{k_j+1} = C_{k_j+1}, \quad \bar{G}_{k_j+1} = A_{k_j+1}, \quad \bar{f}_{k_j+1} = f_{k_j+1}$$

for $i = k_j + 1, \cdots, k_{j+1} - 1$.

Find orthogonal Q_i such that

$$(22) \qquad Q_i^T \begin{bmatrix} \bar{C}_i \\ A_{i+1} \end{bmatrix} = \begin{bmatrix} R_i \\ 0 \end{bmatrix},$$

where $R_i \in R^{n \times n}$ is upper triangular;

Set

$$\begin{bmatrix} E_i \\ \bar{C}_{i+1} \end{bmatrix} \leftarrow Q_i^T \begin{bmatrix} 0 \\ C_{i+1} \end{bmatrix}, \quad \begin{bmatrix} G_i \\ \bar{G}_{i+1} \end{bmatrix} \leftarrow Q_i^T \begin{bmatrix} \bar{G}_i \\ 0 \end{bmatrix}, \quad \begin{bmatrix} g_i \\ \bar{f}_{i+1} \end{bmatrix} \leftarrow Q_i^T \begin{bmatrix} \bar{f}_i \\ f_{i+1} \end{bmatrix}.$$

end (for)

$$\text{Set } \tilde{A}_j = \bar{G}_{k_{j+1}}, \qquad \tilde{C}_j = \bar{C}_{k_{j+1}}, \qquad \tilde{f}_j = \bar{f}_{k_{j+1}}.$$

Clearly, if we knew the values of s_{k_j+1} and $s_{k_{j+1}+1}$, the values of $s_{k_j+2}, \cdots, s_{k_{j+1}}$ could be recovered by the simple back-substitution:

for $i = k_{j+1} - 1, k_{j+1} - 2, \cdots, k_j + 1$

$$(23) \qquad \text{Solve } R_i s_{i+1} = g_i - G_i s_{k_j} - E_i s_{i+2}.$$

In fact, these "separator" variables can be found by forming and solving a reduced system by taking the last n rows of each (processed) partition, and the boundary conditions. This reduced system has the form

$$
\begin{bmatrix} B_a & & & & B_b \\ \tilde{A}_1 & \tilde{C}_1 & & & \\ & \tilde{A}_2 & \tilde{C}_2 & & \\ & & \ddots & \ddots & \\ & & & \tilde{A}_P & \tilde{C}_P \end{bmatrix} \begin{bmatrix} s_{k_1+1} \\ s_{k_2+1} \\ s_{k_3+1} \\ \vdots \\ s_{k_{P+1}+1} \end{bmatrix} = \begin{bmatrix} d \\ \tilde{f}_1 \\ \tilde{f}_2 \\ \vdots \\ \tilde{f}_P \end{bmatrix} . \tag{24}
$$

The form of (24) is obviously identical to the form of the original system (7), so this immediately suggests that it may be possible to apply the whole process recursively, that is, (24) could be partitioned into $P_2 \leq P/2$ pieces, and the algorithm described above could be applied to obtain a smaller reduced system. This process could be repeated for, say, L levels, so at the innermost level a $(P_L + 1)n$-dimensional system would remain.

In a parallel implementation of the algorithm, the number of levels L to be used and the number of processors to be used at each level depend on the number of available processors on the physical machine, and, on a distributed-memory machine, on the cost of interprocessor communication. The "extreme" cases are as follows:

- A one-level version ($P = 1, k_2 = k$), in which (7) is reduced to the $2n \times 2n$ system

$$
\begin{bmatrix} B_a & B_b \\ \tilde{A}_1 & \tilde{C}_1 \end{bmatrix} \begin{bmatrix} s_1 \\ s_{k+1} \end{bmatrix} = \begin{bmatrix} d \\ \tilde{f}_1 \end{bmatrix}, \tag{25}
$$

  which is then solved by QR factorization. This is the "serial" version of the algorithm.
- A two-level version, in which P processors are used to do the reduction and back-substitution (23), and the system (24) is solved by the one-level algorithm just described. It is easy to see that, assuming that each partition contains about k/P rows of blocks, the time required for the reduction phase is proportional to $(k/P - 1)n^3$. The time required to solve (24) on a single processor is proportional to $(P + 1)n^3$. The latter operation will not be a bottleneck for the whole process, provided that $k \gg P^2$.
- A "cyclic reduction" version, in which P is equal to the number of available processors (assuming that $P \leq k/2$), and then $P_2 = P/2$, $P_3 = P/4$, $\cdots$, $P_L = 1$. Here, clearly, $L = \log_2 P + 1$. The total computation time required will be proportional to $(k/P - 1 + \log_2 P)n^3$, while the communication time will be proportional to $n^2 \log_2 P$. With regard to complexity, the algorithm is optimal: when $k = P \log_2 P$, the execution time is $O(\log_2 P)$ on P processors, while the serial time is $O(P \log_2 P)$.

The cyclic reduction variant is also appropriate for implementation on a single *vector* processor. In this environment, we could choose $P = k/2, P_2 = k/4, P_3 = k/8, \cdots$, and so $L = \log_2 k + 1$. At level l, the reduction and back-substitution processes can be coded so that most of the arithmetic involves vectors of length $k/2^l$. At low levels, a reversal of the loop nesting can be used to ensure that vector lengths do not fall below n.

It is not difficult to see that the factorization and solution phases can be separated, provided that the Householder vectors are stored (in the locations vacated by the zeroed elements). This feature is useful when quasi linearization is used to solve a first-order nonlinear BVP. The same coefficient matrix (and its factorization) can be used for a number of consecutive iterations to produce "chord method" approximations to Newton steps.

It is important to note that the scheme proposed above is simply standard Householder QR factorization applied to an initial row- and column-permuted form of the original matrix. Thus, we can apply the standard stability analysis for $(k+1)n$-dimensional matrices from Lawson and Hanson [10] to obtain error bounds for the computed solutions. We have refined these bounds to take into account the structure of the matrix. In addition, to allay any possible concerns about instability at the level of the $O(\mathbf{u}^2)$ terms, we have removed these terms using the style of analysis in Wilkinson [20]. The relevant results are stated and proved in Appendix B. Here, we summarize the analysis in the following lemma and theorem.

LEMMA 4.1. *Let $s = (s_1^T, s_2^T, \cdots, s_{k+1}^T)^T$ denote the true solution of the system* (7), *and suppose that $\hat{s}$ is the approximate solution obtained by using the structured QR factorization and back-substitution. Suppose that the orthogonal matrices used in the factorization are all constructed from Householder reflectors, and that n, k, and $\mathbf{u}$ satisfy*

$$(12n+51)n(k+1)\mathbf{u} \le 0.1. \tag{26}$$

Then $\hat{s}$ is the exact solution of a perturbed system

$$(A_D + \delta A_D)\hat{s} = f_D + \delta f_D, \tag{27}$$

where

$$\|\delta A_D\|_F \le (1.106)(12n+51)(k+2)n\mathbf{u}\|A_D\|_F, \tag{28}$$

$$\|\delta f_D\|_2 \le (1.106)(12n+51)(k+1)n\mathbf{u}\|f_D\|_2. \tag{29}$$

Proof. See Appendix B. □

THEOREM 4.2. *Suppose that the conditions of Lemma* 4.1 *are satisfied. Then, provided that*

(i) *$\bar{h}$ is chosen small enough that* $\text{cond}_\infty(A_{DP}) \le 4k\kappa$,

(ii) $8c_6\kappa\mathbf{u} \le 1$, *where* $c_6 = (1.106)k(k+2)^2n^2(12n+51)$,

the following relative error bound applies:

$$\frac{\|\hat{s}-s\|_\infty}{\|s\|_\infty} \le 16c_6\kappa\mathbf{u}.$$

Suppose that the fundamental and particular solutions that are used to construct A_S and f_S in (4) *are calculated to a tolerance of τ, that is,*

$$\|A_S - \bar{A}_S\|_\infty \le \tau\|A_S\|_\infty, \qquad \|f_S - \bar{f}_S\|_\infty \le \tau\|f_S\|_\infty. \tag{30}$$

Let $\tilde{s}$ be the solution obtained by using a structured QR algorithm, where (26) *holds. Then, provided that*

(iii) $\tau k\kappa\gamma \le \frac{1}{2}$,

(iv) $6c_6\mathbf{u}\kappa\gamma \le 1$,

we have

$$\frac{\max_{i=1,\cdots,k+1}\|\tilde{s}_i - y(t_i)\|_\infty}{\max_{i=1,\cdots,k+1}\|y(t_i)\|_\infty} \le 4\gamma\kappa[k\tau + 3c_6\mathbf{u} + c_7\mathbf{u}\tau],$$

where $c_7 = 12c_6 k\kappa\gamma$.

Proof. The proof is identical to that of Theorem 2.1, once we convert the Frobenius norm bounds (28) and (29) into ∞-norm bounds. This is done by noting that for an $N \times N$ matrix A,

$$\frac{1}{N^{1/2}}\|A\|_\infty \leq \|A\|_F \leq N^{1/2}\|A\|_\infty.$$

Hence

$$\|\delta A_D\|_\infty \leq (1.106)(k+2)^2 n^2 (12n+51)\mathbf{u}\|A_D\|_\infty. \qquad \square$$

5. Condition estimation.

For purposes of assessing the reliability of the computed solution, it is useful to have an estimate of the conditioning of the discrete system. Such an estimate can be obtained, simultaneously with the factorization and solution process, by adapting the procedure described in Cline et al. [6] to our situation.

We aim to compute an estimate of the quantity

$$\hat{\kappa} = \|A\|_\infty \|R^{-1}\|_\infty,$$

where A is one of the coefficient matrices from (4), (7), (5), and (8), and R is the upper triangular factor produced by the procedure just described. It is easy to show that

$$\frac{1}{\sqrt{(k+1)n}}\hat{\kappa} \leq \mathrm{cond}_\infty(A) \leq \sqrt{(k+1)n}\hat{\kappa}.$$

$\|A\|_\infty$ can of course be calculated directly, and so computation of the estimate of $\|R^{-1}\|_\infty$ is the major part of the task of finding $\hat{\kappa}$. Following [6], we do this by first finding vectors z and v such that

$$R^T v = z,$$

where the components of z are all ± 1 and are chosen by a heuristic that attempts to maximize $\|v\|_\infty$. This is done during the factorization of A; as reduction of each partition into a single row of blocks is performed, the heuristic can be applied to finding the components of z and v corresponding to the rows and columns of the original system that are eliminated. Next, the solution of

$$Rw = v$$

is found concurrently with the solution of the original linear system $Rs = Q^T f$. We then use the estimate

$$\|R^{-1}\|_\infty \approx \frac{\|w\|_\infty}{\|v\|_\infty}.$$

The operation count for calculation of $\hat{\kappa}$ is approximately four times that of doing a single backsolve with R, and the parallel complexity is the same as that of the factorization and solution. Comparisons with the LINPACK condition number estimator of $\mathrm{cond}_1(A)$ (which is based on an LU factorization of A but uses similar heuristics) show that the two estimates are usually within a factor of 3 of each other.

TABLE 1

Operation counts and storage requirements for four algorithms, assuming separated end conditions. k = number of meshpoints, n = dimension of y, p = number of left-hand end conditions, R = number of right-hand sides.

Algorithm	Operation count	Storage
LU (row pivoting)	$k[\frac{5}{3}n^3 + 3pn^2 + R(4n^2 + 2pn)]$	$kn(2n + p)$
Structured QR	$k[\frac{46}{3}n^3 + (15R + 30)n^2]$	$4kn^2$
Normal equations	$k[\frac{38}{3}n^3 + 12Rn^2]$	$4kn^2$
DECOMP/SOLVE	$k[\frac{2}{3}n^3 + (4R + 5p)n^2 - 2np^2]$	$2kn^2$

TABLE 2

Operation counts and storage requirements for four algorithms, assuming nonseparated end conditions. k = number of meshpoints, n = dimension of y, R = number of right-hand sides.

Algorithm	Operation count	Storage
LU (row pivoting)	$k[\frac{23}{3}n^3 + 8Rn^2]$	$4kn^2$
Structured QR	$k[\frac{46}{3}n^3 + (15R + 30)n^2]$	$4kn^2$
Normal equations	$k[\frac{62}{3}n^3 + 12Rn^2]$	$5kn^2$
DECOMP/SOLVE	$k[\frac{14}{3}n^3 + 4Rn^2]$	$3kn^2$

6. Computational results. Versions of the structured QR algorithm have been implemented on the Alliant FX/8 vector multiprocessor at Argonne National Laboratory and on the CRAY Y-MP at the North Carolina Supercomputing Center. Performance comparisons were made with

- a row partial pivoting code (two versions, for separated and nonseparated boundary conditions), and
- the DECOMP and SOLVE routines from the PASVA codes [13]. The DECOMP routine uses alternate row and column pivoting (as does the algorithm described in Varah [19] and in §2) but always eliminates by rows.

Approximate operation counts and storage requirements for these algorithms and the normal equations method of Ascher and Chan [1] are given in Tables 1 and 2, for separated and nonseparated boundary conditions, respectively. In tabulating storage requirements, it has been assumed that intermediate information generated during the factorization—namely, the multipliers and Householder vectors—is stored for possible later use with a different right-hand side. In general, the structured QR algorithms require the most operations. This result is not surprising, since it is well known that orthogonal factorization of dense matrices requires about twice as much work as Gaussian elimination. Moreover, in the case of separated end conditions, structured QR generates fill-in that is avoided by the elimination-based methods, and this adds further to the operation count. On the other hand, the operation counts and storage requirements for structured QR are not affected if the end conditions are nonseparated rather than separated, while for the other methods, they increase substantially. The method based on normal equations requires about the same amount of work as structured QR; however, as we noted earlier, it is less stable. We add the caveat that operation counts are notoriously bad predictors of run time for factorizations of narrow-banded matrices. For small n, much of the CPU time is taken up with nonnumerical operations. This is borne out by our results, which show that the QR algorithm does not do as badly as predicted in serial mode.

TABLE 3
Box method, error in first component of computed solution for Problem 1 *for four different linear system solvers.*

	$k = 16$	$k = 64$	$k = 1024$
$\hat{\kappa}$	.17(+2)	.51(+1)	.21(+1)
ROWPP	.21(-2)	.10(-3)	.32(-6)
DECOMP	.22(-2)	.10(-3)	.32(-6)
SQR-1	.21(-2)	.10(-3)	.32(-6)
COMPACT	.22(-2)	.93(+27)	.16(+72)

TABLE 4
Multiple shooting, error in first component of computed solution for Problem 1 *for four different linear system solvers.*

	$k = 16$	$k = 32$	$k = 128$
$\hat{\kappa}$	.85(+6)	.17(+4)	.13(+2)
ROWPP	.45(-3)	.67(-6)	.64(-7)
DECOMP	.45(-3)	.67(-6)	.64(-7)
SQR-1	.45(-3)	.67(-6)	.64(-7)
COMPACT	.51(+72)	.21(+72)	.11(+72)

The linear system solvers described above have been incorporated into both a multiple shooting and finite-difference (box method) framework. The `dverk` code from netlib was used to solve the initial value problems (IVPs) on each interval of multiple shooting, with the global error tolerance parameter set to 10^{-10}. A user-specified number k of equally spaced intervals is used for both methods. For practical codes, the choice of the number of intervals and their lengths (and efficient parallel implementation of this) are important issues, but we focus here on the linear algebra, which typically is the most computationally intensive part of a finite-difference–based code. In accord with the theoretical results of §§2 and 4, virtually no difference was noted between the stability properties of structured QR and row-pivoted LU. As evidence of this, we quote results for the following test problem.

PROBLEM 1 (Ascher and Chan [1]). $a = 0$, $b = 1$, $n = 2$,

$$y'(t) = \begin{bmatrix} -\lambda\cos 2\omega t & \omega + \lambda\sin 2\omega t \\ -\omega + \lambda\sin 2\omega t & \lambda\cos 2\omega t \end{bmatrix} y(t) + f(t),$$

$$B_a = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \qquad B_b = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix},$$

with $f(t)$ and d chosen so that $y(t) = e^t(1,1)^T$.

A fundamental solution is

$$Y(t) = \begin{bmatrix} \cos\omega t & \sin\omega t \\ -\sin\omega t & \cos\omega t \end{bmatrix} \begin{bmatrix} e^{-\lambda t} & 0 \\ 0 & e^{\lambda t} \end{bmatrix},$$

so clearly there exist one growing and one decaying mode. The problem was solved using both multiple shooting and the box method for $\lambda = 200$ and $\omega = 1$. Four different algorithms were used to solve the linear system, namely,

- SQR-1—one-level structured QR;
- ROWPP—LU factorization with row partial pivoting;
- DECOMP—the DECOMP and SOLVE routines from PASVA; and
- COMPACT—compactification, as implemented in the codes D4/S4 described in [21].

TABLE 5

Dimensions of the five test cases, and conditioning of the multiple shooting and finite-difference matrices.

Problem	n	k	ms conditioning	fd conditioning
1a	2	64	.41(+2)	.17(+2)
1b	2	1024	.30(+2)	.30(+2)
2a	4	64	.37(+9)	.83(+3)
2b	4	1024	.26(+2)	.17(+2)
3	3	1024	.20(+1)	.34(+1)

Tables 3 and 4 show the maximum error in the first component of the computed result. Because of its failure to decouple the fundamental solution modes, compactification performs poorly. The accuracy of box method solutions is limited by discretization error, while the multiple shooting solutions are accurate up to the conditioning of the discrete system and the tolerance imposed on the IVP solver. The DECOMP code gives accurate results here because the end conditions are separated. When this is not the case, as in Problem 3 below, DECOMP is known to be unstable.

To test the relative speed of the linear solvers, two further problems from the literature were used in addition to Problem 1.

PROBLEM 2 (Brown and Lorenz [5]). $a = -1$, $b = 1$, $n = 4$,

$$\begin{aligned} -\epsilon y'' - \frac{t}{2}y' + \frac{t}{2}z' + z &= \epsilon\pi^2 \cos \pi t \ + \frac{1}{2}\pi t \sin \pi t, \\ \epsilon z'' &= z, \\ y(-1) = -1 \quad & y(1) = e^{-2/\sqrt{\epsilon}}, \\ z(-1) = 1 \quad & z(1) = e^{-2/\sqrt{\epsilon}}. \end{aligned}$$

(We use $\epsilon = .001$.)

PROBLEM 3 (Mattheij [15]). $a = 0$, $b = \pi$, $n = 3$,

$$\begin{aligned} y'(t) = &\begin{bmatrix} 1 - 19\cos 2t & 0 & 1 + 19\sin 2t \\ 0 & 19 & 0 \\ -1 + 19\sin 2t & 0 & 1 + 19\cos 2t \end{bmatrix} y(t) \\ &+e^t \begin{bmatrix} -1 + 19(\cos 2t \ - \sin 2t) \\ -18 \\ 1 - 19(\cos 2t \ + \sin 2t) \end{bmatrix}, \end{aligned}$$

$$\begin{aligned} y_1(0) &= 1, \\ y_3(0) + y_3(\pi) &= 1 + e^{\pi}, \\ y_2(0) + y_2(\pi) &= 1 + e^{\pi}. \end{aligned}$$

The solution is $y(t) = e^t(1, 1, 1)^T$.

Problems 1 and 2 have separated end conditions, while two of the three end conditions for Problem 3 are nonseparated. We report on five cases (two different values of k were tried for Problems 1 and 2, and values of $\lambda = 1$ and $\omega = 50$ were used in Problem 1). Table 5 gives condition estimates for the multiple shooting and finite-difference matrices.

Results from "scalar" implementations on one processor of the Alliant FX/8 are shown in Table 6. We have tabulated the times required to solve the linear systems. The -Og compiler option was used with each code, so the vector processing capabilities of the Alliant were not used. In addition to the linear solvers already mentioned, we

TABLE 6
Alliant FX/8, one-processor timings for linear system solvers (times in seconds).

Problem	ROWPP	DECOMP	SQR-1	SQR-CR
1a	.041	.081	.071	.098
1b	.439	.660	.813	1.04
2a	.094	.181	.220	.312
2b	1.32	1.95	3.22	4.55
3	1.23	1.44	1.72	2.34

TABLE 7
CRAY Y-MP, one-processor timings for linear system solvers. Vectorized code (times in milliseconds).

Problem	ROWPP	DECOMP	SQR-1	SQR-CR
1a	2.19	3.05	8.63	1.45
1b	34.6	48.3	139.	10.9
2a	5.32	8.38	21.1	6.40
2b	84.4	133.	341.	51.6
3	136.	116.	232.	26.3

tested SQR-CR, which was the cyclic-reduction variant of structured QR. Note that the SQR codes typically take two to three times as long as ROWPP, though the penalty is much smaller when the end conditions are not separated (as in Problem 3). In either case, the overhead for using structured QR is not as great as the operation counts in Tables 1 and 2 would suggest.

Timings for a vectorized implementation on one processor of a CRAY Y-MP are shown in Table 7. In general, the SQR-CR code becomes very competitive, particularly when $n = 2$ or 3, k is large, and/or the end conditions are not separated. This code performs extremely well on Problems 1b and 3. When $n = 4$ (Problem 2), the small amount of vectorization that occurs in the other codes lessens the advantage of SQR-CR, while in Problems 1a and 2a the value of k makes the overall computational task too small to benefit from vectorization.

Table 8 gives results for an eight-processor parallel implementation on the Alliant FX/8. The -Ogc option was used during compilation. Here, SQR-2 refers to the two-level version of structured QR, in which the original system is broken into eight partitions of equal size, which are factorized concurrently. On the largest problem, the parallel efficiency of structured QR (measured by comparing serial SQR-1 to parallel SQR-2) is 87 percent—quite acceptable, given that the solution of the reduced system is an unavoidable bottleneck. The efficiency improves further for still larger problems. Defining speedup as the ratio of the one-processor time for the best serial algorithm to the eight-processor time for the best parallel algorithm, we see, from Table 9, that in three of the five cases good parallel efficiency is attained. The remaining two problems were too small for parallelism to have much effect.

Comparing Tables 6 and 8, it can be seen that ROWPP and DECOMP also speed up a little when extra processors are available. This is because the Alliant is a shared-memory machine. It is important to note that on the current generation of message-passing machines, these algorithms will *not* benefit from multiprocessing unless n is large enough that rows or columns *within each block* can profitably be distributed around the processor array. This is unlikely to happen until n is at least 50 or 100. On the other hand, efficient implementations of multilevel SQR on these machines will be possible for much more typical problem sizes.

TABLE 8
Alliant FX/8, eight-processor timings for linear system solvers (times in seconds).

Problem	ROWPP	DECOMP	SQR-2
1a	.029	.072	.031
1b	.367	.697	.136
2a	.053	.150	.067
2b	.739	1.67	.463
3	.685	1.39	.262

TABLE 9
Alliant FX/8, Ratio of times for ROWPP *(one-processor) to times for* SQR-2 *(eight processors).*

Problem	Speedup
1a	1.3
1b	3.2
2a	1.4
2b	2.9
3	4.7

To summarize, we conclude that the structured QR codes are useful in the following circumstances:

- when the computational task of solving the linear equations is substantial enough to benefit from vectorization or parallelism;
- especially, when the end conditions are not separated;
- on a vector processor, when the value of n is too small (say, only 2 or 3) to allow efficient vectorized factorization of $n \times n$ blocks;
- on the current generation of distributed-memory multiprocessors, unless n is very large and the number of processors is very small;
- on a shared-memory multiprocessor, unless n is quite large (say, greater than 8) and there are fewer than four processors.

A. Appendix A.

We start with a result which is similar to [8, Thm. 3.3.1].

LEMMA A.1. *Suppose that alternate row and column elimination, without pivoting, is applied to an $N \times N$ matrix A with bandwidth b_w to produce computed factors $\hat{L}$, $\hat{B}$, $\hat{U}$. Assume that N, b_w, and the unit roundoff error $\mathbf{u}$ satisfy*

$$N\mathbf{u} < 0.1, \tag{31}$$

$$N\mathbf{u}(2 + 1.06b_w + 2.12b_w\mathbf{u}) < 0.5. \tag{32}$$

Then

$$\hat{L}\hat{B}\hat{U} = A + H,$$

where

$$|H| \le c_3(N-1)\mathbf{u}\{|A| + |\hat{L}||\hat{B}||\hat{U}|\} \tag{33}$$

and

$$c_3 = 5 + 3b_w.$$

Proof. The result is trivially true for $N = 1$. Suppose for induction that (31) holds for matrices of size up to $N - 1$. Let

$$A = \begin{bmatrix} \alpha & w^T \\ v & A_1 \end{bmatrix},$$

where $\alpha \in R$, $A_1 \in R^{(N-1)\times(N-1)}$, etc., and suppose that row elimination will be used to eliminate v. We compute

$$\hat{z} = \frac{1}{\alpha}v + f, \qquad |f| \le \frac{|v|}{\alpha}\mathbf{u}, \tag{34}$$

$$\hat{A}_1 = A_1 - \hat{z}w^T + F, \qquad |F| \le 2\mathbf{u}(|A_1| + |\hat{z}||w|^T). \tag{35}$$

It follows immediately from (35) that

$$|\hat{A}_1| \le (1 + 2\mathbf{u})(|A_1| + |\hat{z}||w|^T). \tag{36}$$

An LBU factorization of $\hat{A}_1$ is then performed, yielding

$$\hat{L}_1\hat{B}_1\hat{U}_1 = \hat{A}_1 + H_1,$$

with

$$|H_1| \le c_3(N-2)\mathbf{u}\{|\hat{A}_1| + |\hat{L}_1||\hat{B}_1||\hat{U}_1|\}. \tag{37}$$

The calculated factors of A are therefore

$$\hat{L} = \begin{bmatrix} 1 & 0 \\ \hat{z} & \hat{L}_1 \end{bmatrix}, \qquad \hat{B} = \begin{bmatrix} \alpha & \hat{w}^T \\ 0 & \hat{B}_1 \end{bmatrix}, \qquad \hat{U} = \begin{bmatrix} 1 & 0 \\ 0 & \hat{U}_1 \end{bmatrix},$$

where $\hat{w}$ is the computed solution of the system $\hat{U}_1^T\tilde{w} = w$. Defining $\bar{b}_w = 1.06b_w$, and noting that $\hat{U}_1^T$ has lower bandwidth b_w, it is easy to show that $\hat{w}$ *exactly* solves

$$(\hat{U}_1^T + \delta_U)\hat{w} = w,$$

where

$$|\delta_U| \le \bar{b}_w\mathbf{u}|\hat{U}_1|^T.$$

Hence

$$|\hat{U}_1^T\hat{w} - w| = |\delta_U\hat{w}| \le |\delta_U||\hat{w}| \le \bar{b}_w\mathbf{u}|\hat{U}_1|^T|\hat{w}|, \tag{38}$$

and so

$$|w| \le (1 + \bar{b}_w\mathbf{u})|\hat{U}_1|^T|\hat{w}|. \tag{39}$$

Now

$$\begin{aligned} &\hat{A}_1 = A_1 - \hat{z}w^T + F, \\ \Rightarrow\ &\hat{z}\hat{w}^T\hat{U}_1 + \hat{L}_1\hat{B}_1\hat{U}_1 = A_1 + F + H_1 - \hat{z}[w - \hat{U}_1^T\hat{w}]^T. \end{aligned}$$

Combining this with (34),

$$\hat{L}\hat{B}\hat{U} = A + \begin{bmatrix} 0 & \hat{w}^T\hat{U}_1 - w^T \\ \alpha f & F + H_1 - \hat{z}[w - \hat{U}_1^T\hat{w}]^T \end{bmatrix}. \tag{40}$$

Now, combining (35), (36), (37), and (39), we find that

$$\begin{aligned} &|F + H_1 - \hat{z}[w - \hat{U}_1^T\hat{w}]^T| \\ &\quad \le |F| + |H_1| + |\hat{z}||w - \hat{U}_1^T\hat{w}|^T \\ &\quad \le [c_3(N-2)\mathbf{u}(1+2\mathbf{u}) + 2\mathbf{u}]|A_1| \\ &\qquad + [c_3(N-2)\mathbf{u}(1+2\mathbf{u})(1+\bar{b}_w\mathbf{u}) + 2\mathbf{u}(1+\bar{b}_w\mathbf{u}) + \bar{b}_w\mathbf{u}]|\hat{z}||\hat{w}|^T|\hat{U}_1| \\ &\qquad + c_3(N-2)\mathbf{u}|\hat{L}_1||\hat{B}_1||\hat{U}_1|. \end{aligned} \tag{41}$$

We now show that

$$c_3(N-2)\mathbf{u}(1+2\mathbf{u})(1+\bar{b}_w\mathbf{u})+2\mathbf{u}(1+\bar{b}_w\mathbf{u})+\bar{b}_w\mathbf{u}\le c_3(N-1)\mathbf{u}. \tag{42}$$

This is equivalent to

$$\begin{aligned}&c_3(N-2)\mathbf{u}(2\mathbf{u}+\bar{b}_w\mathbf{u}+2\bar{b}_w\mathbf{u}^2)+(2\mathbf{u}+\bar{b}_w\mathbf{u}+2\bar{b}_w\mathbf{u}^2)\le c_3\mathbf{u}\\ &\qquad\Leftrightarrow c_3[1-(N-2)\mathbf{u}(2+\bar{b}_w+2\bar{b}_w\mathbf{u})]\ge(2+\bar{b}_w+2\bar{b}_w\mathbf{u}).\end{aligned}$$

Since by assumption (32),

$$1-(N-2)\mathbf{u}(2+\bar{b}_w+2\bar{b}_w\mathbf{u})\ge\tfrac{1}{2},$$

inequality (42) will hold provided that

$$c_3\ge 2(2+\bar{b}_w+2\bar{b}_w\mathbf{u}). \tag{43}$$

From (31), and using $b_w<N$, it is clear that$2\bar{b}_w\mathbf{u}\le 0.5$, so (43) follows trivially.

Since the left-hand side of (42) is the largest of the three coefficients in (41), we can combine (41) and (42) to obtain

$$|F+H_1-\hat{z}[w-\hat{U}_1^T\hat{w}]^T|\le c_3(N-1)\mathbf{u}\{|A_1|+|\hat{z}||\hat{w}|^T|\hat{U}_1|+|\hat{L}_1||\hat{B}_1||\hat{U}_1|\}. \tag{44}$$

Combining (40), (34), (38), and (44), we therefore find that

$$\begin{aligned}&|A-\hat{L}\hat{B}\hat{U}|\\ &=\left|\begin{bmatrix}0 & \hat{w}^T\hat{U}_1-w^T\\ \alpha f & F+H_1-\hat{z}[w-\hat{U}_1^T\hat{w}]^T\end{bmatrix}\right|\\ &\le c_3(N-1)\mathbf{u}\left\{\begin{bmatrix}|\alpha| & |w|^T\\ |v| & |A_1|\end{bmatrix}+\begin{bmatrix}1 & 0\\ |\hat{z}| & |\hat{L}_1|\end{bmatrix}\begin{bmatrix}|\alpha| & |\hat{w}|^T\\ 0 & |\hat{B}_1|\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & |\hat{U}_1|\end{bmatrix}\right\}\\ &=c_3(N-1)\mathbf{u}\{|A|+|\hat{L}||\hat{B}||\hat{U}|\}.\end{aligned}$$

This proves the result for the case in which row elimination is used at stage N. When column elimination is used instead, the proof is analogous. □

LEMMA A.2. *If the alternate row and column elimination is used to solve* (5) *or* (8), *and the assumptions of Lemma* A.1 *hold, with* $N=(k+1)n$ *and* $b_w=2n$, *then the computed solution* $\hat{s}$ *is the exact solution of the perturbed system*

$$(A+E)\hat{s}=f,$$

where

$$\|E\|_\infty\le(k+2)n(5+6n)(1+n^2 2^{n-1})\|A_{DP}\|_\infty\mathbf{u}.$$

Proof. Note first that the pivoting does not alter the sparsity structure of A. We can, therefore, view alternate row and column elimination as being applied to $P^TA\Pi^T$ (where P and Π are permutation matrices) to produce a computed factorization

$$P^TA\Pi^T+H=\hat{L}\hat{B}\hat{U}.$$

It is easy to show that the procedure leading to $\hat{s}$ results in the following sequence of perturbed problems:

$$\begin{aligned} (\hat{L}+\delta_L)\hat{w} &= P^T f, & |\delta_L| &\le (1.06)n\mathbf{u}|\hat{L}|, \\ (\hat{B}+\delta_B)\hat{v} &= \hat{w}, & |\delta_B| &\le 2(1.06)n\mathbf{u}|\hat{B}|, \\ (\hat{U}+\delta_U)\hat{z} &= \hat{v}, & |\delta_U| &\le (1.06)n\mathbf{u}|\hat{U}|, \\ \hat{s} &= \Pi^T \hat{z}. \end{aligned}$$

(The bounds on $|\delta_L|$, $|\delta_B|$, $|\delta_U|$ are a consequence of the maximum number of nonzeros in each row of $\hat{L}$, $\hat{B}$, $\hat{U}$, respectively.) Hence

$$\begin{aligned} &P^T E \Pi^T = H + (\hat{L}+\delta_L)(\hat{B}+\delta_B)(\hat{U}+\delta_U) - \hat{L}\hat{B}\hat{U} \\ &\Rightarrow \|E\|_\infty \le \|H\|_\infty + [4(1.06)n\mathbf{u} + 5(1.06)^2 n^2\mathbf{u}^2 + 2(1.06)^3 n^3 \mathbf{u}^3]\|\hat{L}\|_\infty \|\hat{B}\|_\infty \|\hat{U}\|_\infty. \end{aligned}$$

It follows from (31) that $n\mathbf{u} \le 0.1$, so the coefficient of $\|\hat{L}\|_\infty \|\hat{B}\|_\infty \|\hat{U}\|_\infty$ can be bounded above by $5n\mathbf{u}$. Since element growth in B is bounded by 2^{n-1}, and since all entries in $\hat{L}$ and $\hat{U}$ are bounded by 1, we have

$$\|\hat{B}\|_\infty \le 2^{n-1}\|P^T A \Pi^T\|_\infty \le 2^{n-1}\|A\|_\infty, \qquad \|\hat{L}\|_\infty \le n, \qquad \|\hat{U}\|_\infty \le n.$$

Combining these observations with the result of Lemma A.1, we obtain

$$\begin{aligned} \|E\|_\infty &\le (5+6n)(k+1)n\mathbf{u}\{\|A\|_\infty + \|\hat{L}\|_\infty \|\hat{B}\|_\infty \|\hat{U}\|_\infty\} + 5n\mathbf{u}\|\hat{L}\|_\infty \|\hat{B}\|_\infty \|\hat{U}\|_\infty \\ &\le \{(5+6n)(k+1)n + [(5+6n)n(k+1) + 5n]n^2 2^{n-1}\}\|A\|_\infty \mathbf{u} \\ &\le (k+2)n(5+6n)(1+n^2 2^{n-1})\|A\|_\infty \mathbf{u}, \end{aligned}$$

as required. □

B. Appendix B. We start by stating two results on the rounding error due to Householder reduction. These results are similar to those in Lawson and Hanson [10, pp. 85–89] and Wilkinson [20, pp. 157–162]. They differ from Lawson and Hanson's results in that the $O(\mathbf{u}^2)$ term is explicitly accounted for at every stage, and from Wilkinson's in that we do not assume double-precision accumulation of inner products. Since the proofs are tedious and do not offer any new insight, they are omitted.

LEMMA B.1. *Suppose that an $m_1 \times m_2$ matrix X is multiplied by an $m_1 \times m_1$ Householder reflector Q. Then, provided that*

$$(6m_1 + 18)\mathbf{u} \le 0.1,$$

the computed result Y satisfies

$$Y = Q(X+E),$$

where

$$\|E\|_F \le (7m_1 + 42)\mathbf{u}\|X\|_F.$$

LEMMA B.2. *If Q is a product of r Householder reflectors whose effect is to introduce zeros into the subdiagonals in the first r columns of the $m_1 \times m_2$ matrix X, then provided that*

$$(7m_1 + 42)r\mathbf{u} \le 0.1,$$

the computed result Y satisfies

$$Y = Q(X + E),$$

where

$$\|E\|_F \leq (8m_1 - 4r + 51)r\mathbf{u}\|X\|_F.$$

For the purpose of this Appendix, it is simplest to view the structured factorization process as the application of $k - 1$ orthogonal transformation matrices $Q_1, Q_2, \cdots, Q_{k-1}$ to a row- and column-reordered version of A_D (and the right-hand side f_D), followed by the application of another two matrices Q_k and Q_{k+1} to effect the final reduction of (25). (Q_k and Q_{k+1} reduce the first and last n columns of the coefficient matrix in (25), respectively.) Each of the Q_j, $j = 1, \cdots, k-1$ are products of n Householder reflectors, and each operates on only a small part of the matrix that it multiplies: to be precise, a $2n \times 3n$ submatrix. Since we wish to reduce (7) to (25), it follows that exactly $k - 1$ such transformations are needed.

Proof (Lemma 4.1). Let $\hat{A}_{D,i+1}$ be the transformed version of A_D after i stages of the structured factorization, and $\hat{A}_{D,1} = A_D$. Let $X_{D,i}$ be the submatrix that is actually affected at stage i by the matrix Q_i from (22), and let $\hat{Q}_i$ be the orthogonal matrix that is obtained by embedding Q_i into a $(k+1)n$-dimensional identity matrix. For $i = 1, \cdots, k+1$, we have from Lemma B.2 that $\hat{A}_{D,i}$ and $\hat{A}_{D,i+1}$ are related by

$$\hat{A}_{D,i+1} = \hat{Q}_i(\hat{A}_{D,i} + E_i),$$

where E_i consists of the $2n \times 3n$ "error" submatrix corresponding to the factorization of $X_{D,i}$, padded out with zeros to dimension $(k+1)n$. Hence

$$\|E_i\|_F \leq (8(2n) - 4n + 51)n\mathbf{u}\|X_{D,i}\|_F \leq (12n + 51)n\mathbf{u}\|\hat{A}_{D,i}\|_F.$$

Hence

$$\|\hat{A}_{D,i+1}\|_F \leq [1 + (12n + 51)n\mathbf{u}]\|\hat{A}_{D,i}\|_F \leq [1 + (12n + 51)n\mathbf{u}]^i\|A_D\|_F.$$

The errors made in stages k and $k + 1$ are bounded in the same way, since the submatrices affected at these stages are no larger than those affected at the earlier stages. Under the assumption $(12n + 51)n(k + 1)\mathbf{u} \leq 0.1$, we therefore obtain

$$\|\hat{A}_{D,i+1}\|_F \leq [1 + (1.06)(12n + 51)n(k + 1)\mathbf{u}]\|A_D\|_F \leq (1.106)\|A_D\|_F,$$

for $i = 1, \cdots, k + 1$. Hence

$$\|E_i\|_F \leq (1.106)(12n + 51)n\mathbf{u}\|A_D\|_F.$$

The final (upper triangular) matrix is $\hat{A}_{D,k+2}$, which satisfies

$$\begin{aligned}\hat{A}_{D,k+2} &= \hat{Q}_{k+1}(\hat{A}_{D,k+1} + E_{k+1}) \\ &= \hat{Q}_{k+1}\hat{Q}_k(\hat{A}_{D,k} + E_k) + \hat{Q}_{k+1}E_{k+1} \\ &\vdots \\ &= \hat{Q}_{k+1}\cdots\hat{Q}_1 A_D + \hat{Q}_{k+1}\cdots\hat{Q}_1 E_1 + \hat{Q}_{k+1}\cdots\hat{Q}_2 E_2 + \cdots + \hat{Q}_{k+1}E_{k+1} \\ &= \hat{Q}[A_D + E_D].\end{aligned}$$

Here $\hat{Q} = \hat{Q}_{k+1}\hat{Q}_k \cdots \hat{Q}_1$ is orthogonal, and E_D satisfies the bound

$$\|E_D\|_F \le \sum_{i=1}^{k+1} \|E_i\|_F = (1.106)(12n + 51)n(k+1)\mathbf{u}\|A_D\|_F.$$

Similarly, application of $\hat{Q}$ to the right-hand side f_D, results in a computed vector $\hat{f}_D$, which satisfies

$$\hat{f}_D = \hat{Q}[f_D + \delta f_D],$$

where

$$\|\delta f_D\|_2 \le (1.106)(12n + 51)n(k+1)\mathbf{u}\|f_D\|_2$$

(since $\|v\|_F = \|v\|_2$ when v is a vector). Finally, back-substitution is used on the system with coefficient matrix $\hat{A}_{D,k+2}$ and right-hand side $\hat{f}_D$. The computed solution satisfies

$$(\hat{A}_{D,k+2} + E_S)\hat{s} = \hat{f}_D, \tag{45}$$

where, since $\hat{A}_{D,k+2}$ has at most $3n$ nonzeros per row,

$$\begin{aligned}\|E_S\|_F &\le 3(1.06)n\mathbf{u}\|\hat{A}_{D,k+2}\|_F \\ &\le 3(1.06)n\mathbf{u}[1 + (1.106)(12n+51)n(k+1)\mathbf{u}]\|A_D\|_F \\ &\le 3(1.06)(1.1106)n\mathbf{u}\|A_D\|_F.\end{aligned}$$

Substituting in (45),

$$\begin{aligned}\left[\hat{Q}[A_D + E_D] + E_S\right]\hat{s} &= \hat{Q}[f_D + \delta f_D] \\ \Rightarrow (A_D + E_D + \hat{Q}^T E_S)\hat{s} &= (f_D + \delta f_D).\end{aligned}$$

Defining

$$\delta A_D = E_D + \hat{Q}^T E_S,$$

we have

$$\begin{aligned}\|\delta A_D\|_F &\le \|E_D\|_F + \|E_S\|_F \\ &\le [(1.106)(12n+51)n(k+1) + 3(1.06)(1.1106)n]\,\mathbf{u}\|A_D\|_F \\ &\le (1.106)(12n+51)n(k+2)\mathbf{u}\|A_D\|_F,\end{aligned}$$

as required. □

REFERENCES

[1] U. M. Ascher and P. S. Y. Chan, *On parallel methods for boundary value* ODE*s*, Computing, 46 (1991), pp. 1–17.

[2] U. M. Ascher and R. M. M. Mattheij, *General framework, stability and error analysis for numerical stiff boundary value problems*, Numer. Math., 54 (1988), pp. 355–372.

[3] U. M. Ascher, R. M. M. Mattheij, and R. D. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[4] Z. Bohte, *Bounds for rounding errors in the Gaussian elimination for band systems*, J. Inst. Math. Appls., 16 (1975), pp. 133–142.

[5] D. L. Brown and J. Lorenz, *A high order method for stiff boundary value problems with turning points*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 790–805.

[6] A. K. Cline, C. B. Moler, G. W. Stewart, and J. H. Wilkinson, *An estimate for the condition number of a matrix*, SIAM J. Numer. Anal., 16 (1979), pp. 368–375.

[7] J. C. Diaz, A. Fairweather, and P. Keast, FORTRAN *packages for solving certain almost block diagonal linear systems by modified alternate row and column elimination*, ACM Trans. Math. Software, 9 (1983), pp. 358–375.

[8] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Second Edition, The Johns Hopkins University Press, Baltimore, MD, 1989.

[9] H. B. Keller, *Accurate difference methods for two-point boundary value problems*, SIAM J. Numer. Anal., 11 (1974), pp. 305–320.

[10] C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[11] M. Lentini, *Parallel solution of special large block tridiagonal systems:* TPBVP, manuscript, 1989.

[12] M. Lentini, M. R. Osborne, and R. D. Russell, *The close relationships between methods for solving two-point boundary value problems*, SIAM J. Numer. Anal., 22 (1985), pp. 280–309.

[13] M. Lentini and V. Pereyra, *An adaptive finite difference solver for nonlinear two-point boundary value problems with mild boundary layers*, SIAM J. Numer. Anal., 14 (1977), pp. 91–111.

[14] R. M. M. Mattheij, *The conditioning of linear boundary value problems*, SIAM J. Numer. Anal., 19 (1982), pp. 963–978.

[15] ———, *Decoupling and stability of algorithms for boundary value problems*, SIAM Rev., 27 (1985), pp. 1–44.

[16] M. R. Osborne, *Aspects of the numerical solution of boundary value problems with separated boundary conditions*, manuscript, Computer Research Group, Australian National University, Canberra, Australia, 1978.

[17] M. Paprzycki and I. Gladwell, *Solving almost block diagonal systems on parallel computers*, Parallel Comput., 17 (1991), pp. 133–153.

[18] V. Pereyra, *Iterated deferred corrections for nonlinear boundary value problems*, Numer. Math., 8 (1968), pp. 111–125.

[19] J. M. Varah, *Alternate row and column elimination for solving certain linear systems*, SIAM J. Numer. Anal., 13 (1976), pp. 71–75.

[20] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford University Press, London, U.K., 1965.

[21] S. J. Wright and V. Pereyra, *Adaptation of a two-point boundary value problem solver to a vector-multiprocessor environment*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 425–449.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 3, pp. 765–770, May 1992

009

THREE-DIMENSIONAL DELAUNAY TRIANGULATIONS FOR FINITE ELEMENT APPROXIMATIONS TO A SECOND-ORDER DIFFUSION OPERATOR*

FRANK W. LETNIOWSKI†

Abstract. The second-order diffusion operator given by $\mathcal{L}u = \nabla \cdot (\mathbf{K}\nabla u)$ is studied in terms of finite element numerical solutions. When a standard Galerkin finite element approximation of the operator is arranged in a specific manner, a condition of positive connection values is imposed that is necessary to produce an M-matrix in a linear system when boundary nodes are properly handled. In two dimensions, the condition is achieved when a Delaunay triangulation is imposed. However, it is shown that a three-dimensional Delaunay triangulation does not generally produce a discretisation satisfying the condition. Further, it is generally not possible to produce a three-dimensional triangulation that satisfies the positive interior connection condition.

Key words. finite element method, three-dimensional triangulation, Delaunay triangulation

AMS(MOS) subject classification. 65N30

1. Introduction.

Three-dimensional triangulations have been studied frequently in the context of finite element discretisations [1], [3], [12], [15]. A set of tetrahedra are selected to cover a polyhedron that approximates a region. A common triangulation approach is the three-dimensional Delaunay triangulation [8], [2], [14], where the tetrahedra satisfy a sphere criterion.

This paper considers the effect of triangulation on the finite element approximation of the second-order diffusion operator. Example areas of application include groundwater contamination problems, petroleum reservoir simulation, heat transfer, semiconductor device modelling, and Navier–Stokes fluid flow.

For a dependent variable u, the action of the diffusion operator, $\mathcal{L}$, is defined by

$$\mathcal{L}u = \nabla \cdot (\mathbf{K}\nabla u), \tag{1}$$

where in general $\mathbf{K}$ is possibly nonconstant and a tensor. The assumption in this paper is that $\mathbf{K}$ is symmetric positive definite with constant components.

2. Galerkin finite element discretisation.

The operator $\mathcal{L}$ is approximated using the Galerkin finite element method.

Let N_i be a $\mathcal{C}^0$ linear basis function for the ith node of the discretisation. In the usual way, the unknown u is approximated as

$$u \approx \sum_j u_j N_j := \hat{u}. \tag{2}$$

Define

$$v_i = \{j \text{ s.t. node } j \text{ is adjacent to node } i\}.$$

Consequently,

$$(\mathcal{L}\hat{u}, N_i) = \sum_{j \in v_i} \Gamma_{ij}(u_j - u_i), \tag{3}$$

* Received by the editors October 22, 1990; accepted for publication (in revised form) February 5, 1991. This work was supported by the Natural Sciences and Engineering Research Council of Canada through a postgraduate scholarship, and by the Information Technology Research Centre, which is funded by the province of Ontario.

† NOVA Corporation of Alberta, P.O. Box 2535, Station M, Calgary, Alberta, Canada T2P 2N6.

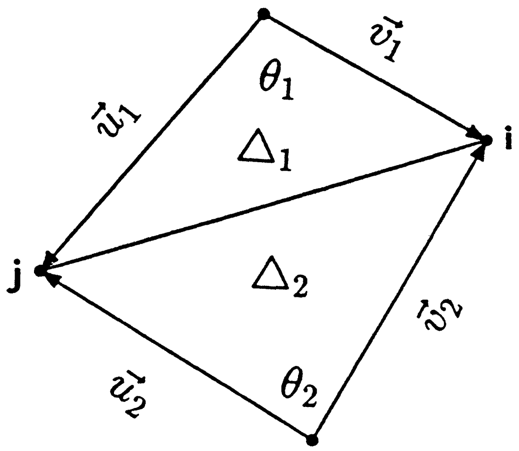


FIG. 1. *Interior connection for triangular elements.*

where

$$\Gamma_{ij} = -\int_R (\mathbf{K}\nabla N_j) \cdot \nabla N_i \, dV, \tag{4}$$

which will be called the *connection value* for the connection between nodes i and j. We have assumed either Dirichlet or Neumann boundary conditions on ∂R.

Define the matrix A such that

$$\begin{aligned} A_{ij} &= \Gamma_{ij}, \qquad i \neq j, \\ A_{ii} &= -\sum_{j \neq i} \Gamma_{ij}. \end{aligned} \tag{5}$$

When Γ_{ij} are positive, the matrix A is an M-matrix when there is at least one Dirichlet node. It is well known that with an M-matrix, the approximation to $\mathcal{L}$ satisfies a discrete maximum principle. An M-matrix is also desirable for iterative sparse matrix methods [11].

Without loss of generality, we perform a linear transformation and set $\mathbf{K}$ to the identity matrix for the remainder of this paper.

3. Two dimensions. In two dimensions, it is always possible to achieve positive interior connection values under the above assumptions. A connection between two nodes is called *interior* if the line segment joining the two nodes does not form part of the boundary.

For triangular elements, an interior connection is shared by exactly two triangles (see Fig. 1). The value of the connection Γ_{ij} is [5]

$$\Gamma_{ij} = \tfrac{1}{2}[\cot\theta_1 + \cot\theta_2]; \tag{6}$$

hence the requirement for positivity is

$$\theta_1 + \theta_2 \leq \pi. \tag{7}$$

A triangulation that satisfies condition (7) is the Delaunay triangulation [5], [7], [6], which satisfies the *circle criterion*: the circumcircle of each triangle in the triangulation contains no nodes of the discretisation in its interior.

Any valid triangulation of a two-dimensional convex region may be transformed into a Delaunay triangulation through a series of local transformations [7], [6]. This series of transformations must converge in a finite number of steps. The local transformation procedure is an *edge swap*, which is described in [9].

In summary, the following theorem holds.

THEOREM 3.1. *In the transformed region, a Delaunay triangulation of a two-dimensional convex region satisfies the positive interior connection value criterion.*

A connection that is situated along the boundary of the region is the edge of only one triangular element. In this case, if the connection value is negative, it may be necessary to add boundary nodes [4] to ensure that (3) produces an M-matrix.

4. Three dimensions. It is natural to speculate that a three-dimensional Delaunay triangulation satisfies the criterion of positive interior connection values; however, the theorem and corollary in this section demonstrate that this is not true in general.

It is well known that any polyhedron may be divided into a finite number of tetrahedra. Such a discretisation is called a three-dimensional triangulation.

One possible triangulation for a set of nodes is the three-dimensional Delaunay triangulation [8], which has been used for three-dimensional mesh generation [1], [3], [12], [15]. The tetrahedra in the discretisation satisfy the *sphere condition*: the circumsphere of the four vertices of any tetrahedron in the triangulation contains no vertices in its interior. The local transformation that changes a local triangulation into one satisfying the sphere condition is a face swap, which is described in [8].

THEOREM 4.1. *In general, a three-dimensional triangulation satisfying the positive interior connection criterion may not be constructed using strictly local transformations.*

Proof. Consider the local region defined by the six three-dimensional points:

$$\begin{aligned} \mathsf{A} &= (-2,-2,0.5),\\ \mathsf{B} &= (0,-2,0.1),\\ \mathsf{C} &= (-2,0,0.1),\\ \mathsf{D} &= (0,0.1,0),\\ \mathsf{E} &= (-2,-2,-0.25),\\ \mathsf{F} &= (-2,-2,1.5). \end{aligned}$$

Consider Figs. 2, 3, and 4, which are the only possible triangulations of the above six points from the transformations described in [8]. The triangulations in each figure produce the following tetrahedra:

Fig. 2: ABDE, ABDF, ACDE, and ACDF;
Fig. 3: ABCF, BCDF, ABCE, and BCDE;
Fig. 4: ABDF, ACDF, ABCE, BCDE, and ABCD.

Table 1 gives the values for the interior connections, AD and BC, for each of the figures. Each of the connection values is negative, which breaches the positive connection criterion. Note also that any tetrahedron that might exist exterior to this local region cannot have AD or BC as one of its edges; that is, all tetrahedra affecting the connection AD or BC are included in the figures. □

COROLLARY 4.2. *A three-dimensional Delaunay triangulation may not satisfy the condition of positive interior connections.*

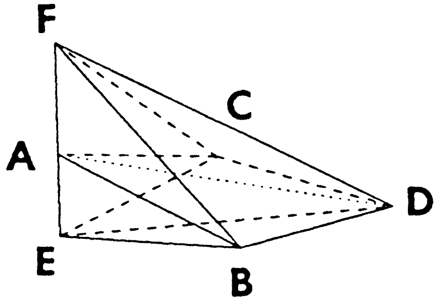


FIG. 2. *Sample triangulation with* AD *connection.*

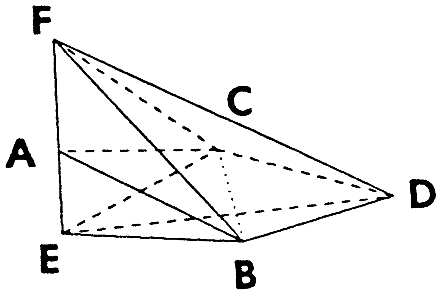


FIG. 3. *Sample triangulation with* BC *connection.*

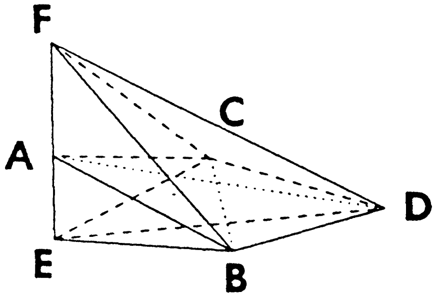


FIG. 4. *Sample Delaunay triangulation.*

TABLE 1
Interior connections for the triangulation example.

	Connection	Γ_{ij}
Fig. 2	AD	−0.01458
Fig. 3	BC	−2.0027
Fig. 4	AD	−2.208
	BC	−1.4694

TABLE 2
Circumcentres for the triangulation example.

Tetrahedron	Circumcentre	Sphere condition?
ABDE	$(-1.035, -0.9464, 0.125)$	no
ACDE	$(-0.942, -1.035, 0.125)$	no
ABDF	$(-0.86, -0.9048, 1.0)$	yes
ACDF	$(-0.907, -0.86, 1.0)$	yes
ABCD	$(-0.919, -0.919, 0.706)$	yes
ABCF	$(-0.86, -0.86, 1.0)$	no
DBCF	$(-0.908, -0.908, 0.931)$	no
ABCE	$(-1.035, -1.035, 0.125)$	yes
DBCE	$(-0.965, -0.965, -0.273)$	yes

Proof. Table 2 lists the nine tetrahedra in Figs. 2–4, gives the coordinates of their circumcentres, and indicates whether or not the tetrahedra satisfy the sphere condition. The table identifies Fig. 4 as the Delaunay triangulation; however, from Table 1 it is seen that the interior connections AD and BC of this triangulation are negative. □

Note that from Table 1, the sizes of the negative connections AD and BC of Figs. 3 and 4 are on the order of 100 times larger than the connection AD given in Fig. 2. In a specific example given in [10], it is determined that the size of negative connections may be more important than the number of negative connections. Thus, *the Delaunay triangulation is not, in general, the best triangulation for a three-dimensional finite element grid, in terms of the positive connection criterion.*

5. Conclusions. In this paper, a condition of positive connection values was imposed. This condition produces an M-matrix, and hence a discrete maximum principle, for a linear diffusion problem when there is at least one Dirichlet node. In two dimensions, this condition may be satisfied by selecting a Delaunay triangulation when boundary nodes are handled properly. It is shown by counterexample that a three-dimensional Delaunay triangulation does not, in general, satisfy the condition.

Moreover, this paper illustrates that, in three dimensions, a triangulation producing positive interior connections may not, in general, be obtained with strictly local transformations. This suggests that, in three dimensions, it may be very difficult to produce a triangulation that always gives an M-matrix, and hence a maximum principle, for a linear diffusion problem with at least one Dirichlet node.

Acknowledgement. The author would like to thank Professor P. A. Forsyth for discussions and guidance during the preparation of this document. I would also like to thank the referees for their helpful comments.

REFERENCES

[1] T. J. BAKER, *Development and trends in three-dimensional mesh generation,* Appl. Numer. Math., 5 (1989), pp. 275–304.

[2] A. BOWYER, *Computing Dirichlet tessellations,* Comput. J., 24 (1981), pp. 162–166.

[3] J. C. CAVENDISH, D. A. FIELD, AND W. H. FREY, *An approach to automatic three-dimensional finite element mesh generation,* Internat. J. Numer. Methods Engrg., 21(1985), pp. 329–347.

[4] P. A. FORSYTH, *A control volume finite element approach to NAPL groundwater contamination,* SIAM J. Sci. Statist. Comput., 12 (1991), pp. 1029–1057.

[5] P. A. Forsyth, *A control volume finite element method for local mesh refinement in thermal reservoir simulation,* Paper SPE 18415, Tenth Society of Petroleum Engineers Symposium on Reservoir Simulation, Houston, TX, 1989.

[6] B. Joe, *Finite element triangulation of complex regions using computational geometry,* Ph.D. thesis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 1984.

[7] ———, *Delaunay triangular meshes in convex polygons,* SIAM J. Sci. Statist. Comput., 7(1986), pp. 514–539.

[8] ———, *Three-dimensional triangulations from local transformations,* SIAM J. Sci. Statist. Comput., 10(1989), pp. 718–741.

[9] C. L. Lawson, *Transforming triangulations,* Discrete Math., 3(1972), pp. 365–372.

[10] F. W. Letniowski, *Numerical methods for nonaqueous phase liquid groundwater contamination in three dimensions,* Master's thesis, Department of Applied Mathematics, University of Waterloo, Waterloo, Ontario, Canada, 1989.

[11] ———, *An overview of preconditioned iterative methods for sparse matrix equations,* Res. Report CS-89-26, University of Waterloo, Waterloo, Ontario, Canada, 1989.

[12] V. Ph. Nguyen, *Automatic mesh generation with tetrahedron elements,* Internat. J. Numer. Methods Engrg., 18(1982), pp. 273–289.

[13] C. S. Rafferty, M. R. Pinto, and R. W. Dutton, *Iterative methods in semiconductor device simulation,* IEEE Trans. Comput. Aided Design, CAD-4(1985), pp. 462–471.

[14] D. F. Watson, *Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes,* Comput. J., 24(1981), pp. 167–172.

[15] M. A. Yerry and M. S. Shephard, *Automatic three-dimensional mesh generation by the modified-octree technique,* Internat. J. Numer. Methods Engrg., 20(1984), pp. 1965–1990.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 3, pp. 771–793, May 1992

010

A PARALLEL NONLINEAR LEAST-SQUARES SOLVER: THEORETICAL ANALYSIS AND NUMERICAL RESULTS*

THOMAS F. COLEMAN† AND PAUL E. PLASSMANN‡

Abstract. The authors recently proposed a new parallel algorithm, based on the sequential Levenberg–Marquardt method, for the nonlinear least-squares problem. The algorithm is suitable for message-passing multiprocessor computers.

In this paper a parallel efficiency analysis is provided and computational results are reported. The experiments were performed on an Intel iPSC/2 multiprocessor with 32 nodes: this paper presents experimental results comparing the given parallel algorithm with sequential MINPACK code executed on a single processor. These experimental results show that essentially full efficiency is obtained for problems where the row size is sufficiently larger than the number of processors.

Key words. hypercube computer, Levenberg–Marquardt, nonlinear least squares, message-passing multiprocessor, parallel algorithms, QR factorization, trust-region algorithms

AMS(MOS) subject classifications. 65H10, 65F05, 65K05, 65K10, 90C30

1. Introduction. Let $F : \mathbf{R}^n \mapsto \mathbf{R}^m$, with $m \geq n$, be a continuously differentiable function. The nonlinear least-squares problem is to find a local minimum of the function

$$\psi(x) = \frac{1}{2}\|F(x)\|_2^2 = \frac{1}{2}\sum_{i=1}^{m} f_i^2(x), \tag{1.1}$$

where f_i is the ith component of F. Recently, Coleman and Plassmann [CP89] proposed a parallel implementation of the well-known Levenberg–Marquardt algorithm [L44], [M63] for solving this problem when the Jacobian of $F(x)$ is dense.[1] In this paper we present a theoretical analysis of our parallel method, as well as experimental results obtained on an Intel iPSC/2 hypercube. The experimental results are obtained on a hypercube multiprocessor; however, we feel that the algorithm is not limited to this architecture. In fact, all that is required of the multiprocessor interconnection topology is support of a ring embedding and means for efficient gather and broadcast operations.

There are three main computational tasks that need to be addressed in a parallel implementation of the Levenberg–Marquardt algorithm [2]:

1. Evaluation or approximation of the Jacobian matrix $J(x)$.

* Received by the editors June 22, 1988; accepted for publication (in revised form) February 20, 1991.

† Computer Science Department and Center for Applied Mathematics, Cornell University, Ithaca, New York 14853. The research of this author was partially supported by the Applied Mathematical Sciences Research Program (KC-04-02) of the Office of Energy Research of the U.S. Department of Energy under grant DE-FG02-86ER25013.A000.

‡ Center for Applied Mathematics, 305 Sage Hall, Cornell University, Ithaca, New York 14853. Present address, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439. The research of this author was partially supported by the Computational Mathematics Program of the National Science Foundation under grant DMS-8706133 and by the U.S. Army Research Office through the Mathematical Sciences Institute, Cornell University.

[1] Plassmann [P90] considers the large sparse case.

[2] For a more detailed description of the Levenberg–Marquardt algorithm, including step acceptance and convergence criteria, we refer the interested reader to the excellent article by Moré [M78].

2. The QR factorization of $J(x)$,

$$J = Q\begin{bmatrix} R \\ 0 \end{bmatrix},$$

where Q is an orthogonal matrix and R is upper-triangular, in order to solve the least-squares problem

$$\begin{bmatrix} R \\ 0 \end{bmatrix} s \overset{\text{L.S.}}{=} -Q^T F. \tag{1.2}$$

3. The computation of the Levenberg–Marquardt parameter λ_*, and vector s_*, satisfying

$$(J^T J + \lambda_* D^T D)s_* = -J^T F, \tag{1.3}$$

such that $\|Ds_*\|_2 = \Delta$, where D is a diagonal scaling matrix and Δ is a positive scalar representing the "trust region" size. Computationally, this means solving least-squares systems of the form

$$\begin{bmatrix} R \\ 0 \\ \lambda^{1/2} D \end{bmatrix} s(\lambda) \overset{\text{L.S.}}{=} -\begin{bmatrix} Q^T F \\ 0 \end{bmatrix} \tag{1.4}$$

for different values of λ.

We address these computational issues in the remainder of the paper. In §2 we summarize the issues involved with respect to the parallel finite-difference approximation of the Jacobian matrix. In §3 we summarize the row-oriented parallel QR factorization proposed in [CP89], provide a new complexity analysis, and present numerical results. A theoretical analysis of the parallel algorithm for determining the Levenberg–Marquardt parameter is given in §4, along with numerical results. Finally, we present experimental results for the entire method and conclusions in §5.

2. Parallel approximation of the Jacobian. It is often the case that the number of rows of the Jacobian is much larger than the number of columns. For the QR factorization stage this suggests a row-oriented method, where the rows of the Jacobian are distributed to processors. This data distribution achieves a better load balancing than a column-oriented method, and results in an algorithm whose efficiency depends on the ratio m/p rather than n/p, where p is the number of processors. Experience has shown that computational costs involved in the QR factorization stage often dominate the Jacobian approximation stage. Thus, we have chosen to pursue a row-oriented QR factorization algorithm.[3] We would like to take advantage of this data distribution in approximating the Jacobian whenever possible. Let I_i, $i = 1, \cdots, p$, be a partition of the rows of J, where I_i is the set of row indices assigned to processor i and let $F_{I_i}(x) = \{f_j(x) \mid j \in I_i\}$ be the corresponding function blocks. We

[3] If a column-oriented Jacobian approximation scheme is used, one must convert this column-oriented data distribution into a row-oriented distribution for the QR factorization stage [JH88], [MVV87], [SS85]. Of course, for sufficiently large n this problem can be avoided by using a column-oriented QR factorization algorithm. We did not take such an approach because it is usually the case that $m \gg n$. However, there exist good column-oriented QR factorization algorithms [B88], [CG88], [M87], and in §4 we describe an efficient column-oriented algorithm for determining the Levenberg–Marquardt parameter.

say that the function F is *block separable* if there exists a partition of the rows such that the evaluation of each function block is computationally independent.

Suppose the function is block separable relative to the partition I_i, $i = 1, \cdots, p$ and let $J_{I_i}^T(x)$ be the set of the rows of the Jacobian estimated at the point x. The jth column of the Jacobian can be estimated in parallel by having each processor compute its block of row components according to the formula

$$J_{I_i}^T(x)e_j \cong \frac{F_{I_i}(x + \tau e_j) - F_{I_i}(x)}{\tau}. \tag{2.1}$$

However, often the evaluation of $F(x)$ is not completely separable; there may be some amount of redundant computation due to common factors that must be computed for each partition of the function $F_{I_i}(x)$, $i = 1, \cdots, p$. If this redundant computation is inexpensive relative to communication cost entailed by using a column-oriented scheme, then we consider this computational overhead tolerable. All of the test problems considered in the experimental section fall into this category. Otherwise, if the redundant computation required by such a partition of the rows is deemed too expensive, a column-oriented approach to approximating $J(x)$ must be adopted.

A subtle problem occurs when n/p is small, the evaluation of $F(x)$ is expensive and not separable, and therefore the estimation of the Jacobian is computationally expensive relative to the QR factorization. Suppose a step $s^{(k)}$ is to be considered at the kth iteration of the algorithm; $F(x^{(k)} + s^{(k)})$ must be evaluated to determine if it meets certain acceptance criteria. When this computation is relatively expensive and not separable, and therefore must be done on one processor, then the remainder of the processors will remain idle during this computation. This can result in detrimental effects on the efficiency of the entire implementation. Byrd, Schnabel, and Shultz [BSS88] and Coleman and Li [CL87] note that this problem can be alleviated somewhat by guessing, based on the previous iteration, whether the proposed point will be accepted. If acceptance is assumed, the Jacobian at $x^{(k)} + s^{(k)}$ can begin to be approximated by idle processors. If we guess that the proposed iterate will not be accepted, then idle processors could evaluate the function at some additional points that might fare better with the acceptance criteria. These ideas were not implemented in our code, but could easily be added. Nevertheless, for $n/p \gg 1$, the computation required to estimate the Jacobian will always dominate these isolated function evaluations.

3. A parallel row-oriented Householder QR algorithm.

In this section we analyze and experiment with the parallel row-oriented Householder QR factorization proposed in [CP89]. We show that this algorithm is more efficient than previous hybrid (Householder/Givens) factorization algorithms. The efficiency of the parallel QR factorization used to solve (1.2) is of paramount importance because a completely new approximation to the Jacobian is computed for each iteration. Consequently, a full QR factorization is also required. For the test problems considered in this paper, we find that the QR factorization is always a major (and sometimes the dominant) computational cost. An additional advantage of this algorithm is that, unlike the hybrid scheme, it produces the same Householder vectors that would be produced by a standard sequential Householder QR algorithm. This property is advantageous in situations where the same system must be solved for multiple right-hand sides. Finally, we show that column pivoting can be introduced into the algorithm with only a slight increase in the computation and communication complexity. In our implementation column pivoting is important because the QR factorization can then be used to estimate matrix rank.

Column-oriented methods have dominated the work on parallel QR algorithms; however, two row-oriented algorithms have been considered previously [CP86], [PR87]. These two algorithms are very similar: to reduce each column of the matrix a reduction involving only data local to each processor is performed, followed by a global reduction requiring communication between the processors. The reduction of rows local to a processor yields one row per processor with a nonzero in the column being reduced to upper-triangular form. This approach has the advantage that all these reductions and matrix updates will be local to the processors, and with the wrap mapping[4] of rows the computational load will be well balanced. Following this local stage is a global stage: a minimum-depth spanning tree is embedded in the hypercube, rooted at the processor where the nonzero for the column under consideration should reside. Rows are communicated up this tree and the leading nonzero is annihilated by a Givens rotation with respect to the parent's row. These rows are then updated with this rotation and the result communicated back to the child. The hypercube topology allows this global reduction process to take place in $\log(p)$ steps. Of these two algorithms, the one presented by Pothen and Raghavan [PR87] seems to be the most efficient, since Householder reductions, as opposed to Givens, are used in the local stage.

Our algorithm is computationally more efficient than the hybrid approach: the full Householder vector is calculated and the intermediate Givens reductions are avoided. However, our challenge is to obtain the same communication complexity as the hybrid approach. We meet this challenge by noticing that computation of the Householder vector and the subsequent rank-one update to the matrix can be combined to halve the number of messages that seem to be required at first glance.

To review the algorithm given in [CP89], consider the QR factorization of an $m \times n$ matrix A. At step j of the factorization, the first $j-1$ rows of R and the Householder vectors have been computed; we need only consider the $(m-j+1) \times (n-j+1)$ lower right submatrix of A, denoted by $A^{(j)}$, with columns $a_k^{(j)}$, $k = j, \cdots, n$. The Householder transformation, $P^{(j)}$, to reduce the first column of $A^{(j)}$, $a_j^{(j)}$, is

$$P^{(j)} = \left[I - 2\frac{v_{(j)} v_{(j)}^T}{v_{(j)}^T v_{(j)}} \right], \tag{3.1}$$

where $v_{(j)} = a_j^{(j)} \pm \|a_j^{(j)}\|_2 e_j$. To determine $a_k^{(j+1)}$, $k = j+1, \cdots, n$, we need to compute the corresponding rank-one update to $A^{(j)}$:

$$\begin{aligned} a_k^{(j+1)} &= a_k^{(j)} - \frac{2}{v_{(j)}^T v_{(j)}} v_{(j)}^T a_k^{(j)} v_{(j)} \\ &= a_k^{(j)} - \alpha_k^{(j)} v_{(j)}, \end{aligned} \tag{3.2}$$

with $\alpha_k^{(j)}$ defined as shown. Let *leader* designate the processor that holds row j. Note that $v_{(j)}$ agrees with $a^{(j)}$ except in the first component. Therefore, the portions of the inner product $v_{(j)}^T a_k^{(j)}$ local to each processor are just $a_j^{(j)T} a_k^{(j)}$ except on *leader*, where $a^{(j)}$ and $v_{(j)}$ differ in the first component. We can take advantage of this fact

[4] The specific row-oriented distribution we consider is a wrapping of rows onto processors. Thus, if the processors on the ring are numbered $0, 1, 2, \cdots, p-1$, then row k of the Jacobian is assigned to processor $(k-1) \mathrm{mod}(p)$.

and combine the communication to compute $v_{(j)}$ with the communication required for the rank-one update to the remainder of the matrix. An outline of the resulting algorithm is given as Algorithm 3.1. For this description we use the notation $[a_j^{(j)}]_{I_i}$ to represent the subvector of $a_j^{(j)}$ with components given by the index set I_i. The $\tilde{\alpha}$ vector is a work vector used in the computation of $\|a_j^{(j)}\|_2$ and the constants $\alpha_k^{(j)}$, $k = j+1, \cdots, n$.

```
Index Set: $I_i$ {set of row indices assigned to processor $i$}

Proc ($i$) : {program for processor $i$}
    For $j = 1, \cdots, n$ do
        If ($i$ =leader) Delete $\{j\}$ from $I_i$;
        Compute dot products for $k = j, \cdots, n$
            $\tilde{\alpha}_k = [a_j^{(j)}]_{I_i}^T [a_k^{(j)}]_{I_i}$;
        Combine $[\tilde{\alpha}_j, \cdots, \tilde{\alpha}_n]$ using gather-sum;
        If ($i$ = leader) then
            Compute first component of $v^{(j)}$ and the coefficients
                $[\alpha_{j+1}^{(j)}, \cdots, \alpha_n^{(j)}]$ and broadcast the result;
        endif
        Update columns, $k = j+1, \cdots, n$
            $[a_k^{(j+1)}]_{I_i} = [a_k^{(j)}]_{I_i} - \alpha_k^{(j)} [v^{(j)}]_{I_i}$;
    enddo
```

ALGORITHM 3.1. *A parallel row-oriented Householder* QR *algorithm.*

In Fig. 3.1 we exhibit the efficiencies of this algorithm compared to the hybrid algorithm described by Pothen and Raghavan in [PR87] as a function of the number of rows. (The data points in the figure are experimental results obtained on a 32 node iPSC/2 hypercube with 4.5 Mbytes of memory per node. All the experimental results presented in this paper are from implementations done on this machine.) The dotted lines in the figure are plots of a theoretical model of the efficiencies of the algorithms that will be presented later in this section. For this plot the number of columns is fixed at 100. The efficiencies shown were calculated by dividing the time taken by an efficient sequential implementation of the algorithm run on one processor by the product of the time taken by the parallel implementation and the number of processors used. In this case our parallel implementations were compared with the MINPACK QR factorization subroutine QRFAC executed on a single processor of the hypercube, and the efficiencies shown were computed from the execution times of these programs. In Table 3.1 we show some representative execution times for our implementations of the hybrid algorithm (Hybrid) and Algorithm 3.1, as compared to the sequential QR factorization program (Single processor).

There is a subtle point in solving (1.2): the orthogonal matrix Q does not need to be saved if the right-hand side of the equation is updated along with the rows of the Jacobian. To achieve this goal, the right-hand side is treated like an additional column to the matrix J, and is distributed across the processors in the same wrap mapping and updated along with the corresponding rows of the Jacobian.

Column pivoting can be added to Algorithm 3.1. The column norms of the

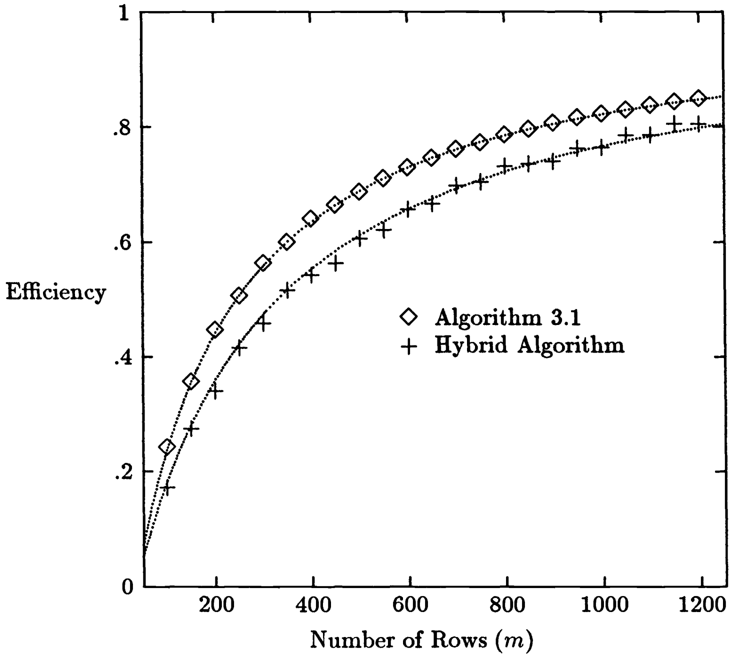


FIG. 3.1. *Efficiencies of Algorithm* 3.1 *and Hybrid on the iPSC/2* ($n = 100, p = 32$).

matrix A are initialized at the beginning. They are updated after each stage of the computation to obtain the column norms of $A^{(j)}$. For example, suppose at stage j the column norms $\|a_k^{(j)}\|_2$, $k = 1, \cdots, n$, are known by *leader*. The column of maximum norm, $k_{\max}$, is determined by *leader* and the result is broadcast. Columns j and $k_{\max}$ are then interchanged by all processors. After stage j of the algorithm the updated norms can be obtained from the formula

$$\|a_k^{(j+1)}\|_2 = \|a_k^{(j)}\|_2 \left(1 - \left(\frac{[a_k^{(j+1)}]_j}{\|a_k^{(j)}\|_2}\right)^2\right)^{1/2}, \tag{3.3}$$

for $k = j{+}1, \cdots, n$. The results are then sent to the next leader (i.e., the next processor on the ring) for stage $j + 1$ of the QR algorithm. Note that numerical cancellation is a potential problem in computing these norm updates. However, circumstances that would result in this problem can be monitored and the suspect column norm can be recomputed. A standard way to monitor for numerical cancellation is to keep track of the products of the multiplicative factors in (3.3) that have been obtained since the last explicit calculation of the column norm. When this product is sufficiently less than one, then there is the possibility of cancellation error, and the column norm is recomputed. In our implementation the recomputation is done by broadcasting a special notifier to the other processors instead of the column pivot. The required column norms are then recomputed and the result gathered at *leader*. Our observation has been that recomputation of the column norms is rarely required and therefore does not significantly affect the efficiency of the algorithm.

TABLE 3.1
Execution times of Algorithm 3.1 *and Hybrid on the iPSC/2 hypercube.*

Execution times (sec) without pivoting					
n	m	Single processor	p	Hybrid	Algorithm 3.1
100	200	31.60	8	5.17	4.86
			16	3.60	3.08
			32	2.98	2.27
100	400	69.44	8	9.89	9.64
			16	5.86	5.45
			32	4.11	3.46
100	800	145.28	8	19.41	19.20
			16	10.59	10.22
			32	6.38	5.84
100	1600	299.86	8	39.21	39.07
			16	20.47	20.15
			32	11.25	10.76
200	400	246.85	8	35.02	34.12
			16	20.44	19.01
			32	14.10	11.77
200	800	543.14	8	73.01	72.18
			16	39.09	37.82
			32	23.01	21.12
400	400	775.93	8	113.11	109.32
			16	66.47	61.04
			32	46.81	37.97

Another potential concern for numerical stability might be the possibility of overflow from the way the $\tilde{\alpha}_k$ are computed in Algorithm 3.1. We note that these partial sums can be scaled by the most recent approximation to the column norms available to all the processors. We did not find it necessary to include this scaling in our implementation.

Figure 3.2 exhibits a graph comparing the efficiencies of Algorithm 3.1 with and without pivoting. In Table 3.2 we include some representative times from these experiments. The efficiencies are again computed by comparing the running times of the parallel algorithms to running times of the MINPACK QR subroutine QRFAC on a single processor. As before, these results were obtained on a 32 node iPSC/2 hypercube with the number of columns fixed at 100. The data points are the experimental results and the dotted curves are theoretical approximations to these efficiencies, which we will now describe.

The efficiencies observed for the row-oriented Householder algorithm can be explained by a simple model for the communication overhead involved and consideration of computational imbalances between the processors. The efficiency is computed by the formula

$$\text{efficiency} = \frac{t_{\text{seq}}}{p\, t_{\text{parallel}}}, \tag{3.4}$$

where t_{seq} is the execution time of the sequential algorithm and t_{parallel} is the execution time of the parallel algorithm on p processors. The parallel execution time can be considered to consist of three parts: (1) the optimal time, t_{seq}/p, (2) the computational imbalance relative to the optimal distribution of work, t_{comp}, and (3) the communication overhead demanded by the parallel algorithm, t_{comm}. Hence, we have

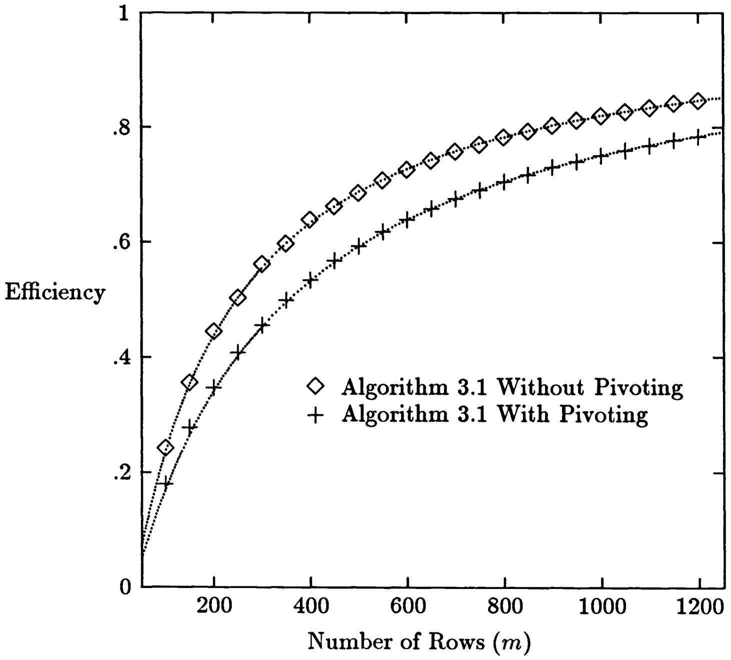


FIG. 3.2. *Efficiencies of Algorithm* 3.1 *with and without column pivoting* ($n = 100$, $p = 32$).

that

$$t_{\text{parallel}} = \frac{t_{\text{seq}}}{p} + t_{\text{comp}} + t_{\text{comm}} \tag{3.5}$$

and (3.4) can be rewritten as

$$\text{efficiency} = \frac{1}{1 + \frac{t_{\text{comm}} + t_{\text{comp}}}{t_{\text{seq}}} p}. \tag{3.6}$$

The sequential execution time of the Householder QR algorithm measured in old-style flops is

$$t_{\text{seq}} = n^2(m - n/3). \tag{3.7}$$

In the discussion that follows, we use (3.7) to define the length of time we consider to be one flop. On the iPSC/2 this time was experimentally determined to be 19.15μsec. However, this definition can be tricky since, for example, an add, multiply, or divide can take varying lengths of time to execute depending on how the code is written. Consequently, some of the coefficients in the following formulae had to be obtained experimentally and are not simple multiples of the above-defined flop.

To approximate the computational imbalance we consider two dominant terms. The first is due to the variation of the number of rows assigned to the processors, and the second term is due to the idle time of processors during the computation of the α's in Algorithm 3.1. Work is not quite equally distributed to the processors with the row wrapping. On the average, half the processors are assigned an

TABLE 3.2
Execution times of Algorithm 3.1 with pivoting on the iPSC/2 hypercube.

Execution times (sec) with pivoting				
n	m	Single processor	p	Algorithm 3.1
100	200	32.36	8	5.62
			16	3.82
			32	3.01
100	400	70.60	8	10.44
			16	6.22
			32	4.21
100	800	147.10	8	20.13
			16	11.03
			32	6.61
100	1600	303.11	8	40.07
			16	21.01
			32	11.56
200	400	249.97	8	36.88
			16	21.65
			32	14.36
200	800	547.77	8	75.15
			16	40.57
			32	23.77
400	400	785.60	8	119.11
			16	70.66
			32	47.52

extra row; hence, the remaining processors are idle during the portion of the Householder update corresponding to this extra row. The Householder update to a row of length k requires $2k$ flops, resulting in a computational imbalance over the entire factorization of $\frac{1}{2}\sum_{k=1}^{n} 2k = n^2/2$ flops. The total idle time of processors during the accumulations of sums in the computation of an $\tilde{\alpha}$-vector of length k is approximately $(\log(p) - 1)k\tau_{\text{add}}$, where τ_{add} is the time required for an add. Summing this expression from $k = 1$ to n yields $\frac{1}{2}(\log(p) - 1)n^2\tau_{\text{add}}$. Finally, the processor *leader* requires some length of time, say, β_1, to compute each element of the α-vector and time β_2 per element to update the column norms. These computations result in a total imbalance of $(\beta_1 + \beta_2)n^2/2$. Combining these contributions yields an approximation for t_{comp}, in flops, of

$$t_{\text{comp}} \approx (\beta_1 + \beta_2 + (\log(p) - 1)\tau_{\text{add}} + 1)n^2/2. \tag{3.8}$$

In our implementation, the times β_1 and β_2 were determined to be 22.4μsec and 110.6μsec and τ_{add} was found to be 11.2μsec.

The communication overhead for Algorithm 3.1 includes the time required for the accumulation and broadcast of the α-vectors. This overhead is $\sum_{k=1}^{n} 2\log(p)\tau(k)$, where $\tau(k)$ is the time required to send a double precision vector of length k between neighboring processors. An additional time of $n\log(p)\tau(1) + \sum_{k=1}^{n}\tau(k)$ is required to broadcast the pivot and transfer the column norms. Combining these two terms yields an approximation to the communication overhead of

$$t_{\text{comm}} \approx (2\log(p) + 1)T(n) + n\log(p)\tau(1), \tag{3.9}$$

where we define $T(n)$ to be $\sum_{k=1}^{n}\tau(k)$.

For the iPSC/2 the function $\tau(k)$ is, fortunately, empirically simple to describe; the cost function is essentially linear over large ranges of vector lengths k. Experimentally, we determined that a good approximation to this cost function is given by

$$\tau(k) \approx \tau_1 + \gamma_1 k, \qquad 1 \leq k \leq 12, \tag{3.10}$$
$$\tau(k) \approx \tau_2 + \gamma_2 k, \qquad 13 \leq k.$$

The startup times, τ_1 and τ_2, were determined to be 378μsec and 702μsec. The incremental costs, γ_1 and γ_2, are 1.19μsec/value and 2.87μsec/value. With these coefficients, the term $T(n)$ in (3.9) can be approximated, for $n \geq 13$, by

$$T(n) \approx \gamma_2 n^2/2 + \tau_2 n + 78(\gamma_1 - \gamma_2) + 12(\tau_1 - \tau_2). \tag{3.11}$$

After substituting these coefficients into the equations for t_{comp} and t_{comm}, (3.6) was plotted, along with the experimental results for Algorithm 3.1, in Figs. 3.1, 3.3, and 3.4. To model the efficiency of the row-oriented Householder algorithm without pivoting we need only eliminate the β_2 term from the equation for t_{comp} and also the communication overhead due to pivoting in the equation for t_{comm}. The resulting modeling function is plotted in Figs. 3.1 and 3.2.

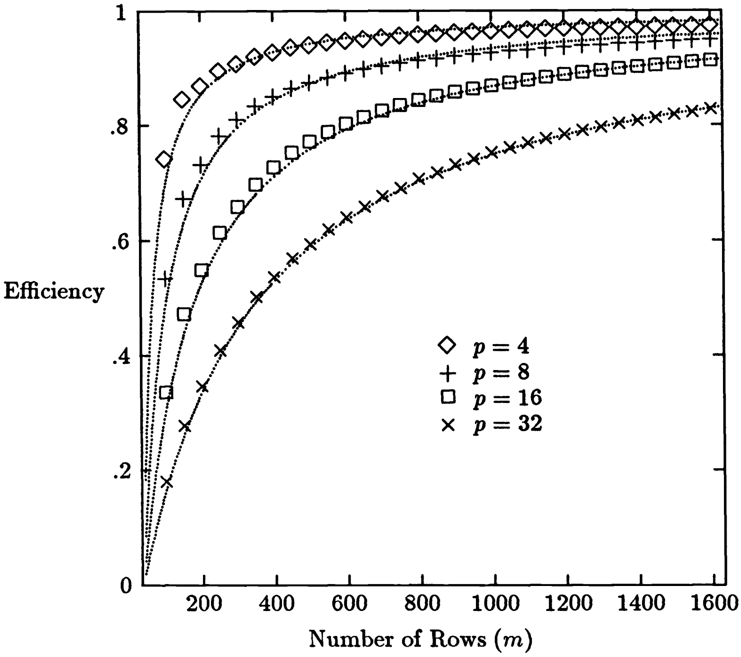


FIG. 3.3. *Efficiencies of Algorithm* 3.1 *with pivoting* ($m = 100$).

Finally, to model the hybrid algorithm we note that only t_{comp} must be modified from the analysis of the efficiency of Algorithm 3.1. Instead of accumulating sums as in Algorithm 3.1, the hybrid algorithm performs a nonlocal binary reduction of rows by Givens rotations. The binary reduction of rows of length k by Givens rotations

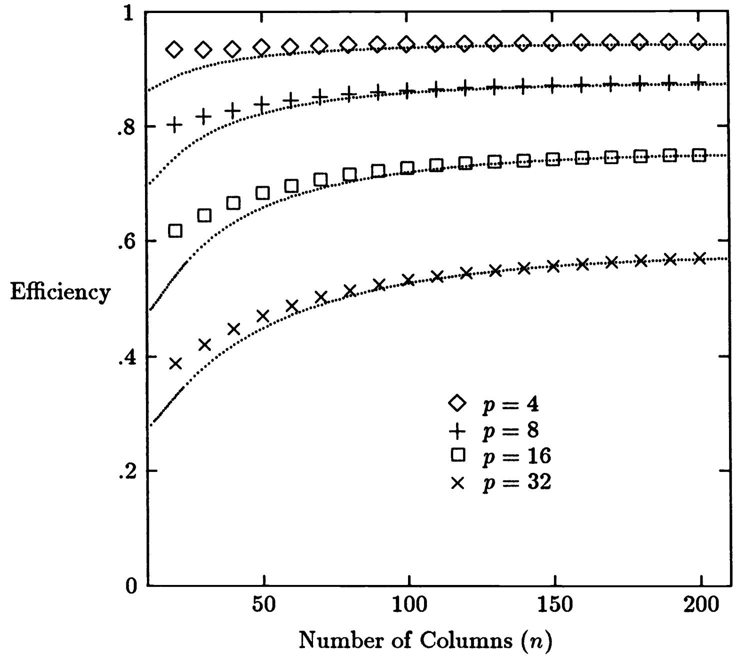


FIG. 3.4. *Efficiencies of Algorithm* 3.1 *with pivoting* ($m = 400$).

entails a total idle time for the processors of approximately $(\log(p) - 1)k\tau_{\text{Givens}}$, where τ_{Givens} is the time required to apply a Givens rotation to a 2-vector. For the iPSC/2, τ_{Givens} was measured to be approximately 37.3μsec. Summing this value from $k = 1$ to n yields total time $\frac{1}{2}(\log(p) - 1)n^2\tau_{\text{Givens}}$. Including the term for the differing number of rows assigned to processors we have that computational imbalance for the hybrid algorithm measured in flops is

$$t_{\text{comp}}^{(\text{hybrid})} \approx ((\log(p) - 1)\tau_{\text{Givens}} + 1)n^2/2. \tag{3.12}$$

Substituting this expression into (3.6) along with the expression for t_{comm} without pivoting we obtain the efficiency modeling function plotted in Fig. 3.1.

4. A parallel implementation of the Levenberg–Marquardt algorithm. To determine the Levenberg–Marquardt parameter, the matrix in (1.4) must be reduced to upper-triangular form. This reduction is computationally intensive: $n(n+1)/2$ Givens rotations and the corresponding row updates, or $O(n^3)$ flops. Note that the work required in this reduction is independent of m, the number of rows. Algorithm 4.1 details a parallel method to accomplish this reduction. In the algorithmic description let S represent storage for an upper-triangular matrix that is initially set equal to the matrix $\sqrt{\lambda}I$ in (1.4). Remember that the rows of R and S are wrapped onto an embedded ring of processors, as described in §1.

Algorithm 4.1 proceeds in n stages, which have been indexed by $j = 0, \cdots, n-1$ in the description. At stage j of Algorithm 4.1 the superdiagonal of S that is a distance j from the main diagonal is eliminated by Givens rotations. After n stages, the upper-triangular matrix S has been completely zeroed and the updated upper-triangular matrix R is still wrapped onto the processors in the same manner as at

```
Index Set:  $I_i$ {set of row indices assigned to processor $i$}
Functions:  next {returns number of next processor in the ring},
            prev {returns number of previous processor in the ring}

Proc ($i$) : {program for processor $i$}
    For $j = 0, \cdots, n-1$ do
        If ($j \neq 0$) receive rows $S^T_{k-j}$, $k \in I_j$, from processor prev ($i$);
        For $k \in I_j$ do
            Compute Givens rotation to zero the bottom
                of the vector $(R_{k,k}, S_{k-j,k})^T$;
            Update rows $R^T_k$ and $S^T_{k-j}$ with above Givens rotation;
            If ($k = j+1$) Delete {$k$} from $I_i$;
        enddo
        Send rows $S^T_{k-j}$, $k \in I_j$, to processor next ($i$);
    enddo
```

ALGORITHM 4.1. *A parallel row-oriented* R-S *reduction.*

the start of the algorithm. As the leading nonzero of each row of S is eliminated and the corresponding rows updated, the rows of S move around the embedded ring in a systolic manner. Although the work at each stage is not completely balanced, the processor doing the most work rotates around the ring. This imbalance is somewhat offset by the required communication. Experimental results of the efficiency of this algorithm as a function of the number of columns are presented as data points in Fig. 4.1. Also plotted in the figure are the modeling functions for these efficiencies, which we will develop below.

Similar to the analysis of the QR factorization algorithms, we can model the observed efficiencies of Algorithm 4.1. First note that the total sequential work, measured in flops, is given by the formula

$$\begin{aligned} t_{\text{seq}} &\approx \sum_{k=1}^{n} 4\left(\frac{k^2}{2}\right) \\ &\approx \frac{2}{3}n^3, \end{aligned} \tag{4.1}$$

since the application of each Givens rotation to a 2-vector requires four flops. As before, we use the above equation to define the length of time we consider to be a flop. On the iPSC/2, this time was determined to be 10.9μsec.

At each step of the outer loop in Algorithm 4.1 there is a processor assigned the longest rows relative to other processors. Each of these rows differs from the average row length by $p/2$ elements. Since each processor has approximately k/p rows at step $n-k$, we have that the computational imbalance is bounded by

$$\begin{aligned} t^{(\text{row})}_{\text{comp}} &\approx \sum_{k=1}^{n} 4\left(\frac{k}{p}\right)\left(\frac{p}{2}\right) \\ &\approx n^2. \end{aligned} \tag{4.2}$$

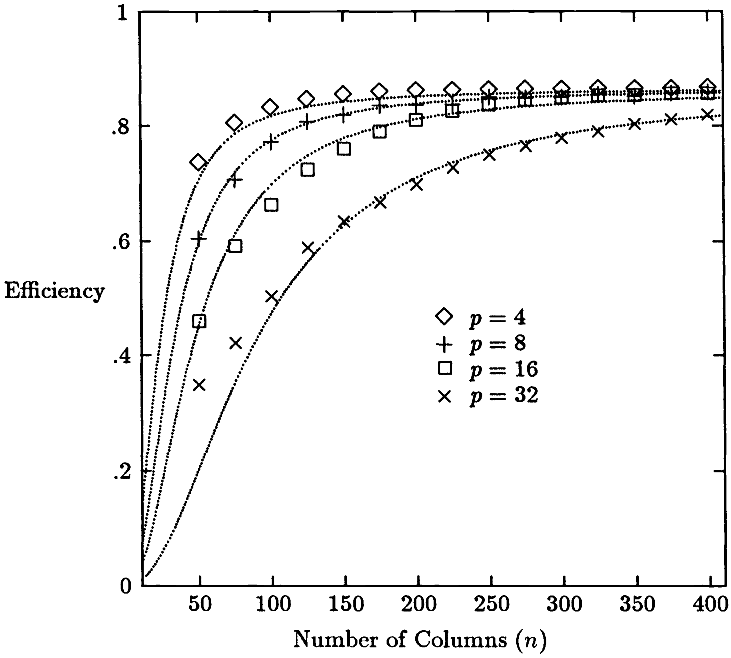


FIG. 4.1. *Efficiencies of Algorithm* 4.1 *on the iPSC/2.*

To compute the communication overhead we define the function $\rho(k)$ to be the length of time for all the processors on an embedded ring to send synchronously a double precision vector of length k to their neighbors. Experimentally, this function is essentially linear; hence, we introduce the approximation

$$\rho(k) \approx \rho_0 + \phi k. \tag{4.3}$$

For the iPSC/2 we determined values of 1105μsec for ρ_0 and 6.85μsec/value for ϕ.

At iteration $n-k$ of Algorithm 4.1 the message length is approximately $k^2/(2p)$; hence, we have that

$$\begin{aligned} t_{\text{comm}}^{(\text{row})} &\approx \sum_{k=1}^{n} \rho\left(\frac{k^2}{2p}\right) \\ &\approx \rho_0 n + \phi \frac{n^3}{6p}. \end{aligned} \tag{4.4}$$

Using (3.6), we obtain the following modeling function for the efficiency of Algorithm 4.1:

$$\text{efficiency}^{(\text{row})} \approx \frac{1}{1 + \phi/4 + \frac{3}{2}(p/n + \rho_0 p/n^2)}. \tag{4.5}$$

After substituting the necessary coefficients into (4.5), the resulting efficiency functions were plotted in Fig. 4.1. Note that for large n the efficiencies do not asymptotically approach 1, but rather approach the constant $1/(1+\phi/4)$, which is independent of the number of processors used.

```
Index Set: $K_i$ {set of column indices assigned to processor $i$}

Proc ($i$) : {program for processor $i$}
    For $j = 0, \cdots, n-1$ do
        If ($j \neq 0$) Receive Givens vector $g$ from processor prev ($i$);
        For $k \in K_j$ do
            For $l = \min(j, p-1), \cdots, 1$ do
                Update rows $R_{k-l}^T$ and $S_{k-j}^T$ with Givens rotation $g_{k-l}$;
            enddo
            Compute Givens rotation to zero the bottom
                of the vector $(R_{k,k}, S_{k-j,k})^T$;
            Update rows $R_k^T$ and $S_{k-j}^T$ with above Givens rotation;
            Update $g_k$ in Givens vector;
            If ($k = j + 1$) Delete {$k$} from $I_i$;
        enddo
        Send Givens vector $g$ to processor next ($i$);
    enddo
```

ALGORITHM 4.2. *A parallel column-oriented* R-S *reduction.*

A column-oriented approach is also possible and is presented as Algorithm 4.2. Experimental results for this algorithm are compared to those of Algorithm 4.1 in Table 4.1 and also plotted in Fig. 4.2. For Algorithm 4.2 the columns, as opposed to the rows, of R and S are wrapped onto the ring of processors. Rather than communicating rows of S between neighboring processors, the Givens rotations are stored in vectors g that rotate around the ring. Once the algorithm has been running for more than p steps, i.e., $j \geq p-1$, then the Givens vector g is completely filled with updates that need to be applied once received. The order in which these rotations are applied in the l loop is important. Since they operate on the same row of S, the rotations must be applied from the oldest to the most recent. Also, by row R_k^T we mean the nonzero components of row R_k^T that are local to processor i. These components are given by the index set K_i.

Even though Algorithm 4.2 is a bit more complicated, the total number of messages that have to be sent is the same as in Algorithm 4.1. However, for large n/p, the total number of values that have to communicated is actually less. For an average step j in Algorithm 4.2 we need only communicate the single Givens vector g of length $O(n)$ between neighboring processors. For the row-oriented version we need to communicate $O(n/p)$ rows of S of length $O(n)$ between processors. In practice, the rows of S are combined into one long message that results in the same number of communication startups as appear in Algorithm 4.2. The message startup cost, measured in equivalent flops, for the Intel iPSC hypercube is very expensive and is normally the dominant factor in the communication cost of an algorithm. For large n/p, however, the average message lengths are extremely different; hence, in comparing Figs. 4.1 and 4.2, it is apparent that the column-oriented version is asymptotically superior. By the same argument, for small n/p, the row-oriented version is superior. This crossover in the observed efficiencies of the two algorithms can be explained by also modeling the efficiency of Algorithm 4.2.

TABLE 4.1
Execution times of Algorithms 4.1 and 4.2 on the iPSC/2 hypercube.

Execution times (sec)				
n	Single processor	p	Algorithm 4.1	Algorithm 4.2
50	1.06	8	0.22	0.21
		16	0.15	0.15
		32	0.09	0.12
100	7.78	8	1.26	1.20
		16	0.74	0.72
		32	0.48	0.49
150	25.54	8	3.90	3.59
		16	2.11	2.01
		32	1.25	1.25
200	59.67	8	8.97	8.07
		16	4.63	4.37
		32	2.67	2.59
300	198.28	8	29.43	25.95
		16	14.66	13.68
		32	7.95	7.56
400	466.14	8	67.91	60.27
		16	34.24	31.28
		32	17.79	16.71

The computational imbalance of Algorithm 4.2 is the same as that for the row-oriented algorithm. Hence, we need only modify the expression for the communication overhead in the efficiency model. Since at iteration $n-k$ in the column-oriented algorithm, each processor sends k Givens rotations to its ring neighbor, we have that

$$t_{\mathrm{comm}}^{(\mathrm{col})} \approx \sum_{k=1}^{n} \rho(2k) \approx \rho_0 n + \phi n^2. \tag{4.6}$$

Combining this expression with the bound for the computational imbalance obtained earlier we obtain an approximation to the efficiency of Algorithm 4.2:

$$\text{efficiency}^{(\mathrm{col})} \approx \frac{1}{1+\frac{3}{2}((1+\phi)p/n + \rho_0 p/n^2)}. \tag{4.7}$$

Comparing (4.5) and (4.7) we note that they are equal for $n^* = 6p$. Experimentally, this crossover in the efficiencies of the two algorithms can be observed in Fig. 4.3. In this figure the crossover appears to occur near $n = 150$, close to the value of $n^* = 192$ predicted by the efficiency modeling functions.

Finally, we note that there are two possible ways to improve the asymptotic performance of the row-oriented algorithm. The first would be to wrap blocks of rows, say, b rows, onto the processors instead of single rows. This would decrease the length of messages sent at each iteration by a factor of $1/b$. Of course, this approach would also increase the computational imbalance by a factor of b. Following the analysis done above for the row-oriented algorithm we find that the optimum block size is $b^* = \sqrt{\phi n/(6p)}$. For this value of b, the asymptotic efficiency of the algorithm is improved to $1/(1+\sqrt{3\phi p/(2n)})$. However, note that for the value of ϕ determined above and for $p = 32$, n must be greater than 610 for the efficiency to improve when

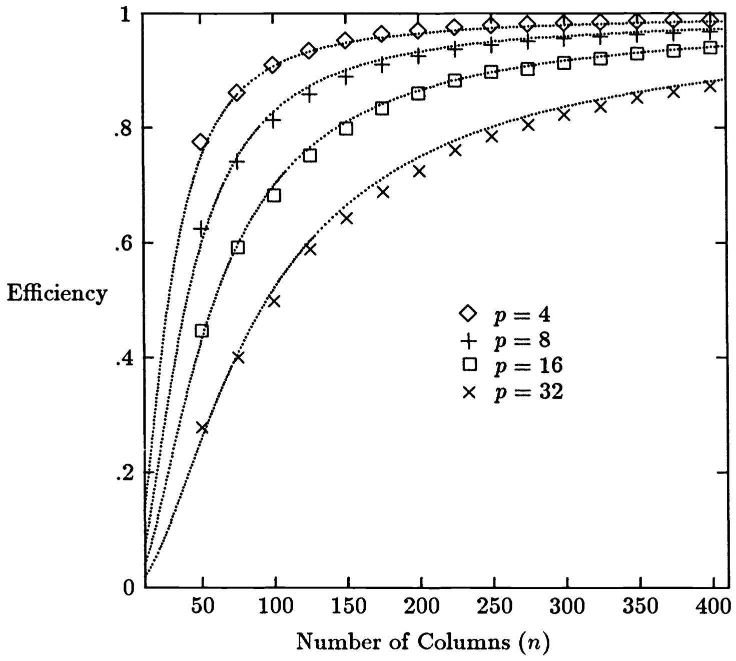


FIG. 4.2. *Efficiencies of Algorithm* 4.2 *on the iPSC/2.*

using block size $b = 2$ instead of block size $b = 1$. A second possible improvement would be to decrease the length of the messages sent in the row-oriented method by postponing the application of the Givens rotations. Unfortunately, both of these algorithms are very complicated and were not implemented.

The reduction of the matrix in (1.4) to upper-triangular form is the major task in a parallel algorithm for determining the Levenberg–Marquardt parameter λ_*, and we have shown that there exist effective algorithms to perform this reduction. However, efficient solution of triangular systems is also important in this context. In fact, for each iteration involving a solution of (1.4) there are two associated triangular solutions that are used to bracket the solution λ_* [M78]. Recently, much work has been done on the parallel solution of triangular systems [C86], [LC88], [LC89], [HR88]. We used the triangular solution algorithms developed by Li and Coleman in our implementations, but it should be noted that the efficiencies of these algorithms are not nearly as good as those of Algorithms 4.1 and 4.2. This difference is what accounts for the discrepancies between the efficiencies shown in Fig. 4.1 and the efficiencies reported in the next section for solving for the Levenberg–Marquardt parameter. This effect is apparent since even though there is an $O(n)$ difference between the amount of work required for Algorithm 4.1 and the corresponding triangular solutions, their communication costs are comparable. The importance of efficient parallel triangle system solvers has also been observed in the parallel solution of systems of nonlinear equations [CL87].

5. Experimental results and conclusions. These algorithms were implemented on a 32 node Intel iPSC/2 hypercube with 4.5 MBytes of memory per node in Green Hills Fortran-386 and run under version R3.2 of the iPSC operating system.

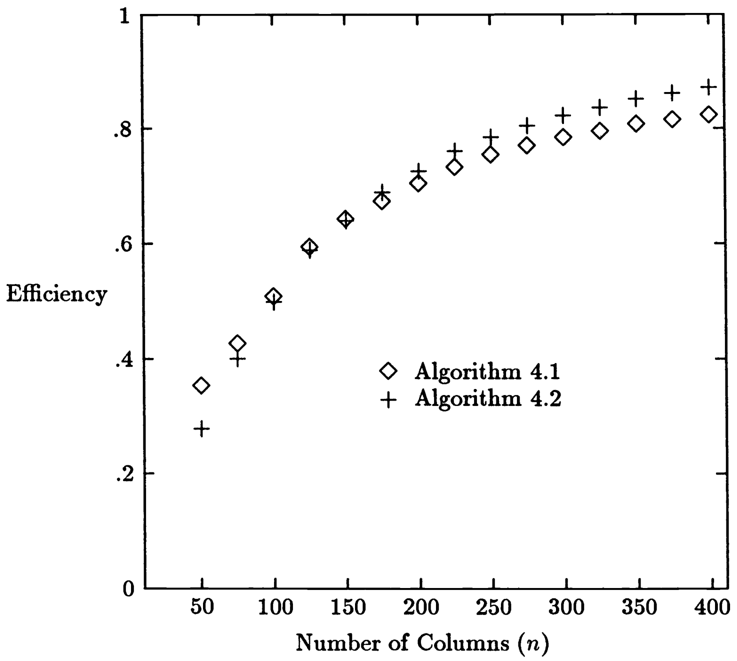


FIG. 4.3. *Efficiencies of Algorithms* 4.1 *and* 4.2 ($p = 32$).

The efficiencies shown below were calculated by dividing the running time of MINPACK [MGH80] code on a single processor by the number of processors used times the running times of the parallel algorithms. Therefore, a large number corresponds to greater efficiency. The comparison is fair, as both programs generate the same sequence of iterates and consequently do the same number of Jacobian approximations, QR factorizations, and Newton iterations in computing the Levenberg–Marquardt parameter.

TABLE 5.1
Description of test functions.

Problem number	Function form	Characteristics	F eval. cost	J approx. cost	F eval. separable?
1	$F = Ax - e$	$A \in \mathbf{R}^{m \times n}$, full rank, Ax easily evaluated, e an m-vector of ones	$O(m)$	cheap	no
2	$F = Ax - e$	$A \in \mathbf{R}^{m \times n}$, low rank, dense	$O(mn)$	expensive	yes
3	$f_i = \sum_{i,k} A_{ijk} x_j x_k - 1$	A sparse	$O(m)$	cheap	yes
4	$f_i = (\exp(x_j \tau_i) - \exp(\alpha_j \tau_i))$	Constrained, $x < 0$ for $\alpha < 0$ and $\tau > 0$	$O(mn)$	expensive	yes

The test problems used to obtain the experimental results are described in

TABLE 5.2
Experimental results of parallel algorithms compared with MINPACK.

Efficiencies compared to MINPACK (% time spent in routine)									
Prob	n	m	p	Total	QR	L-M	R-S	Tri-s	J-appr
1	100	250	8	.732 (100.0)	.757 (94.1)	.182 (0.8)	 (0.0)	.202 (0.7)	.498 (3.0)
			16	.531 (100.0)	.585 (88.2)	.045 (2.4)	 (0.0)	.047 (2.1)	.326 (3.4)
			32	.322 (100.0)	.391 (80.0)	.023 (2.9)	 (0.0)	.023 (2.6)	.191 (3.5)
1	100	500	8	.849 (100.0)	.859 (96.8)	.189 (0.4)	 (0.0)	.202 (0.4)	.624 (2.3)
			16	.709 (100.0)	.745 (93.2)	.045 (1.5)	 (0.0)	.047 (1.4)	.451 (2.7)
			32	.518 (100.0)	.576 (88.1)	.023 (2.1)	 (0.0)	.023 (2.0)	.291 (3.0)
1	100	1000	8	.908 (100.0)	.915 (97.5)	.181 (0.2)	 (0.0)	.202 (0.2)	.737 (1.9)
			16	.822 (100.0)	.850 (94.9)	.046 (0.8)	 (0.0)	.047 (0.8)	.591 (2.1)
			32	.694 (100.0)	.733 (93.0)	.023 (1.4)	 (0.0)	.023 (1.3)	.423 (2.5)
1	200	250	8	.739 (100.0)	.752 (95.9)	.355 (0.5)	 (0.0)	.396 (0.4)	.408 (2.8)
			16	.561 (100.0)	.586 (93.5)	.068 (2.0)	 (0.0)	.070 (1.8)	.247 (3.5)
			32	.361 (100.0)	.396 (89.0)	.041 (2.1)	 (0.0)	.044 (1.9)	.137 (4.0)
1	200	500	8	.851 (100.0)	.860 (97.6)	.354 (0.2)	 (0.0)	.396 (0.2)	.518 (1.7)
			16	.733 (100.0)	.758 (95.2)	.068 (1.1)	 (0.0)	.070 (1.0)	.342 (2.3)
			32	.562 (100.0)	.599 (92.5)	.042 (1.4)	 (0.0)	.044 (1.3)	.204 (2.9)

Table 5.1. Shown is the functional form of the test problems, and the computational complexity of evaluating each function is given in the column labeled "F eval. cost." We also make a subjective determination as to whether estimation of the Jacobian is cheap or expensive relative to its QR factorization. To be more specific about the functions used for testing, problem 1 has an $O(m)$ evaluation cost because of a special form for the matrix A: $A_{ij} = -2/m$, $i \neq j$; $A_{ii} = 1-2/m$. The matrix A used in problem 2 is of low rank: $A_{ij} = ij + O(\epsilon)$ perturbations, where ϵ is the machine precision. For problem 3 the tensor A has a bandwidth of 2, allowing for function evaluation with $O(m)$ cost. The constants used for problem 4 were $\alpha_j = -j$ and $\tau_i = i/m$. In the final column of Table 5.1 we note whether we consider evaluation of these functions to be separable.

Tables 5.2–5.5 summarize the experimental results obtained by comparing our parallel algorithms with the MINPACK code running on a single processor for solving the test problems described in Table 5.1. The efficiencies and the fraction of the total parallel running time spent in each of six sections of the programs are detailed. The six sections refer to: QR, the QR factorization of the Jacobian approximation; L-M, computation of the Levenberg–Marquardt parameter; R-S, the R-S reduction

TABLE 5.3
Experimental results of parallel algorithms compared with MINPACK.

Efficiencies compared to MINPACK (% time spent in routine)									
Prob	n	m	p	Total	QR	L-M	R-S	Tri-s	J-appr
2	100	250	8	.811 (100.0)	.722 (30.9)	.700 (30.8)	.752 (27.7)	.203 (2.8)	.962 (37.1)
			16	.648 (100.0)	.520 (34.3)	.498 (34.6)	.638 (26.1)	.058 (7.9)	.986 (28.9)
			32	.457 (100.0)	.349 (36.0)	.310 (39.2)	.488 (24.1)	.023 (13.8)	.956 (21.0)
2	100	500	8	.873 (100.0)	.833 (37.9)	.657 (11.3)	.740 (9.7)	.201 (1.1)	.972 (48.2)
			16	.780 (100.0)	.686 (41.1)	.477 (14.0)	.631 (10.2)	.058 (3.3)	.983 (42.6)
			32	.623 (100.0)	.531 (42.4)	.288 (18.5)	.481 (10.7)	.023 (6.5)	.956 (35.0)
2	100	1000	8	.930 (100.0)	.902 (41.5)	.616 (2.1)	.733 (1.7)	.202 (0.2)	.979 (54.5)
			16	.867 (100.0)	.797 (43.8)	.387 (3.1)	.625 (1.8)	.058 (0.7)	.998 (49.8)
			32	.795 (100.0)	.695 (46.0)	.284 (3.8)	.487 (2.1)	.023 (1.6)	.956 (47.7)
2	200	250	8	.841 (100.0)	.711 (24.7)	.812 (39.7)	.825 (38.5)	.394 (1.1)	.963 (35.0)
			16	.739 (100.0)	.527 (29.3)	.725 (39.1)	.788 (35.4)	.111 (3.4)	.962 (30.8)
			32	.580 (100.0)	.339 (35.7)	.571 (38.9)	.688 (31.8)	.044 (6.7)	.961 (24.2)
2	200	500	8	.900 (100.0)	.849 (31.6)	.816 (22.6)	.828 (21.9)	.395 (0.6)	.973 (45.2)
			16	.822 (100.0)	.739 (33.2)	.711 (23.7)	.780 (21.2)	.111 (2.1)	.957 (42.0)
			32	.701 (100.0)	.576 (36.4)	.550 (26.1)	.679 (20.8)	.044 (4.5)	.957 (35.8)

described in §5; Tri-s, the solution of triangular systems; and J-appr, the approximation of the Jacobian by forward differences. We include in J-appr the function evaluations necessary to estimate the Jacobian, but not the function evaluations done to test the step acceptance criteria. These function evaluations are included in the total time but not in one of the six sections. Their efficiency is comparable to that of the Jacobian approximation but their fraction of the total running time is much smaller.

These results were chosen to illustrate an average-case behavior of the parallel algorithms in solving nonlinear least-squares problems. For a particular problem the fraction of time spent in the different routines and the number of iterations required for convergence can vary dramatically for various choices of a starting point, initial trust region size, and termination tolerances. For the above problems the MINPACK tolerances were set to $\sqrt{\epsilon}$, where ϵ is the machine precision. The initial trust region size was given by $100\|\hat{x}_0\|_2$, where $\hat{x}_0$ is the starting point normalized by the 2-norms of the columns of the initial Jacobian approximation. For problems 1, 2, and 3, an n-vector of ones was used as the starting point, and for problem 4 the starting point $x_j = -(j + 0.1)$ was employed. For the problem sizes shown, problem 1 required 2 to 3 iterations (Jacobian approximations) for convergence; problem 2 used 10 to

TABLE 5.4
Experimental results of parallel algorithms compared with MINPACK.

Efficiencies compared to MINPACK (% time spent in routine)									
Prob	n	m	p	Total	QR	L-M	R-S	Tri-s	J-appr
3	100	250	8	.742 (100.0)	.758 (94.9)	.182 (0.8)	(0.0)	.202 (0.7)	.527 (3.2)
			16	.545 (100.0)	.586 (90.2)	.046 (2.5)	(0.0)	.047 (2.2)	.345 (3.6)
			32	.344 (100.0)	.391 (85.3)	.023 (3.1)	(0.0)	.023 (2.8)	.207 (3.8)
3	100	500	8	.847 (100.0)	.862 (93.9)	.627 (3.2)	.751 (2.5)	.198 (0.6)	.641 (2.3)
			16	.710 (100.0)	.747 (90.8)	.388 (4.4)	.641 (2.5)	.057 (1.7)	.467 (2.7)
			32	.518 (100.0)	.577 (85.9)	.216 (5.7)	.492 (2.3)	.023 (3.0)	.302 (3.0)
3	100	1000	8	.911 (100.0)	.920 (96.1)	.633 (1.7)	.754 (1.3)	.202 (0.3)	.744 (1.9)
			16	.828 (100.0)	.855 (94.0)	.389 (2.5)	.642 (1.4)	.058 (0.9)	.600 (2.1)
			32	.686 (100.0)	.738 (90.3)	.216 (3.7)	.491 (1.5)	.023 (1.9)	.433 (2.5)
3	200	250	8	.745 (100.0)	.752 (86.6)	.784 (9.9)	.817 (9.2)	.396 (0.6)	.453 (2.6)
			16	.577 (100.0)	.586 (86.0)	.652 (9.2)	.789 (7.3)	.111 (1.7)	.279 (3.3)
			32	.383 (100.0)	.395 (84.7)	.459 (8.7)	.686 (5.6)	.044 (2.8)	.156 (3.9)
3	200	500	8	.853 (100.0)	.862 (97.5)	.354 (0.3)	(0.0)	.396 (0.2)	.549 (1.8)
			16	.738 (100.0)	.760 (95.6)	.068 (1.1)	(0.0)	.070 (1.1)	.367 (2.3)
			32	.571 (100.0)	.601 (93.6)	.042 (1.4)	(0.0)	.044 (1.3)	.222 (3.0)

12 iterations. Problem 3 required 7 to 8 iterations to converge and more than 10 iterations were required for problem 4 to converge; the results shown in Table 5.5 were taken from the first 10 iterations.

From these results it is apparent that either the QR factorization or the Jacobian approximation is the dominant computational cost for these problems. When the QR factorization cost dominates, the implementation is more efficient as m, the number of rows, increases. The cost of computing the Levenberg–Marquardt parameter can also be significant; this computation could dominate the QR factorization for problems that require a disproportionately large number of R-S reductions and for which the ratio m/n is close to one. This effect would occur in problems where the Jacobian is rank-deficient or the function is very nonlinear (which would require a small trust region in order to ensure an accurate quadratic model of the function). But again, for a fixed ratio m/n, the efficiency in solving for the Levenberg–Marquardt parameter increases as m increases. As expected, the parallel triangular system solutions are very inefficient when compared to the QR factorization and R-S reductions. However, the time required for these solutions composes only a small fraction of

TABLE 5.5
Experimental results of parallel algorithms compared with MINPACK.

				Efficiencies compared to MINPACK (% time spent in routine)					
Prob	n	m	p	Total	QR	L-M	R-S	Tri-s	J-appr
4	100	250	8	.905 (100.0)	.745 (10.8)	.708 (18.0)	.759 (16.3)	.202 (1.4)	.980 (69.2)
			16	.814 (100.0)	.570 (12.7)	.517 (22.1)	.647 (17.2)	.058 (4.5)	.970 (62.9)
			32	.677 (100.0)	.371 (16.2)	.330 (28.8)	.492 (18.8)	.023 (9.3)	.975 (52.1)
4	100	500	8	.946 (100.0)	.814 (11.9)	.704 (9.5)	.756 (8.6)	.201 (0.8)	.996 (76.4)
			16	.885 (100.0)	.747 (12.2)	.514 (12.2)	.644 (9.5)	.058 (2.5)	.971 (73.3)
			32	.796 (100.0)	.558 (14.6)	.329 (17.1)	.490 (11.2)	.023 (5.6)	.978 (65.5)
4	100	1000	8	.973 (100.0)	.866 (12.5)	.703 (4.4)	.755 (4.0)	.202 (0.4)	0.994 (80.8)
			16	.941 (100.0)	.850 (12.3)	.511 (5.9)	.644 (4.5)	.058 (1.2)	.987 (79.5)
			32	.879 (100.0)	.703 (13.9)	.326 (8.6)	.491 (5.5)	.023 (2.8)	.979 (74.9)
4	200	250	8	.902 (100.0)	.682 (8.1)	.819 (33.2)	.831 (32.3)	.395 (0.8)	.980 (57.8)
			16	.850 (100.0)	.525 (10.0)	.740 (34.6)	.797 (31.7)	.111 (2.7)	.979 (54.5)
			32	.743 (100.0)	.350 (13.0)	.586 (38.2)	.693 (31.9)	.044 (6.0)	.978 (47.6)
4	200	500	8	.945 (100.0)	.805 (10.1)	.821 (17.8)	.833 (17.3)	.393 (0.4)	.996 (71.0)
			16	.904 (100.0)	.717 (10.9)	.739 (19.0)	.797 (17.3)	.111 (1.5)	.979 (69.1)
			32	.837 (100.0)	.572 (12.6)	.585 (22.2)	.694 (18.5)	.044 (3.5)	.978 (64.0)

the total computation time of the parallel implementation, and therefore it does not significantly decrease the overall efficiency. However, note that for fixed problem size the fraction of total time spent solving triangular systems does increase significantly as the number of processors is increased.

For functions whose evaluation is very expensive we observe that the computation required for the approximation of the Jacobian by forward differences can equal or exceed the computation required for these other tasks. For the expensive test functions we considered, the function evaluation was separable, and hence the row-oriented Jacobian approximation algorithm yielded efficiencies comparable to the QR factorization and Levenberg–Marquardt parameter solves. If the function evaluation were expensive and not separable, one would have to resort to a column-oriented Jacobian approximation algorithm, as described in §3. In this case the efficiency of the implementation would depend on the number of columns being sufficiently large. Another possibility would be to obtain an algebraic expression for the Jacobian and to evaluate the Jacobian directly rather than using forward differences; the algebraic evaluation of the Jacobian could be separable even when the function itself is not.

When the function evaluation is expensive and not separable, a column-oriented implementation suggests itself. In this context, the improved efficiency of using a pipelined, column-oriented QR factorization is tempting. However, complete pivoting destroys the pipelining aspect of these column-oriented algorithms. A local pivoting strategy is possible [B88], but the effect of a different pivoting strategy on the solution of these nonlinear problems would have to be tested.

In summary, we have observed good efficiencies when solving moderate-sized nonlinear least-squares problems on the Intel hypercube. For the test problems considered we noted that the efficiency of our parallel implementation improved as the ratio m/p increased, where m is the number of rows of the Jacobian and p is the number of processors. We also point out that it is possible to solve much larger problems than those we have described above (which had to be run on one processor for comparison). The efficiencies for such larger problems would be correspondingly better. A related topic is the solution of large sparse problems: Plassmann [P90] has considered the general case, and future consideration should be given to problems that have special structure. For example, the row-oriented approach could also be used if the Jacobian were banded and would be efficient in solving problems with a block structure.

Acknowledgments. The work reported in this paper was partially completed with the assistance of computing facilities of the Advanced Computing Research Institute at the Cornell Center for Theory and Simulation in Science and Engineering, which is supported by the National Science Foundation and New York State. We would also like to acknowledge discussions at Oak Ridge National Laboratory (as part of their Numerical Linear Algebra Year) and thank the referees for a number of constructive comments.

REFERENCES

[B88] C. Bischof, QR *factorization algorithms for coarse-grained distributed systems,* Tech. Report 88-939, Computer Science Department, Cornell University, Ithaca, NY, 1988.

[BSS88] R. Byrd, R. Schnabel, and G. Shultz, *Parallel quasi-Newton methods for unconstrained optimization,* Tech. Report CU-CS-396-88, Department of Computer Science, University of Colorado, Boulder, CO, 1988.

[C86] R. M. Chamberlain, *An algorithm for* LU *factorization with partial pivoting on the hypercube,* Tech. Report CCS 86/11, Department of Science and Technology, Chr. Michelsen Institute, Bergen, Norway, 1986.

[CP86] R. M. Chamberlain and M. J. D. Powell, QR *factorisation for linear least-squares problems on the hypercube,* Tech. Report CCS 86/10, Department of Science and Technology, Chr. Michelsen Institute, Bergen, Norway, 1986.

[CG88] E. Chu and A. George, QR *factorization of a dense matrix on a hypercube multiprocessor,* Tech. Report ORNL/TM-10691, Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN, 1988.

[C84] T. F. Coleman, *Large sparse numerical optimization,* Lecture Notes in Computer Science 165, G. Goos and J. Hartmanis, eds., Springer-Verlag, New York, 1984.

[CL87] T. F. Coleman and G. Li, *Solving systems of nonlinear equations on a message-passing multiprocessor,* SIAM J. Sci. Statist. Comput., 11 (1990), pp. 1116–1135.

[CP89] T. F. Coleman and P. E. Plassmann, *Solution of nonlinear least squares problems on a multiprocessor,* in Lecture Notes in Computer Science 384, Parallel Computing 1988, G. A. van Zee and J. G. G. van de Vorst, eds., Springer-Verlag, Berlin, New York, 1989, pp. 44–60.

[HR88] M. T. Heath and C. H. Romine, *Parallel solution of triangular systems on distributed memory multiprocessors,* SIAM J. Sci. Statist. Comput., 9 (1988), pp. 558–588.

[JH88] L. Johnsson and D. T. Ho, *Algorithms for matrix transposition on boolean n-cube configured ensemble architectures,* SIAM J. Matrix Anal. Appl., 9 (1988), pp. 419–454.

[L44] K. Levenberg, *A method for the solution of certain non-linear problems in least squares,* Quart. Appl. Math., 2 (1944), pp. 164–168.

[LC89] G. Li and T. F. Coleman, *A new method for solving triangular systems on distributed memory message-passing multiprocessors,* SIAM J. Sci. Statist. Comput., 10 (1989), pp. 382–396.

[LC88] ———*A parallel triangular solver for a distributed memory multiprocessor,* SIAM J. Sci. Statist. Comput., 9 (1988), pp. 485–502.

[M63] D. W. Marquardt, *An algorithm for least squares estimation of non-linear parameters,* SIAM J. Appl. Math., 11 (1963), pp. 431–441.

[MVV87] O. M. McBryan and E. F. Van de Velde, *Hypercube algorithms and implementations,* SIAM J. Sci. Statist. Comput., 8 (1987), pp. s227–s287.

[M87] C. Moler, *Matrix computations on distributed memory multiprocessors,* Tech. Report, Intel Scientific Computers, Beaverton, OR, 1987.

[M78] 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.

[MGH80] J. J. Moré, B. Garbow, and K. Hillstrom, *User guide for MINPACK-*1, Argonne National Laboratory Report ANL-80-74, Argonne, IL, 1980.

[P90] P. E. Plassmann, *Sparse Jacobian estimation and factorization on a multiprocessor,* in Large-Scale Numerical Optimization, T.F. Coleman and Y. Li, eds., Society for Industrial and Applied Mathematics, 1990, pp. 152–179.

[PR87] A. Pothen and P. Raghavan, *Distributed orthogonal factorization: Givens and Householder algorithms,* Tech. Report, Department of Computer Science, The Pennsylvania State University, State College, PA, 1987.

[SS85] Y. Saad and M. H. Schultz, *Data communication in hypercubes*, Yale University Research Report DCS/RR-428, Department of Computer Science, Yale University, New Haven, CT, 1985.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 3, pp. 794–814, May 1992

011

WELL-CONDITIONED ITERATIVE SCHEMES FOR MIXED FINITE-ELEMENT MODELS OF POROUS-MEDIA FLOWS*

MYRON B. ALLEN†, RICHARD E. EWING†, AND PENG LU‡

Abstract. Mixed finite-element methods are attractive for modeling flows in porous media since they can yield pressures and velocities having comparable accuracy. In solving the resulting discrete equations, however, poor matrix conditioning can arise both from spatial heterogeneity in the medium and from the fine grids needed to resolve that heterogeneity. This paper presents two iterative schemes that overcome these sources of poor conditioning. The first scheme overcomes poor conditioning resulting from the use of fine grids. The idea behind the scheme is to use spectral information about the matrix associated with the discrete version of Darcy's law to precondition the velocity equations, employing a multigrid method to solve mass-balance equations for pressure or head. This scheme still exhibits slow convergence when the permeability or hydraulic conductivity is highly variable in space. The second scheme, based on the first, uses mass lumping to precondition the Darcy equations, thus requiring more work per iteration and minor modifications to the multigrid algorithm. However, the scheme is insensitive to heterogeneities. The overall approach should also be useful in such applications as electric field simulation and heat transfer modeling when the media in question have spatially variable material properties.

Key words. mixed finite elements, iterative solution schemes, heterogeneous porous media

AMS(MOS) subject classification. 65

1. Introduction.

We consider methods for solving discrete approximations to the equations governing single-fluid flow in a porous medium. If the flow is steady and two-dimensional with no gravity drive, Darcy's law and the mass balance take the following forms:

$$\begin{aligned} \mathbf{u} &= -K \operatorname{grad} p \quad \text{in } \Omega, \\ \operatorname{div} \mathbf{u} &= f \quad \text{in } \Omega. \end{aligned} \tag{1.1}$$

Here $\mathbf{u}, p$, and f represent the Darcy velocity, pressure, and source term, respectively. For simplicity, we take the spatial domain to be a square, scaled so that $\Omega = (0,1) \times (0,1)$. The coefficient $K(x,y)$ is the *mobility*, defined as the ratio of the permeability of the porous medium to the dynamic viscosity of the fluid. In applications to underground flows, the structure of K may be quite complex, depending on the lithology of the porous medium and the composition of the fluid. We assume, however, that this ratio is bounded and integrable on $\bar{\Omega}$ and satisfies $K \geq K_{\text{inf}} > 0$. We impose the boundary condition $p = 0$ on $\partial\Omega$, so that p effectively represents the deviation in pressure from a reference value known along $\partial\Omega$.

Scientists modeling contaminant flows in groundwater or solvent flows in oil reservoirs often need accurate finite-element approximations of $\mathbf{u}$ and p simultaneously. For this reason, mixed finite-element methods for solving the system (1.1) are particularly attractive, since they can yield approximations to $\mathbf{u}$ and p that have comparable accuracy [1], [5], [9]. The key to achieving such approximations is the use of appropriate piecewise polynomial trial spaces, such as those proposed by

* Received by the editors April 12, 1990; accepted for publication (in revised form) February 20, 1991. This research was supported by National Science Foundation grant RII-8610680, Office of Naval Research grant 0014-88-K-0370, and the Wyoming Water Reseach Center.

† Department of Mathematics, University of Wyoming, Laramie, Wyoming 82071-3036.

‡ Department of Mathematics, University of Georgia, Athens, Georgia 30605.

Raviart and Thomas [11]. As we review in §2, if we use the lowest-degree Raviart–Thomas spaces, the mixed formulation yields systems of discrete equations that have the form

$$\begin{aligned} AU + NP &= 0, \\ N^T U &= F. \end{aligned} \tag{1.2}$$

Here, U and P signify vectors containing nodal values of the trial functions for $\mathbf{u}$ and p, defined on a grid over Ω, and A and N are matrices. As we illustrate below, the matrix A contains information about the spatially varying material property K, while N and N^T are essentially finite-difference matrices.

Equations (1.2) can be quite difficult to solve efficiently, for the following reasons. When K varies over short distances, accurate finite-element approximations require fine grids on Ω. For example, one might choose grids fine enough to allow reasonable approximations of K by piecewise constant functions. Fine grids, however, typically yield poorly conditioned matrix equations. For classical stationary iterative schemes, this increase in the condition number of the system leads to slow convergence, no matter how "nice" K may be [2, §4.11]. The problem is compounded whenever K exhibits large spatial variations, as can occur near lithologic changes in the porous medium or sharp contacts between fluids of different viscosity. In such problems, as we shall demonstrate, the poor conditioning associated with spatial variability typically aggravates that associated with the fine grids needed to resolve the physics of the problem. Thus, in problems with significant material heterogeneity, methods that are relatively insensitive to these two sources of poor conditioning can have considerable utility.

In this paper we discuss two iterative schemes for the mixed-method equations (1.2). The first scheme possesses convergence rates that are independent of the fineness of the grid. The second scheme, derived from the first, also overcomes the sensitivity to the spatial structure of K, at the expense of somewhat more computation per iteration. Briefly, the first scheme proceeds as follows: Let $(U^{(0)}, P^{(0)})$ be initial guesses for the value of (U, P). Then the kth iterate for (U, P) is the solution of

$$\begin{pmatrix} \omega I & N \\ N^T & 0 \end{pmatrix} \begin{pmatrix} U^{(k)} \\ P^{(k)} \end{pmatrix} = \begin{pmatrix} 0 \\ F \end{pmatrix} + \begin{pmatrix} \omega I - A & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} U^{(k-1)} \\ P^{(k-1)} \end{pmatrix}, \tag{1.3}$$

where I stands for the identity matrix and ω signifies a parameter, discussed below, that is related to the spectral radius $\rho(A)$ of A. For each iteration level k, the main computational work in (1.3) is to solve a linear system of the form $(\omega^{-1} N^T N) P^{(k)} = G^{(k-1)}$. However, the matrix $\omega^{-1} N^T N$ remains vulnerable to the poor conditioning associated with fine grids. We overcome this difficulty by using a multigrid scheme to solve for $P^{(k)}$, thereby greatly reducing the computational work in each iteration.

An interesting feature of this approach is that $N^T N$ is essentially the matrix associated with the five-point difference approximation to the Laplace operator with Dirichlet boundary conditions. Hence, the multigrid portion of the scheme does not encounter the variable coefficient, and the algorithm is particularly simple. The price paid for this simplicity, as we shall see, is sensitivity to the poor conditioning associated with spatial variability.

To overcome this second source of trouble, we modify the first scheme to get new ones of the form

$$\begin{pmatrix} D & N \\ N^T & 0 \end{pmatrix} \begin{pmatrix} U^{(k)} \\ P^{(k)} \end{pmatrix} = \begin{pmatrix} 0 \\ F \end{pmatrix} + \begin{pmatrix} D - A & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} U^{(k-1)} \\ P^{(k-1)} \end{pmatrix}, \tag{1.4}$$

where D denotes a diagonal matrix that we compute from A. This new class of schemes requires us to invert N^TDN, which we again do using a multigrid method to preserve h-independence of the convergence rate. While the multigrid method must now accommodate spatially varying coefficients, the overall scheme possesses the advantage that its convergence rate is independent of the spatial structure of K, provided K is piecewise constant on the grids of interest.

Our paper has the following format. In §2 we review the mixed finite-element method that we use. Section 3 describes the first iterative scheme in more detail and analyzes its convergence. In §4 we discuss the application of multigrid ideas to the first scheme. Much of the motivation and groundwork for the second class of iterative schemes resides in §§3 and 4. In §5 we present some numerical results for this algorithm. Section 6 describes the modifications necessary to produce the second class of iterative schemes and presents numerical results illustrating good convergence rates even in the presence of heterogeneities.

2. A mixed finite-element method. We begin with a brief review of the mixed finite-element method, following the notation of Ewing and Wheeler [8]. Let $H(\text{div}, \Omega) = \{\mathbf{v} \in L^2(\Omega) \times L^2(\Omega) : \text{div } \mathbf{v} \in L^2(\Omega)\}$. The variational form for (1.1) is as follows: Find a pair $(\mathbf{u}, p) \in H(\text{div}, \Omega) \times L^2(\Omega)$ such that

$$\begin{aligned} &\int_\Omega \frac{\mathbf{u} \cdot \mathbf{v}}{K} dx\, dy - \int_\Omega p \text{ div } \mathbf{v}\, dx\, dy = 0 \quad \forall\, \mathbf{v} \in H(\text{div}, \Omega), \\ &\int_\Omega (\text{div } \mathbf{u} - f) q\, dx\, dy = 0 \quad \forall\, q \in L^2(\Omega). \end{aligned} \tag{2.1}$$

By our assumptions on K, there exist constants $K_{\text{inf}}, K_{\text{sup}}$ such that $0 < K_{\text{inf}} \le K \le K_{\text{sup}}$. Implicit in these equations is also the assumption that K^{-1} is integrable on $\bar{\Omega}$.

To discretize the system (2.1), let $\Delta_x = \{0 = x_0 < x_1 < \cdots < x_m = 1\}$ be a set of points on the x-axis and $\Delta_y = \{0 = y_0 < y_1 < \cdots < y_n = 1\}$ a set of points on the y-axis. Let $\Delta_h = \Delta_x \times \Delta_y$ be the rectangular grid on Ω with nodes $\{(x_i, y_j)\}_{i=0,j=0}^{m,n}$. The mesh of this grid is

$$h = \max_{i,j}\{x_i - x_{i-1},\, y_j - y_{j-1}\}.$$

We assume throughout the paper that Δ_x and Δ_y are quasi-uniform in the sense that $x_i - x_{i-1} \ge \alpha h$ and $y_j - y_{j-1} \ge \alpha h$ for some fixed $\alpha \in (0, 1)$. With Δ_h we associate a finite-element subspace $\mathbf{Q}_h \times V_h$ of $H(\text{div}, \Omega) \times L^2(\Omega)$. The "velocity space" is $\mathbf{Q}_h = Q_h^x \times Q_h^y$, where Q_h^x and Q_h^y are both tensor-product spaces of one-dimensional, finite-element spaces. In particular, we use the lowest-order Raviart–Thomas spaces in which Q_h^x contains functions that are piecewise linear and continuous on Δ_x and piecewise constant on Δ_y. Similarly, Q_h^y contains functions that are piecewise linear and continuous on Δ_y and piecewise constant on Δ_x. The "pressure space" V_h consists of functions that are piecewise constant on Δ_h.

Given these approximating spaces, the corresponding mixed finite-element method for solving (2.1) is as follows: Find a pair $(\mathbf{u}_h, p_h) \in \mathbf{Q}_h \times V_h$ such that

$$\begin{aligned} &\int_\Omega \frac{\mathbf{u}_h \cdot \mathbf{v}_h}{K} dx\, dy - \int_\Omega p_h \text{ div } \mathbf{v}_h dx\, dy = 0 \qquad \forall\, \mathbf{v}_h \in \mathbf{Q}_h, \\ &\int_\Omega (\text{div } \mathbf{u}_h - f) q_h dx\, dy = 0 \qquad \forall\, q_h \in V_h. \end{aligned} \tag{2.2}$$

This finite-element discretization yields approximations $\mathbf{u}_h$ and p_h whose global errors are both $O(h)$ in the norm $\|\cdot\|_{L^2(\Omega)}$. Ewing, Lazarov, and Wang [6] also prove superconvergence results that guarantee smaller errors at special points in Ω. This phenomenon appears in our numerical examples in §5. In contrast, standard approaches solve for approximations to p and then numerically differentiate to compute $\mathbf{u} = -K \operatorname{grad} p$, thereby losing an order of accuracy in the velocity field [1].

To see the linear algebraic equations implied by (2.2), suppose $\mathbf{u}_h$ and p_h have the expansions

$$\mathbf{u}_h(x,y) = \left(\sum_{i=0}^{m} \sum_{j=1}^{n} U^x_{i,j} \phi^x_{i,j}(x,y), \sum_{i=1}^{m} \sum_{j=0}^{n} U^y_{i,j} \phi^y_{i,j}(x,y) \right),$$

$$p_h(x,y) = \sum_{i=1}^{m} \sum_{j=1}^{n} P_{i,j} \psi_{i,j}(x,y).$$

Here, $\phi^x_{i,j}$, $\phi^y_{i,j}$, and $\psi_{i,j}$ signify elements in the standard nodal bases for Q^x_h, Q^y_h, and V_h. Define the column vectors $U \in \mathbb{R}^{2mn+m+n}$, $P \in \mathbb{R}^{mn}$ containing the nodal unknowns as follows:

$$\begin{aligned} U^T = (&U^x_{0,1}, U^x_{1,1}, \cdots, U^x_{m,1}, \cdots, U^x_{0,n}, U^x_{1,n}, \cdots, U^x_{m,n}, \\ &U^y_{1,0}, U^y_{1,1}, \cdots, U^y_{1,n}, \cdots, U^y_{m,0}, U^y_{m,1}, \cdots, U^y_{m,n}), \end{aligned} \tag{2.3}$$

$$P^T = (P_{1,1}, P_{2,1}, \cdots, P_{m,1}, \cdots, P_{1,n}, P_{2,n}, \cdots, P_{m,n}).$$

Figure 1 shows how to associate these coefficients with nodes on a spatial grid Δ_h with $m = 4$, $n = 3$.

With these bases, the problem (2.2) has a matrix representation of the form

$$\begin{pmatrix} A & N \\ N^T & 0 \end{pmatrix} \begin{pmatrix} U \\ P \end{pmatrix} = \begin{pmatrix} 0 \\ F \end{pmatrix}. \tag{2.4}$$

Here A is a symmetric, positive definite matrix having the block structure

$$A = \begin{pmatrix} A^x & 0 \\ 0 & A^y \end{pmatrix},$$

in which $A^x \in \mathbb{R}^{(m+1)n \times (m+1)n}$ and $A^y \in \mathbb{R}^{m(n+1)\times m(n+1)}$ have entries of the form

$$\int_\Omega \frac{\phi^x_{i,j}\phi^x_{k,\ell}}{K} dx\, dy, \qquad \int_\Omega \frac{\phi^y_{i,j}\phi^y_{k,\ell}}{K} dx\, dy,$$

respectively. Note that these entries contain information about the spatially varying coefficient K. The matrix N has the block structure

$$N = \begin{pmatrix} N^x \\ N^y \end{pmatrix},$$

where $N^x \in \mathbb{R}^{(m+1)n\times mn}$ and $N^y \in \mathbb{R}^{m(n+1)\times mn}$ have entries given, respectively, by

$$\int_\Omega \psi_{i,j} \frac{\partial \phi^x_{k,\ell}}{\partial x} dx\, dy, \qquad \int_\Omega \psi_{i,j} \frac{\partial \phi^y_{k,\ell}}{\partial y} dx\, dy.$$

By calculating these integrals, one readily confirms that N^x and N^y reduce to the usual difference approximations to $\partial/\partial x$ and $\partial/\partial y$. The vector $F \in \mathbb{R}^{mn}$ has entries given by the integrals $\int_\Omega f \psi_{i,j} dx\, dy$. The appendix to this paper gives more detail on the construction of A and N.

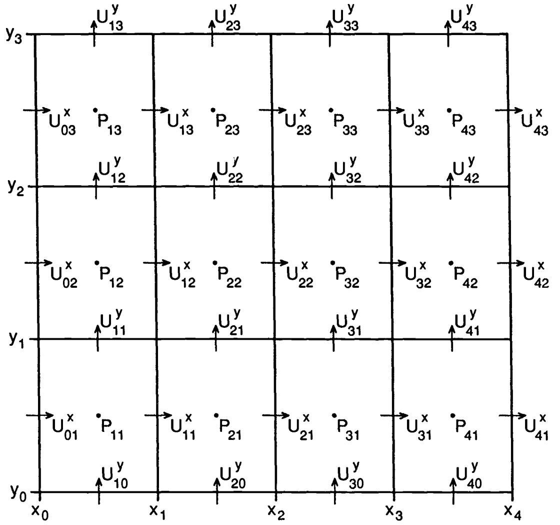


FIG. 1. *Sample* 4×3 *rectangular grid on* $\Omega = (0,1) \times (0,1)$, *showing locations of the nodal unknowns in the velocity and pressure trial functions.*

3. An h-independent iterative method.

Our first iterative scheme for solving the discrete system (2.4) is as follows.

ALGORITHM 1. Beginning with initial guess $\left(U^{(0)}, P^{(0)}\right)^T$ for (U, P), the kth iterate $\left(U^{(k)}, P^{(k)}\right)^T$ is the solution of

$$\begin{pmatrix} \omega I & N \\ N^T & 0 \end{pmatrix} \begin{pmatrix} U^{(k)} \\ P^{(k)} \end{pmatrix} = \begin{pmatrix} 0 \\ F \end{pmatrix} + \begin{pmatrix} \omega I - A & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} U^{(k-1)} \\ P^{(k-1)} \end{pmatrix}, \tag{3.1}$$

where $I \in \mathbb{R}^{(2mn+m+n)\times(2mn+m+n)}$ is the identity matrix and ω is a parameter chosen to satisfy $\omega \geq \rho(A)$.

Here, $\rho(A)$ denotes the spectral radius of the matrix A. Later in this section we discuss a practical way to pick ω that does not require detailed knowledge of the spectrum of A.

Computationally, Algorithm 1 has the following compact form: Given an initial guess $\left(U^{(0)}, P^{(0)}\right)^T$, compute $\left(U^{(k)}, P^{(k)}\right)^T$ by executing three steps:

$$\text{(i) } G^{(k-1)} \leftarrow -F + \omega^{-1} N^T (\omega I - A) U^{(k-1)}, \tag{3.2}$$

$$\text{(ii) Solve } \omega^{-1} N^T N P^{(k)} = G^{(k-1)}, \tag{3.3}$$

$$\text{(iii) } \omega U^{(k)} \leftarrow (\omega I - A) U^{(k-1)} - N P^{(k)}. \tag{3.4}$$

In each iteration, the main computational work is to solve for $P^{(k)} = \omega (N^T N)^{-1} G^{(k-1)}$. An easy calculation shows that the matrix $\omega^{-1}(N^T N)$ is positive definite, being proportional to the standard five-point, finite-difference Laplace operator applied to $P^{(k)}$. Therefore, we expect the numerical solution for $P^{(k)}$ using stationary iterative methods to be plagued by poor conditioning when the grid mesh h is small.

This observation leads us to use a multigrid scheme to get approximations to $P^{(k)}$. (In fact, any fast solver for the five-point discrete Laplacian operator would be appropriate here.) Such a device preserves the h-independence of the overall scheme's convergence rate. We discuss this facet of the algorithm in more detail in the next section. For now let us analyze the convergence properties of the overall iterative scheme, assuming an efficient "black-box" solver for $P^{(k)}$.

We begin by writing (3.1) as a stationary iterative scheme

$$\begin{pmatrix} U^{(k)} \\ P^{(k)} \end{pmatrix} = L + M \begin{pmatrix} U^{(k-1)} \\ P^{(k-1)} \end{pmatrix}, \tag{3.5}$$

where

$$L = \begin{pmatrix} \omega I & N \\ N^T & 0 \end{pmatrix}^{-1} \begin{pmatrix} 0 \\ F \end{pmatrix},$$

$$M = \begin{pmatrix} \omega I & N \\ N^T & 0 \end{pmatrix}^{-1} \begin{pmatrix} \omega I - A & 0 \\ 0 & 0 \end{pmatrix}.$$

The convergence of Algorithm 1 depends on the spectral radius of the matrix M, for which the following proposition gives a bound.

PROPOSITION 3.1. *Let*

$$0 < \lambda_{\min} \le \cdots \le \lambda_{\max} \tag{3.6}$$

be the eigenvalues of the matrix A, and let $\omega \ge \lambda_{\max}$. Then the spectral radius of M obeys the estimate

$$\rho(M) \le 1 - \frac{\lambda_{\min}}{\omega}. \tag{3.7}$$

Proof. Let $\lambda \neq 0$ be an eigenvalue of M with eigenvector $(U_\lambda, P_\lambda)^T$. Thus

$$M \begin{pmatrix} U_\lambda \\ P_\lambda \end{pmatrix} \equiv \begin{pmatrix} \omega I & N \\ N^T & 0 \end{pmatrix}^{-1} \begin{pmatrix} \omega I - A & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} U_\lambda \\ P_\lambda \end{pmatrix} = \lambda \begin{pmatrix} U_\lambda \\ P_\lambda \end{pmatrix}, \tag{3.8}$$

so

$$(\omega I - A) U_\lambda = \lambda(\omega U_\lambda + N P_\lambda), \tag{3.9a}$$

$$0 = \lambda N^T U_\lambda. \tag{3.9b}$$

Since $(U_\lambda, P_\lambda)^T \neq 0$, (3.9a) shows that $U_\lambda \neq 0$; however, U_λ may be complex. Let U_λ^H denote its Hermitian conjugate. If we multiply (3.9a) by U_λ^H, observe that N is a real matrix, and apply (3.9b), we obtain

$$\begin{aligned} U_\lambda^H (\omega I - A) U_\lambda &= \lambda \omega U_\lambda^H U_\lambda + \lambda (N^T U_\lambda)^H P_\lambda \\ &= \lambda \omega U_\lambda^H U_\lambda. \end{aligned}$$

This equation allows us to conclude that

$$0 < |\lambda| = \left| \frac{U_\lambda^H (I - \omega^{-1} A) U_\lambda}{U_\lambda^H U_\lambda} \right| \leq \rho(I - \omega^{-1} A),$$

which implies

$$\rho(M) \leq \rho(I - \omega^{-1} A). \tag{3.10}$$

Also, by (3.6) and the fact that $\omega \geq \lambda_{\max}$, we have

$$\rho(I - \omega^{-1} A) \leq 1 - \frac{\lambda_{\min}}{\omega}.$$

These last two inequalities imply the desired bound (3.7). □

If we choose $\omega = \lambda_{\max} = \rho(A)$, then the estimate (3.7) for the spectral radius of the iteration matrix M becomes

$$\rho(M) \leq 1 - \frac{\lambda_{\min}}{\lambda_{\max}}.$$

To estimate $\lambda_{\min}/\lambda_{\max}$, the following proposition is helpful.

PROPOSITION 3.2. *For the matrix A appearing in (2.4), there exist constants k_0 and k_1, independent of h, such that*

$$k_0 h^2 U^T U \leq U^T A U \leq k_1 h^2 U^T U. \tag{3.11}$$

Proof. The representation of $\mathbf{u}_h$ given in (2.3) leads to the identity

$$U^T A U = \int_\Omega \frac{1}{K} |\mathbf{u}_h|^2 dx\, dy = \sum_{i=1}^{m} \sum_{j=1}^{n} \int_{\Omega_{i,j}} \frac{1}{K} |\mathbf{u}_h|^2 dx\, dy,$$

where $\Omega_{i,j} = (x_{i-1}, x_i) \times (y_{j-1}, y_j)$. Since K is bounded and integrable on $\Omega_{i,j}$, the mean value theorem for integrals [10, pp. 184–185] guarantees the existence of a number $K_{i,j}$, satisfying $\inf_{\Omega_{i,j}} K \leq K_{i,j} \leq \sup_{\Omega_{i,j}} K$, such that

$$\int_{\Omega_{i,j}} \frac{1}{K} |\mathbf{u}_h|^2 dx\, dy = \frac{1}{K_{i,j}} \int_{\Omega_{i,j}} |\mathbf{u}_h|^2 dx\, dy.$$

(If K^{-1} is continuous on $\bar{\Omega}_{i,j}$, then K^{-1} actually assumes the value $K_{i,j}^{-1}$ somewhere on $\Omega_{i,j}$.) Calculating the last integral using our basis for $\mathbf{Q}_h$, we get

$$U^T A U = \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{a_{ij}}{6K_{ij}} \left[\begin{pmatrix} U_{i-1,j}^x \\ U_{i,j}^x \end{pmatrix}^T \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} U_{i-1,j}^x \\ U_{i,j}^x \end{pmatrix} \right.$$
$$\left. + \begin{pmatrix} U_{i,j-1}^y \\ U_{i,j}^y \end{pmatrix}^T \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} U_{i,j-1}^y \\ U_{i,j}^y \end{pmatrix} \right],$$

where a_{ij} signifies the area of $\Omega_{i,j}$. To simplify notation, we notice that the 2×2 matrix appearing in each term of this sum is positive definite. This observation allows us to define a new norm on $\mathbb{R}^2$ as follows:

$$\left\| \begin{pmatrix} U_1 \\ U_2 \end{pmatrix} \right\|_A^2 = \begin{pmatrix} U_1 \\ U_2 \end{pmatrix}^T \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} U_1 \\ U_2 \end{pmatrix}.$$

If $\|\cdot\|_2$ denotes the usual Euclidean norm on $\mathbb{R}^2$, then it is easy to check that $\|\cdot\|_2^2 \le \|\cdot\|_A^2 \le 3\|\cdot\|_2^2$. In terms of the new norm,

$$U^TAU = \sum_{i=1}^{m}\sum_{j=1}^{n} \frac{a_{ij}}{6K_{ij}} \left[\left\| \begin{pmatrix} U_{i-1,j}^x \\ U_{i,j}^x \end{pmatrix} \right\|_A^2 + \left\| \begin{pmatrix} U_{i,j-1}^y \\ U_{i,j}^y \end{pmatrix} \right\|_A^2 \right].$$

The quantity U^TU is easier to calculate:

$$U^TU = \sum_{i=0}^{m}\sum_{j=1}^{n} |U_{ij}^x|^2 + \sum_{i=1}^{m}\sum_{j=0}^{n} |U_{ij}^y|^2. \tag{3.12}$$

Now we use the bounds on K and the quasi uniformity of Δ_h to observe that

$$\begin{aligned} U^TAU &\ge \frac{\alpha^2h^2}{6K_{\sup}} \sum_{i=1}^{m}\sum_{j=1}^{n} \left[\left\| \begin{pmatrix} U_{i-1,j}^x \\ U_{i,j}^x \end{pmatrix} \right\|_A^2 + \left\| \begin{pmatrix} U_{i,j-1}^y \\ U_{i,j}^y \end{pmatrix} \right\|_A^2 \right] \\ &\ge \frac{\alpha^2h^2}{6K_{\sup}} \sum_{i=1}^{m}\sum_{j=1}^{n} \left[\left\| \begin{pmatrix} U_{i-1,j}^x \\ U_{i,j}^x \end{pmatrix} \right\|_2^2 + \left\| \begin{pmatrix} U_{i,j-1}^y \\ U_{i,j}^y \end{pmatrix} \right\|_2^2 \right] \\ &\ge \frac{\alpha^2h^2}{6K_{\sup}} \left[\sum_{i=0}^{m}\sum_{j=1}^{n} \left\| \begin{pmatrix} 0 \\ U_{i,j}^x \end{pmatrix} \right\|_2^2 + \sum_{i=1}^{m}\sum_{j=0}^{n} \left\| \begin{pmatrix} 0 \\ U_{i,j}^y \end{pmatrix} \right\|_2^2 \right] \\ &= \frac{\alpha^2h^2}{6K_{\sup}} U^TU. \end{aligned}$$

This observation establishes the first inequality in (3.11), since we can take $k_0 = \alpha^2/6K_{\sup}$. To prove the second inequality in (3.11), we rewrite (3.12) as follows:

$$\begin{aligned} U^TU = &\frac{1}{2}\sum_{i=0}^{m}\sum_{j=0}^{n} \left[\left\| \begin{pmatrix} 0 \\ U_{i,j}^x \end{pmatrix} \right\|_2^2 + \left\| \begin{pmatrix} 0 \\ U_{i,j}^y \end{pmatrix} \right\|_2^2 \right] \\ &+\frac{1}{2}\sum_{i=1}^{m+1}\sum_{j=1}^{n+1} \left[\left\| \begin{pmatrix} U_{i-1,j}^x \\ 0 \end{pmatrix} \right\|_2^2 + \left\| \begin{pmatrix} U_{i,j-1}^y \\ 0 \end{pmatrix} \right\|_2^2 \right], \end{aligned}$$

where we agree that $U_{ij}^x = 0$ if either $j = 0$ or $j = n + 1$, and $U_{ij}^y = 0$ if either $i = 0$ or $i = m + 1$. Hence,

$$\begin{aligned} U^TU &\ge \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n} \left[\left\| \begin{pmatrix} U_{i-1,j}^x \\ U_{i,j}^x \end{pmatrix} \right\|_2^2 + \left\| \begin{pmatrix} U_{i,j-1}^y \\ U_{i,j}^y \end{pmatrix} \right\|_2^2 \right] \\ &\ge \frac{1}{6}\sum_{i=1}^{m}\sum_{j=1}^{n} \left[\left\| \begin{pmatrix} U_{i-1,j}^x \\ U_{i,j}^x \end{pmatrix} \right\|_A^2 + \left\| \begin{pmatrix} U_{i,j-1}^y \\ U_{i,j}^y \end{pmatrix} \right\|_A^2 \right] \\ &\ge \frac{K_{\inf}}{h^2} U^TAU. \end{aligned}$$

We conclude that $U^TAU \le k_1h^2U^TU$, where $k_1 = 1/K_{\inf}$. □

If we apply Proposition 3.2 to the case when U is an eigenvector of A associated with the eigenvalue $\lambda_{\min}$ or $\lambda_{\max}$, respectively, we find that $\lambda_{\min} \geq \alpha^2 h^2/6K_{\sup}$ and $\lambda_{\max} \leq h^2/K_{\inf}$. Therefore, provided we choose $\omega \geq \lambda_{\max}$ in Algorithm 1, the spectral radius of our iteration matrix M obeys the bound

$$\rho(M) \leq 1 - \frac{\alpha^2 K_{\inf}}{6K_{\sup}}. \tag{3.13}$$

Notice that the right side of this inequality is a constant independent of h. This is the sense in which the convergence rate of Algorithm 1 is independent of h.

Two remarks about the practical implications of the estimate (3.13) are in order. First, the bound on $\rho(M)$ depends strongly on the nature of the coefficient $K(x, y)$. In particular, if $K_{\inf}/K_{\sup}$ is very small, reflecting a high degree of heterogeneity in the physical problem, then we can expect the actual convergence of the algorithm to be slow, albeit independent of grid mesh. Several examples in §5 confirm this expectation. Second, even though the bound (3.13) suggests choosing $\omega = \lambda_{\max}$ to accelerate iterative convergence, this choice is impractical owing to the expense of calculating $\lambda_{\max}$. In practice, we typically pick $\omega = \|A\|_\infty \geq \lambda_{\max}$. This choice is easily computable as the maximum row sum of A, and it preserves h-independence of convergence rate, even though it may be theoretically nonoptimal.

4. Application of a multigrid solver. As we have mentioned, the computation of the pressure iterate $P^{(k)}$ in step (ii) of Algorithm 1 is inefficient if we use direct schemes or classical stationary iterative methods on fine grids. However, the fact that $\omega^{-1}N^TN$ is essentially the finite-difference Laplacian operator motivates us to reduce the computational work for each iteration by calculating an approximation to the kth pressure iterate by using several cycles of a multigrid method on the system (3.3). We refer the reader to [3] for a discussion of the multigrid approach and for a Fortran code applicable in the context of our problem. The modified scheme is as follows.

ALGORITHM 2. Begin with an initial guess $\left(U^{(0)}, P^{(0)}\right)^T$, and suppose that we have computed $\left(U^{(k-1)}, P^{(k-1)}\right)^T$. Compute a new approximation $\left(U^{(k)}, P^{(k)}\right)^T$ using the following steps:

1. Compute the residual,

$$G^{(k-1)} \leftarrow -F + N^T(I - \omega^{-1}A)U^{(k-1)}. \tag{4.1}$$

2. Let $\hat{P}^{(k)}$ denote the exact solution of the problem

$$\omega^{-1}N^TN\hat{P}^{(k)} = G^{(k-1)}. \tag{4.2}$$

   Calculate an approximation $P^{(k)}$ of $\hat{P}^{(k)}$ by applying r cycles of the multigrid algorithm [3] to (4.2), using $P^{(k-1)}$ as initial guess. (We discuss the choice of r below.)

3. Compute $U^{(k)}$ as in Algorithm 1:

$$\omega U^{(k)} \leftarrow (\omega I - A)U^{(k-1)} - NP^{(k)}. \tag{4.3}$$

Multigrid methods for solving elliptic problems have an advantage that is quite relevant to the conditioning problems associated with fine grids: Each cycle has a convergence rate that is independent of h [4, Chap. 4]. Therefore, we need only show that we can choose a *fixed* number r of multigrid cycles such that each iteration of Algorithm 2 reduces the error norm by an appropriate factor close to $\rho(M)$. We do this in Proposition 4.1. Since the factor is independent of h, Algorithm 2 has convergence rate independent of h.

We begin by defining norms on the "pressure" and "velocity" spaces that will make the proof easier. Any $p_h \in V_h$ has a representation

$$p_h(x,y) = \sum_{i,j} P_{i,j}\psi_{i,j}(x,y).$$

Taking advantage of the fact that N^TN is positive definite, we compute a norm of the vector

$$P = (P_{1,1}, P_{2,1}, \cdots, P_{m,1}, \cdots, P_{1,n}, P_{2,n}, \cdots, P_{m,n})^T$$

by setting $||P||_h^2 = P^T(\omega^{-1}N^TN)P$. On the other hand, any $\mathbf{u}_h \in \mathbf{Q}_h$ has a representation

$$\mathbf{u}_h(x,y) = \left(\sum_{i,j} U^x_{i,j}\phi^x_{i,j}(x,y), \sum_{i,j} U^y_{i,j}\phi^y_{i,j}(x,y) \right).$$

We compute a norm of the vector

$$U = (U^x_{0,1}, U^x_{1,1}, \cdots, U^x_{m,1}, \cdots, U^x_{0,n}, U^x_{1,n}, \cdots, U^x_{m,n},$$

$$U^y_{1,0}, U^y_{1,1}, \cdots, U^y_{1,n}, \cdots, U^y_{m,0}, U^y_{m,1}, \cdots, U^y_{m,n})^T$$

by setting $||U||^2_\omega = \omega U^TU$.

The norm $||\cdot||_\omega$ is just a scalar multiple of the Euclidean distance function $||\cdot||_2$, and since ω is a constant related to $\rho(A)$, $||\cdot||_\omega$ is actually a discrete analog of the Euclidean norm $||\cdot||_{L^2(\Omega)\times L^2(\Omega)}$ on the velocity space by Proposition 3.2. This norm is appropriate for measuring the convergence of velocity iterates $U^{(k)}$ to the true discrete approximation U. Also, since N^TN is just the positive definite matrix associated with the five-point difference approximation to the Laplace operator, the norm $||\cdot||_h$ is appropriate for measuring the rapidity with which the pressure iterates satisfy the discrete pressure equation (3.3) as the iterations progress. Ultimately, we want to relate our results to more familiar norms such as $||\cdot||_2$ and $||\cdot||_\infty$; for this step we shall rely on the equivalence of norms for finite-dimensional Euclidean spaces.

In the following proposition, we assume $\nu = \rho(I - \omega^{-1}A) < 1$. Thus ν is an upper bound on $\rho(M)$. Suppose the multigrid iteration used to approximate $\hat{P}^{(k)}$ in step (ii) of Algorithm 1 has convergence rate $\mu \in (0,1)$. This implies that, after r multigrid cycles for $P^{(k)}$ using $P^{(k-1)}$ as initial guess,

$$\left|\left|\hat{P}^{(k)} - P^{(k)}\right|\right|_h \le \mu^r \left|\left|\hat{P}^{(k)} - P^{(k-1)}\right|\right|_h. \tag{4.4}$$

PROPOSITION 4.1. *For any $\nu' \in (\nu, 1)$, there exists a number r of multigrid cycles such that*

$$\left|\left|P - P^{(k)}\right|\right|_h + \left|\left|U - U^{(k)}\right|\right|_\omega \le \nu' \left(\left|\left|P - P^{(k-1)}\right|\right|_h + \left|\left|U - U^{(k-1)}\right|\right|_\omega \right),$$

where (P, U) is the solution of the problem (2.4) and $\left(P^{(k)}, U^{(k)}\right)$ is the approximation to (P, U) produced by the kth iteration of Algorithm 2.

Proof. Suppose we compute $\hat{U}^{(k)}$ according to (3.4) with the exact (nonmultigrid) pressure iterate $\hat{P}^{(k)}$. Thus,

$$\omega\hat{U}^{(k)} = (\omega I - A)U^{(k-1)} - N\hat{P}^{(k)}, \tag{4.5}$$

where $\hat{P}^{(k)}$ satisfies (4.2). Then from (2.4), (4.1), (4.2), and (4.5), we have

$$\omega\left(U - \hat{U}^{(k)}\right) + N\left(P - \hat{P}^{(k)}\right) = (\omega I - A)\left(U - U^{(k-1)}\right), \tag{4.6}$$

$$N^T\left(U - \hat{U}^{(k)}\right) = 0. \tag{4.7}$$

Multiplying (4.6) by $\left(U - \hat{U}^{(k)}\right)^T$ and using the identity (4.7), we get

$$\begin{aligned}\left|\left|U - \hat{U}^{(k)}\right|\right|_\omega^2 &= \left(U - \hat{U}^{(k)}\right)^T (\omega I - A)\left(U - U^{(k-1)}\right)\\ &\le \left|\left|U - \hat{U}^{(k)}\right|\right|_\omega \left|\left|(I - \omega^{-1}A)\left(U - U^{(k-1)}\right)\right|\right|_\omega\\ &\le \rho(I - \omega^{-1}A)\left|\left|U - \hat{U}^{(k)}\right|\right|_\omega \left|\left|U - U^{(k-1)}\right|\right|_\omega .\end{aligned}$$

Therefore, the velocity iterates obey the estimate

$$\left|\left|U - \hat{U}^{(k)}\right|\right|_\omega \le \nu\left|\left|U - U^{(k-1)}\right|\right|_\omega .$$

Similarly, multiplying (4.6) by $\left[\omega^{-1}N\left(P - \hat{P}^{(k)}\right)\right]^T$, we get

$$\begin{aligned}\left|\left|P - \hat{P}^{(k)}\right|\right|_h^2 &= \left(P - \hat{P}^{(k)}\right)^T N^T\omega^{-1}(\omega I - A)\left(U - U^{(k-1)}\right)\\ &\le \left|\left|\omega^{-1}N\left(P - \hat{P}^{(k)}\right)\right|\right|_\omega \left|\left|(I - \omega^{-1}A)\left(U - U^{(k-1)}\right)\right|\right|_\omega\\ &\le \left|\left|P - \hat{P}^{(k)}\right|\right|_h \rho(I - \omega^{-1}A)\left|\left|U - U^{(k-1)}\right|\right|_\omega .\end{aligned}$$

Hence, the pressure iterates obey the bound

$$\left|\left|P - \hat{P}^{(k)}\right|\right|_h \le \nu\left|\left|U - U^{(k-1)}\right|\right|_\omega .$$

Now we derive bounds on $\left|\left|P - P^{(k)}\right|\right|_h$ and $\left|\left|U - U^{(k)}\right|\right|_\omega$ in terms of their values at the previous iterative level. For $\left|\left|P - P^{(k)}\right|\right|_h$, we use the triangle equality and the multigrid estimate (4.4) to get

$$\begin{aligned}\left|\left|P - P^{(k)}\right|\right|_h &\le \left|\left|P - \hat{P}^{(k)}\right|\right|_h + \left|\left|\hat{P}^{(k)} - P^{(k)}\right|\right|_h\\ &\le \left|\left|P - \hat{P}^{(k)}\right|\right|_h + \mu^r\left|\left|\hat{P}^{(k)} - P^{(k-1)}\right|\right|_h \qquad (4.8)\\ &\le \left|\left|P - \hat{P}^{(k)}\right|\right|_h + \mu^r\left(\left|\left|P - \hat{P}^{(k)}\right|\right|_h + \left|\left|P - P^{(k-1)}\right|\right|_h\right).\end{aligned}$$

But the original iterative scheme (3.5) implies that

$$\begin{pmatrix} U - \hat{U}^{(k)} \\ P - \hat{P}^{(k)} \end{pmatrix} = M \begin{pmatrix} U - U^{(k-1)} \\ P - P^{(k-1)} \end{pmatrix}.$$

So, in light of the inequality (3.1) bounding $\rho(M)$ by ν, we have

$$\left|\left|\hat{P} - P^{(k)}\right|\right|_h \leq \rho(M) \left|\left|P - P^{(k-1)}\right|\right|_h \leq \nu \left|\left|P - P^{(k-1)}\right|\right|_h.$$

This inequality allows us to simplify (4.8), getting

$$\left|\left|P - P^{(k)}\right|\right|_h \leq (\nu + \mu^r + \nu\mu^r) \left|\left|P - P^{(k-1)}\right|\right|_h. \tag{4.9}$$

Turning to $\left|\left|U - U^{(k)}\right|\right|_\omega$, we use (4.3), multiplied by ω^{-1}, to write

$$\left(U - U^{(k)}\right) = (I - \omega^{-1}A)\left(U - U^{(k-1)}\right) + \omega^{-1}N\left(P - P^{(k)}\right).$$

This identity implies that

$$\begin{aligned} \left|\left|U - U^{(k)}\right|\right|_\omega &\leq \left|\left|(I - \omega^{-1}A)\left(U - U^{(k-1)}\right)\right|\right|_\omega + \left|\left|\omega^{-1}N\left(P - P^{(k)}\right)\right|\right|_\omega \\ &\leq \nu \left|\left|U - U^{(k-1)}\right|\right|_\omega + \left|\left|P - P^{(k)}\right|\right|_h \\ &\leq \nu \left|\left|U - U^{(k-1)}\right|\right|_\omega + (\nu + \mu^r + \nu\mu^r) \left|\left|P - P^{(k-1)}\right|\right|_h \\ &\leq (\nu + \mu^r + \nu\mu^r)\left(\left|\left|P - P^{(k-1)}\right|\right|_h + \left|\left|U - U^{(k-1)}\right|\right|_\omega\right). \end{aligned} \tag{4.10}$$

Combining the inequalities (4.9) and (4.10), we get

$$\begin{aligned} &\left|\left|P - P^{(k)}\right|\right|_h + \left|\left|U - U^{(k)}\right|\right|_\omega \\ &\qquad \leq (\nu + \mu^r + \nu\mu^r)\left(\left|\left|P - P^{(k-1)}\right|\right|_h + \left|\left|U - U^{(k-1)}\right|\right|_\omega\right). \end{aligned}$$

Since $\mu < 1$, $\mu^r + \nu\mu^r \to 0$ as $r \to \infty$. We can therefore choose r large enough so that $\nu + \mu^r + \nu\mu^r + \nu \leq \nu' < 1$. In this way,

$$\left|\left|P - P^{(k)}\right|\right|_h + \left|\left|U - U^{(k)}\right|\right|_\omega \leq \nu' \left(\left|\left|P - P^{(k-1)}\right|\right|_h + \left|\left|U - U^{(k-1)}\right|\right|_\omega\right). \qquad \square$$

In view of the norm equivalence mentioned earlier, Proposition 4.1 leads us to expect that, if we choose ω as prescribed in §3, then the computed convergence rate

$$\bar{\mu} = \lim_{k \to \infty} \left[\frac{\|P - P^{(k)}\|_\infty + \|U - U^{(k)}\|_\infty}{\|P - P^{(0)}\|_\infty + \|U - U^{(0)}\|_\infty}\right]^{1/k} \tag{4.11}$$

should be a constant independent of h as $h \to 0$. In fact, for "generic" initial guesses, the contribution from the eigenvector associated with the largest magnitude eigenvalue of M will eventually dominate the error. We therefore expect $\bar{\mu}$ to give good approximations to $\rho(M)$ in computational practice [2, p. 129].

5. Numerical examples of *h*-independence. To test our results, we apply Algorithm 2 to several versions of the following boundary-value problem:

$$\begin{aligned} -\operatorname{div}[K(x,y)\operatorname{grad} p(x,y)] &= f(x,y), && (x,y) \in \Omega, \\ p(x,y) &= 0, && (x,y) \in \partial\Omega. \end{aligned} \tag{5.1}$$

We use the lowest-order, mixed finite-element method on grids with $h = 2^{-\ell}$, where $\ell = 4, 5, 6, 7, 8$. Each iteration of the solution scheme includes $r = 2$ V-cycles of the multigrid algorithm described in [3], where the coarsest grid in each cycle has mesh 2^{-1}, and the finest has mesh $2^{-\ell}$. We use the following realizations of the coefficient $K(x,y)$:

$$\begin{aligned} K_{\text{I}}(x,y) &= 1, \\ K_{\text{II}}(x,y) &= e^{-x-y}, \\ K_{\text{III}}(x,y) &= \begin{cases} 1 & \text{if } x < y, \\ 0.1 & \text{if } x \geq y, \end{cases} \\ K_{\text{IV}}(x,y) &= K_{\text{II}}(x,y) \cdot K_{\text{III}}(x,y), \\ K_{\text{V}}(x,y) &= \begin{cases} 1 & \text{if } x < y, \\ 0.01 & \text{if } x \geq y. \end{cases} \end{aligned}$$

To confirm the convergence properties of the mixed finite-element method as $h \to 0$, we examine the exact and numerical solutions to (5.1) using $K = K_{\text{II}}$ and taking $f(x,y)$ to be the function that results when the solution is $p(x,y) = x(1-x)\sin(\pi y) + y(1-y)\sin(\pi x)$. We compute the nodal error indicators $\|U_{\text{exact}} - U\|_\infty$ and $\|P_{\text{exact}} - P\|_\infty$, where U_{exact} and P_{exact} stand for the vectors of nodal values of the exact solutions $\mathbf{u}$ and p, and U and P are vectors containing nodal values of the finite-element approximations on a uniform grid of mesh h. Figure 2 shows plots of $\log\|U_{\text{exact}} - U\|_\infty$ and $\log\|P_{\text{exact}} - P\|_\infty$ versus $\log h$ having least-squares slopes of 1.899 and 2.000, respectively. These results suggest that the nodal values of U and P are accurate to $O(h^2)$, corroborating the equal-order accuracy available in the Raviart–Thomas subspaces and indicating superconvergent nodal values in accordance with the work of Ewing, Lazarov, and Wang [6].

To check the convergence properties of the iterative scheme, we examine the behavior of the ratio $\bar{\mu}$, defined in (4.11), for each of the choices of K. Our results, shown in Fig. 3, support the expectation that, as $h \to 0$, the convergence rate of the scheme tends to a constant independent of h. Notice however that, as K exhibits more spatial variation, the convergence of the algorithm becomes slower. Any effects of variability in K on the conditioning of the discrete equations still influence this first algorithm; the only effects of poor conditioning that we have eliminated so far are those associated with grid refinement.

6. Modified schemes for heterogeneous media. To mitigate the difficulties associated with spatial variability, we modify the first iterative scheme (3.1) to get a class of new schemes having the following form.

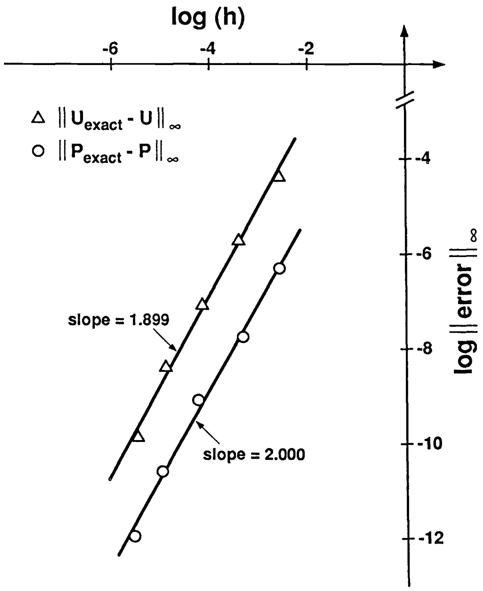


FIG. 2. *Convergence plot for the mixed finite-element scheme for Poisson's equation, using lowest-order Raviart–Thomas trial spaces. The plots demonstrate the rate of decrease in the nodal errors as* $h \to 0$.

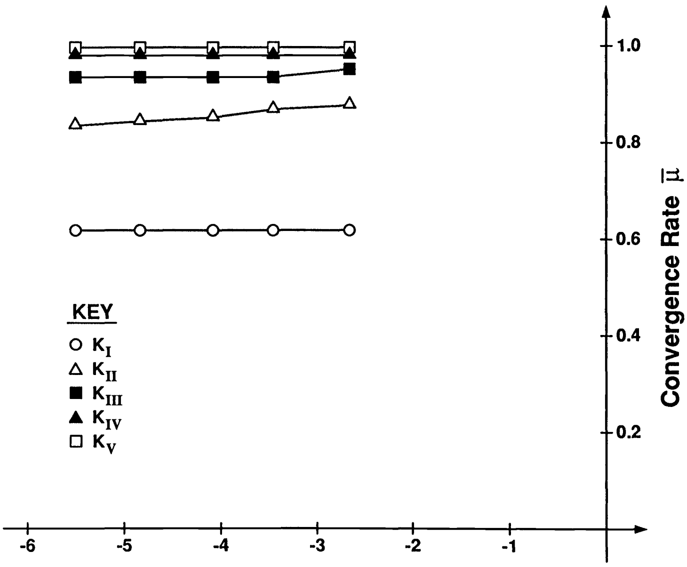


FIG. 3. *Rate of convergence* $\bar{\mu}$ *versus grid mesh* h *for Algorithm* 2, *using the various choices of coefficient* $K(x, y)$.

ALGORITHM 3. Given initial guess $\left(U^{(0)}, P^{(0)}\right)^T$, the kth iterate $\left(U^{(k)}, P^{(k)}\right)^T$ is the solution of

$$\begin{pmatrix} D & N \\ N^T & 0 \end{pmatrix} \begin{pmatrix} U^{(k)} \\ P^{(k)} \end{pmatrix} = \begin{pmatrix} 0 \\ F \end{pmatrix} + \begin{pmatrix} D-A & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} U^{(k-1)} \\ P^{(k-1)} \end{pmatrix}. \tag{6.1}$$

Here, the "preconditioning" matrix $D \in \mathbb{R}^{(2mn+m+n)\times(2mn+m+n)}$ is a diagonal matrix whose choice we discuss below.

When we construct D properly, the iteration matrix

$$M = \begin{pmatrix} D & N \\ N^T & 0 \end{pmatrix}^{-1} \begin{pmatrix} D-A & 0 \\ 0 & 0 \end{pmatrix} \tag{6.2}$$

has spectral radius that is independent of both h and the structure of K. The price we pay for this benefit is apparent in the computational form of the new algorithm:

$$\text{(i) } G^{(k-1)} \leftarrow -F + N^T D^{-1}(D-A)U^{(k-1)}, \tag{6.3}$$

$$\text{(ii) Solve } N^T D^{-1} N P^{(k)} = G^{(k-1)}, \tag{6.4}$$

$$\text{(iii) } U^{(k)} \leftarrow D^{-1}(D-A)U^{(k-1)} - D^{-1}NP^{(k)}. \tag{6.5}$$

In contrast to (3.3), solving for $P^{(k)}$ in the new scheme calls for the inversion of $N^T D^{-1} N$ instead of $N^T N$. Therefore, we must modify the multigrid segment of the algorithm to accommodate variable coefficients. As we discuss, this modification is fairly easy to make. This section establishes criteria for the construction of D, gives two examples that satisfy these criteria, comments on the multigrid solver used, and presents computational results.

As with the original scheme presented in §3, the key to the convergence of the new scheme is the spectral radius of the iteration matrix M defined in (6.2). The following proposition gives sufficient conditions under which $\rho(M) < 1$.

PROPOSITION 6.1. *Suppose D is a diagonal matrix with positive entries on the diagonal, and suppose there exist constants $b_1, b_2 \in (0,1)$ such that*

$$b_1 \le \frac{U^H A U}{U^H D U} \le 2 - b_2$$

for all vectors $U \in \mathbb{C}^{(m+1)n+m(n+1)}$. Then the iteration matrix M defined in (6.2) *satisfies*

$$0 < \rho(M) \le \max\{1-b_1, 1-b_2\} < 1. \tag{6.6}$$

Proof. Let $\lambda \neq 0$ be an eigenvalue of M with associated eigenvector $(U_\lambda, P_\lambda)^T$, as in Proposition 3.1. Then steps similar to those yielding (3.9) show that

$$\begin{aligned}(D-A)U_\lambda &= \lambda(DU_\lambda + NP_\lambda), \\ 0 &= \lambda N^T U_\lambda.\end{aligned}$$

Thus $U_\lambda^H(D-A)U_\lambda = \lambda U_\lambda^H D U_\lambda$, which is nonzero since D is positive definite. Therefore,

$$|\lambda| = \left| 1 - \frac{U_\lambda^H A U_\lambda}{U_\lambda^H D U_\lambda} \right|.$$

Hence, using the hypothesized bounds on $U_\lambda^H A U_\lambda / U_\lambda^H D U_\lambda$, we have the desired inequalities (6.6). □

To use this proposition, we need estimates on $U^H AU$. Given the structure of A as shown in the Appendix, one can calculate a useful expression for $U^H AU$, assuming $U \in \mathbb{C}^{(m+1)n+m(n+1)}$ has the form $(U^x, U^y)^T$ indicated in (2.3). In particular,

$$U^H AU = \frac{1}{3} S(U) + \frac{1}{6} R(U),$$

where, in the notation of the Appendix,

$$S(U) = \sum_{i=1}^{m} \sum_{j=1}^{n} \left(T^{\mathrm{I}}_{i,j} |U^x_{i-1,j}|^2 + T^{\mathrm{III}}_{i,j} |U^x_{i,j}|^2 + T^{\mathrm{IV}}_{i,j} |U^y_{i,j-1}|^2 + T^{\mathrm{VI}}_{i,j} |U^y_{i,j}|^2 \right),$$

$$R(U) = \sum_{i=1}^{m} \sum_{j=1}^{n} \left[T^{\mathrm{II}}_{i,j} \left(\bar{U}^x_{i,j} U^x_{i-1,j} + \bar{U}^x_{i-1,j} U^x_{i,j} \right) + T^{\mathrm{V}}_{i,j} \left(\bar{U}^y_{i,j} U^y_{i,j-1} + \bar{U}^y_{i,j-1} U^y_{i,j} \right) \right].$$

Here, $\bar{z}$ denotes the complex conjugate of z. The coefficients $T^{\mathrm{I}}_{i,j}, \cdots, T^{\mathrm{VI}}_{i,j}$ appearing in these expressions are values depending on $K(x,y)$ and arising from applications of the mean value theorem for integrals over each cell $\bar{\Omega}_{i,j}$ in the finite-element grid Δ_h. By using the inequality $|w|^2 + |z|^2 \geq -2|w||z|$, we can estimate $R(U)$ as follows:

$$\begin{aligned} -2 \sum_{i=1}^{m} \sum_{j=1}^{n} \left(T^{\mathrm{II}}_{i,j} |U^x_{i,j}| |U^x_{i-1,j}| + T^{\mathrm{V}}_{i,j} |U^y_{i,j}| |U^y_{i,j-1}| \right) &\leq R(U) \\ &\leq \sum_{i=1}^{m} \sum_{j=1}^{n} \left(T^{\mathrm{II}}_{i,j} |U^x_{i-1,j}|^2 + T^{\mathrm{II}}_{i,j} |U^x_{i,j}|^2 + T^{\mathrm{V}}_{i,j} |U^y_{i,j-1}|^2 + T^{\mathrm{V}}_{i,j} |U^y_{i,j}|^2 \right). \end{aligned} \tag{6.7}$$

In general, the estimates $0 < K_{\mathrm{inf}} \leq K \leq K_{\mathrm{sup}}$ may be too coarse to provide enough control on the coefficients $T^{\mathrm{I}}_{i,j}, \cdots, T^{\mathrm{VI}}_{i,j}$ for constructing a reasonable preconditioner D. Strictly speaking, the necessary level of control will be available only if we have information about the *local* variation of K on each cell $\bar{\Omega}_{i,j}$.

In practice, however, we rarely have such fine-scale knowledge of K, and even if we did we would not try to use it in calculating the Galerkin integrals $\int_\Omega K^{-1}\mathbf{u} \cdot \mathbf{v}\,dx\,dy$ exactly. Instead, most practical codes use approximate quadrature schemes that effectively treat K^{-1} as piecewise polynomial. In fact, as we suggested in §1, for sufficiently fine grids it is reasonable to treat K^{-1} as piecewise constant. In such applications, we can use the second inequality in (6.7), together with the identities $T^{\mathrm{II}}_{i,j} = T^{\mathrm{V}}_{i,j} = T_{i,j}$, to show that

$$U^H AU = \frac{1}{3} S(U) + \frac{1}{6} R(U) \leq \frac{1}{2} S(U).$$

Similarly, the first inequality in (6.7), together with the identities $T^{\mathrm{I}}_{i,j} = T^{\mathrm{III}}_{i,j} = T^{\mathrm{IV}}_{i,j} =$

$T^{\rm VI}_{i,j} = T_{i,j}$, shows that

$$\begin{aligned} U^H AU &= \frac{1}{6}S(U) + \frac{1}{6}[S(U) + R(U)] \\ &\geq \frac{1}{6}S(U) + \frac{1}{6}\sum_{i=1}^{m}\sum_{j=1}^{n} T_{i,j}\left[\left(|U^x_{i-1,j}| - |U^x_{i,j}|\right)^2 + \left(|U^y_{i,j-1}| - |U^y_{i,j}|\right)^2\right] \\ &\geq \frac{1}{6}S(U). \end{aligned}$$

In summary, $\frac{1}{6}S(U) \leq U^H AU \leq \frac{1}{2}S(U)$ whenever K is piecewise constant on the grid Δ_h.

Now consider the choice $D = \frac{2}{3}\text{lump}(A)$, where

$$[\text{lump}(A)]_{i,j} = \begin{cases} 0 & \text{if } i \neq j, \\ \sum\limits_j A_{i,j} & \text{if } i = j. \end{cases}$$

This is the matrix that results when we add entries along each row of A and assign the sum to the diagonal entry in that row. Gonzales and Wheeler [9] use this "mass lumping" idea to improve conditioning in mixed finite-element discretizations of petroleum reservoir problems. This choice of D is also a simple instance of a preconditioner developed in [7] for other iterative schemes. It is a straightforward matter to show that, when K is piecewise constant, $U^H\text{lump}(A)U = \frac{1}{2}S(U)$, so $U^H DU = \frac{1}{3}S(U)$. As a consequence,

$$b_1 = \frac{1}{2} \leq \frac{U^H AU}{U^H DU} \leq \frac{3}{2} = 2 - b_2.$$

Therefore, by Proposition 6.1, $\rho(M) \leq \frac{1}{2}$, and the iterative scheme converges with a rate independent of h and K. According to our remarks at the end of §4, we expect the ratio of error norms between successive iterates to approach $\frac{1}{2}$ as the iteration counter $k \to \infty$.

As an even simpler example, consider the choice $D = \text{diag}(A)$, where

$$[\text{diag}(A)]_{i,j} = \begin{cases} 0 & \text{if } i \neq j, \\ A_{i,i} & \text{if } i = j, \end{cases}$$

is the matrix A stripped of its off-diagonal entries. This choice has the attractive feature that it is trivial to compute from A. With D defined in this way, we once again find that $U^H DU = \frac{1}{3}S(U)$ when K is piecewise constant on Δ_h. Therefore, $\rho(M) \leq \frac{1}{2}$, and this iterative scheme also converges with a rate independent of h and K.

Either choice of D requires us to solve a matrix equation of the form

$$N^T D^{-1} N P^{(k)} = G^{(k-1)}$$

at each iteration. To do this, we use two cycles of a multigrid scheme in which the Jacobi iteration is the smoother, the coarse-to-fine interpolation is bilinear, and the fine-to-coarse restriction is accomplished using half-injection [4, p. 65]. This scheme preserves the h-independence of the overall algorithm's convergence rate and appears

TABLE 1
Convergence rates for various coefficients and grids.

	Grid Mesh h				
Coefficient	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
K_{I}	0.4933	0.4988	0.4993	0.4995	0.4999
K_{II}	0.4966	0.4995	0.4988	0.4997	0.4999
K_{III}	0.4948	0.4982	0.4991	0.4998	0.4999
K_{IV}	0.4947	0.4980	0.4992	0.4998	0.4999
K_{V}	0.4939	0.4978	0.4989	0.4999	0.5000

to handle the variable coefficient K effectively. Alternative multigrid implementations are certainly possible here.

To test the convergence rate of Algorithm 3, we apply it to the boundary-value problems described in §5, using the preconditioner $D = \frac{2}{3}\text{lump}(A)$. Table 1 shows values of the convergence rate $\bar{\mu}$ computed for each choice of coefficient K, for each of five different values of the grid mesh h. All of the tabulated values are very close to the spectral radius estimate $\rho(M) \leq \frac{1}{2}$. We conclude that this scheme converges at a rate independent of both grid mesh h and the heterogeneity reflected in the mobility coefficient K.

7. Conclusions. Poor conditioning associated with heterogeneity and fine spatial grids is a common problem. While this paper focuses on steady flows in porous media, similar equations and results apply in other fields. Two obvious applications for (1.1) arise in heat transfer, where temperature plays the role of pressure and heat flux plays the role of the Darcy velocity, and in electrostatics, where the electric potential and the electric field serve as the analogs of pressure and Darcy velocity, respectively. In either case, mixed finite-element methods can give useful approximations. However, heterogeneity, either in the thermal diffusivity or in the dielectric coefficient, can lead to poor conditioning in precisely the same way as it does for porous media. One virtue of the mixed finite-element formulation is that it permits us to attack the two sources of poor conditioning separately, exploiting multigrid ideas to reduce the sensitivity to fine grids and using spectral information associated with the material coefficient to reduce the sensitivity to heterogeneity.

Appendix: Matrix structure of the finite-element equations. The mixed finite-element equations (2.2) give rise to integral equations having the following forms. For the x-velocity equation,

$$\int_{\Omega} \left(K^{-1} u_h^x \, \phi_{i,j}^{x} \; - \; p_h \, \frac{\partial \phi_{i,j}^{x}}{\partial x} \right) dx \, dy = 0, \quad i = 0, \cdots, m, \quad j = 1, \cdots, n.$$

For the y-velocity equation,

$$\int_{\Omega} \left(K^{-1} u_h^y \, \phi_{i,j}^{y} \; - \; p_h \, \frac{\partial \phi_{i,j}^{y}}{\partial y} \right) dx \, dy = 0, \quad i = 1, \cdots, m, \quad j = 0, \cdots, n.$$

For the mass balance,

$$\int_{\Omega} \left(\frac{\partial u_h^x}{\partial x} + \frac{\partial u_h^y}{\partial y} - f \right) \psi_{i,j} dx \, dy = 0, \quad i = 1, \cdots, m, \quad j = 1, \cdots, n.$$

The following integrals appearing in these expressions involve no spatially varying coefficients and hence are easy to compute using the bases for $\mathbf{Q}_h$ and V_h:

$$\int_\Omega p_h \frac{\partial \phi_{i,j}^x}{\partial x} dx\, dy, \quad \int_\Omega p_h \frac{\partial \phi_{i,j}^y}{\partial y} dx\, dy, \quad \int_\Omega \frac{\partial u_h^x}{\partial x} \psi_{i,j} dx\, dy, \quad \int_\Omega \frac{\partial u_h^y}{\partial y} \psi_{i,j} dx\, dy.$$

However, the remaining integrals involve the spatially varying functions $K^{-1}(x,y)$ and $f(x,y)$. We compute these integrals using the mean value theorem for integrals [10, pp. 184–185] as follows: Since K^{-1} is bounded and integrable on each cell $\overline{\Omega}_{i,j}$, there exist numbers $T^{\rm I}_{i,j}, T^{\rm II}_{i,j}, T^{\rm III}_{i,j}$ such that

$$\int_\Omega K^{-1}\, \phi_{s,t}^x\, \phi_{i,j}^x\, dx\, dy = \begin{cases} T^{\rm II}_{i,j}/6, & t=j, \quad s=i-1; \\ (T^{\rm I}_{i,j} + T^{\rm III}_{i+1,j})/3, & t=j, \quad s=i; \\ T^{\rm II}_{i+1,j}/6, & t=j, \quad s=i+1\,. \end{cases}$$

Here, $T^{\rm I}_{i,j}/[(x_i - x_{i-1})(y_j - y_{j-1})]$ is a number lying between the upper and lower bounds of K^{-1} on the cell $\overline{\Omega}_{i,j}$, and similarly for $T^{\rm II}_{i,j}$ and $T^{\rm III}_{i,j}$. Analogous calculations show that

$$\int_\Omega K^{-1}\, \phi_{s,t}^y\, \phi_{i,j}^y\, dx\, dy = \begin{cases} T^{\rm V}_{i,j}/6, & t=j-1, \quad s=i; \\ (T^{\rm IV}_{i,j} + T^{\rm VI}_{i,j+1})/3, & t=j, \quad s=i; \\ T^{\rm V}_{i,j+1}/6, & t=j+1, \quad s=i\,. \end{cases}$$

The calculations of $\int_\Omega f\, \psi_{i,j}\, dx\, dy$ can proceed similarly.

Now let us adopt the following orderings for the vectors of unknown nodal coefficients:

$$U^x = \begin{bmatrix} U_{0,1}^x \\ \vdots \\ U_{m,1}^x \\ \vdots \\ U_{0,n}^x \\ \vdots \\ U_{m,n}^x \end{bmatrix}, \quad U^y = \begin{bmatrix} U_{1,0}^y \\ \vdots \\ U_{1,n}^y \\ \vdots \\ U_{m,0}^y \\ \vdots \\ U_{m,n}^y \end{bmatrix}, \quad P = \begin{bmatrix} P_{1,1} \\ \vdots \\ P_{m,1} \\ \vdots \\ P_{1,n} \\ \vdots \\ P_{m,n} \end{bmatrix}.$$

Then the entire algebraic system arising from (2.2) has the structure

$$\begin{bmatrix} A^x & 0 & N^x \\ 0 & A^y & N^y \\ (N^x)^T & (N^y)^T & 0 \end{bmatrix} \begin{bmatrix} U^x \\ U^y \\ P \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ F \end{bmatrix}.$$

Here,

$$A^x = \begin{bmatrix} A_1^x & & \\ & \ddots & \\ & & A_n^x \end{bmatrix} \in \mathbb{R}^{(m+1)n \times (m+1)n}\,,$$

where each block $A_j^x \in \mathbb{R}^{(m+1)\times(m+1)}$ has the tridiagonal structure

$$A_j^x = \frac{1}{6}\begin{bmatrix} 2T_{1,j}^{\mathrm{III}} & T_{1,j}^{\mathrm{II}} & & & \\ T_{1,j}^{\mathrm{II}} & 2(T_{1,j}^{\mathrm{III}} + T_{2,j}^{\mathrm{I}}) & T_{2,j}^{\mathrm{II}} & & \\ & & \ddots & & \\ & & & T_{m,j}^{\mathrm{II}} & 2T_{m,j}^{\mathrm{III}} \end{bmatrix}.$$

Similarly,

$$A^y = \begin{bmatrix} A_1^y & & \\ & \ddots & \\ & & A_m^y \end{bmatrix} \in \mathbb{R}^{m(n+1)\times m(n+1)},$$

where each block $A_i^y \in \mathbb{R}^{(n+1)\times(n+1)}$ has the tridiagonal form

$$A_i^y = \frac{1}{6}\begin{bmatrix} 2T_{i,1}^{\mathrm{VI}} & T_{i,1}^{\mathrm{V}} & & & \\ T_{i,1}^{\mathrm{V}} & 2(T_{i,1}^{\mathrm{VI}} + T_{i,2}^{\mathrm{IV}}) & T_{i,2}^{\mathrm{V}} & & \\ & & \ddots & & \\ & & & T_{i,n}^{\mathrm{V}} & 2T_{i,n}^{\mathrm{VI}} \end{bmatrix}.$$

Finally, the two "difference" matrices N^x and N^y have the following structures:

$$N^x = \begin{bmatrix} N_1^x & & \\ & \ddots & \\ & & N_n^x \end{bmatrix} \in \mathbb{R}^{n(m+1)\times nm},$$

where

$$N_j^x = (y_j - y_{j-1})\begin{bmatrix} 1 & & & \\ -1 & 1 & & \\ & -1 & 1 & \\ & & \ddots & \ddots \\ & & & 1 \\ & & & -1 \end{bmatrix} \in \mathbb{R}^{(m+1)\times m},$$

while

$$N^y = \begin{bmatrix} N_{1,1}^y & \dots & N_{1,n}^y \\ \vdots & & \\ N_{m,1}^y & \dots & N_{m,n}^y \end{bmatrix} \in \mathbb{R}^{(n+1)m\times nm},$$

where

$$N_{i,j}^{y} = (x_i - x_{i-1}) \begin{array}{cc} \left[\begin{array}{ccc} & \vdots & \\ \cdots & 1 & \cdots \\ \cdots & -1 & \cdots \\ & \vdots & \end{array}\right] & \begin{array}{l} \\ \leftarrow \text{row } j \\ \leftarrow \text{row } j+1 \\ \\ \end{array} \\ \begin{array}{c} \uparrow \\ \text{column } i \end{array} & \end{array} \in \mathbb{R}^{(n+1)\times m} .$$

REFERENCES

[1] M. B. Allen, R. E. Ewing, and J. V. Koebbe, *Mixed finite-element methods for computing groundwater velocities*, Numer. Meth. P.D.E., 3 (1985), pp. 195–207.

[2] G. Birkhoff and R. E. Lynch, *Numerical Solution of Elliptic Problems*, Society for Industrial and Applied Mathematics, Philadelphia, 1984.

[3] A. Brandt, *Multi-level adaptive solutions to boundary-value problems*, Math. Comp., 31 (1977), pp. 333–390.

[4] W. L. Briggs, *A Multigrid Tutorial*, Society for Industrial and Applied Mathematics, Philadelphia, 1987.

[5] J. Douglas, R. E. Ewing, and M. F. Wheeler, *The approximation of the pressure by a mixed method in the simulation of miscible displacement*, RAIRO Anal. Numér., 17 (1983), pp. 17–33.

[6] R. E. Ewing, R. D. Lazarov, and J. Wang, *Superconvergence of the velocities along the Gaussian lines in mixed finite element methods*, SIAM J. Numer. Anal., 28 (1991), pp. 1015–1029.

[7] R. E. Ewing, R. D. Lazarov, P. Lu, and P. S. Vassilevski, *Preconditioning indefinite systems arising from the mixed finite-element discretization of second-order elliptic systems*, in Preconditioned Conjugate Gradient Methods, Lecture Notes in Mathematics 1457, O. Axelsson and L. Kolotilina, eds., Springer-Verlag, Berlin, 1990, pp. 280–343.

[8] R. E. Ewing and M. F. Wheeler, *Computational aspects of mixed finite element methods*, in Numerical Methods for Scientific Computing, R. S. Stepelman, ed., North-Holland, Amsterdam, 1983, pp. 163–172.

[9] R. Gonzales and M. F. Wheeler, *Mixed finite element methods for petroleum reservoir engineering problems*, in Proceedings, Sixth International Conference on Computing Methods in Applied Sciences and Engineering, INRIA, Versailles, France, 1983, North-Holland, Amsterdam, 1984, pp. 639–658.

[10] M. E. Munroe, *Introduction to Measure and Integration*, Addison-Wesley, Cambridge, MA, 1953.

[11] P. A. Raviart and J. M. Thomas, *A mixed finite element method for 2nd order elliptic problems*, in Mathematical Aspects of Finite Element Methods, Lecture Notes in Mathematics 606, I. Galligani and E. Magenes, eds., Springer-Verlag, Berlin, 1977, pp. 292–315.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 3, pp. 815–825, May 1992

EFFICIENT HIGH ACCURACY SOLUTIONS WITH GMRES(m)*

KATHRYN TURNER† AND HOMER F. WALKER†

Abstract. Consideration of an abstract improvement algorithm leads to the following principle, which is similar to that underlying iterative refinement: By making judicious use of relatively few high accuracy computations, high accuracy solutions can be obtained very efficiently by the algorithm. This principle is applied specifically to GMRES(m) here; it can be similarly applied to a number of other "restarted" iterative linear methods as well. Results are given for numerical experiments in solving a discretized linear elliptic boundary value problem and in computing a step of an inexact Newton method using finite differences for a discretized nonlinear elliptic boundary value problem.

Key words. GMRES(m), "restarted" iterative linear methods, high accuracy solutions, iterative refinement, large-scale linear and nonlinear systems, elliptic boundary value problems

AMS(MOS) subject classifications. 65F10, 65H10

1. Introduction. Our specific objective is to outline efficient algorithms for obtaining high accuracy with GMRES(m), the restarted version of the generalized minimal residual (GMRES) method of Saad and Schultz [10] for numerically solving general nonsingular linear systems. These algorithms are based on general observations which have broader applicability and are of interest in their own right, and we begin with them.

We consider a very abstract *improvement algorithm* for a problem in which the solution $x^* \in \mathbb{R}^n$ is characterized as a zero of a *residual function* $r : \mathbb{R}^n \to \mathbb{R}^n$, i.e., $r(x^*) = 0$. The algorithm centers around an unspecified *correction process* which, given a current approximate solution x, computes a correction z such that $x + z$ is, one hopes, an improved approximation of x^*. Our presumption is that the accuracy in the computed value of z is limited by the accuracy in the computed value of $r(x)$. Therefore, we explicitly require in the algorithm that $r(x)$ be computed accurately, by which we mean as accurately as is reasonably possible or affordable, or in some sense especially accurately. For similar reasons, on which we expand in the following, we require $x + z$ to be computed accurately in the same sense.

Algorithm GI: General Improvement Algorithm

```
LET AN INITIAL $x$ BE GIVEN.
UNTIL TERMINATION, DO:
  COMPUTE $r(x)$ ACCURATELY.
  DETERMINE $z$ BY THE CORRECTION PROCESS.
  UPDATE $x \longleftarrow x + z$ ACCURATELY.
END DO.
```

The termination criterion may be based on the smallness of $\|r(x)\|$, or on some related but different feature, as in the iterative refinement example below.

Our interest is in the limits of accuracy which can be obtained by Algorithm GI. Since the actual error in x is unknown, we take "limits of accuracy" to mean limits of reduction of $\|r(x)\|$, where $\|\cdot\|$ is an appropriate norm on $\mathbb{R}^n$. If we assume that

* Received by the editors May 29, 1990; accepted for publication (in revised form) February 6, 1991.

† Department of Mathematics and Statistics, Utah State University, Logan, Utah 84322-3900. The work of the second author was supported by United States Department of Energy grant DE-FG02-86ER25018, National Science Foundation grant DMS-0088995, and United States Air Force grant AFOSR-91-0294, all with Utah State University.

the accuracy in the computed value of z is limited only by the relative error in the computed value of $r(x)$, and not by the smallness of $\|r(x)\|$ per se, then it is clear that accuracy is limited only by the accuracy which can be maintained in computing $r(x)$ and in updating $x \longleftarrow x + z$.

A familiar example of Algorithm GI is *iterative refinement* for improving approximate solutions of a system of linear equations. (See, e.g., Golub and Van Loan [7, §3.5.3] as a general reference.) Consider a linear problem written as follows:

$$\text{(1)} \qquad \text{Given } A \in \mathbb{R}^{n\times n} \text{ and } b \in \mathbb{R}^n, \quad \text{find } x^* \in \mathbb{R}^n \text{ such that } Ax^* = b.$$

Suppose that we have computed factors of A and have used them to compute an initial x. In iterative refinement, we take $r(x) \equiv b - Ax$ and, at each step, determine a correction z by solving $Az = r(x)$ using the previously computed factors of A. Iterative refinement is typically applied when we can afford to do most computing only in single precision, but full single precision accuracy is desired in the computed solution. Thus A, b, x, $r(x)$, and z are stored in single precision, and the factors of A are computed and stored in single precision. The accuracy in z depends on the relative error in the computed value of $r(x)$ and not on the smallness of $\|r(x)\|$, provided all computations remain within the floating point range of the machine. However, if $r(x)$ is small, then special care must be taken to compute it accurately. The usual prescription is to compute it in double precision in a well-known way. Unless A is very ill conditioned, this results in sufficient accuracy in $r(x)$ and z to provide improvement in x until z is so small that x and $x + z$ have the same single precision representation, i.e., the updating $x \longleftarrow x + z$ results in no change, at which point x is the single precision representation of the exact solution and the algorithm terminates. Since only single precision accuracy is desired in x, there is no need to form $x + z$ with more than single precision accuracy. If higher accuracy *were* desired in x, then it could be obtained by maintaining x in double precision and forming $x + z$ with z represented in double precision; however, this is unlikely to be of practical interest.

This discussion illustrates our guiding principle: By making judicious use of relatively few high accuracy calculations within Algorithm GI, specifically in computing $r(x)$ and in updating $x \longleftarrow x + z$, we can obtain essentially the accuracy that would be realized by using high accuracy calculations throughout, while maintaining storage requirements and execution times only modestly above those which would be required if only low accuracy calculations were used.

We now turn to the case of particular interest, viz., that of problem (1) in which $r(x) \equiv b - Ax$ and the correction process consists of one cycle of GMRES(m). Although we focus exclusively on GMRES(m), we emphasize that Algorithm GI and the guiding principle drawn from it are more broadly applicable. In particular, there are a number of other "restarted" iterative linear methods which could be used in place of GMRES(m) in the algorithms formulated below, e.g., a restarted version of a method of Axelsson [1], the iterative Arnoldi method of Saad [9, §3.2], and GCR(m) given by Elman [5] and Eisenstat, Elman, and Schultz [4].

For completeness, we give very brief descriptions of GMRES and GMRES(m) here; for more detailed descriptions, see Saad and Schultz [10] or Walker [12], [13]. The GMRES method begins with an initial approximate solution x and initial residual $r(x) = b - Ax$. At the kth iteration, a correction z is characterized as the solution of the least-squares problem

$$\text{(2)} \qquad \min_{\bar{z}\in\mathcal{K}_k} \|b - A(x + \bar{z})\|_2 = \min_{\bar{z}\in\mathcal{K}_k} \|r(x) - A(\bar{z})\|_2,$$

where $\mathcal{K}_k$ is the kth Krylov subspace generated by A and $r(x)$, defined by

$$\mathcal{K}_k \equiv \text{span}\{r(x), Ar(x), \cdots, A^{k-1}r(x)\}.$$

The correction z is not actually computed by the method unless a termination criterion is satisfied. If it is computed, then the least-squares problem (2) is solved by using a basis of $\mathcal{K}_k$ to reduce (2) to a k-dimensional least-squares problem for the coefficients of the basis elements. The basis used in practice is the orthonormal *Arnoldi basis*, generated inductively by

$$\begin{aligned} v_1 &= r(x)/\|r(x)\|_2, \\ v_{j+1} &= P^{\perp}_{\mathcal{K}_j} Av_j/\|P^{\perp}_{\mathcal{K}_j} Av_j\|_2 \quad \text{for } j = 1, \cdots, k-1. \end{aligned} \tag{3}$$

Here, $P^{\perp}_{\mathcal{K}_j} v$ denotes the orthogonal projection of a vector v onto the orthogonal complement of $\mathcal{K}_j$; it can be computed using either the modified Gram–Schmidt process [10] or Householder transformations [12]. We use the modified Gram–Schmidt process below since it is used more often in practice. Note that the process (3) fails for some $j < k$ if and only if $P^{\perp}_{\mathcal{K}_j} Av_j = 0$, i.e., $Av_j \in \mathcal{K}_j$, in which case it can be shown that GMRES produces $z \in \mathcal{K}_j$ such that $r(x + z) = 0$, i.e., $x + z = x^*$ (see [10, Prop. 2, p. 865]). The process (3) is implemented as the GMRES iterations proceed, i.e., v_1 is determined and stored initially and at the kth iteration, (only) v_{k+1} is generated and added to storage. At the kth iteration, $k + 1$ n-vectors must be stored, and orthogonalizing Av_k against $v_1, \cdots, v_k$ requires $O(kn)$ arithmetic operations.

Because storage and arithmetic costs increase as the GMRES iterations proceed, the algorithm usually implemented in practice is GMRES(m), where m is some maximum allowable number of iterations which is prescribed at the outset, typically much less than n. In one cycle of GMRES(m), beginning with a current approximate solution x, up to m GMRES iterations are performed; the iterations are terminated when either the residual norm has been reduced sufficiently or the full m iterations have been taken. The residual norm can be monitored during the iterations without actually computing the correction or the residual; see below. After the iterations have been performed, the cycle is concluded with the computation of the correction z. If $\|r(x + z)\|_2$ is not sufficiently small, then additional cycles may be performed as needed.

We give below a detailed outline of one cycle of GMRES(m) as used here. In it, for each i, J_i denotes a Givens rotation which "rotates" components i and $i + 1$ of vectors on which it acts; see, e.g., [7, §5.1] for definitions and properties of Givens rotations. Also, for each k, $(J_k\rho)_{k+1}$ denotes the $(k + 1)$st component of the vector $J_k\rho$ and $w_1, \cdots, w_{k+1}$ denote the first $k + 1$ components of the vector w.

Algorithm: One Cycle of GMRES(m)

```
LET $r(x) \neq 0$, $m$, AND TOL$> 0$ BE GIVEN.
INITIALIZE:
    SET $v_1 = r(x)/\|r(x)\|_2$ AND $w = (\|r(x)\|_2, 0, \cdots, 0)^T \in \mathbb{R}^{m+1}$.
ITERATE:
    FOR $k = 1, 2, \cdots, m$, DO:
        EVALUATE $v_{k+1} = Av_k$.
        FOR $i = 1, \cdots, k$, DO:
            SET $\rho_i = v_i^T v_{k+1}$.
            OVERWRITE $v_{k+1} \longleftarrow v_{k+1} - \rho_i v_i$.
        END DO.
```

```
            SET $\rho_{k+1} = \|v_{k+1}\|_2$.
            IF $\rho_{k+1} \neq 0$, OVERWRITE $v_{k+1} \longleftarrow v_{k+1}/\rho_{k+1}$.
            SET $\rho = (\rho_1, \cdots, \rho_{k+1}, 0, \cdots, 0)^T \in \mathbb{R}^{m+1}$.
            IF $k > 1$, OVERWRITE $\rho \longleftarrow J_{k-1} \cdots J_1 \rho$.
            FIND $J_k$ SUCH THAT $(J_k\rho)_{k+1} = 0$ AND OVERWRITE
              $\rho \longleftarrow J_k\rho$ AND $w \longleftarrow J_k w$.
            SET $R_k = [\rho]$ IF $k = 1$ AND $R_k = [R_{k-1}, \rho]$ IF $k > 1$.
            IF $|w_{k+1}|$ <TOL, THEN GO TO SOLVE.
        END DO.
    SOLVE:
        LET $k$ BE THE FINAL ITERATION NUMBER REACHED, LET
          $\bar{R}_k \in \mathbb{R}^{k\times k}$ BE THE UPPER TRIANGULAR MATRIX DETERMINED
          BY THE FIRST $k$ ROWS OF $R_k$, AND SET $\bar{w} = (w_1, \cdots, w_k)^T \in \mathbb{R}^k$.
        SOLVE $\bar{R}_k y = \bar{w}$ FOR $y$.
        FORM $z = [v_1, \cdots, v_k]y$.
```

The test $|w_{k+1}|$ <TOL at the final step of the iteration is explained by the fact that if the correction z were formed, then we would have $\|r(x+z)\|_2 = |w_{k+1}|$, at least in exact arithmetic; see [10, Prop. 1, p. 862]. Also, we note that $\bar{R}_k$ is always nonsingular.

Our general efficient high accuracy algorithm centering around GMRES(m) is the following adaptation of Algorithm GI.

Algorithm EHA: Efficient High Accuracy Algorithm

```
    LET AN INITIAL $x$ BE GIVEN. UNTIL TERMINATION, DO:
        COMPUTE $r(x)$ ACCURATELY.
        DETERMINE $z$ BY ONE CYCLE OF GMRES($m$).
        UPDATE $x \longleftarrow x + z$ ACCURATELY.
    END DO.
```

Algorithm EHA terminates when the residual norm becomes less than a given tolerance TOL> 0. If $\|r(x)\|_2$ <TOL prior to beginning a GMRES(m) cycle, then the algorithm terminates immediately; if this criterion is met during a GMRES(m) cycle, then the algorithm terminates as soon as the updating $x \longleftarrow x + z$ is done.

In the following, we explore applications of Algorithm EHA. In §2, we show how the steps in Algorithm EHA can be specified so that essentially all of the accuracy obtainable from a full double precision implementation of GMRES(m) can be obtained with storage requirements and arithmetic costs which, in a typical problem, are only a little greater than those of a single precision implementation. In §3, we consider the computation of inexact Newton steps for a nonlinear problem by means of Algorithm EHA, in which products of the Jacobian (matrix) with vectors are approximated by finite differences. We show that by using high-order finite difference approximations in the computation of each $r(x)$ and low-order finite difference approximations in each cycle of GMRES(m), we can obtain essentially the accuracy that could be obtained by using high-order finite difference approximations throughout, but at much less cost.

2. Obtaining double precision accuracy efficiently. Here we show how Algorithm EHA can be used to obtain essentially double precision accuracy efficiently in the solution of a linear problem (1). By double precision accuracy, we mean the accuracy which could be obtained by using double precision arithmetic throughout; we do not mean that the result will be the exact solution rounded to double precision.

In applying Algorithm EHA, we use three double precision vectors: one for the approximate solution x, one for the right-hand side b, and one for the matrix-vector product Ax. A subroutine to compute Ax in double precision is needed in addition to a subroutine that is used within GMRES(m) to compute products of A with other vectors in single precision. We apply Algorithm EHA as follows: Given a double precision approximate solution x, we first compute $r(x)$ by computing Ax in double precision, subtracting Ax from b in double precision, and then rounding the difference to single precision. A cycle of GMRES(m) is then carried out, entirely in single precision, producing a single precision correction z. The update of the approximate solution is $x \longleftarrow x + \text{dble}\,(z)$, where $\text{dble}\,(z)$ indicates that z is represented in double precision in the addition. This representation is done componentwise and does not require that z be converted in toto to a double precision vector.

2.1. Numerical experiments. We conducted numerical experiments on nonsymmetric linear systems arising from the discretization of the elliptic boundary value problem

$$\begin{aligned} \Delta w + cw + d\frac{\partial w}{\partial x} &= f \qquad \text{in } D, \\ w &= 0 \qquad \text{on } \partial D, \end{aligned} \tag{4}$$

where $D = [0,1] \times [0,1]$ and $c \geq 0$ and d are constants. In the experiments reported here, we took $f \equiv 1$ and used a 100×100 mesh of equally spaced discretization points in D, so that the resulting linear systems were of dimension $10,000$. Discretization was by the usual second-order centered differences.

We compared Algorithm EHA as specified above with methods which differed only in that either exclusively single or exclusively double precision arithmetic and storage were used throughout. We refer to the latter two methods as the full single precision (FSP) and full double precision (FDP) methods, respectively. We note that in the context of these experiments, Algorithm EHA requires only a little more storage than the FSP method, which requires about half that of the FDP method. The additional storage required by Algorithm EHA over the FSP method arises mainly from the double precision storage of b, x, and Ax. In the double precision computation of Ax, the simplicity of A in these experiments allows the nonzero entries of A to be represented by a few nonzero constants.

In GMRES(m), we used $m = 10$ in all of the experiments reported here because that seemed to be more effective than other choices we tried. All computing was done on a Sun Microsystems SPARCstation 1 using the SunOS FORTRAN compiler.

In our first set of experiments, we took $c = d = 10$ and used right preconditioning with a fast Poisson solver from FISHPACK [11], which is very effective for these fairly small values of c and d. We first started each method with zero as the initial approximate solution and allowed it to run for 40 GMRES(m) iterations, after which the limit of residual norm reduction had been reached. Figure 1 shows plots of the logarithm of the Euclidean norm of the residual versus the number of GMRES(m) iterations for the three methods. We note that in Fig. 1 and in all other figures below, the plotted residual norms were not the values maintained by GMRES(m), but rather were computed as accurately as possible "from scratch." That is, at each GMRES(m) iteration, the current approximate solution was formed and its product with the coefficient matrix was subtracted from the right-hand side, all in double precision. It was important to compute the residual norms in this way because the values maintained by GMRES(m) become increasingly untrustworthy as the limits of residual norm reduction are neared; see [12]. It is seen in Fig. 1 that Algorithm EHA achieved the same ultimate level of residual norm reduction as the FDP method and required only a few more GMRES(m) iterations to do so.

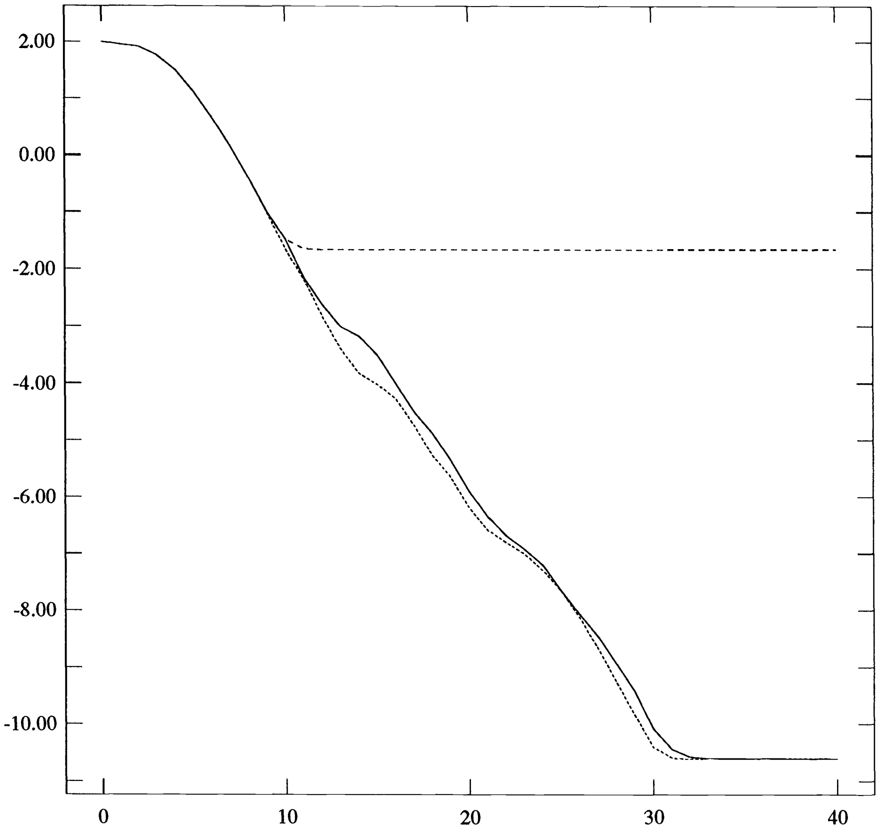


FIG. 1. Log$_{10}$ *of the residual norm versus the number of* GMRES(m) *iterations for* $c = d = 10$ *with fast Poisson preconditioning. Solid curve: Algorithm* EHA*; dotted curve:* FDP *method; dashed curve:* FSP *method.*

In order to assess the relative amount of computational work required by Algorithm EHA and the FDP method to reach comparable levels of residual norm reduction, we observed in each of 20 trials the GMRES(m) iteration numbers and run times required by each method to reduce the residual norm by a factor of 10^{-12}. Since timing was of major interest, the residual norms were not computed "from scratch" in these trials; instead, the values maintained by GMRES(m) were used. These values were trustworthy because 10^{-12} is significantly larger than the limiting residual norm reduction factor for these methods evident in Fig. 1. In each trial, the components of the initial approximate solution were obtained by generating uniformly distributed random numbers over $[-1, 1]$. The means and standard deviations of the run times are given in Table 1. Each method required exactly 30 GMRES(m) iterations to reach termination in every trial, which accounts for the small standard deviations.

In our second set of experiments, we took $c = d = 100$ and carried out trials analogous to those in the first set above. No preconditioning was used in these experiments, both because we wanted to compare the methods without preconditioning and because the fast Poisson preconditioning used in the first set of experiments is not

TABLE 1

Statistics over 20 *trials of run times required to reduce the residual norm by a factor of* 10^{-12}. *Fast Poisson preconditioning;* $c = d = 10$. *Each method took* 30 GMRES(m) *iterations to terminate in every trial.*

Method	Mean Run Time (Seconds)	Standard Deviation
EHA	28.77	.03125
FDP	45.80	.06442

cost effective for these large values of c and d. We first allowed each method to run for 600 GMRES(m) iterations, starting with zero as the initial approximate solution, after which the limit of residual norm reduction had been reached. The results are shown in Fig. 2. We note that Algorithm EHA reached the same ultimate level of residual norm reduction as the FDP method but required about ten percent more GMRES(m) iterations to reach this level.

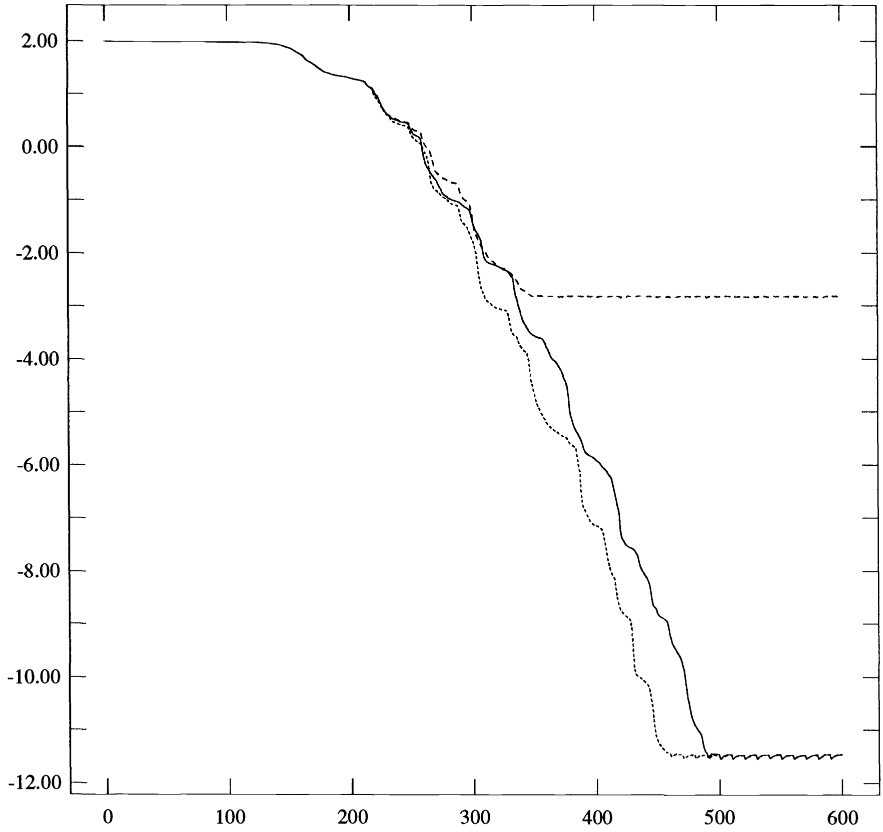


FIG. 2. Log$_{10}$ *of the residual norm versus the number of* GMRES(m) *iterations for* $c = d = 100$ *with no preconditioning. Solid curve: Algorithm* EHA; *dotted curve:* FDP *method; dashed curve:* FSP *method.*

We then observed in each of 20 trials the numbers of GMRES(m) iterations and run times required by Algorithm EHA and the FDP method to reduce the residual norm by a factor of 10^{-12}. In these trials, the initial approximate solutions were obtained by generating random components as in the previous such trials. In contrast

TABLE 2
Statistics over 20 *trials of* GMRES(m) *iteration numbers and run times required to reduce the residual norm by a factor of* 10^{-12}. *No preconditioning;* $c = d = 100$.

Method	Mean Number of Iterations	Standard Deviation	Mean Run Time (Seconds)	Standard Deviation
EHA	346.2	34.18	91.57	9.068
FDP	345.9	35.56	153.0	15.72

to the previous trials, there was in these trials significant variation in both the numbers of GMRES(m) iterations and the run times for each method. Consequently, the means and standard deviations of the GMRES(m) iteration counts as well as the run times are given in Table 2.

3. Computing inexact Newton steps using finite differences.

We now consider an inexact Newton method [3] for the solution of a nonlinear problem, written as follows:

$$\text{Given } F : \mathbb{R}^n \to \mathbb{R}^n, \quad \text{find } u^* \in \mathbb{R}^n \text{ such that } F(u^*) = 0. \tag{5}$$

Given an approximate solution u of (5), an inexact Newton method determines a step x such that $F'(u)x \approx -F(u)$ and updates $u \longleftarrow u + x$; such a method is of interest when computing the exact Newton step $-F'(u)^{-1}F(u)$ is infeasible. We shall apply Algorithm EHA to the computation of a single inexact Newton step, in which u remains fixed and $r(x) \equiv -F(u) - F'(u)x$.

We focus on the case in which $F'(u)$ cannot be used analytically. In this case, we might approximate $F'(u)$ using finite differences. However, with GMRES(m), $F'(u)$ or an approximation of it is not explicitly needed; we require only its action on vectors v, which can be approximated, e.g., to first, second, fourth, and sixth order, respectively, by

$$\frac{1}{\delta}[F(u+\delta v) - F(u)], \tag{6}$$

$$\frac{1}{2\delta}[F(u+\delta v) - F(u-\delta v)], \tag{7}$$

$$\frac{1}{6\delta}\left[8F\left(u+\frac{\delta}{2}v\right) - 8F\left(u-\frac{\delta}{2}v\right) - F(u+\delta v) + F(u-\delta v)\right], \tag{8}$$

$$\begin{aligned}\frac{1}{90\delta}\Bigg[256F\left(u+\frac{\delta}{4}v\right) - 256F\left(u-\frac{\delta}{4}v\right) - 40F\left(u+\frac{\delta}{2}v\right)\\ + 40F\left(u-\frac{\delta}{2}v\right) + F(u+\delta v) - F(u-\delta v)\Bigg].\end{aligned} \tag{9}$$

In an inexact Newton method, $F(u)$ is already available; therefore, each of (6)–(9) requires a number of new F-evaluations equal to its order.

We apply Algorithm EHA in this context by using a higher-order formula, e.g., one of (7)–(9), in the residual computation preceding each cycle of GMRES(m) and using a lower-order formula, e.g., the first-order formula (6), for matrix-vector products required within GMRES(m).

3.1. Numerical experiments. We conducted numerical experiments in computing inexact Newton steps for discretizations of a *modified Bratu problem*, given by

$$\begin{aligned} \Delta w + ce^{w} + d\frac{\partial w}{\partial x} &= f \qquad \text{in } D, \\ w &= 0 \qquad \text{on } \partial D, \end{aligned} \tag{10}$$

where c and d are constants. The actual Bratu problem has $d = 0$ and $f \equiv 0$. It provides a simplified model of nonlinear diffusion phenomena, e.g., in combustion and semiconductors, and has been considered by Glowinski, Keller, and Rheinhardt [6], as well as by a number of other investigators; see [6] and the references therein. See also problem 3 by Glowinski and Keller and problem 7 by Mittelmann in the collection of nonlinear model problems assembled by Moré [8]. The modified problem (10) has been used as a test problem for inexact Newton methods by Brown and Saad [2].

In our experiments, we took $D = [0,1] \times [0,1]$, $f \equiv 0$, $c = d = 10$, and discretized (10) using the usual second-order centered differences over a 100×100 mesh of equally spaced points in D. In GMRES(m), we took $m = 10$ and used fast Poisson right preconditioning as in the experiments in §2. The computing environment was as described in §2. All computing was done in double precision.

Letting F denote the nonlinear function obtained by discretizing (10) and letting u denote an approximate solution of the discretized problem, we used (6)–(9) in approximating Jacobian-vector products $F'(u)v$. The value of δ used for each formula was chosen according to the usual heuristic when F is evaluated to full machine precision: If p is the order of the formula, then $\delta = \mathbf{u}^{1/(p+1)}$, where $\mathbf{u}$ is unit roundoff (taken to be 10^{-16} here).

The methods we compared in our experiments are the following:

1. Four methods, each of which used one of (6), (7), (8), or (9) exclusively for all Jacobian-vector products. We refer to these below as FD1, FD2, FD4, and FD6, respectively.
2. Three versions of Algorithm EHA, each of which used one of (7), (8), or (9) exclusively in the accurate evaluation of initial residuals and then used (6) exclusively in the GMRES(m) cycles. We refer to these below as EHA2, EHA4, and EHA6, respectively.
3. A method in which all Jacobian-vector products were evaluated analytically. We refer to this as method A.

In the first set of experiments, we allowed each method to run for 40 GMRES(m) iterations, starting with zero as the initial approximate solution, after which the limit of residual norm reduction had been reached. The results are shown in Fig. 3. In Fig. 3, the top curve was produced by method FD1. The second curve from the top is actually a superposition of the curves produced by methods EHA2 and FD2; the two curves are visually indistinguishable. Similarly, the third curve from the top is a superposition of the curves produced by methods EHA4 and FD4, and the fourth curve from the top, which lies barely above the bottom curve, is a superposition of the curves produced by methods EHA6 and FD6. The bottom curve was produced by method A.

In the second set of experiments, our purpose was to assess the relative amount of computational work required by the methods which use higher-order differencing to reach comparable levels of residual norm reduction. We compared pairs of methods EHA2 and FD2, EHA4 and FD4, and EHA6 and FD6 by observing in each of 20

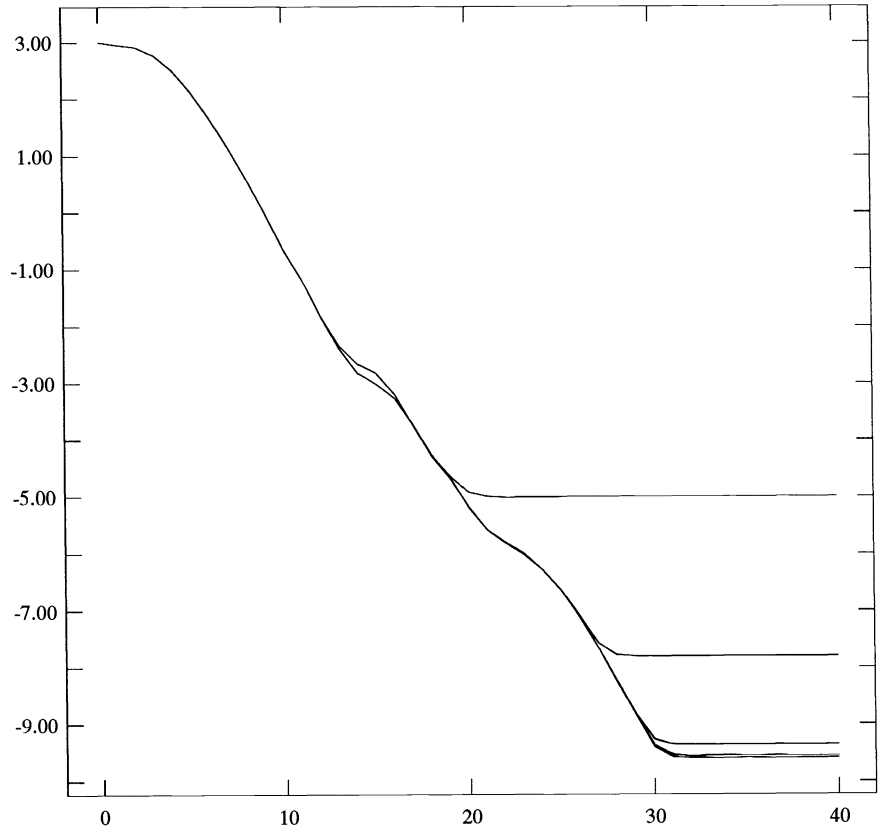


FIG. 3. Log_{10} *of the residual norm versus the number of* GMRES(m) *iterations for the finite difference methods.*

TABLE 3
Statistics over 20 *trials of* GMRES(m) *iteration numbers, F-evaluations, and run times required to reduce the residual norm by a factor of ϵ. For each method, the number of* GMRES(m) *iterations and F-evaluations was the same in every trial.*

Method	ϵ	Number of Iterations	Number of F-Evaluations	Mean Run Time (Seconds)	Standard Deviation
EHA2	10^{-10}	26	32	47.12	.1048
FD2	10^{-10}	26	58	53.79	.1829
EHA4	10^{-12}	30	42	56.76	.1855
FD4	10^{-12}	30	132	81.35	.3730
EHA6	10^{-12}	30	48	58.56	.1952
FD6	10^{-12}	30	198	100.6	.3278

trials the number of GMRES(m) iterations, number of F-evaluations, and run time required by each method to reduce the residual norm by a factor of ϵ, where for each pair of methods ϵ was chosen to be somewhat greater than the limiting ratio of final to initial residual norms obtainable by the methods. In these trials, the initial approximate solutions were obtained by generating random components as in the similar experiments in §2. We note that for every method, the numbers of GMRES(m) iterations and F-evaluations required before termination did not vary at all over the 20 trials. The GMRES(m) iteration counts, numbers of F-evaluations, and means and standard deviations of the run times are given in Table 3.

Acknowledgments. We thank John Dennis and Gene Golub for stimulating conversations relating to this work.

REFERENCES

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

[2] P. N. Brown and Y. Saad, *Hybrid Krylov methods for nonlinear systems of equations*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 450–481.

[3] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, *Inexact Newton methods*, SIAM J. Numer. Anal., 19 (1982), pp. 400–408.

[4] S. C. Eisenstat, H. C. Elman, and M. H. Schultz, *Variational iterative methods for nonsymmetric systems of linear equations*, SIAM J. Numer. Anal., 20 (1983), pp. 345–357.

[5] H. C. Elman, *Iterative methods for large, sparse, nonsymmetric systems of linear equations*, Ph.D. thesis, Department of Computer Science, Yale University, New Haven, CT, 1982.

[6] R. Glowinski, H. B. Keller, and L. Rheinhart, *Continuation-conjugate gradient methods for the least-squares solution of nonlinear boundary value problems*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 793–832.

[7] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Second Edition, The Johns Hopkins University Press, Baltimore, MD, 1989.

[8] J. J. Moré, *A collection of nonlinear model problems*, in Computational Solutions of Nonlinear Systems of Equations, E. L. Allgower and K. Georg, eds., Lectures in Applied Mathematics, Vol. 26, American Mathematical Society, Providence, RI, 1990, pp. 723–762.

[9] Y. Saad, *Krylov subspace methods for solving large unsymmetric linear systems*, Math. Comp., 37 (1981), pp. 105–126.

[10] Y. Saad and M. H. Schultz, GMRES*: A generalized minimal residual method for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.

[11] P. N. Swarztrauber and R. A. Sweet, *Efficient* FORTRAN *subprograms for the solution of elliptic partial differential equations*, ACM Trans. Math. Software, 5 (1979), pp. 352–364.

[12] H. F. Walker, *Implementation of the* GMRES *method using Householder transformations*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 152–163.

[13] ———, *Implementations of the* GMRES *method*, Computer Phys. Comm., 53 (1989), pp. 311–320.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 3, pp. 826–839, May 1992

A GRID-BASED SUBTREE-SUBCUBE ASSIGNMENT STRATEGY FOR SOLVING PARTIAL DIFFERENTIAL EQUATIONS ON HYPERCUBES*

MO MU† AND J. R. RICE†

Abstract. The authors propose a grid-based subtree-subcube assignment strategy for solving PDE problems on hypercubes. A complexity analysis of the communication is given for the proposed approach and the standard subtree-subcube assignment when applied to a variant of nested dissection indexing and a nonsymmetric sparse solver. The new assignment reduces communication cost using p processors by a factor of $O(\log\ p)$ in message startups and a factor of about two in traffic volume. Using a modified nested dissection indexing, the total effect on message startups is a reduction by a factor of $O(\log\ N)$ compared to previous approaches, where N is the number of unknowns. This grid-based assignment strategy achieves the optimal order in both traffic volume and startups; it provides good load balancing and as much parallelism as is inherent in the underlying algorithm. Some experimental results are presented which confirm the increased communication efficiency.

Key words. sparse matrices, parallel methods, hypercubes, partial differential equations, communication efficiency

AMS(MOS) subject classifications. 65F50, 65N55

1. Introduction.

Parallel sparse solvers for elliptic partial differential equation (PDE) problems on a distributed memory, message passing (DMMP) multiprocessor can be generally divided into three major components: *assignment*, *indexing*, and *solution*. By assignment we mean that the discrete equations and the associated subtasks are assigned to processors. Indexing means that equations/unknowns are given an order, which establishes a system of linear algebraic equations. Solution means applying a numerical method to solve this linear system. Potentially, one can apply any algorithm to each of these three components. Various combinations of these algorithms will, of course, perform differently. Our Parallel Ellpack system provides a test bed for such performance experiments [11], [16].

Most parallel implementations of sparse solvers are based on elimination trees which represent the sparse structure of a linear system (e.g., [6], [13], [2]). A better shaped elimination tree (balanced, wide, and short) implies higher parallelism. The elimination tree shape is, in turn, determined by indexing because it determines the sparse structure. Indexings of the nested dissection type are appropriate for generating well-shaped elimination trees. See [13] and [3] for further references to earlier work in this area.

This paper mainly deals with one of the three components, the assignment issue, for a hypercube multiprocessor. An ideal assignment strategy should keep the load balanced, exploit fully the parallelism inherent in the problem, and minimize the communication requirement. For a nested dissection type indexing and a general purpose sparse solver, an attractive *subtree-subcube* approach is proposed by [7] which has the lowest order of traffic volume. However, for most current commercially available hypercube multiprocessors, both the traffic volume and the number of communication startups strongly affect efficiency. We observe that this assignment does

* Received by the editors May 17, 1989; accepted for publication (in revised form) January 23, 1991.

† Computer Science Department, Purdue University, West Lafayette, Indiana 47907. The work of the first author was supported in part by National Science Foundation grant CCR-8619817. The work of the second author was supported in part by Air Force Office of Scientific Research grant 88-0243, and Strategic Defense Initiative Office contract DAAL03-86-K-0106.

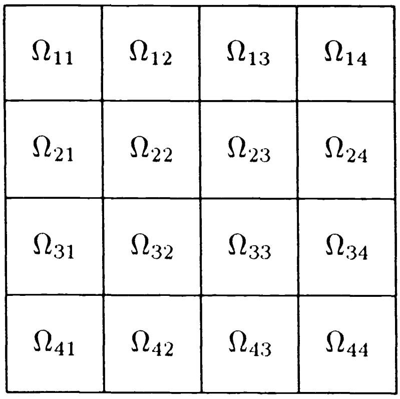
not achieve the lowest order of startups, and thus propose a new assignment strategy, *grid-based subtree-subcube*, which minimizes the startup cost while also retaining the other desirable properties.

A geometric (or physical) approach to develop parallel sparse solvers for solving PDE problems is suggested in [13], [16] which has several advantages over some standard algebraic approaches (e.g., [6], [2]). For example, it naturally allows nonsymmetric structure in the linear system, avoids symbolic factorization, leads to a well-shaped (block) elimination tree, and also allows flexible combinations of various techniques in different regions according to the local information about the geometry and physics of the PDE problem. Basically, the approach is based on a domain decomposition in a nested dissection manner. For indexing, we use *nested dissection* modified by locally using *minimum degree ordering* combined with an idea from the *multifrontal method.* This, together with the domain decomposition, implies a block elimination tree. Various parallel sparse solvers can then be devised based on this block elimination tree.

2. Background.

2.1. Domain decomposition and block elimination tree.

For the readers' convenience, this section briefly describes our geometric approach to developing a parallel sparse solver and the associated algorithms used in indexing and solution. For more details, see [13], [15]–[17]. For simplicity, we consider a PDE problem on a rectangular domain Ω. The approach can be extended easily to general domains. Suppose we have $p\,(= 2^{2d})$ processors available, 2^d in each direction. By domain decomposition Ω is divided into p subdomains Ω_{ij}, $i,\ j = 1,\ 2,\ \cdots, p^{1/2}$ as shown in Fig. 2.1. One puts a local grid on each subdomain Ω_{ij} and discretizes the local problem whose solution U_{ij} only depends on unknowns at grid points of $\partial\Omega_{ij}$, the boundary of Ω_{ij}.



Ω_{11}	Ω_{12}	Ω_{13}	Ω_{14}
Ω_{21}	Ω_{22}	Ω_{23}	Ω_{24}
Ω_{31}	Ω_{32}	Ω_{33}	Ω_{34}
Ω_{41}	Ω_{42}	Ω_{43}	Ω_{44}

FIG. 2.1. *Domain decomposition of a rectangle for $d = 2$.*

Initially, all interior unknowns U_{ij} are eliminated locally as in the standard domain decomposition approach. This step is obviously totally parallel. Then, all processors participate in eliminating interface unknowns. To exploit more parallelism, we use dissection in alternating directions to partition the interface set into several levels suitable for a hypercube machine. Each level consists of several *separators*, groups of unknowns which separate regions. The partition, which we call the *incomplete* *nested dissection decomposition* (borrowing the term used in [10]), is shown by Fig. 2.2 with circles representing the subdomain Ω_{ij}, and boxes representing the *separators.* For simplicity, these are all called *subdomains* of this domain decomposition and are numbered from top level to bottom level as shown in Fig. 2.2.

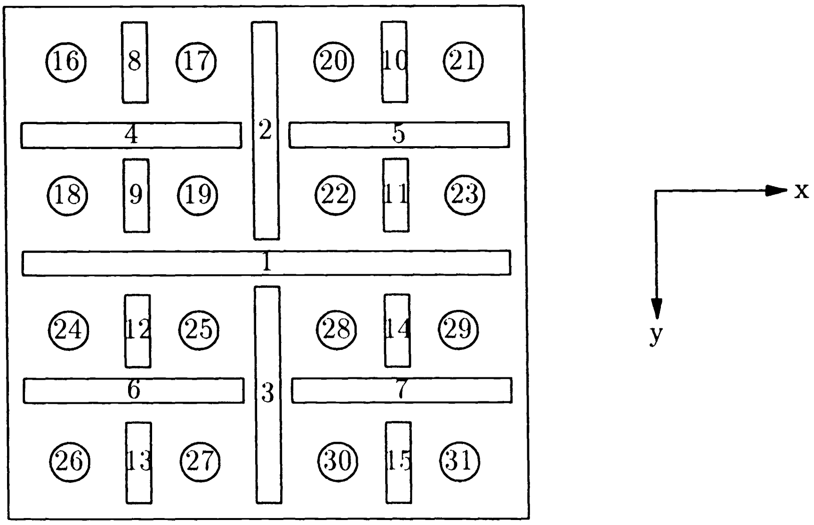


FIG. 2.2. *Partition of the subdomain interfaces in Fig.* 2.1 *using the incomplete nested dissection. The circles* (16–31) *represent the* 16 *groups of subdomain unknowns and the boxes represent groups of interface unknowns or separators. All boxes of the same size in each direction are on the same level of the elimination tree.*

This domain decomposition naturally inherits certain parallelism because the PDE discretization process leads to a local or boundary dependence property for interfaces. For example, if we consider the union of subdomains 16, 17, and 8 as a more general subdomain Ω'_8 then the local interior solution set U'_8 is uniquely determined by unknowns on $\partial\Omega'_8$. This relation holds similarly for groups at higher levels of the dissection decomposition for the unknowns arising in PDE applications. This local dependency can be described by a binary tree as shown in Fig. 2.3 with each tree node corresponding to a subdomain in the decomposition as shown in Fig. 2.2. We call it a *block elimination tree* because it plays a similar role as the standard *elimination tree* [2] does in exploiting parallelism. However, each node corresponds to, in the former case, a block of unknowns/equations, while in the latter case, a single unknown/equation. In other words, we are seeking the parallelism in the block sense no matter what local properties the linear system has within each subdomain locally. This block elimination tree has the following properties: (a) Each node corresponds to a set of unknowns from one location; local ordering and lack of symmetry in the linear system do not affect the tree structure. (b) Eliminating a node only has effects on its ancestors. (c) The eliminations of nodes that are not descendants/ancestors of one another are independent of one another. It thus implies several basic rules for assignment, indexing and solution, as in Fig. 2.3, to exploit the full parallelism inherent in the block elimination tree.

2.2. Indexing. It is natural to index the unknowns/equations globally according to the order of the dissection decomposition, i.e., if unknowns u and v are in nodes

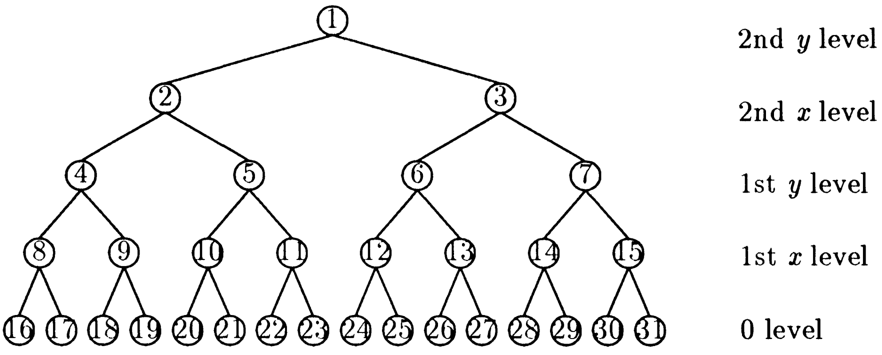


FIG. 2.3. *Block elimination tree produced by the incomplete nested dissection domain decomposition. The numbering of nodes corresponds to the groups of unknowns in Fig.* 2.2; *the* x, y *levels refer to the directions of the bisection.*

S and T, respectively, and S is on a higher level than T is, then the index of u is larger than that of v. This variation of nested dissection is incomplete in the sense that each separator is a line, instead of a cross as usual. It is also incomplete in the sense that we do not perform the dissection all the way until each node on the bottom level contains a single unknown/equation. Potentially, any linear system solver can be applied locally within each node without affecting the good shape of the block elimination tree. This allows us to choose appropriate local indexings in different regions for different purposes and thus to mix nested dissection with other indexings to improve performance without affecting the parallelism. For example, within each subdomain on the bottom level, elimination occurs sequentially in a single processor while the local problem is still sparse, so the fill-in issue here is a major concern. For example, the *minimum degree* ordering may have less fill-in than nested dissection does [12], especially for irregular domains. In the following, we will generically use *minimum degree* to mean whatever indexing gives the best performance on the sequential problems on the lowest level. The classic minimum degree ordering is just one reasonable choice here. Furthermore, if we keep the elimination front in Ω_{ij} as far as possible from $\partial\Omega_{ij}$, it helps to reduce the communication cost. Denote the boundary layer of grid points in Ω_{ij} (those next to $\partial\Omega_{ij}$) by B_{ij} as shown in Fig. 2.4. Without loss of generality *assume* that only unknowns on B_{ij} are related to those on $\partial\Omega_{ij}$ in the linear system, such as the *five-point star* discretization would generate. So we first eliminate the unknowns on $\Omega_{ij} - B_{ij}$ using the minimum degree ordering. There is no communication required at this stage. Then unknowns on B_{ij} are eliminated. Notice that the local indexing we use here is the minimum degree combined with the multifrontal method idea. We call the indexing described above the *modified nested dissection.*

Processing the boundary B_{ij} last in Ω_{ij} might increase the fill-in and computational cost. Application of the argument in [5] shows that the effects are very minor. And, as the size of the domains Ω_{ij} increases, this small effect becomes less important. More specifically, if one performs operation counts for nested dissection applied to all of Ω_{ij} and for nested dissection applied to $\Omega_{ij} - B_{ij}$ followed by B_{ij} elimination, one finds that not only is the order (in terms of k) the same, but the leading coefficients are the same. We have not experimentally measured this effect since we

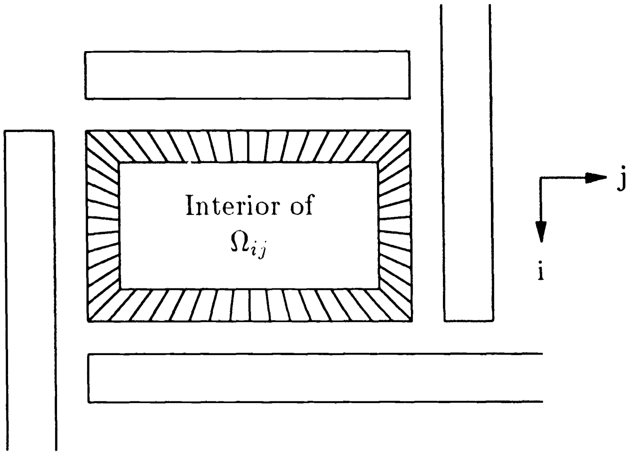


FIG. 2.4. *View of a typical subdomain Ω_{ij} of the* PDE *discretization. There are k^2 unknowns (grid points) in Ω_{ij}, $(k-2)^2$ in the interior and $n_0 = 4(k-1)$ in B_{ij}, the boundary layer (dashed area). The separators (shown as boxes) are lines of grid points separating the subdomains.*

expect it to be small or even totally absent. On the other hand, for existing and contemplated DMMP machines, these algorithms are usually communication bound, so that processing the boundary B_{ij} last appears to be the correct choice for algorithm designs.

As far as the nodes on higher levels (separator subdomains) are concerned, local indexing does not affect fill-in and communication significantly since the local systems are almost dense when the elimination front arrives. In the following, a *wrapping ordering* is assumed for each of these nodes locally.

No pivoting is assumed in the above discussion because we are mainly interested in solving elliptic PDEs where the matrix problems are usually numerically stable even though the matrices are often nonsymmetric. However, using this framework, one can still perform certain local pivoting within each $\Omega_{ij} - B_{ij}$ or within separators.

2.3. Solution. There are four kinds of potential parallelism here. First, elimination steps in independent nodes of the block elimination tree can execute simultaneously. We call this the *outer parallelism.* Second, if there are several processors available for a single node, we can also exploit *inner parallelism* within the node. This does not occur at leaf nodes if each leaf node has only one processor, as in usual cases, even though it represents a sparse subproblem. For the other nodes we apply various efficient parallel dense solvers to exploit the inner parallelism. Third, the task to modify an equation to eliminate an unknown (or simply, a *modification*) is independent for different equations just as for dense matrices. Finally, fourth, modifications, even on the same equation, due to independent descendant nodes can be performed in arbitrary order and hence in parallel. Further, modifications of one equation are vector operations whose components can be processed in parallel.

The potential parallelism inherent in the block elimination tree provides many ways to develop parallel algorithms for different matrix properties and machine architectures. Fan-in [1], multifrontal [2], and fan-out [8] organizations are commonly used in parallel sparse solvers. The fan-in scheme is currently known to be efficient in communication [1]. But it is strictly restricted to symmetric matrices when used on DMMP machines because forming the so-called aggregate update column (or row) needs the information from not only a set of columns (or rows), but also a row (or column) for the corresponding multipliers. This causes substantial communication expense for nonsymmetric matrices, even if they are structurally symmetric. We do not assume structural symmetry in our analysis. We believe that practical solvers will have the matrix distributed to the processors either by rows or by columns.

One can easily see that a realistic sparse solver essentially involves the following communication assumption for handling nonsymmetric linear systems on DMMP machines.

ASSUMPTION 3.1. *For Gauss elimination of nonsymmetric matrices on a DMMP machine, each pivot equation must be sent to those processors which hold equations related to the pivot equation.*

In other words, if $a_{mk} \neq 0$, $m > k$, then the processor holding the mth equation must receive the kth equation (after it becomes a pivot equation). This communication assumption is enough for the communication complexity analysis in §4 so that further details about a particular solution algorithm are not used. The multifrontal and fan-out schemes both satisfy this assumption. We recently presented a new organization of Gauss elimination for nonsymmetric problems in [17] which achieves much higher speedup than the fan-in, multifrontal, and fan-out methods do even for symmetric problems with comparable sizes. This very efficient algorithm (which is basically of fan-out type) still satisfies this assumption except for the bottom level in the elimination tree. That level, however, does not affect the analysis in §4. In summary, the analysis includes all potential solution algorithms for solving nonsymmetric problems on DMMP machines except the fan-in scheme which is not applicable here.

3. Assignment strategy.

By assigning an unknown to a processor we mean assigning both the problem data and the factorization subtask associated with this unknown. To achieve high parallelism, load balancing, and low communication costs based on the block elimination tree, we want to (a) avoid assigning independent nodes to the same processor, and (b) assign processors to a single node so as to have minimal communication connections. The standard wrapping assignment is not as effective here even though it achieves load balancing. Recall that the primary goal in parallel algorithms is efficiency and that load balancing is not a necessary condition to achieve maximum efficiency.

In [7] an attractive *subtree-subcube with local wrapping* assignment is proposed. We refer to it as the *standard* subtree-subcube assignment. It is a top to bottom process. First, the root node of the elimination tree is assigned to the whole hypercube and then the hypercube is split into two subcubes to which the two descendant subtrees are assigned. This process goes on recursively until all subtrees become assigned to single processors. The assignment within each node is simply in a wrapping manner. Note that: (a) eliminating an unknown in a node need not affect all of its ancestor nodes, and (b) even when effects occur in some ancestor nodes, they need not affect all equations in them. Geometrically, the effect of elimination spreads in a multifrontal manner with each processor starting with one subdomain and continuing to work on "merged" domains containing it as interface (separator) unknowns are eliminated. However, one cannot represent these properties completely by elimination trees even though they may affect the parallel efficiency very much. This suggests that unknown assignments be made in a multifrontal manner with a processor responsible only for those unknowns located at the fronts of some nodes to which the processor has been assigned. If several processors (usually a subcube) correspond to the same set of unknowns, then local wrapping can be applied within this set. We call this the *grid-based subtree-subcube* assignment which is defined more precisely as follows.

This is a bottom to top assignment process. Let us denote the levels in the block elimination tree from bottom to top by 0, 1st x, 1st y, 2nd x, 2nd y, $\cdots$, ith x, ith y, and so on. The x and y refer to the horizontal (j) and vertical (i) directions, respectively. The process is related to Fig. 2.2 where level zero consists of the leaves, 1st x consists of the eight separators 8–15, 1st y consists of the four separators 4–7, 2nd x consists of separators 2 and 3, and 2nd y consists of separator 1. The first step is to map the given hypercube to a two-dimensional grid (for domain decomposition) by the well-known *gray code* such that adjacent processors are directly connected and Ω_{ij} is assigned to proc.essor P_{ij}. This defines the assignment at the leaves of the block elimination tree, the 0 level. Next, we subdivide each separator on the ith x level and the ith y level into 2^{i-1} and 2^i *segments*, respectively. This subdivision of segments corresponds to the natural geometric segments along the separators of the domain decomposition. We take care of the intersection points of adjacent segments in each separator by adding an intersection point to its top (left) segment for the $x(y)$ direction. Then we assign each segment on the ith $x(y)$ level to the closest 2^i processors in the $x(y)$ direction. The assignment within each segment uses wrapping. This scheme is illustrated in Fig. 3.1. Thus, processors P_{11} and P_{12} are assigned to the interface unknowns of separator 8 (the upper left box in Fig. 3.1). For comparison, Fig. 3.2 illustrates the standard subtree-subcube assignment strategy.

The potential for reducing communication is seen by examining separator 4 which is the top, left horizontal box in Figs. 2.2 and 3.2. In Fig. 3.1 we see those unknowns divided into two separate groups (segments). In the grid-based subtree-subcube assignment (Fig. 3.1), the processors P_{11} and P_{21} only handle the interface between the two subdomains (16 and 18) they handled at the lower level. In the standard subtree-subcube assignment (Fig. 3.2), the processors P_{11} and P_{21} are part of a group of processors handling the four subdomains (16, 17, 18, and 19). Thus P_{11} and P_{21} must now obtain information about subdomains 17 and 19 at this step while this is not required in the grid-based subtree-subcube assignment. This reduction in the communication occurs in a similar manner for every separator.

4. Communication complexity analysis. Throughout this section, we use the communication Assumption 3.1. The communication complexity, of course, depends on indexing schemes, too. We will make it clear which indexing is used whenever necessary.

In [7] it is proved that the total amount (or volume) of communication for the standard subtree-subcube assignment is $O(pN)$. This order is optimal in the sense of minimizing traffic volume for nested dissection algorithms; the grid-based subtree-subcube assignment has this same optimal order, as the analysis in [7] applies to all types of subtree-subcube assignments. We give an analysis which provides estimates for the communication complexity of startups as well as of traffic volume for both assignments when applied to the modified nested dissection indexing and general nonsymmetric solution algorithms. The analysis does not apply to the fan-in scheme because the fan-in algorithm does not satisfy Assumption 3.1. But this algorithm is not applicable to nonsymmetric problems which we are considering here.

Suppose that a set of q processors composes a connected subgraph in the hypercube architecture and it is involved in communicating a piece of message. We assume that each nonroot processor receives the message exactly once from one of its direct neighbors. This occurs in most procedures for multicasting, i.e., sending a message

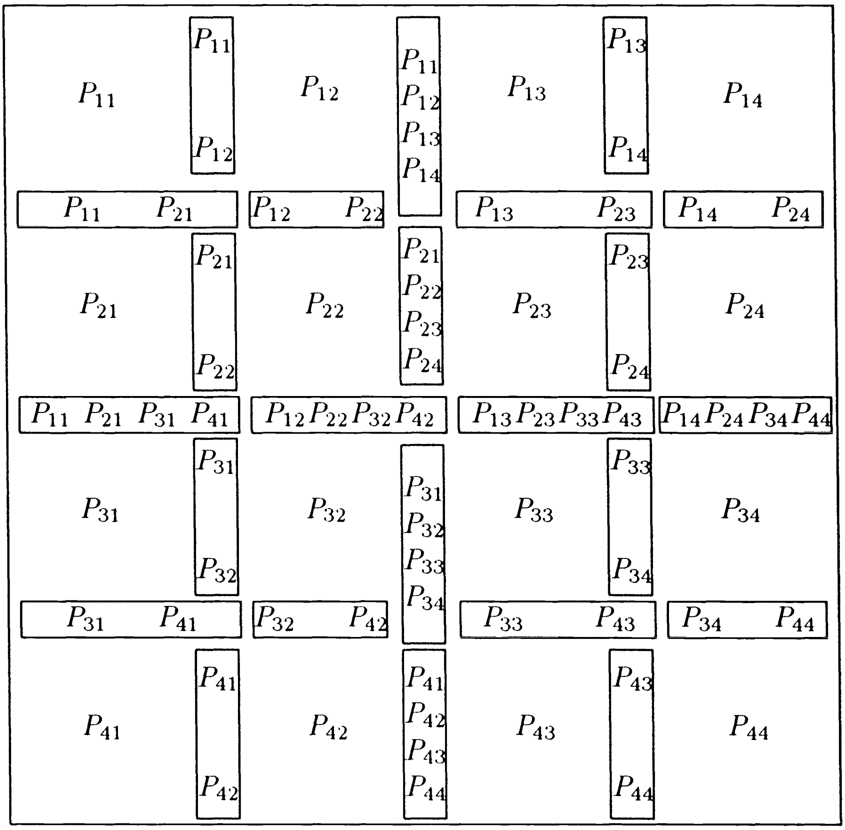


FIG. 3.1. *Grid-based subtree-subcube assignment for* 16 *processors. Within the subdomain interfaces we show how the processors are assigned to unknowns in parts of the separators.*

from one processor to a list of other processors. Therefore the total number of startups required in such a communication process is equal to $q - 1$. Putting a $k \times k$ grid on each subdomain Ω_{ij}, we have the total number of grid points on Ω equal to $N = n \times n$ with $n = p^{1/2}\ k\ +\ p^{1/2}\ -\ 1$. Without loss of generality assume that there is a correspondence between each unknown and a grid point, as in the *five-point star* discretization.

Let *startup*(n, p) denote the number of startups required for communication corresponding to the $n \times n$ grid and the p processors assigned to this grid. From Assumption 3.1, in eliminating a particular unknown we need to count the cost for communicating a piece of multiplier message (a vector) among those processors holding unknowns which are currently related to this unknown. First we estimate *startup*(n, p) for the grid-based subtree-subcube assignment and the modified nested dissection indexing using the recursive analysis similar to the argument used in [7].

THEOREM 4.1. *Suppose $d \geq 3$ and there are $N = n \times n$ unknowns and $p = 4^d$ processors. Assume that a nonsymmetric sparse solver uses the grid-based subtree-subcube assignment, the modified nest dissection indexing, and a solution algorithm which satisfies Assumption* 3.1. *Then, on a DMMP machine the number of startups satisfies*

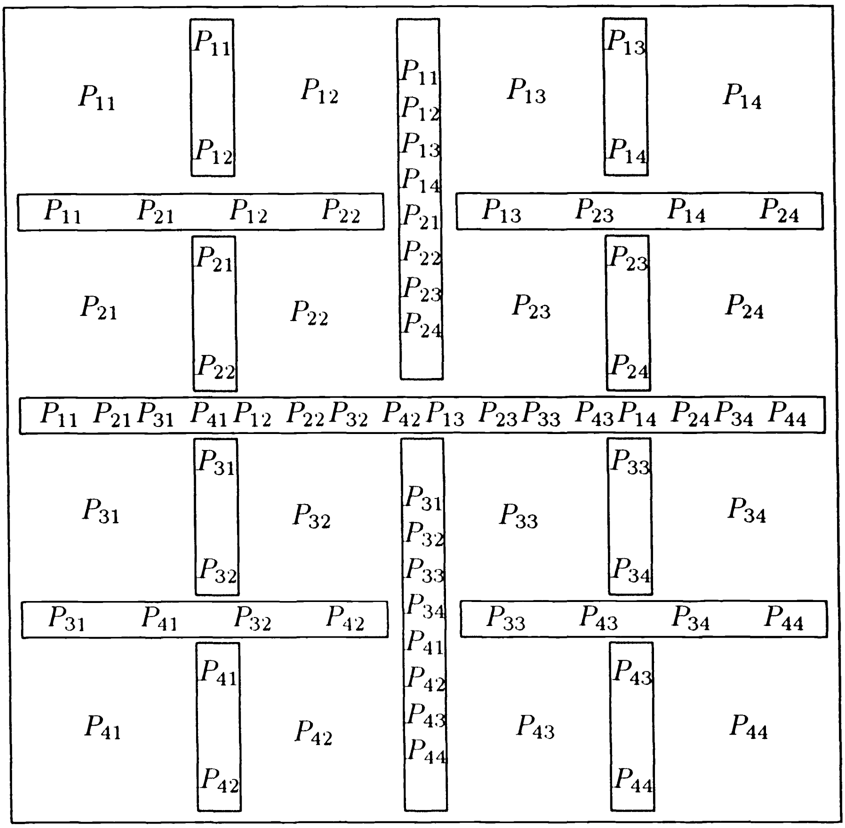

P_{11}
P_{11} P_{12}
P_{12}
P_{11} P_{12} P_{13} P_{14} P_{21} P_{22} P_{23} P_{24}
P_{13}
P_{13} P_{14}
P_{14}
P_{11} P_{21} P_{12} P_{22}
P_{13} P_{23} P_{14} P_{24}
P_{21}
P_{21} P_{22}
P_{22}
P_{23}
P_{23} P_{24}
P_{24}
P_{11} P_{21} P_{31} P_{41} P_{12} P_{22} P_{32} P_{42} P_{13} P_{23} P_{33} P_{43} P_{14} P_{24} P_{34} P_{44}
P_{31}
P_{31} P_{32}
P_{32}
P_{31} P_{32} P_{33} P_{34} P_{41} P_{42} P_{43} P_{44}
P_{33}
P_{33} P_{34}
P_{34}
P_{31} P_{41} P_{32} P_{42}
P_{33} P_{43} P_{34} P_{44}
P_{41}
P_{41} P_{42}
P_{42}
P_{43}
P_{43} P_{44}
P_{44}

FIG. 3.2. *Standard subtree-subcube assignment for* 16 *processors. Within each box unknowns are assigned in wrapping manner to processors shown in the box.*

$$startup(n,p) \le 10np \; + \; 80np^{1/2}. \tag{4.1}$$

Proof. Notice that the "cross" set on the top level consisting of two separators on the dth x level and one on the dth y level divides the $n \times n$ grid and the set of p processors into four $\frac{n}{2} \times \frac{n}{2}$ subgrids and four groups of $\frac{p}{4}$ processors. We introduce the function $f_2(m,\; q)$ as the number of startups required due to the outside processors on the subgrid border for eliminating unknowns in each $m \times m$ subgrid with two border sides on the top level "cross" set and each side having q processors assigned to it. So, $startup(n,p)$ is bounded by three terms:

$$startup(n,p) \le 4\; startup(\tfrac{n}{2},\; \tfrac{p}{4}) \; + \; 4f_2(\tfrac{n}{2},\; \tfrac{p}{2}) \; + \; \{2(\tfrac{n}{2}p) \; + \; n\, p\,\}, \tag{4.2}$$

The third term in (4.2) corresponds to the number of startups required for eliminating unknowns in three separators of the "cross" set.

To estimate $f_2(\frac{n}{2},\; \frac{p}{2})$, we examine one lower level by using the "cross" set in the $\frac{n}{2} \times \frac{n}{2}$ grid to further divide it into four $\frac{n}{4} \times \frac{n}{4}$ subgrids and we have

$$f_2(\tfrac{n}{2},\; \tfrac{p}{2}) \le f_2(\tfrac{n}{4},\; \tfrac{p}{4}) \; + \; 2f_1(\tfrac{n}{4},\; \tfrac{p}{4}) \; + \; \{[\tfrac{n}{4}(\tfrac{p}{4} \; + \; \tfrac{p}{8}) \; + \; \tfrac{n}{4}\tfrac{p}{8}] \; + \; \tfrac{n}{2}\,(\tfrac{p}{4} \; + \; \tfrac{p}{4})\} \tag{4.3}$$

where the function f_1 has a meaning similar to f_2, but with only one border side on the top level "cross" set. The third term in (4.3) comes from eliminating three separators in the "cross" set of this $\frac{n}{2} \times \frac{n}{2}$ grid.

Similarly, we go to the next lower level to estimate the function f_1 as follows:

$$f_1(\tfrac{n}{4},\ \tfrac{p}{4}) \leq 2f_1(\tfrac{n}{8},\ \tfrac{p}{8}) + \{\tfrac{n}{8} \cdot \tfrac{p}{8} + \tfrac{n}{4} \cdot \tfrac{p}{8}\}. \tag{4.4}$$

Notice that on the bottom level, i.e., the domain decomposition level, we have

$$f_1(\tfrac{n}{p^{1/2}},\ p^{1/2}) \leq (4 \cdot \tfrac{n}{p^{1/2}}) \cdot \tfrac{p^{1/2}}{2} \ = \ 2n. \tag{4.5}$$

By recursively using (4.4) till the bottom level, we get

$$\begin{aligned} f_1(\tfrac{n}{4},\tfrac{p}{4}) &\leq \tfrac{3}{64}np + 2f_1(\tfrac{n}{8},\tfrac{p}{8}) \\ &\leq \tfrac{3}{64}np(1 + \tfrac{1}{2} + \tfrac{1}{2^2} + \cdots) + 2f_1(\tfrac{n}{p^{1/2}},p^{1/2}) \\ &\leq \tfrac{3}{32}np + 4n. \end{aligned} \tag{4.6}$$

Similarly, on the bottom level we have

$$f_2(\tfrac{n}{p^{1/2}},\ p^{1/2}) \leq (4 \cdot \tfrac{n}{p^{1/2}}) \cdot p^{1/2} \ = \ 4n. \tag{4.7}$$

Substituting f_1 from (4.6) and recursively using (4.3) and then using (4.7), we obtain (4.8)

$$\begin{aligned} f_2(\tfrac{n}{2},\tfrac{p}{2}) &\leq \tfrac{9}{16}np + 8n + f_2(\tfrac{2}{4},\tfrac{p}{4}) \\ &\leq \tfrac{9}{16}np(1 + \tfrac{1}{4} + \tfrac{1}{4^2} + \cdots) + 8n(1 + \tfrac{1}{2} + \tfrac{1}{2^2} + \cdots) + f_2(\tfrac{n}{p^{1/2}},p^{1/2}) \\ &\leq \tfrac{3}{4}np + 16n + 4n \\ &= \tfrac{3}{4}np + 20n. \end{aligned}$$

We substitute (4.8) into (4.2) to obtain

$$startup(n,p) \leq 5np \ + \ 80n \ + \ 4startup(\tfrac{n}{2},\ \tfrac{p}{4}). \tag{4.9}$$

We observe that

$$startup(\tfrac{n}{p^{1/2}},1) \ = \ 0. \tag{4.10}$$

We apply (4.9) recursively and then use (4.10) to obtain

$$\begin{aligned} startup(n,p) &\leq 5np(1 \ + \ \tfrac{1}{2} \ + \ \tfrac{1}{2^2} \ + \ \cdots) \ + \ 80n(1 \ + \ 2 \ + \ 2^2 \ + \ \cdots \ + \ 2^{d-1}) \\ &\leq 10np \ + \ 80np^{1/2}. \end{aligned}$$

This completes the proof. □

This analysis gives an upper bound. We see that the bound is sharp in terms of the orders for n and p for a pipelined algorithm (one where a processor forwards information needed by another processor as soon as it becomes available). The final step (node 1 of the elimination tree) involves the n unknowns on the separator 1; see Fig. 2.2. This step involves a dense matrix where all p processors must exchange information with all other processors about all the remaining n equations. If we maintain pipelining, then each multiplier vector must be broadcast as soon as it is available. Thus np message startups are required. Thus we see that the grid-based subtree-subcube assignment is of optimal order with respect to startups; we thus have the following.

COROLLARY. *With $d \geq 3$, p processors, $n \times n$ unknowns, and a pipelined algorithm, we have*

$$\mathit{startup}(n,p) \;=\; O(pn).$$

One could consider a partially pipelined algorithm where several multiplier vectors are collected before being broadcast. This would be similar in philosophy to loop unrolling (see [4]); it has the disadvantage of delaying computations on other processors. While the optimal strategy will depend on the exact characteristics of the hypercube, we believe that the optimal strategy will usually involve only a small number of vectors to be collected together before broadcasting so the order will remain the same.

Changing the new assignment in Theorem 4.1 to the standard subtree-subcube one, we see that the functions f_1 and f_2 do not depend on the second argument, the number of processors. Therefore, an extra factor $\log_2 p$ is introduced into the estimate for $\mathit{startup}(n,p)$. Furthermore, if the indexing in Theorem 4.1 is also changed to the standard nested dissection, then the factor $\log_2 p$ will become $\log_2 n$ because the number of the dissection levels increases from $\log_2 p$ to $\log_2 n$. This is also pointed out in [19]. These two facts are stated as follows.

THEOREM 4.2. *Change the subtree-subcube assignment in Theorem* 4.1 *from grid-based to standard. Then we have*

$$\mathit{startup}(n,p) \;=\; O(pn \;\log_2 p).$$

Further, if the indexing is replaced by the standard nested dissection, then we have

$$\mathit{startup}(n,p) \;=\; O(pn \;\log_2 n).$$

We can also bound the traffic volume for both assignments. Specifically, let V_g and V_s denote the total communication volume for the grid-based subtree-subcube assignment and the standard subtree-subcube assignment, respectively.

THEOREM 4.3. *With the assumptions of Theorems* 4.1 *and* 4.2 *we have*

$$V_g \leq 14.4pN \;+\; O(\log_2 pN), \tag{4.11}$$

$$V_s \leq 31.85pN \;+\; O(\log_2 pN). \tag{4.12}$$

We do not give the proof for Theorem 4.3 here. It can be found in [14]. From Theorem 4.3 we see that the bound on V_s is about twice the bound on V_g. Both have the same optimal order $O(pN)$ as seen by examining the elimination at node 1. It is intuitively clear and not hard to show that both algorithms are load balanced, i.e., the volume of messages from each processor is $O(N)$.

5. Implementation and performance. Serious experimental studies are needed to measure the value of the many variations in sparse matrix algorithms for hypercubes. This is a difficult challenge because of the sheer number of variations and the difficulty is further compounded by (a) the lack of a stable architecture, and (b) the lack of good performance analysis tools for hypercubes. Thus, even if we knew everything now, the next generation of hypercubes is likely to have such a different balance of hardware capabilities that we would have to start over. Even now, we face the problem of whether poor performance observed is due to an inherent flaw in the algorithm/method or to a terrible implementation of some underlying system service. In view of this, we settle for performance comparison of these two assignment strategies, which are implemented as two *assignment* modules in our Parallel Ellpack and are combined with the *indexing* module MODIFIED NESTED DISSECTION and two *solution* modules PARALLEL SPARSE [15], which is basically a fan-out scheme, and NEW GAUSS ELIMINATION [17], on our two hypercubes (a first generation NCUBE with 128 processors and a second generation NCUBE with 64 processors).

We solve a *Poisson* equation with *Dirichlet* boundary condition on a rectangle domain using 16 processors by our geometric approach with *five-point star* discretization. We only consider the communication cost and timing of the *LU* factorization phase. While this problem is actually symmetric, we still treat it as a nonsymmetric one since we want to examine the effects of these two assignments on general problems. In the following, we use *standard* and *new* to briefly denote the *standard subtree-subcube* and the *grid-based subtree-subcube* assignments, respectively.

Figure 5.1 shows the number of messages versus $N^{1/2}$ for both *standard* and *new*. The total message volume versus N for both *standard* and *new* is shown in Fig. 5.2. PARALLEL SPARSE is used as the solution module for both of these figures. We observe a substantial reduction in communication requirements using the *new* assignment strategy. These data also firmly support the analysis in §4. If the standard nested dissection indexing is used with the "standard" curve, then the number of messages is increased.

Applying the new assignment and indexing strategies to the NEW GAUSS ELIMINATION module, we get a speedup of 4 for solving a 23 × 23 grid problem using four processors, and a speedup of 15.7 for solving a 61 × 61 grid problem using 16 processors. Notice that even for symmetric solvers, the best speedup reported previously for 16 processors is 10.62. This is for solving a 125 × 63 L-shaped grid problem with nine-point star by the fan-in scheme on an Intel iPSC/2 hypercube [1]. For more detailed performance data, see [18].

6. Conclusion.

We propose a grid-based subtree-subcube assignment strategy appropriate for sparse matrix algorithms to solve PDE problems on hypercubes. The change from the standard subtree-subcube assignment is motivated by the multifrontal method used in PDE solvers where interfaces between subdomains are handled by the processors assigned to the subdomains. For both assignments we give a complexity analysis of the communication costs of the elimination considering both the total traffic volume and the number of message startups. The analysis is given for a PDE problem on a rectangular domain but the arguments can be extended to more general domains. We show that our assignment has, theoretically, about half the total communication volume as the standard assignment and the number of communication startups is improved by a factor of $\log_2 p$. With the modified nested dissection indexing, the number of startups is then improved in total by a factor of $\log_2 n$. This approach achieves the optimal order in traffic volume and number of startups. Experimental measurements indicate that these theoretical improvements occur in practice. This assignment also provides load balancing and exploits fully the parallelism inherent in the problem. It is applied to nonsymmetric problems.

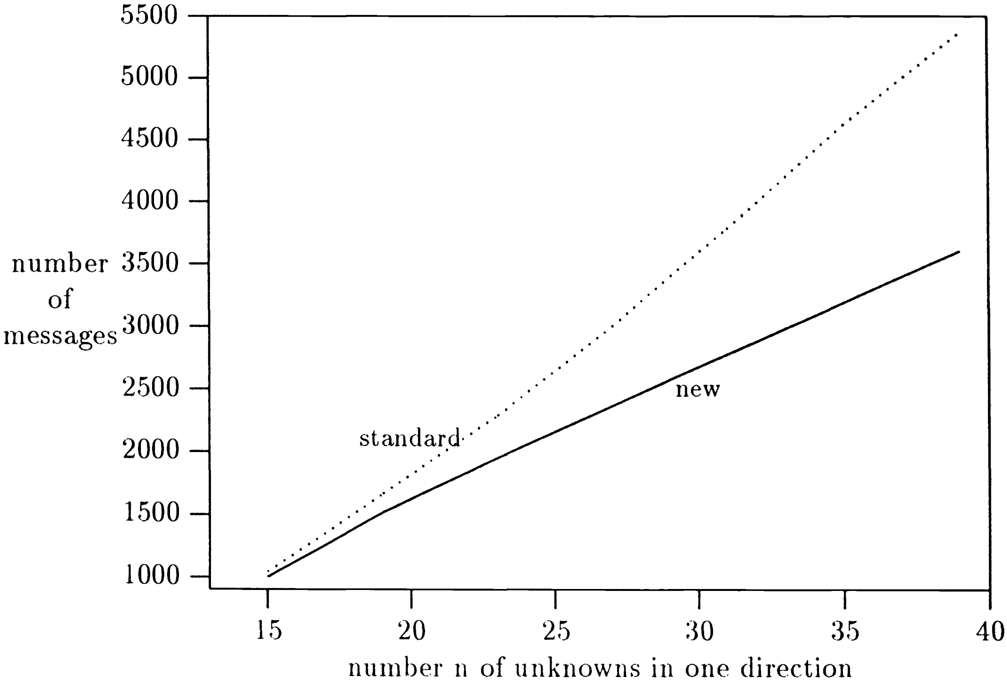


FIG. 5.1. *Number of messages S versus number $n = N^1/2$ of unknowns in one direction with 16 processors.*

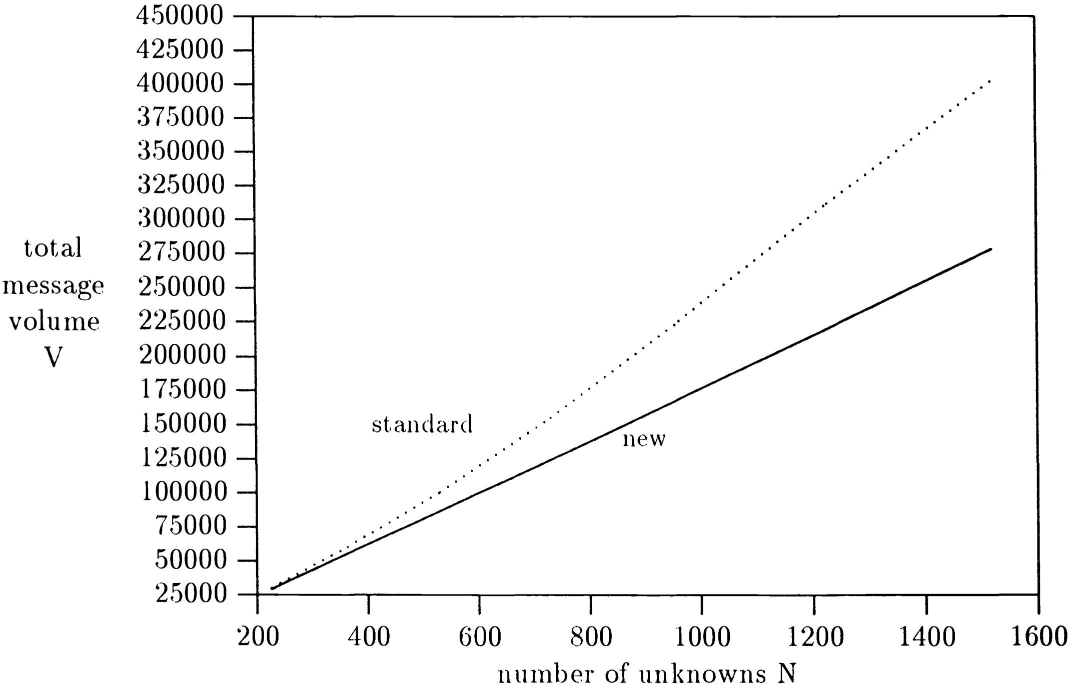


FIG. 5.2. *Total message volume V versus number N of unknowns with 16 processors.*

Acknowledgment. We appreciate the helpful comments and suggestions of the referees which substantially improved the presentation.

REFERENCES

[1] C. Ashcraft, S. Eisenstat, and J. Liu (1990), *A fan-in algorithm for distributed sparse numerical factorization*, SIAM J. Sci. Statist. Comput., 11, pp. 593–599.

[2] I. S. Duff (1986), *Parallel implementation of multifrontal schemes*, Parallel Comput., 3, pp. 193–204.

[3] K. A. Gallivan, M. T. Heath, E. Ng, J. M. Ortega, B. W. Peyton, R. J. Plemmons, C. H. Romine, A. H. Sameh, R. G. Voigt (1990), *Parallel algorithms for matrix computations*, Society for Industrial and Applied Mathematics, Philadelphia, PA.

[4] G. A. Geist and C. H. Romine (1988), *LU factorization algorithms on distributed-memory multiprocessor architectures*, SIAM J. Sci. Statist. Comput., 9, pp. 639–649.

[5] A. George (1973), *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10, pp. 345–363.

[6] A. George, M. Heath, J. Liu, and E. Ng (1988), *Sparse Cholesky factorization on a local-memory multiprocessor*, SIAM Sci. Statist. Comput., 9, pp. 327–340.

[7] A. George, J. Liu, and E. Ng (1987), *Communication reduction in parallel sparse Cholesky factorization on a hypercube*, in Hypercube Multiprocessors, M. Heath, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, pp. 576–586.

[8] ——— (1988), *A data structure for sparse* QR *and* LU *factors*, SIAM J. Sci. Statist. Comput., 9, pp. 100–121.

[9] A. George and E. Ng (1988), *Parallel sparse Gaussian elimination with partial pivoting*, Tech. Report ORNL/TM-10866, Oak Ridge National Laboratory, Oak Ridge, TN.

[10] A. George, W. G. Poole, and R. G. Voigt (1978), *Incomplete nested dissection for solving n by n grid problems*, SIAM J. Numer. Anal., 15, pp. 662–673.

[11] E. Houstis and J. R. Rice (1989), *Parallel Ellpack*, Math. Comput. Simulation, 31, pp. 497–508.

[12] J. W. H. Liu (1989), *The minimum degree ordering with constraints*, SIAM J. Sci. Statist. Comput., 10, pp. 1136–1145.

[13] M. Mu and J. R. Rice (1989), *LU factorization and elimination for sparse matrices on hypercubes*, in Fourth Conference on Hypercube Concurrent Computers and Applications, Monterey, CA, March, 1989, Golden Gate Enterprises, Los Altos, CA, 1990, pp. 681–684.

[14] ——— (1989), *A grid-based subtree-subcube assignment strategy for solving PDEs on hypercubes*, CSD-TR-869, CER-89-12, Computer Science Department, Purdue University, West Lafayette, IN.

[15] ——— (1990), *Parallel sparse: Data structures and algorithm*, CSD-TR 974, CER-90-17, Computer Science Department, Purdue University, West Lafayette, IN.

[16] ——— (1990), *The structure of parallel sparse matrix algorithms for solving partial differential equations on hypercubes*, CSD-TR 976, CER-90-19, Computer Science Department, Purdue University, West Lafayette, IN.

[17] ——— (1990), *A new organization of sparse Gauss elimination for solving PDEs*, CSD-TR-991, CER-90-22, Computer Science Department, Purdue University, West Lafayette, IN.

[18] ——— (1992), *Performance of PDE Sparse Solvers on Hypercubes*, in Unstructured Scientific Computation on Scalable Multiprocessors, P. Mehrotra, J. Saltz, and R. Voigt, eds., MIT Press, Cambridge, MA, pp. 345–370.

[19] E. Zmijewski (1989), *Limiting communication in parallel sparse Cholesky factorization*, TRCS89-18, Department of Computer Science, University of California, Santa Barbara, CA.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 4, pp. 841-859, July 1992

001

PARALLEL METHODS FOR SOLVING NONLINEAR BLOCK BORDERED SYSTEMS OF EQUATIONS*

XIAODONG ZHANG†, RICHARD H. BYRD‡, AND ROBERT B. SCHNABEL‡

Abstract. A group of parallel algorithms, and their implementation for solving a special class of nonlinear equations, are discussed. The type of sparsity occurring in these problems, which arise in VLSI design, structural engineering, and many other areas, is called a *block bordered* structure. The explicit method and several implicit methods are described, and the new *corrected implicit method* for solving block bordered nonlinear problems is presented. The relationship between the two types of methods is analyzed, and some computational comparisons are performed. Several variations and globally convergent modifications of the implicit method are also described. Parallel implementations of these algorithms for solving block bordered nonlinear equations are described, and experimental results on the Intel hypercube that show the effectiveness of the parallel implicit algorithms are presented. These experiments include a fairly large circuit simulation that leads to a multilevel block bordered system of nonlinear equations.

Key words. systems of nonlinear equations, parallel computation

AMS(MOS) subject classifications. 65H10, 65W05

1. Introduction.

1.1. Definition of block bordered nonlinear problems. In this paper we present a group of parallel algorithms and their implementations for solving a special class of nonlinear equations, instances of which occur in VLSI design, structural engineering, and many other areas. The class of sparsity occurring in these problems is called a block bordered structure. In such a problem the general system of n nonlinear equations in n unknowns may be grouped into $q+1$ subvectors $x_1, \cdots, x_{q+1}$ and $f_1, \cdots, f_{q+1}$ such that the nonlinear system of equations has the form

$$\begin{aligned} & f_i(x_i, x_{q+1}) = 0, \qquad i = 1, \cdots, q \\ & f_{q+1}(x_1, \cdots, x_{q+1}) = 0 \end{aligned} \tag{1.1}$$

where

$$x_i \in R^{n_i},$$
$$f_i \in R^{n_i}, \qquad i = 1, \cdots, q+1,$$
$$\sum_{i=1}^{q+1} n_i = n.$$

The block bordered Jacobian matrix of (1.1) is

$$\begin{pmatrix} A_1 & & & & B_1 \\ & A_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & A_q & B_q \\ C_1 & C_2 & \cdots & C_q & P \end{pmatrix}, \tag{1.2}$$

* Received by the editors January 16, 1990; accepted for publication (in revised form) May 14, 1991. This research was partially supported by Air Force Office of Scientific Research grant AFOSR-85-0251.

† Division of Mathematics and Computer Science, University of Texas, San Antonio, Texas 78285-0664 (zhang@ringer.cs.utsa.edu).

‡ Department of Computer Science, University of Colorado, Boulder, Colorado 80309-0430 (richard@cs.colorado.edu and bobby@cs.colorado.edu).

where

$$A_i = \frac{\partial f_i}{\partial x_i} \in R^{n_i \times n_i}, \quad i = 1, \cdots, q,$$

$$B_i = \frac{\partial f_i}{\partial x_{q+1}} \in R^{n_i \times n_{q+1}}, \qquad i = 1, \cdots, q,$$

$$C_i = \frac{\partial f_{q+1}}{\partial x_i} \in R^{n_{q+1} \times n_i}, \qquad i = 1, \cdots, q,$$

$$P = \frac{\partial f_{q+1}}{\partial x_{q+1}} \in R^{n_{q+1} \times n_{q+1}}.$$

Newton's method is the fundamental approach for solving a general nonlinear system of equations. Several parallel algorithms for solving general systems of nonlinear equations that are based upon Newton's method have been developed and implemented on various parallel computers. These parallel algorithms consist mainly of solving the linear Jacobian system in parallel. Many parallel algorithms have been developed for solving linear systems, such as as parallel factorizations, parallel SOR methods, parallel red-black methods, parallel multicolor, and others (see e.g., Ortega and Voigt [1985], O'Leary and White [1985], White [1986], Fontecilla [1987], and Coleman and Li [1990]).

In the case of very large nonlinear problems we cannot expect a single parallel algorithm to efficiently handle all the instances of the system of nonlinear equations problem, but rather the algorithm must take into account the sparsity structure and other special characteristics of the problem. In fact, many nonlinear problems arising in applications have their own special sparsity structure. Parallel algorithms taking advantage of this special structure may be much more efficient than the algorithms ignoring the special structure. This paper is an instance of developing special algorithms for a special, important structure.

1.2. Background on block bordered problems. Block bordered problems of the form (1.1) arise in many areas of science and engineering, and a few algorithms have been developed to efficiently solve linear block bordered systems of equations. In applications such as structural engineering, large spatial models may be divided into q regions such that each region only interacts directly with neighboring regions. The variables x_i for each region are chosen so that the model can determine their values, given the values of the linking variables (the x_{q+1}) at the boundaries of the regions. The linking variables are tied together by a $(q+1)$st set of equations representing the interactions between the regions. Thus the equilibrium equations for such a model will be of the form (1.1). In addition, the Jacobian matrix is symmetric. These problems, and parallel algorithms for solving the linear block bordered systems that arise from them, are discussed in Farhat and Wilson [1986] and Nour-Omid and Park [1986].

Mu and Rice [1989] study parallel Gaussian elimination for the block bordered matrices arising from the discretization of partial differential equations (PDEs). Christara and Houstis [1988], [1989] implement a domain decomposition spline collocation method and a preconditioned conjugate gradient (PCG) method for this linear block bordered system on both NCUBE/7 and Sequent multiprocessors.

All the work described above concerns parallel methods for solving linear block bordered equations. Our research is to develop, implement, and analyze parallel methods for solving nonlinear block bordered problems. To our knowledge, no one has done similar work.

Block bordered equations also arise in VLSI circuit design, where parts of the circuits may be divided into regions. The concept of macromodeling the circuit is to decompose the circuit into subcircuits and to analyze the subcircuits separately (see Rabbat, Sangiovanni-Vincentelli, and Hsieh [1979] and Rabbat and Sangiovanni-Vincentelli [1980]). Macromodeling of the circuit results in a system of nonlinear equations of form (1.1). A_i and B_i $(i=1, \cdots, q)$ in the Jacobian matrix are usually used to represent internal and input-output variables in each of the q independent subcircuits. The variables represent voltages and currents. The bottom block f_{q+1} represents the voltages or currents between subcircuits. Since each voltage or current is used only in one block of equations f_i plus possibly the bottom block f_{q+1}, the nonzero columns of the B_i's (and A_i) are disjoint, meaning that the matrices $B_i, \cdots B_q$ in (1.2) also follow a block diagonal pattern. In addition, since f_{q+1} describes the point-to-point connections of voltage and current, it is a linear function. The size of the function f_{q+1} depends on the number of connections among the subcircuits.

We have studied parallel methods for solving block bordered nonlinear equations extensively from both theoretical and practical viewpoints. Section 2 considers the explicit method and several implicit methods for solving block bordered nonlinear equations, and presents a new implicit approach—the corrected implicit method. A mathematical analysis of the two types of methods and a computational comparison in a sequential context is made. Section 3 briefly discusses techniques used to make these methods globally convergent. In § 4, we give a group of parallel algorithms for solving block bordered nonlinear systems of form (1.1), which may be implemented and distributed on both shared and distributed memory multiprocessors. The implementations and experimental results of these algorithms on the Intel hypercube, a distributed memory multiprocessor, are presented in § 5. Finally, our conclusions and some future research directions are summarized in § 6,

2. Explicit and implicit methods.

2.1. Introduction. There are two basic ways in which Newton's method can be applied to the nonlinear block bordered system of equations (1.1), which we refer to as the explicit and implicit approaches. The explicit approach is to simply apply Newton's method to (1.1). This involves iteratively solving the linear system

$$J(X^k)\Delta X^k = -F(X^k), \qquad k=0, 1, \cdots \tag{2.1}$$

for ΔX^k, where $J(X^k)$ is the Jacobian of F, which has the block bordered structure (1.2), and $X=(x_1, \cdots, x_q, x_{q+1})$.

The pure implicit approach is to use each of the q systems of nonlinear equations

$$f_i(x_i, x_{q+1})=0, \qquad i=1, \cdots, q \tag{2.2}$$

to solve for x_i, given a fixed value of x_{q+1}. This means that each of the x_i is implicitly given by a function of x_{q+1}. The whole problem (2.2) is then equivalent to solving

$$f_{q+1}(x_1(x_{q+1}), \cdots, x_q(x_{q+1}), x_{q+1})=0. \tag{2.3}$$

The Jacobian of this system is given by

$$\hat{J}=\frac{\partial f_{q+1}}{\partial x_{q+1}}-\sum_{i=1}^{q} \frac{\partial f_{q+1}}{\partial x_i}\left(\frac{\partial f_i}{\partial x_i}\right)^{-1} \frac{\partial f_i}{\partial x_{q+1}} \tag{2.4}$$

or

$$\hat{J}=P-\sum_{i=1}^{q} C_i A_i^{-1} B_i \tag{2.5}$$

and we may solve (2.3) by Newton's method. We will be considering practical variants of the implicit method that do not calculate $x_i(x_{q+1})$ exactly. We assume here and for the rest of § 2 that the matrices A_i and $\hat{J}$ are nonsingular. The situation where this does not hold will be discussed in § 3.

In this section we describe the explicit and implicit methods and their relations to each other, introduce the new corrected implicit method, and give some simple experimental results using these methods on a sequential computer.

2.2. Algorithms and analysis for the explicit and implicit methods. Newton's method applied to (1.1) in the explicit method consists of solving the following linear equations at iteration k $(k=0, 1, \cdots)$: from $f_i(x)=0$, $i=1, \cdots, q$,

$$A_i\Delta x_i^k + B_i\Delta x_{q+1}^k + f_i(x_i^k, x_{q+1}^k)=0, \tag{2.6}$$

and from $f_{q+1}(x_1^k, \cdots, x_q^k, x_{q+1}^k)=0$,

$$\sum_{i=1}^{q} C_i\Delta x_i^k + P\Delta x_{q+1}^k + f_{q+1}(x_1^k, \cdots, x_q^k, x_{q+1}^k)=0. \tag{2.7}$$

(For simplicity, we omit superscripts k on A_i, B_i, C_i, and P, but note that at least the A_i's and B_i's can change at each iteration.) Solving for Δx_i^k in (2.6) and substituting into (2.7), we obtain

$$\hat{J}\Delta x_{q+1}^k = -f_{q+1}(x_1^k, \cdots, x_q^k, x_{q+1}^k) + \sum_{i=1}^{q} C_iA_i^{-1}f_i(x_i^k, x_{q+1}^k) \tag{2.8}$$

where $\hat{J}$ is given by (2.5). So

$$x_{q+1}^{k+1} = x_{q+1}^k + \Delta x_{q+1}^k \tag{2.9}$$

can be determined from (2.8), and

$$x_i^{k+1} = x_i^k + \Delta x_i^k, \qquad i=1, \cdots, q \tag{2.10}$$

can be determined from (2.6).

In the pure implicit method, Newton's method is applied to (2.3) and gives

$$\hat{J}\Delta x_{q+1}^k + f_{q+1}(x_1(x_{q+1}^k), \cdots, x_q(x_{q+1}^k), x_{q+1}^k)=0, \tag{2.11}$$

where $x_i(x_{q+1}^k)$ $(i=1, \cdots, q)$ is implicitly determined by solving the nonlinear system

$$f_i(x_i, x_{q+1}^k)=0 \tag{2.12}$$

for x_i. To turn this into a practical computational procedure, we use a second (or inner) Newton process on (2.12) to calculate $x_i(x_{q+1}^k)$, which solves (2.12) approximately. For each $i=1, \cdots, q$, this yields the inner iteration

$$\hat{A}_i\Delta x_i^{k,j-1} + f_i(x_i^{k,j-1}, x_{q+1}^k)=0, \quad i=1, \cdots, q, \quad j=1, 2, \cdots, I_{in}. \tag{2.13}$$

Here $x_i^{k,0}=x_i^k$, I_{in} is the number of inner iterations, and $\hat{A}_i = A_i$ if it is only evaluated once at the beginning of each outer iteration; or, it may be evaluated up to I_{in} times. At the end of each inner iteration, we set

$$x_i^{k,j} = x_i^{k,j-1} + \Delta x_i^{k,j-1}, \qquad i=1, \cdots, q. \tag{2.14}$$

When we exit the inner iterations, we set

$$x_i(x_{q+1}^k) = x_i^{k+1} = x_i^{k,j}. \tag{2.15}$$

Then x_{q+1} is determined from (2.11) and

$$x_{q+1}^{k+1} = x_{q+1}^k + \Delta x_{q+1}^k. \tag{2.16}$$

The implicit Newton iteration (2.11) is also considered by Rabbat, Sangiovanni-Vincentelli, and Hsieh [1979], but instead of focusing on the number of inner iterations, they assume that (2.12) is solved to some tolerance. By considering a method with a fixed number of inner iterations, we are led to the following theorems, which show a close relationship between the explicit method and the implicit methods just described. We are also led to a new method—the corrected implicit method.

THEOREM 1. *If f_{q+1} is linear, the matrices A_i and $\hat{J}$ are nonsingular, and only one inner Newton iteration is applied to solve for $x_i^{k,j}$ $(i=1,\cdots,q)$ in the implicit method, i.e., $I_{in}=1$, then for a fixed k, the steps Δx_{q+1}^k are identical in both methods.*

Proof. In this case $x_i^{k+1}=x_i^k-A_i^{-1}f_i(x_i^k,x_{q+1}^k)$ and $f_{q+1}(x_1^{k+1},\cdots,x_q^{k+1},x_{q+1}^k)=f_{q+1}(x_1^k,\cdots,x_q^k,x_{q+1}^k)-\sum_{i=1}^q C_iA_i^{-1}f_i(x_i^k,x_{q+1}^k)$. Substituting this into (2.11) gives

$$\hat{J}\Delta x_{q+1}^k=-f_{q+1}(x_1^k,\cdots,x_q^k,x_{q+1}^k)+\sum_{i=0}^q C_iA_i^{-1}f_i(x_i^k,x_{q+1}^k), \tag{2.17}$$

which is identical to the explicit formula (2.8). □

The corrected implicit method. In the implicit method described above, the steps Δx_i for $i\leqq q$ do not involve any information about f_{q+1} or Δx_{q+1}, and are not the same as in the explicit method. If the value of $x_i^{k+1}(i\leqq q)$ calculated by the implicit method is corrected after each iteration to account for the change in x_{q+1}, however, the implicit method can be made closer to the explicit method and quadratically convergent. The problem may be defined to find a correction term δ such that

$$f_i(x_i^{k+1}+\delta,\ x_{q+1}^{k+1})\approx 0 \tag{2.18}$$

or

$$f_i(x_i^{k+1}+\delta,x_{q+1}^k+\Delta x_{q+1}^k)\approx 0. \tag{2.19}$$

Making a linear approximation to f_i in (2.19) yields the condition

$$f_i(x_i^{k+1},x_{q+1}^k)+A_i\delta+B_i\Delta x_{q+1}^k=0. \tag{2.20}$$

The correction term δ obtained from (2.20) would then be

$$\delta=-A_i^{-1}[f_i(x_i^{k+1},x_{q+1}^k)+B_i\Delta x_{q+1}^k]. \tag{2.21}$$

However, after I_{in} inner iterations of solving for x_i^{k+1}, $f_i(x_i^{k+1},x_{q+1}^k)\approx 0$. Thus we make a further approximation giving the correction term

$$\delta_i=-A_i^{-1}B_i\Delta x_{q+1}^k. \tag{2.22}$$

We call the new implicit method with this correction the *corrected implicit method.* The cost of the correction term (2.22) is small since the matrices $A_i^{-1}B_i$ $(i=1,\cdots,q)$ have been calculated already and in a parallel implementation, the matrix-vector products can be parallelized fully.

We can now see that when f_{q+1} is linear, the explicit method is a special case of the corrected implicit method.

THEOREM 2. *If f_{q+1} is linear, the matrices A_i and $\hat{J}$ are nonsingular, and only one Newton iteration is applied to solve for $x_i^{k,j}$ $(i=1,\cdots,q)$ in the implicit method, i.e., $I_{in}=1$, and the system is corrected by adding $-A_i^{-1}B_i\Delta x_{q+1}$ to $x_i^{k+1}(i\leqq q)$ after each iteration, then the explicit method and implicit method are identical.*

Proof. From Theorem 1, Δx_{q+1}^k is identical for the two methods. Combining (2.14) with $j=1$ and (2.22) gives

$$x_i^{k+1}=x_i^k-A_i^{-1}[f_i(x_i^k,,x_{q+1}^k)-B_i\Delta x_{q+1}^k] \tag{2.23}$$

which is identical to (2.6) in the explicit method. □

This equivalence, together with the standard convergence analysis for Newton's method, gives the following convergence result.

COROLLARY. *Suppose $J(X)$ is continuously differentiable near a solution X^*, the matrices $A_i(x^*)$ and $J(\hat{X}^*)$ are nonsingular, and f_{q+1} is linear. Then the corrected implicit method with $I_{in}=1$ inner iterations per outer iteration is locally quadratically convergent to the solution.*

Quadratic convergence can also be shown for $I_{in}>1$ and when f_{q+1} is nonlinear. The proof is given in Zhang [1989], and is based on the fact that the extra inner iterations tend to move X closer to the solution, and the nonlinearity in f_{q+1} at most adds a term of order $\|X^k - X^*\|^2$ to the error at X^{k+1}.

2.3. Some experiments on a sequential processor. The previous subsection shows that a variant of the implicit method is equivalent to the explicit method, but does not indicate why the implicit method might be preferred. The main reason is that, by using more than one inner iteration per outer iteration in the implicit method, the number of outer iterations can be reduced substantially, which is advantageous, especially for parallel computation. In this subsection we give a first indication of the sort of computational behavior that we have found.

We initially tested the methods discussed in this section on several artificial problems. Here we report results on a simple 20×20 nonlinear block bordered system of quadratic equations that has four 4×4 blocks, $A_1, \cdots, A_4$, and a 4×4 bottom block P, with f_{q+1} linear. In all cases, the starting value of x was close to the solution, and no global strategy (e.g., line search) was used. All these experiments were run on a Pyramid P90 computer.

First we compare the performance of the three methods when only one inner iteration ($I_{in}=1$) is used in the uncorrected implicit and corrected implicit methods (Table 2.1). The explicit method and the corrected implicit method with $I_{in}=1$ are identical in this case (see Theorem 2). Thus the same number of iterations is required to converge to the solutions. The computing times are slightly different since the implementations of the two methods are different. The uncorrected implicit method converges more slowly than the other two methods, which is reasonable since the correction step is needed to make it quadratically convergent.

TABLE 2.1
Experiments with the three methods.

($I_{in}=1$) Outer iterations (seconds)		
Explicit	Implicit	Corrected implicit
13 (0.44)	14 (0.40)	13 (0.40)

Next we increased the number of inner iterations in the uncorrected and corrected implicit methods. The experimental results (Tables 2.2 and 2.3) show that the number of outer iterations is sharply decreased when the number of inner iterations is two. However, the number of outer iterations decreases more slowly as I_{in} increases further. There exists an optimal value I_{in} for computing time in both the methods, but it is problem dependent. Our experiments also show that the corrected implicit method converges a little bit faster than the uncorrected implicit method when $I_{in}>1$, which is consistent with our convergence analysis. In § 5 we will see that for larger problems,

TABLE 2.2
Experiments with the (uncorrected) implicit method.

($I_{in} > 1$) Outer iterations (seconds)				
$I_{in} = 1$	$I_{in} = 2$	$I_{in} = 3$	$I_{in} = 4$	$I_{in} = 5$
14 (0.40)	8 (0.34)	7 (0.40)	6 (0.44)	6 (0.54)

TABLE 2.3
Experiments with the corrected implicit method.

($I_{in} > 1$) Outer iterations (seconds)				
$I_{in} = 1$	$I_{in} = 2$	$I_{in} = 3$	$I_{in} = 4$	$I_{in} = 5$
13 (0.40)	8 (0.38)	6 (0.36)	6 (0.50)	5 (0.54)

the improvements in time for the corrected implicit method with $I_{in} > 1$ can be considerably larger than those seen here. Also, we will see in §§ 4 and 5 that the decrease in iterations is advantageous for parallel computation.

3. Globally convergent modifications of the explicit and implicit methods. The explicit method and corrected implicit method are locally quadratically convergent to the solution. In other words, when the initial solution approximation is good enough, those methods are guaranteed to converge rapidly to a solution. However, it is often hard to find a good initial approximation for nonlinear problems in practice. In addition, many practical problems, such as the circuit equations, are highly nonlinear, and if the current solution approximation is not close enough, a Newton step may easily result in an increase in the function norm. For example, a small change in some voltage difference in a nonlinear circuit equation may result in a great change in an exponential term in a diode or transistor's function evaluation. Also, many block bordered equations result in nearly singular or singular Jacobians in the process of the iterations, for example, because a transistor with an exponential model is turned on at a nearly flat function curve (see Zhang, Byrd, and Schnabel [1989]). For these reasons, the methods need to be modified to handle unacceptable steps and singular Jacobian matrices in order to converge to a solution. In this section, we briefly describe the modifications we have used, which are motivated in part by their appropriateness for parallel distributed computation. They are described in more detail in Zhang [1989]. A global convergence analysis will be given in a forthcoming paper.

We achieve convergence from poor starting points by using a line search. The explicit method is just a standard Newton's method, so we can use a standard line search. That is, we calculate the overall step direction $d^k = (\Delta x_1^k \cdots \Delta x_q^k, \Delta x_{q+1}^k)$ as described in § 2.2, and then set

$$X^{k+1} = X^k + \lambda^k d^k,$$

where the steplength parameter $\lambda^k > 0$ is chosen by a line search procedure that assures sufficient descent on $\|F(X)\|_2$. Our line search is based upon Algorithm A6.3.1 in Dennis and Schnabel [1983].

The implicit method is more complicated since we have both inner and outer iterations. We need to choose the steps $\Delta x_i^{k,j} = (x_i^{k,j+1} - x_i^{k,j})$ in the inner iterations so that the overall step direction

$$d^k = \left(\sum_{j=0}^{I_{in}-1} \Delta x_1^{k,j} + \delta_1^k, \cdots, \sum_{j=0}^{I_{in}-1} \Delta x_q^{k,j} + \delta_q^k, \Delta_{q+1}^k \right), \tag{3.1}$$

where δ_i^k is the correction step (2.22), is a descent direction on $\|F(x)\|_2$. In addition, we would like the calculation of the steps $\Delta x_i^{k,j}$ for different values of i to be independent, so that the calculations of the inner iterations can be parallelized easily and efficiently.

Zhang [1989] shows that if each $\Delta x_i^{k,j}$ is calculated by

$$\Delta x_i^{k,j} = -\lambda_i^{k,j} A_i^{-1} f_i(x_i^{k,j}, x_{q+1}^k), \qquad i = 1, \cdots, q \tag{3.2}$$

(i.e., the step discussed in § 2.2 multiplied by a line search parameter $\lambda_i^{k,j} > 0$), the correction steps δ_i^k are calculated by (2.22), Δx_{q+1}^k is calculated by (2.11) as before, and f_{q+1} is linear, then d^k given by (3.1) satisfies

$$J(X^k)d^k$$

$$= -\left(\sum_{j=0}^{I_{in}-1} \lambda_1^{k,j} f_1(x_1^{k,j}, x_{q+1}^k), \cdots, \sum_{i=0}^{I_{in}-1} \lambda_q^{k,j} f_q(x_q^{k,j}, x_{q+1}^k), f_{q+1}(x_1^k, \cdots, x_q^k, x_{q+1}^k) \right).$$

Since the derivative of $\|F(X)\|_2^2$ in the direction d^k is equal to

$$F(X_k)^T J(X^k) d^k = -\sum_{i=0}^{q} \sum_{j=0}^{I_{in}-1} \lambda_i^{k,j} f_i(x_i^{k,j}, x_{q+1}^k)^T f_i(x_i^{k,0}, x_{q+1}^x) - \|f_{q+1}(X^k)\|^2,$$

it is clear that a sufficient condition for d^k given by (3.1) to be a descent direction on $\|f(X)\|_2$ is that for each $i = 1, \cdots, q$,

$$\sum_{j=0}^{I_{in}-1} \lambda_i^{k,j} f_i(x_i^{k,j}, x_{q+1}^k)^T f_i(x_i^{k,0}, x_{q+1}^k) > 0. \tag{3.3}$$

Note that (3.3) always holds for $I_{in} = 1$ (since each $\lambda_i^{k,1} > 0$), and that it is true for $I_{in} = 2$ if $\|f_i(x_i^{k,1}, x_{q+1}^k)\| < \|f_i(x_i^{k,0}, x_{q+1}^k)\|$ (which any line search will enforce) and $\lambda_i^{k,1} \leqq \lambda_i^{k,0}$. Thus we expect to get a descent direction most of the time. However, since (3.3) can be monitored independently for each i, the following parallel procedure could be used to guarantee that a descent direction is generated. For each j, the procedure calculates each $\Delta x_i^{k,j}$ by (3.2) using a standard line search as mentioned above, and then checks whether the corresponding partial sum of (3.3) is satisfied. If it is not, it sets $\Delta x_i^{k,l} = 0$ for $l = j, \cdots, I_{in} - 1$ and exits the inner iteration for x_i. The outer line search can be performed as in the explicit method.

Our approach for dealing with (nearly) singular Jacobians is based upon the Levenberg-Marquardt approach as described in Dennis and Schnabel [1983]. For a general system of nonlinear equations, if the current Jacobian matrix J is (nearly) singular, this approach modifies the search direction to be $-(J^TJ + \mu I)^{-1} J^T F$, where F is the current function value, and μ is a small positive number. This direction is a descent direction on $\|F(x)\|_2$ and is the solution to the trust region problem

$$\text{minimize}_d \ \|F + Jd\|_2 \quad \text{subject to } \|d\|_2 \leqq \Delta$$

for some $\Delta > 0$. In the limit as $\mu \to 0$, this direction equals $-J^+F$, where J^+ is the pseudoinverse of J.

In the explicit and implicit methods described in § 2, we need to solve systems of linear equations using the matrices $A_1, \cdots, A_q$ and the matrix $\hat{J} = P - \sum_{i=1}^{q} C_i A_i^{-1} B_i$. If any of these matrices, say, M, is nearly singular (i.e., either the factorization detects numerical singularity or the estimated condition number of M is greater than *macheps*$^{-1/2}$) we simply replace M^{-1} by $(M^T M + \mu I)^{-1} M^T$ in the formulas of § 2, where μ is chosen by the trust region strategy described by Dennis and Schnabel [1983], and is thus a function of M and the trust region size. These perturbations again have interpretations in terms of trust regions. Note also that the algorithms for deciding whether to perturb each $A_1, \cdots, A_q$, and for perturbing them if necessary, are totally independent so that they can be performed in parallel.

Combining these perturbation techniques with the inner line searches to assure descent at the outer iteration and global convergence is somewhat more complex, and will be addressed in a future paper. In our implementations, we have simply taken I_{in} inner iterations for each block i, $i = 1, \cdots, q$. We have used a standard line search to choose each $\lambda_i^{k,j}$ (requiring sufficient descent on f_i) but have not checked a condition like (3.3) that assures global descent, as this condition is more restrictive than necessary. To our knowledge, the algorithm has still always produced a descent direction.

The algorithm we implement is summarized below. If J or any A_i below is (nearly) singular, $\hat{J}^{-1}$ or A_i^{-1} is replaced by $(\hat{J}^T \hat{J} + \mu I)^{-1} J^T$ or $(A_i^T A_i + \mu_i I)^{-1} A_i^T$ for a small positive μ or μ_i, respectively. When f_{q+1} is linear, the explicit method is just a special case with $I_{in} = 1$ and each $\lambda_i^{k,1} = 1$.

IMPLICIT METHOD WITH GLOBAL MODIFICATION

1. For $j = 0, \cdots, I_{in} - 1$, calculate $x_i^{k,j+1} = x_i^{k,j} - \lambda_i^{k,j} A_i^{-1} f_i(x_i^{k,j}, x_{q+1}^k)$ where $\lambda_i^{k,j} \geqq 0$, $i = 1, \cdots, q$.
2. Form and factor $\hat{J} = P - \sum_{i=1}^{q} C_i A_i^{-1} B_i$.
3. Calculate $\Delta x_{q+1}^k = -\hat{J}^{-1} f_{q+1}(x_1^{k,I_{in}}, \cdots, x_q^{k,I_{in}}, x_{q+1}^k)$.
4. Calculate the corrections $\delta_i^k = -A_i^{-1} B_i \Delta x_{q+1}^k$ and set $\bar{x}_i^{k+1} = x_i^{k,I_{in}} + \delta_i^k$, $i = 1, \cdots, q$.
5. Calculate $X^{k+1} = X^k - \lambda^k d^k$ where $d^k = (\bar{x}_1^{k+1} - x_1^k, \cdots, \bar{x}_q^{k+1} - x_q^k, \Delta x_{q+1}^k)$.

4. Parallel explicit and implicit algorithms.

4.1. Motivation—LU factorization of block bordered linear equations.

Note that the LU factorization of the block bordered Jacobian matrix

$$\begin{pmatrix} A_1 & & & & B_1 \\ & A_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & A_q & B_q \\ C_1 & C_2 & \cdots & C_q & P \end{pmatrix}$$

is

$$\begin{pmatrix} L_1 & & & & \\ & L_2 & & & \\ & & \ddots & & \vdots \\ & & & L_q & \\ \hat{C}_1 & \hat{C}_2 & \cdots & \hat{C}_q & L_{q+1} \end{pmatrix} \begin{pmatrix} U_1 & & & & \hat{B}_1 \\ & U_2 & & & \hat{B}_2 \\ & & \ddots & & \vdots \\ & & & U_q & \hat{B}_q \\ & & \cdots & & U_{q+1} \end{pmatrix},$$

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

$$A_i = L_i U_i,$$
$$\hat{B}_i = L_i^{-1} B_i,$$
$$\hat{C}_i = C_i U_i^{-1},$$

and

$$L_{q+1}U_{q+1}=\hat{J}=P-\sum_{i=1}^{q} C_iA_i^{-1}B_i=P-\sum_{i=1}^{q} \hat{C}_i\hat{B}_i,$$

provided that the matrices A_i are nonsingular. (This is the same matrix $\hat{J}$ as in § 2.) The calculations of L_i, U_i, $\hat{B}_i$, and $\hat{C}_i$ for each i are independent, and thus can be parallelized very efficiently. The factorization of $\hat{J}$ must follow these calculations and will not parallelize as efficiently, especially on distributed memory multiprocessors, because it will require considerable communication.

A parallel version of the explicit method essentially consists of performing the above factorization in parallel at each iteration. The parallel version of the implicit method that we discuss next will be seen to perform closely related operations. The major difference will be that, by performing more than one inner iteration per outer iteration, it will spend a larger portion of its time on the calculations that parallelize very efficiently (those for blocks $1, \cdots, q$) and a smaller portion of its time on the calculations that parallelize less well—the formation and factorization of $\hat{J}$ and the outer line search. Thus the implicit method can be expected to parallelize more effectively than the explicit method, especially on distributed memory computers. If the two methods require similar amounts of time on sequential computers, as indicated in § 2, then the implicit method can be expected to be faster on parallel computers.

4.2. Parallel algorithms. Below we give a general description of a parallel corrected implicit method that is based upon the sequential method presented in §§ 2 and 3. The parallelism comes mainly from executing all the operations on blocks 1 through q, which have been designed to be independent, concurrently. The parallel explicit method is just the special case with $I_{in}=1$ and no inner line search.

```
INNER ITERATIONS
1. For $i=1, q$, Do in parallel:
   1.1. Factor $A_i$ and estimate its condition number Cond $(A_i)$.
   1.2. If Cond $(A_i)\leqq Tol$ then set $M_i=A_i$, $N_i=I$.
        Else choose $\mu_i>0$, form and factor $M_i=A_i^TA_i+\mu_iI$, set $N_i=A_i^T$.
   1.3. For $j=0, I_{in}-1$, Do:
        Solve $M_i\Delta x_i^{k,j}=-N_if_i(x_i^{k,j}, x_{q+1}^k)$ for $\Delta x_i^{k,j}$.
        Inner line search: $x_i^{k,j+1}=x_i^{k,j}+\lambda_i^{k,j}\Delta x_i^{k,j}$ for some $\lambda_i^{k,j}>0$.
   1.4. Solve $M_iW_i=N_iB_i$ for $W_i$.
   1.5. Calculate $T_i=C_iW_i$.

OUTER ITERATION
*2. Form $\hat{J}=P-\sum_{i=1}^q T_i$.
*3. Factor $\hat{J}$ and estimate its condition number Cond $(\hat{J})$.
*4. If Cond $(\hat{J})\leqq Tol$ then set $M=\hat{J}$, $N=I$.
    Else choose $\mu>0$, form and factor $M=\hat{J}^T\hat{J}+\mu I$, set $N=\hat{J}^T$.
*5. Solve $M\Delta x_{q+1}^k=-Nf_{q+1}(x_1^{k,I_{in}}, \cdots, x_q^{k,I_{in}}, x_{q+1}^k)$ for $\Delta x_{q+1}^k$.
 6. For $i=1, q$, Do in parallel:
    Calculate corrections $\delta_i^k=-W_i\Delta x_{q+1}^k$ and set $\bar{x}_i^{k+1}=x_i^{k,I_{in}}+\delta_i^k$.
*7. Outer line search: $X^{k+1}=X^k+\lambda^k(\bar{x}_1^{k+1}-x_1^k, \cdots, \bar{x}_q^{k+1}-x_q^k, \Delta x_{q+1}^k)$ for some
    $\lambda^k>0$.
```

The steps marked with stars require synchronization (on a shared memory multiprocessor) or communication (on a distributed memory multiprocessor). Step 2 requires synchronization if the matrices T_i are full. In the VLSI problems, however, the nonzero columns of B_i, and hence T_i, are disjoint (see § 1.2) and hence step 2 can be performed in parallel.

On shared memory machines, steps 3-5 can be performed in parallel using standard parallel methods for solving linear equations. On a distributed memory machine, it will only be efficient to perform steps 3-5 in parallel if the dimension of $\hat{J}$ is rather large. In our test problems, $\hat{J}$ was fairly small, so we performed steps 3-5 on one processor, on which we kept P, $\hat{J}$, and x_{q+1}. The remaining data was distributed in the obvious way: A_i, B_i, C_i, and x_i were stored together on one processor that handled block i. Step 7 includes two main operations, the calculation of trial points $\bar{x}^{k+1}$ and the evaluations of F at the points, which are performed in parallel on a shared memory machine, and may be performed in parallel on a distributed memory machine depending on their costs relative to the cost of communication.

5. Experimental results on a hypercube multiprocessor.

5.1. The test problem: A nonlinear block bordered circuit equation. The nonlinear block bordered application we consider for testing is the VLSI circuit simulation problem. Standard circuit simulation methods consist of stiffly stable implicit integration formulae to discretize the differential equations, Newton's method to solve the resulting nonlinear algebraic equations

$$F(X)=0, \tag{5.1}$$

and sparse LU decomposition to solve the linear equations that arise at each iteration

$$J\Delta X=-F(X), \tag{5.2}$$

where $J \in R^{n\times n}$ is the Jacobian matrix of (5.1). Typically, less than 2 percent of the entries of J are nonzero for $n>500$ (see, e.g., Sangiovanni-Vincentelli and Webber [1986]). The Newton iteration is repeated until the solution converges or the upper bound on the number of iterations is reached. The program then decides whether to accept the solution, based on its estimate of local truncation error and the number of iterations required.

As mentioned in § 1.2, partitioning the circuit leads to a block bordered system of nonlinear equations of the form (1.1) (see, e.g., Rabbat, Sangiovanni-Vincentelli, and Hsieh [1979]). Given a circuit network Γ, a group of partitioned subnetworks γ_i, $i=1,\cdots,q$, and the connecting current and voltage equations, the block bordered nonlinear system of equations is defined as follows. Currents between two subnetworks and voltages at the boundary are each represented by two variables, one in each subnetwork, which are set equal to each other by equations of f_{q+1}. Variables x_i $(i=1,\cdots,q)$ are used to represent internal voltages and current variables in each of the q independent subnetworks. Some of these are the current connecting variables among the q subnetworks. The variables x_{q+1} are used to represent the voltage connecting variables among the q subnetworks. Here the equations for voltages and currents are standard current equations involving resistors, transistors, diodes, voltage sources, and other elements. Since the connecting equation f_{q+1} is linear, the coefficient matrices C_i, $i=1,\cdots,q$ for the current connecting functions are constant, and the coefficient matrix P for the voltage connecting function is also constant.

For a very large circuit, the network Γ may be divided into subnetworks recursively, which leads to a multilevel block bordered system of nonlinear equations. In such a case, the diagonal blocks A_i $(i=1,\cdots,q)$ are themselves block bordered matrices. The border elements of the multilevel system represent the connections of the highest level.

We applied our algorithm to a simulation of the 741 op-amp circuit (see, e.g., Sedra and Smith [1982]), which was introduced in 1966 and is currently produced by almost every analog semiconductor manufacturer. The circuit is partitioned into four parts with roughly equal nodes in each subcircuit. A transistor is viewed as a nonlinear three-terminal device in the circuit. Thus, applying the Ebers–Moll transistor model (see Ebers and Moll [1954]), 24, 27, 23, and 27 KCL functions are defined in the first, second, third, and fourth block, respectively. The seven connections among the four blocks result in 14 linear current and voltage connecting functions. The total number of variables is $24+27+23+27+14=115$.

We also used a large analog filter composed of three 741 op-amp circuits (see, e.g., Smith [1971] and Valkenburg [1982]) to construct a two-level block bordered nonlinear system. The analog filter is first partitioned into three parts, each of which contains one 741 op-amp circuit. The first-level block bordered structure is thus formed with three diagonal blocks and one connecting block. Each of the diagonal blocks is a 741 op-amp circuit that is partitioned into the second-level block bordered structure.

5.2. The 741 op-amp circuit simulation on the Intel Hypercube. The nonlinear block bordered equations of the 741 op-amp circuit were solved in parallel on an Intel iPSC1 hypercube using the algorithm of § 4.2. The four blocks of the circuit were distributed among four nodes of the hypercube. For convenience, the steps involving the connection function f_{q+1} (steps 3–5 of the parallel algorithm) were performed on a different node which plays the control role. They could just as well have been done on one of the four nodes. Identical initial values were used as the inputs for all the above experiments, and the convergence tolerances were also the same for those experiments. The solutions of the experiments were verified by comparing them to the solutions computed by the program SPICE, which is a general-purpose circuit simulation program for nonlinear dc, nonlinear transient, and linear ac analysis (see Newton, Pederson, and Sangiovanni-Vincentelli [1988]).

Tables 5.1 and 5.2 show the experimental results for the explicit method. Tables 5.3 and 5.4 list the experimental results for the corrected implicit method with one or more than one inner iterations per outer iteration and with inner line searches. Note that as long as the inner line search is applied, the corrected implicit method even with one inner iteration per outer iteration is not the same as the explicit method.

In Tables 5.1 and 5.3, T_i $(i=1,\cdots,4)$ is the total computing time for all computations for solving the ith diagonal block on node i, T_b is the total computing time for all computations for solving the bottom block on the control node, T_c is the total

TABLE 5.1
The explicit method times for the op-amp 741 circuit.

T_1	T_2	T_3	T_4	T_b	T_c	N_{out}
12.81	14.20	13.45	14.58	3.42	0.43	20

TABLE 5.2
The explicit method parallel performance for the op-amp 741 circuit.

T_s	T_p	sp	eff
58.46	18.43	3.14	78.5%

TABLE 5.3
The corrected implicit method times for the op-amp 741 circuit.

I_{in}	T_1	T_2	T_3	T_4	T_b	T_c	N_{out}
1	11.81	13.06	11.96	13.28	2.84	0.38	18
2	12.60	14.01	12.83	14.45	1.65	0.32	15
3	11.28	12.84	11.46	13.01	1.38	0.25	12
4	18.48	20.57	18.81	21.34	1.32	0.25	11
5	29.04	32.12	29.92	32.89	1.34	0.24	11

TABLE 5.4
The corrected implicit method parallel performance for the op-amp 741 circuit.

I_{in}	T_s	T_p	sp	eff
1	52.95	16.5	3.21	80.25%
2	55.54	16.42	3.38	84.50%
3	49.97	14.64	3.43	85.75%
4	80.52	22.91	3.50	87.50%
5	125.31	34.47	3.60	90.0%

communication time for the computation, N_{out} is the total number of outer iterations required to converge to the solution, and I_{in} in Table 5.3 is the number of inner iterations used in the corrected implicit method. In the performance Tables 5.2 and 5.4, T_s is the computing time for solving the same problem on one node:

$$T_s = \sum_{i=1}^{4} T_i + T_b,$$

T_p is the parallel computing time;

$$T_p = \max(T_i, \cdots, T_4) + T_b + T_c,$$

sp is the speedup of the parallel computation:

$$sp = \frac{T_s}{T_p},$$

and eff is the parallel efficiency defined by

$$eff = \frac{sp}{\text{number of processors}}.$$

Our experiments show that the inner line search and inner iterations indeed speed up the convergence to the solution. For example, the experiment with one inner iteration with inner line search used 18 iterations to converge to the solution. The same experiment without inner line search (explicit method) used 20 iterations. As the number of inner iterations I_{in} is increased, the total number of outer iterations N_{out} decreases from 18 to 11, and the speedup increases because the bottom block computations constitute a smaller percentage of the overall computation. However, the sequential computing time only decreases by a small amount for $I_{in} = 3$, and then increases dramatically because the cost of the extra inner iterations swamps the small savings from the further decrease in the number of outer iterations. Both the sequential and parallel computing times are minimized when the number of inner iterations is $I_{in}=3$, and the speedup in this case, 3.43, is good.

Our experiments also show that the bottleneck computing time T_b of the corrected implicit method is more than 50 percent lower than in the explicit method if more than one inner iteration is applied. The communication time is also lower since the total number of iterations, and hence the time to send the updated variables among the nodes, is less than for the explicit method. Consequently, the advantage of the implicit method over the explicit method should be greater in larger problems where the amount of communication and cost of solving the bottom block are larger.

5.3. Experiments for two-level block bordered circuit equations. We also solved in parallel the two-level system of nonlinear block bordered equations for an analog filter formed by connecting together three blocks of the 741 op-amp circuit. We again used the Intel iPSC1 hypercube multiprocessor. The linearization of these equations at each iteration has the form

$$J\Delta X=-F, \tag{5.3}$$

where J is the two-level block bordered matrix

$$\left(\begin{array}{ccccccccccccc}
A_1^1 & & & B_1^1 & & & & & & & & & \\
 & \ddots & & \vdots & & & & & & & & & \hat{B}_1 \\
 & & A_q^1 & B_q^1 & & & & & & & & & \\
C_1^1 & \cdot & C_q^1 & P^1 & & & & & & & & & \\
 & & & & A_1^2 & & & B_1^2 & & & & & \\
 & & & & & \ddots & & \vdots & & & & & \hat{B}_2 \\
 & & & & & & A_q^2 & B_q^2 & & & & & \\
 & & & & C_1^2 & \cdot & C_q^2 & P^2 & & & & & \\
 & & & & & & & & A_1^3 & & & B_1^3 & \\
 & & & & & & & & & \ddots & & \vdots & \hat{B}_3 \\
 & & & & & & & & & & A_q^3 & B_q^3 & \\
 & & & & & & & & C_1^3 & \cdot & C_q^3 & P^3 & \\
 & \hat{C}_1 & & & & \hat{C}_2 & & & & \hat{C}_3 & & & \hat{P}
\end{array}\right),$$

$$\Delta X=(\Delta x_1^1,\cdots,\Delta x_q^1,\Delta x_{q+1}^1,\Delta x_1^2,\cdots,\Delta x_q^2,\Delta x_{q+1}^2,\Delta x_1^3,\cdots,\Delta x_q^3,\Delta x_{q+1}^3,\Delta\hat{x}_{q+1})^T,$$

and

$$F=(f_1^1,\cdots,f_q^1,f_{q+1}^1,f_1^2,\cdots,f_q^2,f_{q+1}^2,f_1^3,\cdots,f_q^3,f_{q\times1}^3,\hat{f}_{q+1})^T.$$

The system (5.3) could be solved by applying the block bordered solver to each of the three block bordered submatrices, and then solving the whole system by applying the block bordered solver again. Alternatively, the block bordered Jacobian matrix may be reordered to

$$\begin{bmatrix}
A_1^1 & & & & & & & & & B_1^1 & & & \\
& \cdot & & & & & & & & \cdot & & & \hat{B}_1 \\
& & A_q^1 & & & & & & & B_q^1 & & & \\
& & & A_1^2 & & & & & & & B_1^2 & & \\
& & & & \cdot & & & & & & \cdot & & \hat{B}_2 \\
& & & & & A_q^2 & & & & & B_q^2 & & \\
& & & & & & A_1^3 & & & & & B_1^3 & \\
& & & & & & & \cdot & & & & \cdot & \hat{B}_3 \\
& & & & & & & & A_q^3 & & & B_q^3 & \\
C_1^1 & \cdot & C_q^1 & & & & & & & P^1 & & & \\
& & & C_1^2 & \cdot & C_q^2 & & & & & P^2 & & \\
& & & & & & C_1^3 & \cdot & C_q^3 & & & P^3 & \\
& \hat{C}_1 & & & \hat{C}_2 & & & \hat{C}_3 & & & & & \hat{P}
\end{bmatrix},$$

with ΔX reordered to

$$\Delta X = (\Delta x_1^1, \cdots, \Delta x_q^1, \Delta x_1^2, \cdots, \Delta x_q^2, \Delta x_1^3, \cdots, \Delta x_q^3, \Delta x_{q+1}^1, \Delta x_{q+1}^2, \Delta x_{q+1}^3, \Delta \hat{x}_{q+1})^T,$$

and F reordered to

$$F = (f_1^1, \cdots, f_q^1, f_1^2, \cdots, f_q^2, f_1^3, \cdots, f_q^3, f_{q+1}^1, f_{q+1}^2, f_{q+1}^3, \hat{f}_{q+1})^T.$$

In solving this block bordered system, two levels of parallelism can be exploited. Let m be the number of amplifiers in the analog filter and q the number of subcircuits inside each amplifier; here $m = 3$ and $q = 4$. First the $m \times q$ independent operations for solving the diagonal blocks can be performed in parallel. Second, the m independent operations for transforming the matrices P^j, $j = 1, \cdots, m$, and solving the resultant systems of equations can be performed in parallel. Finally, the very bottom block, with the matrix $\hat{P}$, must be transformed and solved.

In our test program, the 12 diagonal block equations of the analog filter were distributed among 12 nodes of the Intel hypercube. The first level of internal connection functions in each amplifier, f_{q+1}^j $(j = 1, \cdots, 3)$, was distributed to three of the 12 nodes, and the second level connection function among the three amplifiers in the analog filter, $\hat{f}_{q+1}$, was handled sequentially by one of the 12 nodes.

Tables 5.5 and 5.6 give the experimental results and the performance of the explicit method for solving the two-level analog filter equation. Tables 5.7-5.11 show the

TABLE 5.5
The explicit method times for the analog filter.

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
13.49	14.93	14.21	15.34	13.44	14.85	14.16	15.46	13.36
T_{10}	T_{11}	T_{12}	T^1	T^2	T^3	T_b	T_c	N_{out}
14.76	14.25	15.63	3.01	3.12	3.17	0.58	1.31	21

TABLE 5.6
The explicit method parallel performance for the analog filter.

T_s	T_p	sp	eff
183.76	20.69	8.88	74.00%

TABLE 5.7
Corrected implicit method times for the analog filter, $I_{in} = 1$.

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
11.54	12.78	12.13	13.13	11.52	12.69	12.10	13.21	11.42
T_{10}	T_{11}	T_{12}	T^1	T^2	T^3	T_b	T_c	N_{out}
12.61	12.20	13.40	2.56	2.67	2.73	0.5	1.12	18

TABLE 5.8
Corrected implicit method times for the analog filter, $I_{in} = 2$.

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
11.83	11.93	11.84	12.09	11.91	12.01	11.89	12.12	11.84
T_{10}	T_{11}	T_{12}	T^1	T^2	T^3	T_b	T_c	N_{out}
12.05	11.94	12.13	1.65	1.65	1.64	0.36	0.73	12

TABLE 5.9
Corrected implicit method times for the analog filter, $I_{in} = 3$.

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
21.25	23.67	23.25	24.51	21.67	23.29	24.21	24.35	21.22
T_{10}	T_{11}	T_{12}	T^1	T^2	T^3	T_b	T_c	N_{out}
23.56	23.21	24.75	1.52	1.51	1.53	0.34	0.69	11

TABLE 5.10
Corrected implicit method times for the analog filter, $I_{in} = 4$.

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
29.03	32.24	29.98	32.75	29.11	32.04	29.87	32.71	29.40
T_{10}	T_{11}	T_{12}	T^1	T^2	T^3	T_b	T_c	N_{out}
32.21	31.05	32.34	1.50	1.51	1.50	0.35	0.69	11

TABLE 5.11
Parallel performance of the corrected implicit method for the analog filter.

I_{in}	T_s	T_p	sp	eff
1	157.19	17.19	8.86	73.83%
2	148.68	14.86	10.01	83.40%
3	283.84	27.31	10.39	86.58%
4	373.08	34.89	10.69	89.08%

experimental results and performance of the corrected implicit method for solving the two-level block bordered analog filter equations with one to four inner iterations. The symbols in the tables have the same meanings as in Tables 5.1–5.4, with the following exception: T_i, $i = 1, \cdots, 12$, is the total time for the 12 first-level blocks, while T^j, $j = 1, 2, 3$, is the time for the three second-level blocks.

Our experimental results show that the corrected implicit method is also more efficient than the explicit method on this larger block bordered system of equations. The total number of iterations N_{out} decreases from 18 to 11 as the number of inner iterations I_{in} is increased from 1 to 4, but the sequential computing time T_b only decreases from 157.19 to 148.68 for $I_{in} = 2$, then increases again. The high speedups for $I_{in} = 3$ and 4 in comparison to the same sequential method are not significant since the large number of inner iterations makes the algorithm inefficient, and the sequential time is suboptimal. For the optimal number of inner iterations, $I_{in} = 2$, the speedup is 10.01 out of 12 processors and the efficiency is 83.40 percent. The computation time improvement over the parallel explicit method is 28 percent, as compared to 19 percent in the sequential case. Our parallel analog filter simulation experiment indicates that applying the implicit method to solving large block bordered circuit equations on a distributed memory multiprocessor can result in high efficiency.

6. Summary and future research. We have introduced a corrected implicit method for solving block bordered systems of nonlinear equations. It allows multiple "inner" iterations, iterations on the variables, and equations of the q diagonal blocks, to be performed per each "outer" iteration, which involves all the variables and equations including the connecting bottom block. If only one inner iteration is performed per outer iteration, no line search is used, and the bottom connecting equations are linear, then the corrected implicit method is identical to the explicit method (Newton's method). When more than one inner iteration is performed per outer iteration, however, the methods are different, and in our experiments the corrected implicit method solves problems in somewhat less time than the explicit method on sequential computers. On parallel computers, the corrected implicit method has a larger advantage over the explicit method because it parallelizes more effectively, since the inner iterations constitute a larger percentage of the total computation and parallelize far better than the outer iterations. On one- and two-level block bordered problems from VLSI circuit design that we tested, the parallel efficiency of the fastest (sequential and parallel) corrected implicit method on an Intel iPSC1 hypercube was about 85 percent.

The methods presented in this paper all assume that the Jacobian matrix is available at each iteration, either analytically or by finite differences, and that it is not too expensive to evaluate. In some applications, however, the nonlinear equations are given by an expensive computational procedure, and analytic or finite difference Jacobians are very expensive to obtain. In such cases, for general systems of nonlinear equations, secant approximations to the Jacobian are used that are based entirely on function values at the iterates (see, e.g., Dennis and Schnabel [1983]). The development of related secant approximations to the Jacobian for block bordered nonlinear equations seems to be an attractive research topic, since it appears possible to construct approximations that retain the block bordered sparsity pattern of the Jacobian, and also allow the factorization of the Jacobian approximation to be updated efficiently.

REFERENCES

C. CHRISTARA [1988], *Spline collocation methods, software and architectures for linear elliptic boundary value problems*, Ph.D. thesis, Computer Science Department, Purdue University, West Lafayette, IN, August, 1988.

C. CHRISTARA AND E. HOUSTIS [1989], *A domain decomposition spline collocation method for elliptic partial differential equations*, in Proc. 4th Conf. Hypercube Concurrent Computers and Applications, Monterey, CA, March 6-8, 1989.

L. CHUA AND P. LIN [1975], *Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques*, Prentice-Hall, Englewood Cliffs, NJ.

T. COLEMAN AND G. LI [1990], *Solving systems of nonlinear equations on a message-passing multiprocessor*, SIAM J. Sci. Statist. Comput., 11, pp. 1116-1135.

J. DENNIS AND R. SCHNABEL [1983], *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ.

J. EBERS AND J. MOLL [1954], *Large-signal behavior of junction transistors*, Proc. IRE, 42, pp. 1761-1772.

C. FARHAT AND E. WILSON [1986], *Concurrent iterative solution of large finite element systems*, Tech. Report, Civil Engineering Department, University of California, Berkeley, CA.

R. FONTECILLA [1987], *A parallel nonlinear Jacobi algorithm for solving nonlinear equations*, Tech. Report, Computer Science Department, University of Maryland, College Park, MD, May.

M. R. GAREY AND D. S. JOHNSON [1979], *Computers and Intractability, A Guide to the Theory of* NP-*Completeness*, W. H. Freeman, San Francisco, CA.

P. GILL, W. MURRAY, AND M. WRIGHT [1981], *Practical Optimization*, Academic Press, New York.

M. MU AND J. RICE [1989], *Solving linear systems with sparse matrices on hypercubes*, Tech. Report CSD-TR-870, Computer Science Department, Purdue University, West Lafayette, IN, February.

A. NEWTON, D. PEDERSON, AND A. SANGIOVANNI-VINCENTELLI [1988], SPICE 3B1 *user's guide*, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA.

B. NOUR-OMID AND K. C. PARK [1986], *Solving structural mechanics problems on a Caltech hypercube machine*, Tech. Report, Mechanical Engineering Department, University of Colorado, Boulder, CO.

J. M. ORTEGA AND R. G. VOIGT [1985], *Solution of partial differential equations on vector and parallel computers*, SIAM Rev., 27, pp. 149-240.

D. P. O'LEARY AND R. E. WHITE [1985], *Multi-splittings of matrices and parallel solution of linear systems*, SIAM J. Algebraic Discrete Meth., 4, pp. 137-149.

N. RABBAT AND H. HSIEH [1976], *A latent macromodular approach to large-scale sparse networks*, IEEE Trans. Circuits and Systems, CAS-23, pp. 745-752.

N. RABBAT, A. SANGIOVANNI-VINCENTELLI AND H. HSIEH [1979], *A multilevel Newton algorithm with macromodeling and latency for the analysis of large-scale nonlinear circuits in the time domain*, IEEE Trans. Circuits and Systems, CAS-26, pp. 733-741.

N. RABBAT AND A. SANGIOVANNI-VINCENTELLI [1980], *Techniques of time-domain analysis of* LSI *circuits*, Tech. Report RC 8351 (#36320), IBM T. J. Watson Research Center, Yorktown Heights, NY, July.

C. ROMINE AND J. ORTEGA [1986], *Parallel solution of triangular systems of equations*, ICASE Tech. Report, National Aeronautics and Space Administration, Hampton, VA.

A. SANGIOVANNI-VINCENTELLI, L. CHEN, AND L. CHUA [1977], *An efficient heuristic cluster algorithm for tearing large-scale networks*, IEEE Trans. Circuits and Systems, CAS-24, pp. 709-717.

A. SANGIOVANNI-VINCENTELLI AND D. WEBBER [1986], *Computer architecture issues in circuit simulation*, in High Speed Computing, R. Wilhelmson, ed., University of Illinois Press, Chicago, IL.

A. SEDRA AND K. SMITH [1982], *Microelectronic Circuits*, CBS College Publishing, New York.

J. SMITH [1971], *Modern Operational Circuit Design*, John Wiley, New York.

M. VALKENBURG [1982], *Analog Filter Design*, CBS College Publishing, New York.

R. S. VARGA [1973], *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, NJ.

R. E. WHITE [1986], *Parallel algorithms for nonlinear problems*, SIAM J. Algebraic Discrete Meth., 7, pp. 137–149.

X. ZHANG [1989], *Parallel computation for the solution of nonlinear block bordered equations and their applications*, Ph.D. thesis, Department of Computer Science, University of Colorado, Boulder, CO, July.

X. ZHANG, R. BYRD, AND R. SCHNABEL [1989], *Solving nonlinear block bordered circuit equations on hypercube multiprocessors*, in Proc. 4th Conf. Hypercube Concurrent Computers and Applications, Monterey, CA, March 6–8.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 4, pp. 860-873, July 1992

AN EFFICIENT SCHEME FOR UNSTEADY FLOW PAST AN OBJECT WITH BOUNDARY CONFORMAL TO A CIRCLE*

MO-HONG CHOU†

Abstract. An efficient finite-difference method is presented for studying unsteady two-dimensional incompressible flow past an object with boundary conformal to a circle. At each time step the computations of vorticity transport and streamfunction are decoupled and based on a body-fitted, prowake, orthogonal grid. To handle the no-slip condition properly and efficiently, a new wall-vorticity conditioning algorithm is proposed and incorporated into the vorticity transport equation. When the object presents sharp edges, this algorithm performs particularly well in that it yields a significant increase in the time step increment that can be used in comparison to other known "decoupled" methods. In the method developed here the vorticity is advanced by an implicit scheme of Crank-Nicholson type, and the streamfunction equation is solved by a multigrid technique based on body-fitted coordinates. As an application, the experiment on flow past an inclined thin ellipse with angle of attack ranging from 5° to 85°, and at Reynolds number 200 and 400, is presented in detail. Depending on the angle of attack, the computed results present in the long run a quasi-steady or quasi-periodic feature, as demonstrated by the flow pattern and by the curves of drag, lift, and moment coefficients. Furthermore, the pivot point, with respect to which the thin ellipse exercises no torque, is also investigated and compared to the asymptotic result based on Kirchhoff theory. Such a result might be useful in modeling a pivoting prosthetic heart valve.

Key words. conformal mapping, Kirchhoff-Rayleigh, Navier-Stokes, Crank-Nicholson, vorticity conditioning, defect correction, multigrid, finite-difference, vector computer

AMS(MOS) subject classifications. 30A24, 65F10, 65M05, 76.65

1. Introduction. In this paper we present a finite-difference method to tackle the problem of unsteady two-dimensional flow past an object with boundary conformal to a circle. Such a problem is encountered, for example, in the modeling of prosthetic heart valves [7]. The numerical simulation is based on the unsteady Navier-Stokes equations, which are written in terms of vorticity and streamfunction. Some important features in our scheme are stated as follows.

We propose a novel, body-fitted, orthogonal grid system which, in contrast to the standard radial coordinates, automatically provides a prowake computational domain. Such a setup facilitates long time computations in the wake region. At each time step, the computations of vorticity and streamfunction are decoupled. The vorticity transport equation is handled by a Crank-Nicholson-type scheme in which a new wall-vorticity algorithm is incorporated. This algorithm provides a mechanism to couple the interior vorticity and the wall vorticity in such a way that the no-slip condition is handled properly and efficiently. This idea of vorticity conditioning is similar to that of Quartapelle's [8] in that the Green's identity plays an important role in both cases. However, as seen in § 2, our implementation is quite different from [8]. Owing to such an approach, the stability of the proposed numerical scheme is increased, and hence we are able to employ a larger time increment than can be used with other known "decoupled" schemes (see [5], for example). The elliptic problem that determines the streamfunction is solved by a multigrid method. The rate of convergence of the iterative method is extremely fast, as compared, for example, to unigrid line SOR [5]. The accuracy of the whole scheme is approximately of the second order in space. The temporal accuracy of the proposed Crank-Nicholson-like scheme, strictly speaking,

* Received by the editors February 20, 1990; accepted for publication (in revised form) March 6, 1991.

† Institute of Mathematics, Academia Sinica, Nankang, Taipei, Taiwan 11529, Republic of China.

is of the first order, since there is a time lag in updating the streamfunction. However, the efficiency we have gained from this scheme is better than the backward Euler scheme. Finally, since the computation domain is doubly connected, the single-valuedness of the surface fluid pressure might be of great concern in the conventional approach to compute the fluid dynamic properties, such as drag and lift (see Wu [12]). We propose an alternative to circumvent this difficulty. The idea has the same spirit as our vorticity conditioning algorithm. Namely, the Green's identity is employed together with auxiliary functions in order to get an integral viewpoint of the desired data. Van Der Vegt [11] has also used such an alternative in conjunction with discrete vortex methods. For flow past an object with sharp edges, this alternative is important in our experience in that it generates much smoother data, especially when we use an unsteady approach to quasi-steady flow problems.

This paper is organized as follows. In § 2 we present the details of our approach for the problem of flow past an inclined thin ellipse. For other bodies, the only adaptation required is the change of the conformal mapping from a circle to the boundary of the object. As we check on the accuracy of our approach, we compare our results with an asymptotic result concerning the pivot point about which the ambient fluid exerts no torque on the ellipse. This asymptotic result is presented in § 3. This result is derived from the classical Kirchhoff–Rayleigh theory and provides elegant information on how this point varies with the angle of attack, which would be quite useful, for example, for designing a pivoting artificial heart valve whose maximum angle of opening (that is, 90° minus the torque-free angle of attack) matches the physiological conditions (see the work of Peskin and McQueen [7]).

The efficiency and accuracy of our approach is demonstrated in § 4 for flow past an inclined thin ellipse. The angle of attack ranges from 5° to 85°. The Reynolds number is taken to be between 200 and 400. Depending on the angle of attack, the computed results present, in the long run, a quasi-steady or quasi-periodic feature, as demonstrated by the flow pattern (streamlines, lines of equal vorticity) and by the curves of drag, lift, and moment coefficients. Among these computed results a particular case of 45° incidence is also compared with the work of Lugt and Haussling [5], where good agreement is shown. Furthermore, the aforementioned zero-torque pivot point is also investigated and compared with the Kirchhoff theory. The Strouhal number, regarded as a function of the angle of attack, is also compared with some relevant experiments [1], [2], [5], and it is shown that our result is within the correct range.

2. Numerical method for Navier–Stokes equations. As sketched in Fig. 1, we consider a uniform flow, with density $\rho \equiv 1$ and velocity $(U_\infty 0)$ at infinity, past a thin ellipse inclined at an angle of attack α measured clockwise from the negative x axis. The thinness of the ellipse is measured by the aspect ratio $\lambda \equiv l/L$, in which $l(L)$ denotes the length of the short(long) axis. In our consideration $\lambda = 0.1$ (or 10 percent) is taken in order to have the results directly compared with existing literature.

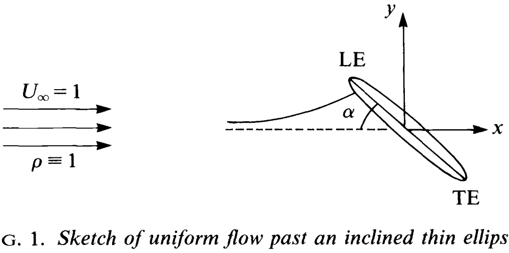


FIG. 1. *Sketch of uniform flow past an inclined thin ellipse.*

The flow is governed by unsteady Navier-Stokes equations, which we write in terms of vorticity(ω) and streamfunction(ψ) as follows.

$$\omega_t + \psi_y\omega_x - \psi_x\omega_y = \frac{2}{R}(\omega_{xx} + \omega_{yy}), \tag{2.1}$$
$$\psi_{xx} + \psi_{yy} = -\omega.$$

System (2.1) is supplemented by some initial and boundary conditions, which will be stated later. Note that in (2.1) the length scale is based on half long axis, i.e., $L/2$, while the Reynolds number R is defined as $R = (LU_\infty/\nu)$ (ν is the kinematic viscosity).

The proposed finite-difference scheme for solving (2.1) starts with the following generation of a body-fitted orthogonal coordinate system. This system is essentially composed of two conformal mappings. Let $z = x + iy$ denote the physical coordinates, and $\zeta = \xi + is$ the transformed coordinates (note that $i = \sqrt{-1}$). Then, the map $z = z(\mathfrak{z}(\zeta))$ is determined as follows.

$$z = \frac{\sqrt{1-\lambda^2}}{2}\left(\mathfrak{z} + \frac{e^{-2i\alpha}}{\mathfrak{z}}\right) \quad \text{for } |\mathfrak{z}| \geqq \sqrt{\frac{1+\lambda}{1-\lambda}} \equiv d; \tag{2.2a}$$

$$\zeta = \frac{-i}{2d}\left(\mathfrak{z} - \frac{d^2}{\mathfrak{z}}\right) + \frac{\Gamma}{\pi i}\operatorname{Log}(\mathfrak{z}/d) \quad \text{for } \zeta \text{ such that } -\Gamma \leqq \xi \leqq \Gamma,\ s \leqq 0. \tag{2.2b}$$

We note that (2.2a) provides desired high resolution in the vicinity of the edges (where $\mathfrak{z} = \pm de^{-i\alpha}$) of the ellipse, as is done in the standard elliptic coordinates. The novelty of our system is due to (2.2b), which generates a prowake computation domain (see Fig. 3) through the introduction of a strong circulation term with strength $2\Gamma \gg 0$. To increase the resolution in the proximity of the ellipse, a quadratic scaling in the s-direction is also introduced. Namely, let a, b be suitable positive constants; then

$$s = -(a\eta + b\eta^2) \equiv f(\eta) \quad \text{for } \eta \geqq 0. \tag{2.2c}$$

We also note that, in our scheme, (2.2a) is the only component that varies with the boundary of the object in consideration.

By virtue of (2.2), the Navier-Stokes equations (2.1) are transformed into the following. For $(\xi, \eta) \in [-\Gamma, \Gamma] \times [0, \eta_\infty]$,

$$(Jf'^2)\omega_t + f'(\psi_\eta\omega_\xi - \psi_\xi\omega_\eta) = \frac{2}{R}\left[\omega_{\eta\eta} - \frac{f''}{f'}\omega_\eta + f'^2\omega_{\xi\xi}\right], \tag{2.3a}$$

$$\psi_{\eta\eta} - \frac{f''}{f'}\psi_\eta + f'^2\psi_{\xi\xi} + (Jf'^2)\omega = 0. \tag{2.3b}$$

In (2.3), $J = |dz/d\zeta|^2$ is the Jacobian of the mapping composed of (2.2a) and (2.2b), and f', f'' denote the first and second derivatives of the function f defined in (2.2c). Note that, in the sequel, J is modified to denote the combined term Jf'^2.

The initial condition of (2.3) is specified by uniform potential flow past the given ellipse. We denote the initial streamfunction by ψ°. With the aid of Fig. 3, in which the mapping (2.2) is sketched, boundary conditions are specified as follows.

(2.4a) $\psi = \dfrac{\partial \psi}{\partial \eta} = 0$ for $\eta = 0$, $-\Gamma \leqq \xi \leqq \Gamma$;

(2.4b) $\psi(-\Gamma, \eta) = \psi(\Gamma, \eta)$, $\omega(-\Gamma, \eta) = \omega(\Gamma, \eta)$ for $0 \leqq \eta \leqq \eta^*$, where, associated with (2.2), $\zeta = \pm\Gamma + if(\eta^*)$ is the point(s) such that $|z'(\zeta)| = \infty$;

(2.4c) $\psi(\pm\Gamma, \eta) = \psi^\circ(\pm\Gamma, \eta)$, $\omega(\pm\Gamma, \eta) = 0$ for $\eta^* \leqq \eta \leqq \eta_\infty$,

(2.4d) $\dfrac{\partial \psi}{\partial \eta} = \dfrac{\partial \psi^\circ}{\partial \eta}$, and ω is updated through (2.3a) with $R = \infty$, for $\eta = \eta_\infty$, $-\Gamma \leqq \xi \leqq \Gamma$.

We note in passing that in our experiment the downstream boundary conditions for ψ, as proposed by Lugt and Haussling [5] and by Mehta and Lavan [6], will lead to abnormal vorticity accumulation along the downstream boundary for long-term computation, unless η_∞ is sufficiently large. To obtain comparable results, condition (2.4d), on the other hand, allows us to use smaller η_∞, which in turn leads to a smaller algebraic system to be solved.

In our computation, the vorticity and streamfunction are decoupled at each time step. According to such an approach the specification of wall vorticity plays a very important part in the stability consideration, especially in the presence of sharp edges. The standard method, which utilizes (2.3b) together with (2.4a), places a severe limitation on the time increment (see [5], for example). To relax this limitation, we propose a new approach, which is based on the following theorem.

THEOREM 1. (As shown in Fig. 2.) *Let Ω be a neighborhood of the ellipse. The boundary $\partial\Omega = B_0 \cup B_1$ with unit outer normal ν. If ω, ψ are such that $\Delta\psi = -\omega$ in Ω and $\psi \equiv$ constant on B_0, then the condition that $\partial\psi/\partial\nu = 0$ on B_0 is equivalent to the following*:

(2.5a) $$\int_\Omega \omega g_n\,dxdy + \oint_{B_1} \left[g_n \frac{\partial \psi}{\partial \nu} - \psi \frac{\partial}{\partial \nu}(g_n) \right] ds = 0 \quad \textit{for } n = 0, 1, 2, \cdots,$$

where, with the aid of (2.2), *the auxiliary functions g_n are given by*

(2.5b) $$g_n(z) \equiv g_n(z(\zeta)) = \exp(i\pi n(\bar{\zeta} + \Gamma)/\Gamma) \quad \textit{for } n = 0, 1, 2, \cdots.$$

The proof of Theorem 1 is a straightforward application of Green's theorem and Fourier analysis, and is therefore omitted.

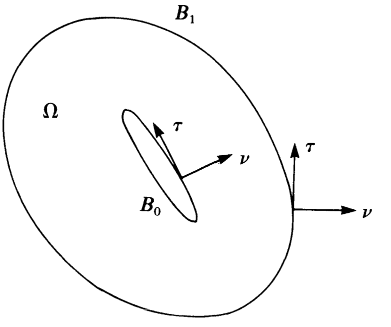


FIG. 2. *Illustration of Theorem* 1 *for wall-vorticity conditioning.*

This idea of replacing the no-slip condition by a set of integral constraints on ω has been used, for example, by Quartapelle [8]. As compared to [8], some major differences are noted below. First, the integration domain Ω is just a small portion of the whole computational domain. Thus, for each n, (2.5) contains a term involving ψ, which, though not known in advance, is assumed to have a smooth time-varying property provided that B_1 is chosen not too close to the ellipse B_0. Second, the test harmonic functions, i.e., $\{g_n\}$, are given analytically. However, the expression of ω in terms of $\{g_n\}$ is not pursued. Thus Theorem 1 is applied to vorticity conditioning in a way quite different from [8], as will be seen below.

The transformed Navier–Stokes equations (2.3) are discretized over a uniform grid. Centered differencing is used to approximate spatial differentiation except for the terms ω_ξ and ω_η. For these two terms, a third-order upwinding involving cubic interpolation [4] is used instead, in order to suppress spurious oscillations that occur as the Reynolds number R gets large, while the mesh is fixed. This type of upwinding, in our experience, is certainly superior to those hybrid schemes combining first-order upwinding and centered difference. In fact, this kind of hybrid scheme will introduce erroneous numerical diffusion around the sharp edges, which eventually destroys the accuracy of the whole computation.

To apply Theorem 1, we couple (2.5) with (2.3a). Before doing so, we assume they have the following discrete forms.

$$J\omega_t + G(\psi, \omega_0)\omega = 0 \quad \text{for (2.3a)}, \tag{2.6a}$$

$$A_0 \cdot \omega_0 + A \cdot \omega + g \cdot \psi = 0 \quad \text{for (2.5)}. \tag{2.6b}$$

The matrices A_0, A, and g in (2.6b), with A_0 invertible, are discrete integral operators corresponding to suitable quadrature formulas. Note that the wall vorticity ω_0 has been separated from the interior vorticity ω. In (2.6a) the convection and diffusion of ω is handled by the operator G, which is discretized according to the discussion of the last paragraph.

By virtue of (2.6), we can discuss the advancement of ω, ω_0 as follows. Let Δt denote the time increment and $(\omega^n, \omega_0^n, \psi^n)$ denote the quantities at the nth time step. Then, $(\omega^{n+1}, \omega_0^{n+1})$ are obtained as the asymptotic steady state of the following coupled ordinary differential equations:

$$\frac{d\omega}{d\tau} + \omega - \omega^n + \frac{\Delta t}{2}\{G(\psi^n, \omega_0)\omega + G(\psi^n, \omega_0^n)\omega^n\}/J = 0, \tag{2.7a}$$

$$\frac{d\omega_0}{d\tau} + \beta[\omega_0 + A_0^{-1}(A \cdot \omega + g \cdot \psi^n)] = 0 \quad \text{for } \tau > 0, \tag{2.7b}$$

$$\omega = \omega^n, \qquad \omega_0 = \omega_0^n \quad \text{at } \tau = 0. \tag{2.7c}$$

Note that the relaxation parameter $\beta > 0$ in (2.7b) is introduced to accelerate the convergence. Also note that (2.7a) presents an implicit scheme for the interior vorticity, which is of Crank-Nicholson type, when the τ-derivative term becomes negligible. Since there is a time lag in updating ψ, the temporal accuracy of this scheme, strictly speaking, is of the first order. However, it is more accurate than the backward Euler scheme.

The ODE (2.7) is solved over a sequence of discrete τ, namely, $\tau = k\Delta\tau$, $k \geqq 0$. Denote the result of the kth step by $(\omega^{n+1,k}, \omega_0^{n+1,k})$. Then $(\omega^{n+1,k+1}, \omega_0^{n+1,k+1})$ is obtained through the following algorithm.

ALGORITHM A.

(1) Defect-correction formula for (2.7a):

$$J\left(\frac{1}{\Delta\tau}+1\right)\delta^{k+1}+\frac{\Delta t}{2}F(\psi^n,\omega_0^{n+1,k})\delta^{k+1}$$

$$=-\left\{J(\omega^{n+1,k}-\omega^n)+\frac{\Delta t}{2}[G(\psi^n,\omega_0^{n+1,k})\omega^{n+1,k}+G(\psi^n,\omega_0^n)\omega^n]\right\}$$

$$\equiv H^k,$$

$$\omega^{n+1,k+1}=\omega^{n+1,k}+\delta^{k+1},$$

where F differs from G only in the convection part in which first-order upwinding is used instead.

(2) Obtain approximate δ^{k+1} for (1) through incomplete LU factorization (ILU).

(3) Substitute $\omega^{n+1,k+1}$ into (2.7b), and solve it for $\omega_0^{n+1,k+1}$ by forward Euler method.

(4) Go back to (1) and repeat the cycle until, for some prescribed tolerance ε, RMS $(H^k)<\varepsilon$ where RMS $(\cdot)$ means the root mean square over the whole computational grid.

Some remarks on the efficiency and accuracy of Algorithm A are made below. Associated with our choice of $\{g_n\}$ in (2.5), the matrix A_0 is the product of an orthogonal matrix and a diagonal one. So the additional effort to obtain A_0^{-1} is negligible. Furthermore, the computation of $A_0^{-1}(A\cdot\omega+g\cdot\psi)$ is fully vectorizable and hence the labor of updating ω_0 through (2.7b) can be largely saved on a vector computer. For fixed Δt, we can adjust $\Delta\tau$ and β such that ILU is an excellent preconditioner, and the stopping criterion can be met in just a few cycles. If the tolerance $\varepsilon=0$, our algorithm is of second-order accuracy in space. For small ε we expect that it is approximately of this order. Finally, the complexity of Algorithm A is well offset by its stability, which allows us to employ much larger Δt, as compared to other "decoupled" methods [5].

After obtaining ω^{n+1}, we must update ψ by solving the transformed Poisson equation (2.3b) together with the boundary conditions (2.4), in which we note that the condition $\partial\psi/\partial\eta=0$ along the ellipse has been dropped. To this end, a three-grid iterative method is adopted, which yields considerably fast convergence as compared, for example, to one-grid line SOR [5]. Such an approach is also more efficient than GMRES [9] in our experiment. The basic ingredients of this scheme are listed as follows. Namely, standard double coarsening strategy on the grid, five-point central difference formula for (2.3b), incomplete LU factorization as smoothing operator, and standard nine-point prolongation and restriction scheduled in a "saw-tooth" manner. For details of this framework, we refer to the work of Sonneveld, Wesseling, and de Zeeuw [10]. The overall efficiency of the proposed Navier–Stokes solver is evidenced in §4.

Finally, we discuss the computations of drag, lift, moment, and the pivot point of zero torque. As a rule, the surface pressure p is calculated through the expression

$$\frac{\partial p}{\partial\tau}=\frac{2}{R}\frac{\partial\omega}{\partial\nu}, \tag{2.8}$$

where τ, ν denote the counterclockwise tangential and outer normal components, respectively. Since the computational domain is doubly connected, the single-valuedness of this computed surface pressure might be of great concern. See, for example, the discussion of Wu [12]. In our experience in small angle-of-attack simulations, formula (2.8) leads to unpleasant fluctuations in determining the pivot point of zero torque, while the variation of this point is supposed to be quite small in these quasi-steady situations. To circumvent this difficulty, we propose the following alternative. The idea has the same spirit as our vorticity conditioning algorithm, Theorem 1, as will be seen in Theorem 2. Van der Vegt [11] has also used such an alternative in conjunction with discrete vortex methods.

THEOREM 2. *Let Ω, B_0, B_1 be as shown in Theorem* 1, *and ω, ψ denote the vorticity and streamfunction, respectively. Then, the drag, lift, and moment exerted by the nondimensionized surface pressure can be expressed as follows.*

$$\text{(2.9a)} \qquad -\frac{d}{dt}\oint_{B_1}\psi d\varphi+\int_{\Omega}\omega\nabla\psi\cdot\nabla\varphi dxdy-\frac{2}{R}\left[\oint_{B_1}\omega d\varphi-\oint_{B_0}\omega d\varphi\right]$$

where the auxiliary function φ satisfies the following partial differential equation (PDE):

$$\text{(2.9b)} \qquad \begin{aligned} &\Delta\varphi=0 \quad \text{in } \Omega,\\ &\frac{\partial\varphi}{\partial\nu}=0 \quad \text{on } B_1,\\ &\frac{\partial\varphi}{\partial\nu}=\frac{\partial g}{\partial\tau} \quad \text{on } B_0,\end{aligned}$$

where $g(x, y)=-y$ for drag; $=x$ *for lift*; $=(x^2+y^2)/2$ *for moment.*

The proof is sketched as follows. By definition, we have (drag, lift, moment)$_p=$ $\oint_{B_0}p(-dy, dx, xdx+ydy)$. Substitute the result of (2.9b) into the last integral. Then, (2.9a) follows from Green's theorem, momentum equation $(\vec{u}_t+(\vec{u}\cdot\nabla)\vec{u}+\nabla p=(2/R)\Delta\vec{u})$, the Gauss theorem, and the physical boundary conditions on the velocity.

The skin friction part is handled as usual, namely,

$$\text{(2.10)} \qquad (\text{drag, lift, moment})_s=\frac{2}{R}\oint_{B_0}\omega(dx, dy, x\,dy-y\,dx).$$

For large R, the contribution of (2.10) is minor, except for the drag component in those cases where α is small, as compared to the pressure part (2.9). The drag, lift, and moment coefficients are defined as

$$\text{(2.11)} \qquad (C_D, C_L)=(\text{drag, lift})/\tfrac{1}{2}\rho U_\infty^2 L \quad \text{and} \quad C_M=\text{moment}/\tfrac{1}{2}\rho U_\infty^2 L^2.$$

Along the long axis of the ellipse, there exists a pivot point with respect to which the ellipse experiences no torque. The signed distance, q, of this point from the ellipse center is given by

$$\text{(2.12)} \qquad q=[C_M/(C_D\cdot\sin\alpha+C_L\cdot\cos\alpha)]\cdot L.$$

If $q<0$ this point is closer to the leading edge than to the trailing edge, and conversely if $q>0$.

3. An asymptotic result for zero-torque pivoting. For studying high Reynolds number flow past a thin ellipse, the classical Kirchhoff-Rayleigh model [3, §§76–77] of steady inviscid flow past a flat plate can be used as a first approximation. Thus we consider a uniform potential flow past a flat plate inclined at an angle of attack α, $0<\alpha\leqq\pi/2$. Behind the plate the wake is occupied by dead fluid, which is bounded by two free streamlines emanating from the edges of the plate. Let the plate, of length L, be uniformly parametrized by q such that $-1\leqq q\leqq 1$, in which $q=-1$ and $+1$ correspond to the leading edge (LE) and the trailing edge (TE), respectively.

With respect to a pivot point q, the ambient fluid exerts a torque T on the plate, which, after being nondimensionized by $\frac{1}{2}\rho U_\infty^2 L^2$, has the following form:

$$T(\alpha, q) = -\frac{\pi \sin \alpha}{4+\pi \sin \alpha}\left[q+\frac{3 \cos \alpha}{2(4+\pi \sin \alpha)}\right]. \tag{3.1}$$

Formula (3.1) is derived by following the procedure as presented in [3]. From (3.1) it is easy to see that the pivot point of zero torque is

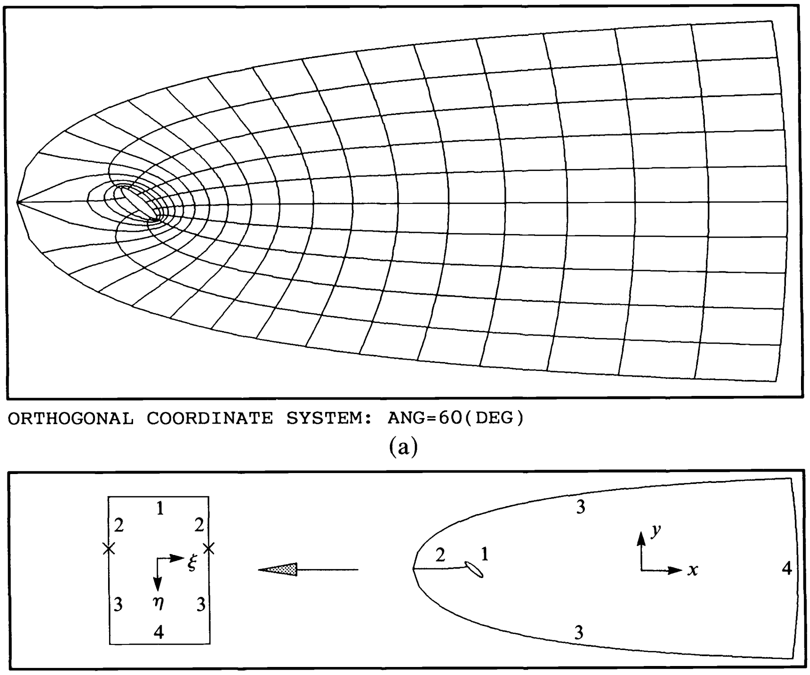
$$q(\alpha) = \frac{-3 \cos \alpha}{2(4+\pi \sin \alpha)} \quad \text{for } 0<\alpha \leqq \pi/2. \tag{3.2}$$

In view of (3.2), it is easy to check that: (1) The zero-torque pivot point is unique for each α with $0<\alpha \leqq \pi/2$; (2) The distance $1+q$ of this point from the leading edge of the plate is a smooth increasing function of α; (3) As $\alpha \to 0+$, $1+q \to \frac{5}{8} > 0$, however. This means that when the plate is placed parallel to the incoming flow, there exists, starting from the leading edge, a wide range for picking a torque-free pivot point.

4. Numerical results.

4.1. Computing facilities. The numeric computations were run on an ETA10-Q108 computer. We do take advantage of this machine's vector facilities. Associated with our program, the vector/scalar speed ratio is 8/1. The numerical results were then loaded down to a VAX8530 computer in which we used GKS to generate the graphic output.

4.2. Computational domain. Figure 3 sketches the result of (2.2). In the transformed $\xi\eta$-plane, we employ a 60×80 uniform grid. The corresponding physical domain



ORTHOGONAL COORDINATE SYSTEM: ANG=60(DEG)

(a)

BOUNDARY OF COMPUTATIONAL DOMAIN

(b)

FIG. 3. (a) *Body-fitted, prowake, orthogonal computational mesh*; (b) *Boundary correspondence between physical and transformed domains.*

is outerscribed by the rectangle $[-7, 50]\times[-17, 17]$, and the downstream boundary is about 25 plate lengths from the ellipse center.

4.3. Vorticity transport with $R=200$ and 400. According to Algorithm A in §2, the interior and wall vorticity are advanced with time increment $\Delta t=0.04$, which is 8 times as large as that used by Lugt and Haussling [5]. To check the temporal accuracy, a comparative experiment with $\Delta t=0.02$ is also performed. For $\alpha\leqq 45°$ the difference is negligible. The cases with higher α will be discussed later in conjunction with the estimate of Strouhal number. A much smaller time increment, e.g., $\Delta t/20$, is used for the first few steps in order to make a smooth transition from the potential flow. In Algorithm A, the pseudo time increment $\Delta\tau$ is chosen as $\Delta\tau=1.5$ for $R=200$, and as $\Delta\tau=1.75$ for $R=400$. The relaxation parameter β for wall vorticity is chosen such that $\beta\cdot\Delta\tau=0.45$. Although such an arrangement of β and $\Delta\tau$ is not optimal, the average reduction rate $\mu\equiv \text{RMS}\,(H^k)/\text{RMS}\,(H^{k+1})$ reaches the value 2.4 for $R=200$, and 2.7 for $R=400$. With $\varepsilon=2\times10^{-5}$, Algorithm A takes few ($\leqq 6$) iterations to advance one time step for angle of attack α ranging from 5° to 85°. The long-term behavior of the vorticity transport presents either a quasi-steady or a quasi-periodic feature, depending on the angle of attack. Figures 4(a)–4(d) show several stages in one period of vortex shedding for $\alpha=60°$ and $R=200$. In our computations, the results for $R=200$ and for $R=400$ are quite similar.

4.4. Streamfunction calculation. As discussed in §2, the transformed Poisson equation (2.3b) was solved by a three-grid iterative method. The stopping criterion is that the RMS (see Algorithm A) of the left-hand side of (2.3b) is less than or equal to 2×10^{-5}. For each time step with $\Delta t=0.04$, the work done to meet this criterion is no more than two "saw-tooth" cycles in which the ILU smoothing is applied twice in each grid level. Such a convergence rate is certainly superior to one-grid line SOR [5], and is also better than GMRES [9] with ILU preconditioning. The streamlines during one period of vortex shedding are also shown in Figs. 4(a)–4(d).

4.5. Drag, lift, moment, and Strouhal number. Figures 5(a)–5(f) present the C_D, C_L, and C_M curves plotted against time. They also show a quasi-steady or quasi-periodic feature. In those quasi-periodic cases ($\alpha\geqq 30°$), we found that these curves reach their respective local extreme (maximum for C_D, C_L, and $-C_M$) around the time when the recirculatory region developed behind the leading edge reaches its maximum and is going to separate from this tip. With higher α, these curves also reach their respective local extreme (maximum for C_D, C_L, and C_M) around the time when the recirculatory region developed behind the trailing edge reaches the same situation as stated above for the leading edge. However, the local C_L-maximum corresponding to trailing-edge vortex shedding is illegible for $R=200$, and becomes appreciable for $R=400$. In view of these results, our work is in agreement with the observation of Lugt and Haussling [5] for $\alpha=45°$ and $R=200$ (note that as compared to (2.11), Lugt's definitions of C_D, C_L, and C_M have an amplification factor of 2, 2, 4, respectively).

In Fig. 6 the Strouhal number, as defined by $S=fL\sin\alpha/U_\infty$ where f is the frequency of vortex shedding, is plotted against α. This curve is expected to be nearly constant, as demonstrated by some relevant experiments [1]. It turns out that, with $\Delta t=0.04$, the computed curve increases slightly with α, and yields a mean 0.20. To improve the accuracy we try again with $\Delta t=0.02$; this curve becomes much flatter and yields a mean 0.18. To judge the correctness, we cite some relevant known results as follows. For an ellipse having the same aspect ratio as ours, the result of Lugt and Haussling [5] with $\alpha=45°$ and $R=200$ indicates that $S=0.16-0.18$; the wind tunnel experiment of Fage and Johansen [1] indicates that $S=0.15$ for inviscid flow past a

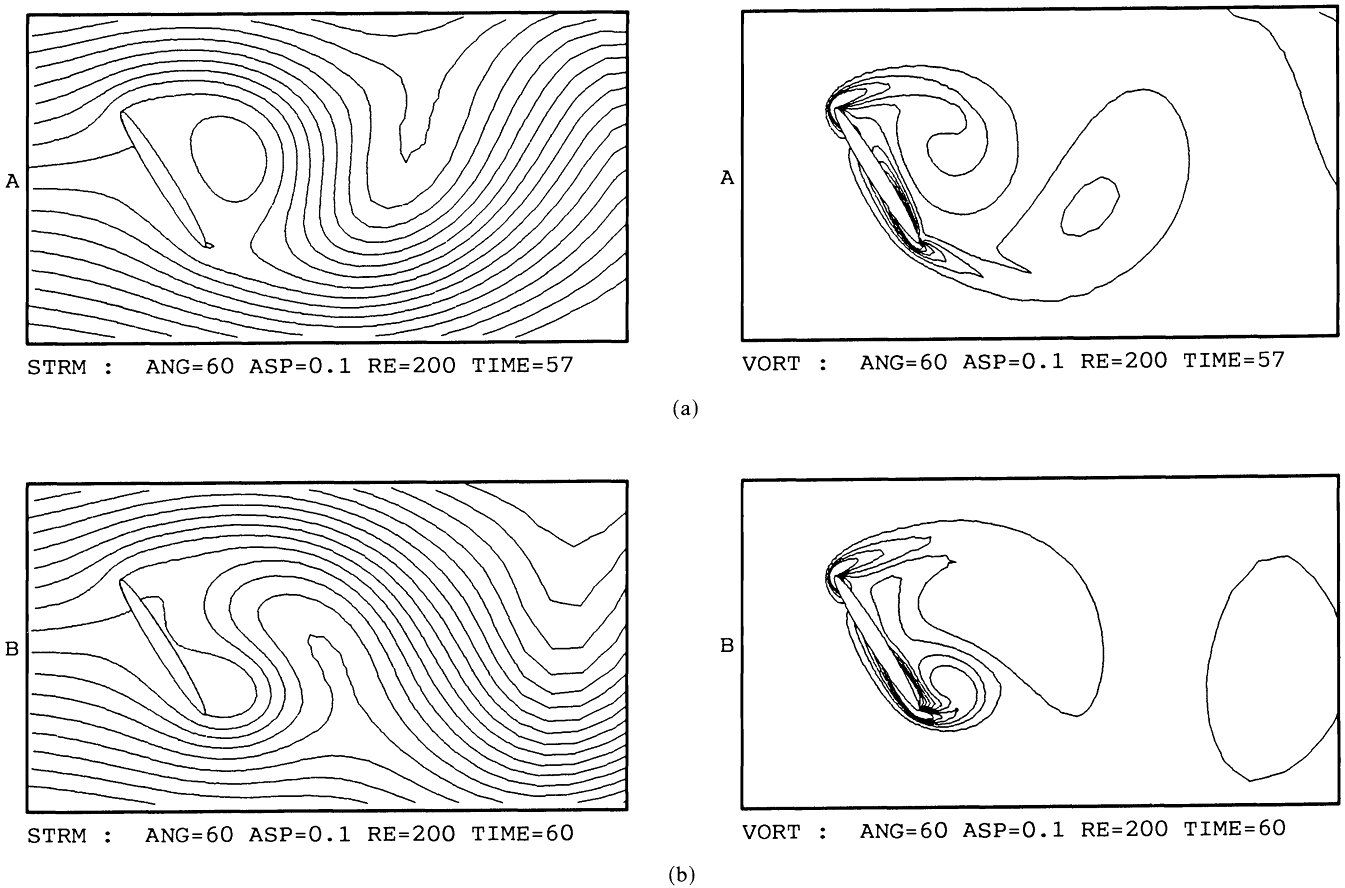


FIG. 4(a), (b). *Streamlines (left) and equivorticity lines (right) during one period of vortex shedding behind the ellipse in which* $\lambda = 0.1$, $\alpha = 60°$, *and* $R = 200$. *The corresponding dimensionless time is* $t = 57$, 60, 63, *and* 66.

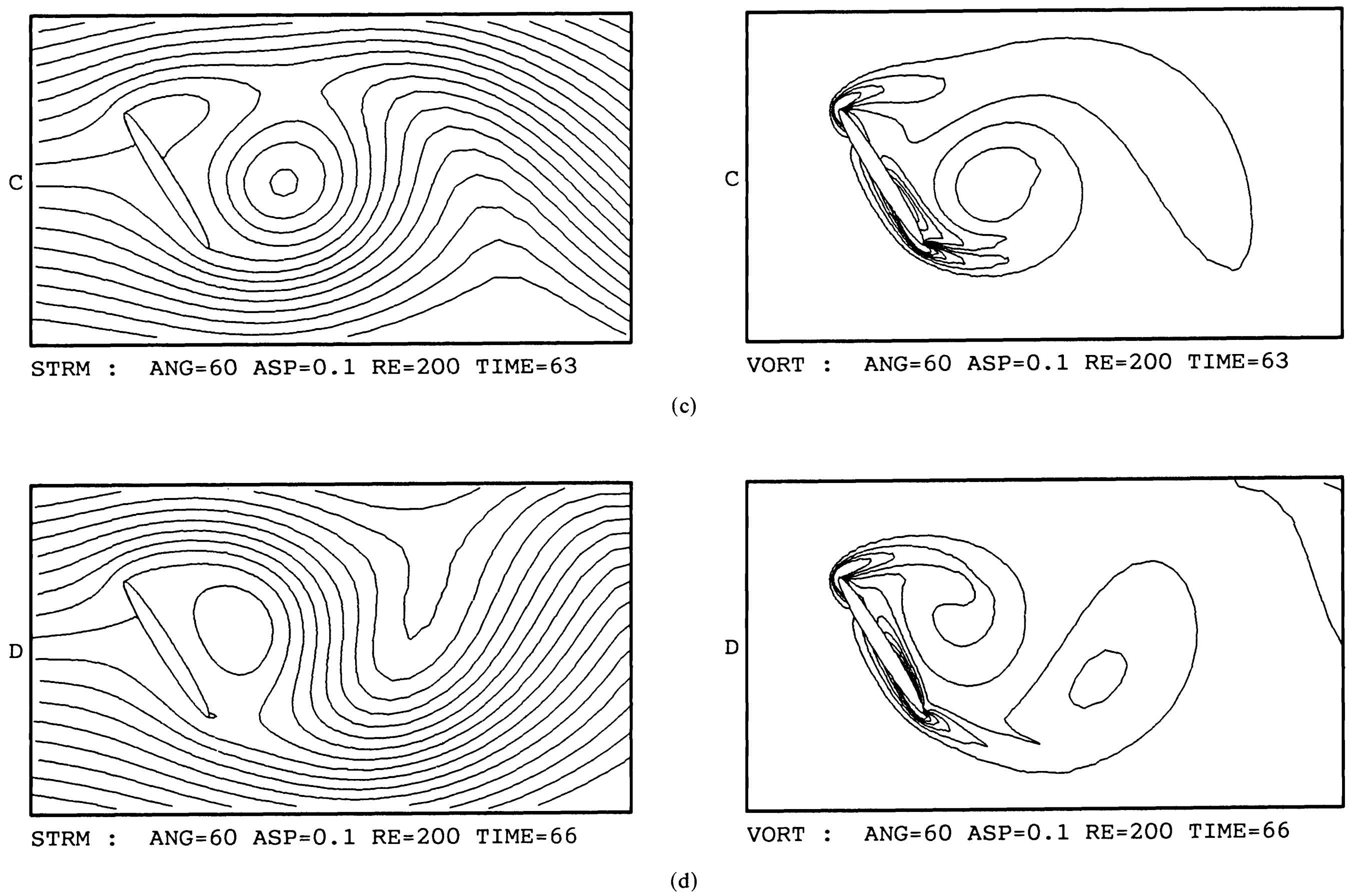
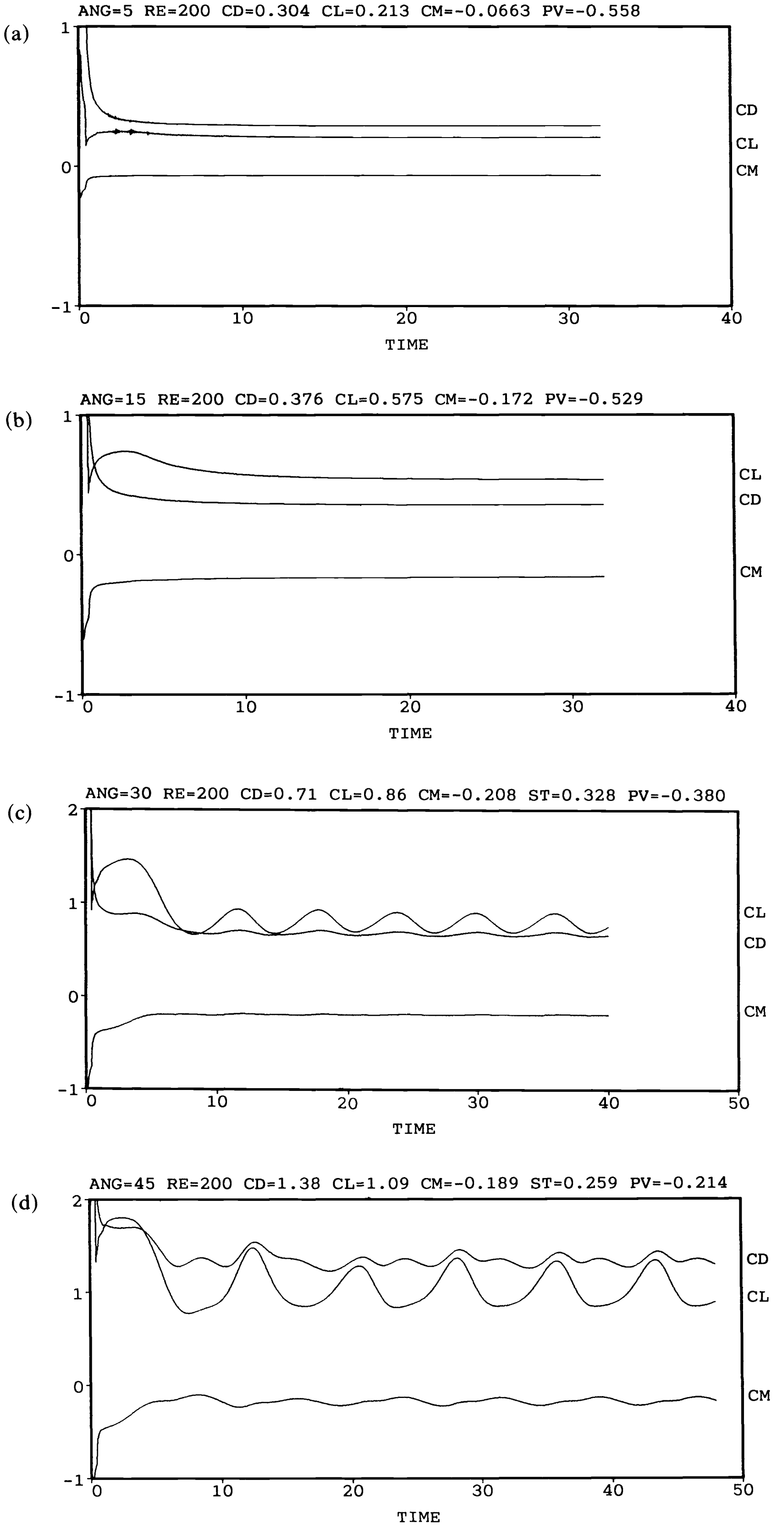
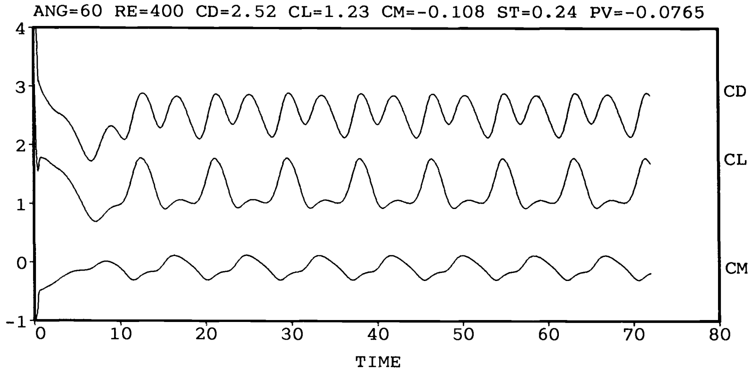


FIG. 4(c), (d). *Streamlines (left) and equivorticity lines (right) during one period of vortex shedding behind the ellipse in which* $\lambda = 0.1$, $\alpha = 60°$, *and* $R = 200$. *The corresponding dimensionless time is* $t = 57$, 60, 63, *and* 66.



(e)



(f)

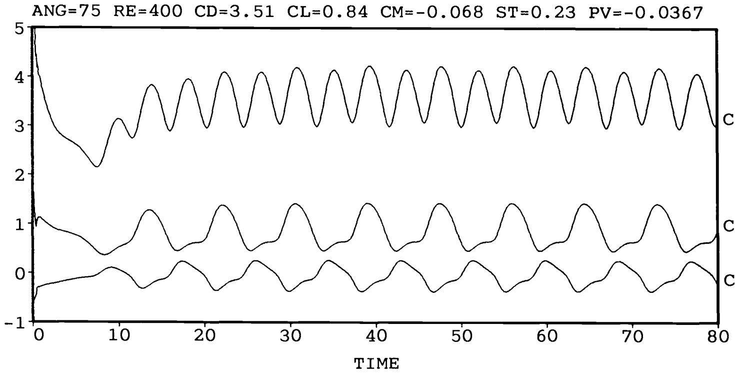


FIG. 5(e), (f). *Drag, lift, and moment coefficients vs. time*: $R = 400$, $\alpha = 60°$, $75°$.

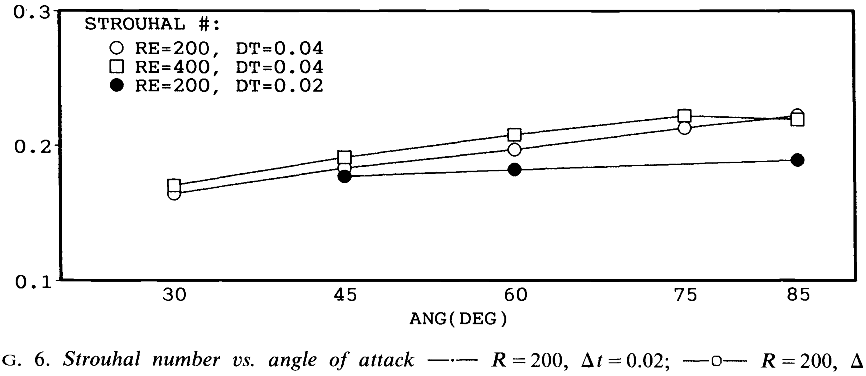


FIG. 6. *Strouhal number vs. angle of attack* —•— $R = 200$, $\Delta t = 0.02$; —○— $R = 200$, $\Delta t = 0.04$; —□— $R = 400$, $\Delta t = 0.04$.

flat plate; an experiment on a 12 percent thick symmetric Joukowski airfoil with $\alpha = 15°$, $30°$, and $R = 1000$ as made by Ghia, Osswald, and Ghia [2] indicates that $S = 0.17 - 0.18$.

4.6. Pivot point of zero torque. The time history of this point is quite similar to a C_M-curve with largely reduced amplitude. In Fig. 7, the mean position of this point scaled by $L/2$ is plotted against α along with the result from Kirchhoff theory as presented in §3. We see that these results show agreement in qualitative behavior. In spite of its simplicity, the Kirchhoff-Rayleigh model also serves as an excellent approximation for high α for finding this zero-torque pivot point.

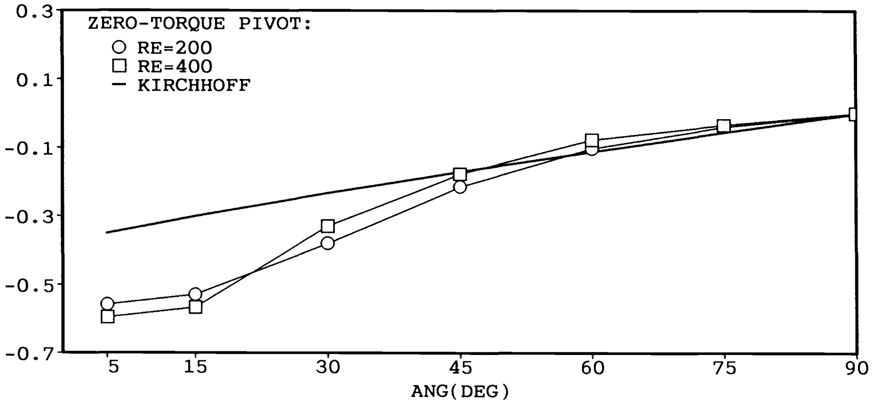


FIG. 7. *Zero-torque pivot point vs. angle of attack*: — *Kirchhoff theory*; —○— $R = 200$; —□— $R = 400$.

Acknowledgments. I would like to thank Professor Charles Peskin at Courant Institute, New York University, for introducing me to the potential application of the present work to the modeling of prosthetic heart valves. I would also like to thank Mr. H. C. Lin for his help in preparing the graphic output.

REFERENCES

[1] A. FAGE AND F. C. JOHANSEN, *On the flow of air behind an inclined flat plate of infinite span*, Proc. Roy. London Soc. Ser. A, 116(1927), pp. 170–197.

[2] K. N. GHIA, G. A. OSSWALD, AND U. GHIA, *Simulation of self-induced unsteady motion in the near wake of a Joukowski airfoil*, Lecture Notes in Engineering 24, C. A. Brebbia and S. A. Orszag, eds., Springer-Verlag, New York, 1986, pp. 118–132.

[3] S. H. LAMB, *Hydrodynamics*, Dover, New York, 1945.

[4] B. P. LEONARD, *Third-order upwinding as a rational basis for computational fluid dynamics, computational techniques and applications:* CTAC-83, J. Noye and C. Fletcher, eds., North-Holland, Amsterdam, 1984, pp. 106–120.

[5] H. L. LUGT AND H. J. HAUSSLING, *Laminar flow past an abruptly accelerated elliptic cylinder at* 45° *incidence*, J. Fluid Mech., 65 (1974), pp. 711–734.

[6] U. B. MEHTA AND Z. LAVAN, *Starting vortex, separation bubble and stall: A numerical study of laminar unsteady flow around an airfoil*, J. Fluid Mech., 67 (1975), pp. 227–256.

[7] C. S. PESKIN AND D. M. MCQUEEN, *Modeling prosthetic heart valves for numerical analysis of blood flow in the heart*, J. Comput. Phys., 37 (1980), pp. 113–132.

[8] L. QUARTAPELLE, *Vorticity conditioning in the computation of two-dimensional viscous flows*, J. Comput. Phys., 40 (1981), pp. 453–477.

[9] Y. SAAD AND M. H. SCHULTZ, GMRES: *A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.

[10] P. SONNEVELD, P. WESSELING, AND P. M. DE ZEEUW, *Multigrid and conjugate gradient methods as convergence acceleration techniques*, IMA Conference Series 3, D. J. Paddon and H. Holstein, eds., Clarendon Press, Oxford, 1985, pp. 117–167.

[11] J. J. W. VAN DER VEGT, *Calculation of forces and moments in vortex methods*, J. Engrg. Math., 22 (1988), pp. 225–238.

[12] J. C. WU, *Numerical boundary conditions for viscous flow problems*, AIAA J., 14 (1976), pp. 1042–1049.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 4, pp. 874-884, July 1992

SOLUTION OF STRUCTURED GEOMETRIC PROGRAMS IN SAMPLE SURVEY DESIGN*

FAIZ A. AL-KHAYYAL†, THOM J. HODGSON‡, GRANT D. CAPPS§¶,
JAMES A. DORSCH§[1], DAVID A. KRIEGMAN§[2], AND PAUL D. PAVNICA§

Abstract. Determination of the optimal design of a survey to measure minority voting participation can be posed as a very large number of highly structured geometric programming problems. The problem formulation is discussed, and an efficient special-purpose algorithm based on relaxation and duality is presented. The approach can also be applied to problems with similar structure.

Key words. stratified sampling, sample survey design, geometric programming, Lagrangian duality

AMS(MOS) subject classifications. 62D05, 90C28

1. Introduction.

Determination of the optimal design of a survey to measure minority voting participation can be posed as a very large number of highly structured geometric programming problems. A special heuristic algorithm was devised to exploit the structure of the thousands of geometric programs that needed to be solved. The heuristic procedure is exact for special cases. In the other cases, a methodology is developed for determining an a posteriori bound on the relative error of the solution obtained by the procedure. We discuss the model formulation, solution procedure, and its application to the survey design problem.

1.1. Survey background and requirements.

A 1978 survey of voting and registration statistics was designed by the U.S. Census Bureau to measure the difference in the voting participation rates between a given minority population and the (white) nonminority population in specified voting jurisdictions across the nation, but concentrated largely in the South. Congress, the Department of Justice, and the Census Bureau jointly identified 955 voting jurisdictions (mostly counties) to be surveyed. The minorities of interest, which varied by jurisdiction, included black, Spanish origin, American Indian, Japanese, Chinese, Filipino, Hawaiian, and Native Alaskan subgroups.

A design goal of the survey was to satisfy a given standard error of the estimated minority/nonminority voting rate differential within 718 jurisdictions. (The sampling procedure was specified a priori in the remaining 237 jurisdictions.) For each jurisdiction, all minorities that comprised three percent or more of the voting age population were considered to be minorities of interest. Frequently, there were two or more specified minorities in a jurisdiction, thereby requiring a sample design that reliably measured several voting rate differentials.

* Received by the editors February 14, 1985; accepted for publication (in revised form) May 30, 1991.

† School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332. The research of this author was supported in part by Office of Naval Research contract N00014-86-K-0173.

‡ Department of Industrial Engineering, North Carolina State University, Raleigh, North Carolina 27695. The research of this author was supported in part by Office of Naval Research contract N00014-86-K-0046.

§ Statistical Methods Division, Bureau of the Census, Washington, D.C. 20233.

¶ Present address, Fleet Analysis Center, Corona, California 91718.

[1] Present address, Defense Intelligence Agency, Washington, D.C. 20340-0001.

[2] Present address, Systems Research and Applications Corporation, Arlington, Virginia 22201.

1.2. Efficient sample survey design. The estimated total cost of conducting the survey was large (approximately \$40,000,000). To keep the costs as low as possible, the survey design was chosen as follows. First, sixteen possible sampling schemes were considered for each jurisdiction. For every jurisdiction, the "optimum" implementation of a given scheme was determined as a solution to the optimization problem of finding the combination of sampling parameters that satisfied the specified error requirements at a minimum cost. Completion of this step required the solution of more than 11,000 such optimization problems. The "best" scheme for each jurisdiction was then taken as the least expensive of the sixteen optimized schemes.

In § 2 we describe how the specification of the optimum version of each scheme in a single jurisdiction can be formulated as an optimization problem, and in § 3 we derive a specialized algorithm for such a problem.

2. Development of a typical optimization problem. In demographic sample surveys, the Census Bureau typically selects populations from three frames (groups). These three frames are mutually exclusive and cover (virtually) all housing units in a typical jurisdiction.

(1) *Permit old construction* (POC) frame consists of all housing units which existed in permit-issuing areas (i.e., areas that issue building permits) at the time of the previous national census. Computer records of these housing units contain demographic information, and hence it is straightforward to perform a stratified sample.

(2) *Permit new construction* (PNC) frame consists of all housing units constructed in permit-issuing areas since the previous census. Within this frame, clusters of eight housing units were systematically selected from a sorted list of building permits.

(3) *Nonpermit area* (NPA) frame (old and new construction) consists of all housing units not in permit-issuing areas. Three basic sample designs were considered for use in this frame. Common to all three designs is the selection of a number of land areas with known boundaries. The selection of housing units within these land areas depends on parameters of the chosen sampling scheme.

We will describe only the first of the sixteen sampling schemes, which illustrates the optimization problem in its most general form. With scheme 1, free parameters are associated only with the first and third frames, and the scheme is defined as follows.

(a) The POC frame sample is divided into n_1 strata, where n_1 is known for each jurisdiction and can be as large as eight. One parameter is associated with each stratum; let $x_j, j=1, \cdots, n_1$, represent the number of housing units to be taken from stratum j.

(b) In the PNC frame, a systematic sample of clusters of housing units is selected, where the expected number of units in each cluster is predetermined to be eight.

(c) The NPA frame sample is selected in three stages and depends on two parameters. First, x_{n_1+1} enumeration districts (EDs) are selected, where an ED contains approximately 300 housing units. These EDs are then divided into segments whose average size is 30 housing units, and x_{n_1+2} segments are chosen from each ED. Finally, a sample of eight housing units is chosen from each of these segments.

To formulate an optimization problem whose solution gives the optimum implementation of this scheme, we must first define the objective function to be minimized with respect to the parameters $\{x_j\}$, $j=1, \cdots, n_1+2$. The cost function, expressed in dollars, is taken as (see [9]):

$$\text{cost} = 17.5 \sum_{j=1}^{n_1} x_j + 87.35 x_{n_1+1} + 89.87 x_{n_1+1} x_{n_1+2} + 17.5 N, \tag{1}$$

where the PNC frame sampling size of N housing units was predetermined for each jurisdiction, and the aim is to minimize (1).

The constraints on $\{x_j\}$ come in two forms. First, the variance associated with scheme 1 for the estimated difference between the nonminority voting rate and that of the ith minority is defined as (see [4], [6]):

$$(2) \qquad V_i = \sum_{j=1}^{n_1} \frac{a_{ij}}{x_j} + \frac{a_{i,n_1+1}}{x_{n_1+1}} + \frac{a_{i,n_1+2}}{x_{n_1+1}x_{n_1+2}} - (f_{1i} - f_{2i} + f_{3i}) \quad \text{for } i = 1, \cdots, m,$$

where m is the total number of minorities. In (2), the quantities $\{a_{ij}\}$ are nonnegative constants (see [9] for details and [4] for a simple example). The constants f_{1i} and f_{3i} reflect the finite population correction factors associated with the POC and NPA frames, respectively. The constant f_{2i} is equal to the variance component associated with PNC frame sampling. The x's are the unknown sample sizes. All of the constants vary by jurisdiction and minority. The reliability requirements for the survey impose m nonlinear constraints of the following form

$$(3) \qquad V_i \leqq \sigma_i^2, \qquad i = 1, \cdots, m,$$

where σ_i is the required standard error for the ith minority.

In addition to the m nonlinear constraints (3), upper and lower bounds are placed on each variable, so that we require

$$(4) \qquad 0 < x_j \leqq u_j, \qquad j = 1, \cdots, n_1 + 2.$$

For $i \leqq j \leqq n_1$, the upper bound u_j is the total number of housing units *available* to be in the sample from stratum j of the POC frame. (Frequently, u_j is much less than the total number of housing units in stratum j, since in some jurisdictions the sample was not chosen from the full census.) The upper bound u_{n_1+1} is the total number of the NPA frame EDs, and u_{n_1+2} is an estimate of the average number of segments per NPA frame ED. It is assumed that the average segment contains approximately 30 housing units, and $30u_{n_1+2}$ is the average number of housing units per NPA frame ED.

In summary, the optimization problem associated with sampling scheme 1 includes the objective function (2), and constraints (3) and (4). (In our work, we did not require that the sample sizes be integers for two reasons. The use of integer variables would substantially increase the algorithmic complexity, and the optimal values were typically very large.) To simplify the notation, let $n = n_1 + 2$, let c_j be the coefficient of the jth term in the objective function (1), and let $b_i = \sigma_i^2 + f_{1i} - f_{2i} + f_{3i}$. The optimization problem thus has the form

$$(\text{P1}) \qquad \begin{aligned} &\text{minimize} && \sum_{j=1}^{n-1} c_j x_j + c_n x_n x_{n-1} \\ &\text{subject to} && \sum_{j=1}^{n-1} \frac{a_{ij}}{x_j} + \frac{a_{in}}{x_n x_{n-1}} \leqq b_i, \qquad i = 1, \cdots, m, \\ &&& 0 < x_j \leqq u_j, \qquad j = 1, \cdots, n. \end{aligned}$$

3. A solution approach. To complete the survey design, approximately 11,000 problems of form similar to (P1) needed to be solved. Obviously, these problems could have been solved using general-purpose nonlinear programming methods. However, a crucial special feature of this application was that in more than 90 percent of the problems, $m \leqq 2$ and it was known a priori that only a single constraint would be satisfied exactly at the solution. We now describe how a special solution procedure was developed to exploit these properties.

A constrained optimization problem of the form

$$(\text{C1}) \qquad \begin{aligned} &\text{minimize} && f(x) \\ &\text{subject to} && g_i(x) \leqq b_i, \qquad i = 1, \cdots, m, \end{aligned}$$

is called a *convex program* whenever f and each g_i are convex functions. (For a definition of convex functions and related properties, see, e.g., Rockafellar [8].) A function h of n real variables $\{x_i\}$, $i=1,\cdots,n$, is said to be a *posynomial* (i.e., a generalized polynomial with positive coefficients) if it can be written as

$$h(x)=\sum_{i=1}^{m} c_i \prod_{j=1}^{n} x_j^{a_{ij}},$$

where $c_i>0$ and a_{ij} are real constants. If f and each g_i are posynomials defined on the positive orthant, then (C1) restricted to the positive orthant becomes a *posynomial geometric program*, a class of problems that can be transformed into equivalent convex programs (see, for example, [1]).

The general approach, to be described in detail below, is to solve a sequence of (simpler) convex programs related to (P1). Consider the following two convex programs:

$$\text{(P2)}\qquad \begin{aligned} &\text{minimize} && \sum_{j=1}^{n} c_j x_j \\ &\text{subject to} && \sum_{j=1}^{n} \frac{a_{ij}}{x_j} \leqq b_i, \qquad i=1,\cdots,m, \\ & && x_j>0, \qquad j=1,\cdots,n, \end{aligned}$$

and

$$\text{(P3)}\qquad \begin{aligned} &\text{minimize} && \sum_{j=1}^{n} c_j x_j \\ &\text{subject to} && \sum_{j=1}^{n} \frac{a_{ij}}{x_j} \leqq b_i, \qquad i=1,\cdots,m, \\ & && 0<x_j\leqq u_j, \qquad j=1,\cdots,n. \end{aligned}$$

Note that (P3) is (P1) with the $x_n x_{n-1}$ term in the objective replaced by x_n, and that (P2) is (P3) without upper bounds on the variables.

The partial Lagrangian function (see, e.g., Bazaraa and Shetty [2, Chap. 6]) for (P2) is given by

$$\text{(5)}\qquad L(x,\lambda)=\sum_{j=1}^{n} c_j x_j+\sum_{i=1}^{m}\lambda_i\left[\sum_{j=1}^{n}\frac{a_{ij}}{x_j}-b_i\right],$$

where x is the n-vector of decision variables and λ is the m-vector of Lagrange multipliers for the general constraints. For fixed $\lambda\geqq 0$ ($\lambda\neq 0$), the function L is strictly convex in x for $x>0$. Its unique feasible minimizer (with respect to x) is given by

$$\text{(6)}\qquad x_j^*(\lambda)=\sqrt{\sum_{i=1}^{m}\frac{\lambda_i a_{ij}}{c_j}}, \qquad j=1,\cdots,n.$$

Substituting (6) into (5) and simplifying yields

$$\text{(7)}\qquad h(\lambda)=L(x^*(\lambda),\lambda)=2\sum_{j=1}^{n}\sqrt{c_j\sum_{i=1}^{m} a_{ij}\lambda_i}-\sum_{i=1}^{m}\lambda_i b_i.$$

It is not difficult to show that h is a concave function and its maximum for $\lambda\geqq 0$ coincides with the minimum value of the objective function in (P2). We thus have the dual of problem (P2)

$$\text{(D2)}\qquad \begin{aligned} &\text{maximize} && h(\lambda) \\ &\text{subject to} && \lambda\geqq 0, \end{aligned}$$

where λ is now called the dual vector. Problem (D2) is also a convex program, so let λ^* denote the unique optimal solution. It follows that the solution of (P2) is defined by $x_j^* = x_j^*(\lambda^*)$.

Problem (D2) can be solved using general methods for bound-constrained optimization (e.g., McCormick [7, Chap. 13]). However, if (P2) has only one constraint, then (D2) has only one variable and its solution is given in closed form as

$$\lambda_1^* = \left(\frac{1}{b_1}\sum_{j=1}^{n}\sqrt{c_j a_{1j}}\right)^2. \tag{8}$$

Thus, the solution to (P2) when $m=1$ is given by

$$x_j^* = x_j^*(\lambda_1^*) = \frac{1}{b_1}\sqrt{\frac{a_{1j}}{c_j}}\sum_{i=1}^{n}\sqrt{c_i a_{1i}}, \tag{9}$$

which corresponds to the well-known Neyman allocation [5]. In the next section, we consider how to use this result to solve (P3) when $m=1$.

3.1. The case $m=1$. When $m=1$, the single general constraint is always active at the optimal solution of (P3), assuming a nonempty feasible set. Let x^* and $\hat{x}$ denote the unique optimal solutions of (P2) and (P3), respectively. Standard optimality conditions then indicate that, for *each* component j for which $x_j^* \geqq u_j$, it holds that $\hat{x}_j = u_j$ (see, e.g., Rockafellar [8, § 28] or Bazaraa and Shetty [2, Chap. 4]). This observation is much stronger than the result for general convex programs (see, e.g., [2, Exercise 4.15]) which only guarantees the existence of a solution, say $\bar{x}$, to a convex program such that *at least* one constraint violated by the optimal solution of a relaxed problem (obtained by dropping that constraint along with, possibly, others) is active at $\bar{x}$. Moreover, in our situation, knowledge of an active (upper bound) constraint at the optimal solution automatically reduces the dimensionality of the problem. These observations suggest the following simple solution procedure for (P3) (assumed feasible), which is based on repeated solution of subproblems obtained from relaxations (P2) by fixing different sets of variables to the bound u.

ALGORITHM A.

Step 0. Set $k=1$, $\bar{b}_k = b_1$, and $J_k = \{1, 2, \cdots, n\}$. Problem (P2) with $m=1$ can be written as

$$\text{(P2M1(k))}\qquad \begin{aligned} &\text{minimize} && \sum_{j\in J_k} c_j x_j \\ &\text{subject to} && \sum_{j\in J_k}\frac{a_{1j}}{x_j} \leqq \bar{b}_k, \\ &&& x_j > 0, \qquad j \in J_k. \end{aligned}$$

Step 1. Let x^k solve problem P2M1(k). The solution is given by (7) as

$$x_j^k = \frac{1}{\bar{b}_k}\sqrt{\frac{a_{1j}}{c_j}}\sum_{i\in J_k}\sqrt{c_i a_{1i}} \quad \text{for } j \in J_k.$$

Step 2. If $x_j^k \leqq u_j$ for all $j \in J_k$, set $x_j^* = x_j^k$ for all $j \in J_k$; STOP, $x^* = \hat{x}$ solves (P3).

Step 3. If $x_j^k > u_j$ for some $j \in J_k$, set $x_j^* = u_j$ for all such j. Define

(a) $F_k = \{j: x_j^k > u_j, j \in J_k\}$

(b) $J_{k+1} = J_k \backslash F_k$

(c) $\bar{b}_{k+1} = \bar{b}_k - \sum_{j\in F_k}\frac{a_{1j}}{u_j}$.

Set $k = k+1$ and return to Step 1.

Remark. The algorithm clearly terminates since, at the unique solution $\hat{x}$ to (P3), we know that $\sum_j a_{1j}/\hat{x}_j = b_1$ and that reduction of problem dimensionality at each iteration does not change the implication on active upper bounds inferred by solving the relaxed problem (P2). The algorithm assumes the feasible set for (P3) to be nonempty. Infeasible problems can be detected by simply verifying $\sum_j a_{1j}/u_j > b_1$.

3.2. The case $m \geqq 2$. For $m \geqq 2$, it is no longer true, in general, that if $x_j^* > u_j$ in the optimal solution of (P2) then it follows that $\hat{x}_j = u_j$ in the optimal solution of (P3). However, the (heuristic) rule of fixing the jth variable to its upper bound whenever $x_j^* > u_j$, and repeatedly solving lower-dimensional relaxations (P2), leads to a feasible solution to (P3) whose closeness to $\hat{x}$ can be measured.

We next present a formal statement of this procedure for finding the solution x^* to (P3) obtained using the foregoing rule. In the next section, we present a method for computing a relative error bound on using x^* as an approximation to the true optimum solution $\hat{x}$ of (P3).

ALGORITHM B.

Step 0. Set $k=1$, $b_i^k = b_i$, and $J_k = \{1, 2, \cdots, n\}$. Problem (P2) can be written as

$$\text{(P2(k))}\qquad \begin{aligned} &\text{minimize} && \sum_{j \in J_k} c_j x_j \\ &\text{subject to} && \sum_{j \in J_k} \frac{a_{ij}}{x_j} \leqq b_i^k, \quad i = 1, \cdots, m, \\ &&& x_j > 0, \quad j \in J_k \end{aligned}$$

and its dual becomes

$$\text{(D2(k))}\qquad \begin{aligned} &\text{maximize} && 2 \sum_{j \in J_k} \sqrt{c_j \sum_{i=1}^m a_{ij} \lambda_i} - \sum_{i=1}^m \lambda_i b_i \\ &\text{subject to} && \lambda_i \geqq 0, \qquad i = 1, \cdots, m. \end{aligned}$$

Step 1. Solve (D2(k)) by any terminating procedure and let λ^k denote the solution. Then the solution to (P2(k)) is given by (4) as

$$x_j^k = \sqrt{\sum_{i=1}^m \frac{\lambda_i^k a_{ij}}{c_j}} \quad \text{for } j \in J_k.$$

Step 2. If $x_j^k \leqq u_j$ for all $j \in J_k$, set $x_j^* = x_j^k$ for all $j \in J_k$; STOP, x^* is an approximate solution for (P3).

Step 3. If $x_j^k > u_j$ for some $j \in J_k$, set $x_j^* = u_j$ for all such j. Define

(a) $F_k = \{j : x_j^k > u_j, j \in J_k\}$

(b) $J_{k+1} = J_k \backslash F_k$

(c) $b_i^{k+1} = b_i^k - \sum_{j \in F_k} \dfrac{a_{ij}}{u_j} \qquad i = 1, \cdots, m.$

Set $k = k+1$ and return to Step 1.

Remark. Here again, we assume that the feasible set of (P3) is nonempty; otherwise, $\sum_j a_{ij}/u_j > b_i$ for at least one i. Convergence of the algorithm depends on the procedure selected in Step 1 for solving (D2(k)). A general variable reduction algorithm (see, e.g., [7, Chap. 13]) such as the reduced gradient method can be used, or a specialized technique such as the projected Newton method using only the diagonal elements of the Hessian matrix and an Armijo-like inexact line search presented in [3]. Moreover, it is easy to show that if one and only one component of an optimal solution to (P2) exceeds its upper bound, then that variable is at its bound in the optimal solution of (P3). Hence, if the optimal solution x^k of problem (P2(k)) has at most one component exceeding its upper bound for *every* k, then the solution obtained by the preceding algorithm is optimal for (P3).

3.3. Bound on relative error. By expending a little more effort, the feasible point x^* can be checked to see if it is in fact optimal for (P3). Otherwise, an error bound can be calculated by just finding a point $\tilde{x}$ that satisfies

$$f(\tilde{x}) < f(\hat{x}) < f(x^*), \tag{10}$$

where the second inequality follows from feasibility of x^*, and then using $[f(x^*) - f(\tilde{x})]/f(\tilde{x})$ as an upper bound on the relative error $[f(x^*) - f(\hat{x})]/f(\hat{x})$ of using x^* to approximate $\hat{x}$. In this way, an a posteriori error bound on the optimal objective function value is available for purposes of assessing the goodness of the heuristic employed.

Suppose Algorithm B terminates in iteration k. Then x^* solves the problem

$$\begin{aligned} \text{minimize} \quad & \sum_{j \in N} c_j x_j \\ \text{subject to} \quad & \sum_{j \in N} \frac{a_{ij}}{x_j} \leqq b_i, \qquad i \in M, \\ & x_j = u_j, \qquad j \in F(k), \\ & x_j > 0, \qquad j \in N, \end{aligned} \tag{11}$$

where $M = \{1, 2, \cdots, m\}$, $N = \{1, 2, \cdots, n\}$, and $F(k) = \bigcup_{i=1}^{k} F_i$ is the index set of variables fixed to their upper bounds. Let v^* and w^* be the multiplier vectors such that (x^*, v^*, w^*) solves the following Karush-Kuhn-Tucker conditions (see, e.g., Bazaraa and Shetty [2, Chap. 4]):

$$\begin{aligned} & c_j = \sum_{i \in M} v_i a_{ij} / x_j^2 - w_j, \qquad j \in F(k), \\ & c_j = \sum_{i \in M} v_i a_{ij} / x_j^2, \qquad j \in N \backslash F(k), \\ & v_i \left[\sum_{j \in N} a_{ij} / x_j - b_i \right] = 0, \qquad i \in M, \\ & v_i \geqq 0, \qquad i \in M. \end{aligned} \tag{12}$$

Here w^* is the vector of multipliers for the active upper bounds. If $w_j^* \geqq 0$ for all $j \in F(k)$, then $x^* = \hat{x}$ is the unique solution of (P3). Otherwise, a point x satisfying (10) can be computed in the following way

The general idea for determining an appropriate error bound is to take $\tilde{x}$ as the optimal solution determined by an application of Algorithm A to the problem of minimizing $\sum_j c_j x_j$ subject to a single functional constraint, which is either chosen from the index set M or constructed as a convex combination of two constraints from M.

Assume $w_j^* < 0$ for at least one $j \in F(k)$. From (12) we have

$$x_j^* = \sqrt{\sum_{i \in M} v_i^* a_{ij} / (c_j + w_j^*)} \quad \text{for all } j \in F(k).$$

It follows that

$$\tilde{x}_j = \begin{cases} \sqrt{\sum_{i \in M} v_i^* a_{ij} / c_j} < x_j^* & \text{for } j \in F(k) \quad \text{such that } w_j^* < 0, \\ x_j^* & \text{otherwise} \end{cases} \tag{13}$$

is infeasible to (P3). We shall use the vector $\tilde{x}$ given by (13) to identify the single constraint to be passed to Algorithm A.

Two cases are distinguished depending on whether or not $\bar{x}$ violates all constraints in M.

Case 1. $\bar{x}$ satisfies at least one constraint.

Let $i \in M$ be the index of a satisfied constraint and $\nu \in M$ be the index of a violated constraint. Then, because of convexity, the convex combination of these two constraints that passes through $\bar{x}$ contains the feasible region; specifically,

$$\sum_{j \in N} \frac{\alpha a_{ij} + (1-\alpha) a_{\nu j}}{x_j} \leqq \alpha b_i + (1-\alpha) b_\nu, \tag{14}$$

where

$$\alpha = \frac{\varphi_\nu(\bar{x}) - b_\nu}{b_i - \varphi_i(\bar{x}) + \varphi_\nu(\bar{x}) - b_\nu}$$

and

$$\varphi_\nu(x) = \sum_{j \in N} \frac{a_{\nu j}}{x_j}.$$

In problem (P2M1(1)) of Algorithm A, take $a_{1j} = \alpha a_{ij} + (1-\alpha) a_{\nu j}$ and $\bar{b}_1 = \alpha b_i + (1-\alpha) b_\nu$ and let $\tilde{x}$ denote the solution obtained for this problem.

Case 2. $\bar{x}$ violates all constraints.

A simple procedure in this case is to choose, for Algorithm A, that violated constraint which is closest to $\bar{x}$ in the direction c. This constraint can be determined as the binding constraint of the easily solvable problem

$$\begin{aligned} &\text{maximize} && \alpha \\ &\text{subject to} && g_i(\alpha) \geqq b_i, \qquad i \in M, \\ & && \alpha \geqq 0, \end{aligned}$$

where

$$g_i(\alpha) = \sum_{j \in N} \frac{a_{ij}}{\bar{x}_j + \alpha c_j}.$$

For both cases above, more elaborate procedures are not difficult to devise, but our aim here is to keep the treatment simple, since a comparative experiment (or theoretical analysis) of different heuristics is not the focus of this study.

An example that illustrates Case 1 is presented next.

3.4. Example. The following example illustrates Case 1.

$$\begin{aligned} &\text{minimize} && 100x_1 + 5x_2 + x_3 \\ &\text{subject to} && \frac{1}{x_1} + \frac{5}{x_2} + \frac{1}{x_3} \leqq 1.6, \\ & && \frac{125}{x_1} + \frac{1}{x_2} + \frac{1200}{x_3} \leqq 252.75, \\ & && x_1, x_2, x_3 > 0, \\ & && x_1 \leqq 15,\ x_2 \leqq 4.5,\ x_3 \leqq 5. \end{aligned} \tag{15}$$

Algorithm B returns the point $x^* = (9.98, 4.5, 5)$ with $f(x^*) = 1025.5$, $k = 2$, and $F(2) = \{2, 3\}$. The triple (x^*, v^*, w^*) with $(v_1^*, v_2^*) = (0, 79.68)$ and $(w_2^*, w_3^*) = (-1.07, 3823.66)$

solves (12), and $\bar{x} = (9.98, 3.99, 5)$ is now available from (13); checking feasibility of $\bar{x}$ fixes $i = 1$ and $v = 2$.

Replace the two functional constraints in (15) by the convex combination determined by (14); namely,

$$\frac{77.88}{x_1} + \frac{2.52}{x_2} + \frac{744.38}{x_3} \leqq 157.31. \tag{16}$$

Algorithm A gives $\tilde{x} = (9.90, 4.5, 5)$, with $f(\tilde{x}) = 1017.08$, as the optimal solution for the problem with (16) as its single constraint, and all other data being the same as (15).

Since $[f(x^*) - f(\tilde{x})]/f(\tilde{x}) = 0.0083$, then $f(x^*)$ is within 0.83 percent of the true optimum value $f(\hat{x})$. Here, we know that the optimal solution is $\hat{x} = (10, 4, 5)$ with $f(\hat{x}) = 1025$, so that our heuristic solution is actually within 0.05 percent of optimality.

4. Solving the survey design model. In this section we describe a solution procedure for the survey design model (P1), which we stated earlier to be related to the simpler convex program (P3). We now make this relationship clear. It is easy to see that (P1) is equivalent to

$$\text{(P4)} \quad \begin{aligned} &\text{minimize} && \sum_{j=1}^{n} c_j y_j \\ &\text{subject to} && \sum_{j=1}^{n} \frac{a_{ij}}{y_j} \leqq b_i, \qquad i = 1, \cdots, m, \\ &&& y_n - u_n y_{n-1} \leqq 0, \\ &&& 0 < y_j \leqq \bar{u}_j, \qquad j = 1, \cdots, n, \end{aligned}$$

where $\bar{u}_j = u_j$ for all $1 \leqq j \leqq n-1$ and $\bar{u}_n = u_n u_{n-1}$. The equivalence follows immediately from the one-to-one correspondence between the feasible points under the transformation defined by $y_j = x_j$ $(1 \leqq j \leqq n-1)$ and $y_n = x_n x_{n-1}$. Note that (P3) is a relaxation of the above problem. For convenience, (P3) is rewritten here as

$$\text{(P5)} \quad \begin{aligned} &\text{minimize} && \sum_{j=1}^{n} c_j y_j \\ &\text{subject to} && \sum_{j=1}^{n} \frac{a_{ij}}{y_j} \leqq b_i, \qquad i = 1, \cdots, m, \\ &&& 0 < y_j \leqq \bar{u}_j, \qquad j = 1, \cdots, n, \end{aligned}$$

If $\hat{y}$ denotes the optimal solution of (P5) and $\tilde{y}$ is the optimal solution of (P4), it is straightforward to show that if $\hat{y}_n > u_n \hat{y}_{n-1}$, then $\tilde{y}_n = u_n \tilde{y}_{n-1}$. Hence, with $y_n = u_n y_{n-1}$, (P4) reduces to

$$\text{(P6)} \quad \begin{aligned} &\text{minimize} && \sum_{j=1}^{n-1} c'_j y_j \\ &\text{subject to} && \sum_{j=1}^{n-1} \frac{a'_{ij}}{y_j} \leqq b_i, \qquad i = 1, \cdots, m, \\ &&& 0 < y_j \leqq \bar{u}_j, \qquad j = 1, \cdots, n-1, \end{aligned}$$

where $c'_j = c_j$ and $a'_{ij} = a_{ij}$ for $1 \leqq j \leqq n-2$ and all i, and $c'_{n-1} = c_{n-1} + c_n u_n$ and $a'_{i,n-1} = a_{i,n-1} + a_{in}/u_n$. Since (P6) has the same form as (P3), then (P1) can be solved by solving at most two problems of the form (P3). The overall algorithm for (P1) is presented next, followed by a remark on bounding the relative error of the heuristic solution obtained.

ALGORITHM C.

Step 1. Solve (P5) and let $\hat{y}$ denote a solution. One of three alternative procedures may be used.

(1) If $m=1$, use Algorithm A, whereupon $\hat{y}$ is the solution to the problem.

(2) If $m \geqq 2$, apply Algorithm A for each of the m constraints and let $y^1, \cdots, y^m$ denote the solutions obtained. For any j, if y^j satisfies all m constraints then $\hat{y}=y^j$ solves (P5). If all m solutions are infeasible to (P5), then use Algorithm B, whereupon $\hat{y}$ is an approximate solution to the problem.

Step 2. If $\hat{y}_n \leqq u_n \hat{y}_{n-1}$, then

$$\hat{y}^* = (\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_{n-1}, \hat{y}_n/\hat{y}_{n-1})$$

is an exact or approximate solution to (P1), depending on whether $\hat{y}$ is an exact or approximate solution to (P5). Otherwise, go to Step 3.

Step 3. Solve (P6) using one of the alternative procedures in Step 1. Let $(y_1^*, y_2^*, \cdots, y_{n-1}^*)$ denote a solution. Then

$$y^* = (y_1^*, y_2^*, \cdots, y_{n-1}^*, u_n y_{n-1}^*)$$

is an exact or approximate solution to (P1), depending on the method used to solve (P6).

Remark. The methodology for bounding the relative error of the heuristic solution (§ 3.3) can be applied to either (P5) or (P6), depending on whether the algorithm terminates in Step 2 or Step 3, respectively.

5. Concluding remarks. In the sample survey design application, sixteen sampling schemes were considered for each of the 718 jurisdictions. The optimum implementation of each scheme required the solution of an optimization problem to determine the least cost stratified sample sizes that met voting participation variance limits. Each problem was either of the form (P1) if both NPA and POC frames were used in the survey design, or of the form (P3) if only POC frames were used. For the 11,488 problems considered, the number of stratified sample sizes n did not exceed nine, and the number of minorities m was four or less. Moreover, experimentation with different stratifications and different techniques for estimating variance resulted in the need to solve these problems several times. Approximately 90 percent of the jurisdictions had only one or two minorities, and many of those had only POC frames. Thus, a substantial proportion of the problems that needed to be solved were of the form (P3) with $m \leqq 2$, for which the approach presented above is ideally suited. Available general optimization procedures could not satisfactorily process the large number of small problems in this application because they did not exploit the highly specialized nature of the survey design optimization problems.

One of the sixteen sampling schemes is a minor variation of the standard scheme that the Census Bureau usually uses for this type of survey. In each jurisdiction, the optimum scheme was compared to the optimized standard scheme (SC). For this purpose, 650 of the 718 jurisdictions were optimized. Schemes for the remaining jurisdictions were preselected. In 155 jurisdictions, the feasible region of SC was empty. That is, SC does not satisfy the variance constraints unless either the nonpermit area (NPA) frame cluster size or the permit new construction (PNC) frame sample size is increased.

For the 495 jurisdictions where SC was feasible, the cost of using SC in all jurisdictions was estimated to be \$15,400,000. In 118 jurisdictions, an optimum scheme different from SC was selected at an estimated savings of about \$400,000. In the 155 jurisdictions where SC was infeasible, the savings from selecting an optimum scheme was estimated to be about \$140,000. Therefore, a conservative estimate of the amount saved on the cost of conducting surveys in the 650 jurisdictions was \$540,000.

A jurisdiction is said to be undercovered if not all housing units have a chance to be surveyed, and in some jurisdictions such an occurrence was unacceptable. By listing the costs of the 16 different schemes, the optimization procedure allowed the analysts to weigh coverage against costs in a precise manner. In some jurisdictions, suboptimal survey designs were chosen to prevent either known or suspected undercoverage. These subjective choices resulted in an estimated \$260,000 loss of savings.

Without optimization, alternatives to the standard Census Bureau scheme could not have been considered while still conducting the survey on schedule. Unfortunately, for unrelated reasons the survey project was canceled at the last moment. However, the model and methodology have been successfully applied in subsequent studies by the bureau (see, e.g., Causey [4]).

Acknowledgments. The authors are grateful to the referees and to the Reviewing Editor, Margaret H. Wright, who provided many constructive comments.

REFERENCES

[1] M. AVRIEL, ED., *Advances in Geometric Programming*, Plenum Press, New York, 1980.

[2] M. S. BAZARAA AND C. M. SHETTY, *Nonlinear Programming: Theory and Algorithms*, John Wiley, New York, 1979.

[3] D. P. BERTSEKAS, *Projected Newton methods for optimization problems with simple constraints*, SIAM J. Control Optim., 20 (1982), pp. 221-246.

[4] B. D. CAUSEY, *Computational aspects of optimal allocation in multivariate stratified sampling*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 322-329.

[5] W. G. COCHRAN, *Sampling Techniques*, Third Edition, John Wiley, New York, 1977.

[6] M. H. HANSEN, W. N. HURWITZ, AND W. G. MADOW, *Sample Survey Methods and Theory*, John Wiley, New York, 1963.

[7] G. P. MCCORMICK, *Nonlinear Programming: Theory, Algorithms, and Applications*, John Wiley, New York, 1983.

[8] R. T. ROCKAFELLAR, *Convex Analysis*, Princeton University Press, Princeton, NJ, 1970.

[9] *Sampling Specifications for the* 1978 *Survey of Voting and Registration Statistics*, Memorandum 2, Bureau of the Census, Washington, D.C., March 20, 1978.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 4, pp. 885–903, July 1992

BLOCK M-MATRICES AND COMPUTATION OF INVARIANT TORI*

LUCA DIECI[†] AND JENS LORENZ[‡]

Abstract. In this work a generalization of nonsingular M-matrices to block matrices is proposed, where positivity of numbers is replaced by positive definiteness of blocks. In addition, the outer diagonal blocks are multiples of the identity. This generalization is arrived at by studying the matrices that arise from discretizing linear first-order systems of partial differential equations (PDEs) where each equation has the same principal part. These PDEs occur in the study of invariant tori of dynamical systems. In this paper, a first-order discretization of these PDEs is investigated and block M-matrix properties are used to establish stability and error estimates. To obtain higher-order convergence, an error expansion is proved, which legitimates Richardson's extrapolation. Some of the numerical and algorithmic aspects of the proposed discretization are discussed and briefly contrasted to others. Some numerical examples to illustrate the theory and to highlight the interplay between attractivity and smoothness of the tori versus accuracy of the approximation are also presented.

Key words. invariant tori, numerical computation, PDEs with same principal part, upwinding, extrapolation

AMS(MOS) subject classifications. 65L99, 65N07, 34C99

1. Introduction.

In [DLR], we considered the problem of computing invariant tori of dynamical systems numerically. Our approach required the solution of an associated system of partial differential equations (PDEs). We used the leap-frog discretization for this system, and proved stability and second-order convergence for linear constant coefficient problems. The extension to linear variable coefficients has been the main motivation for this paper. Here, we provide such an extension for an upwind scheme; under suitable assumptions, we prove stability and first-order convergence. To prove stability, we are led to define a class of block matrices that generalizes the usual class of nonsingular M-matrices. This generalization is necessary, because the PDE is not scalar. We prove that the *block M-matrices* are invertible, and we give explicit bounds on the norm of their inverse. This allows us to show stability.

An outline of the paper is as follows. In the remainder of the introduction we set some notation. In §2 we define block M-matrices and, extending a basic property of usual M-matrices, we obtain a bound on their inverse. In §3, we consider the application of block M-matrices to the computation of invariant tori. Under certain assumptions, an invariant torus can be parametrized by the solution of a system of first-order PDEs, where each equation of the system has the same principal part. For linear systems of PDEs of this type, we investigate a first-order upwind scheme. The discretization leads to a block M-matrix if the coefficient matrix of the zero-order term in the PDE system is positive definite. (Positive definiteness of this zero-order coefficient implies attractivity of the torus.) It is then possible to prove stability and

*Received by the editors August 13, 1990; accepted for publication (in revised form) April 25, 1991. This work was supported in part by National Science Foundation grant DMS-8802762 and Department of Energy grant DE-AS03-76ER72012.

[†]School of Mathematics, Georgia Institute of Technology, Atlanta, Georgia 30332 (dieci@math.gatech.edu).

[‡]Department of Mathematics and Statistics, University of New Mexico, Albuquerque, New Mexico 87131 (lorenz@altona.unm.edu).

first-order convergence. Under smoothness assumptions, we show that the discretization error admits an expansion in terms of powers of the mesh sizes. This result can be used in conjunction with Richardson's extrapolation to obtain higher-order procedures. In §4, we present some of the implementation considerations, and two examples, to illustrate the theory and to stress the relation between smoothness and attractivity of the tori and accuracy of the computed approximations.

We consider block square matrices of the form

$$A = \begin{pmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & & \vdots \\ A_{n1} & \cdots & A_{nn} \end{pmatrix}, \tag{1.1}$$

where each $A_{ij} \in \mathbb{R}^{m \times m}$. For matrices $B, C \in \mathbb{R}^{m \times m}$, possibly unsymmetric, we write

$$B \geq_q C$$

if and only if the quadratic form defined by $B - C$ is positive semidefinite, i.e.,

$$\mathbf{x}^T B \mathbf{x} \geq \mathbf{x}^T C \mathbf{x} \quad \forall \mathbf{x} \in \mathbb{R}^m.$$

For vectors $\mathbf{a},\mathbf{b} \in \mathbb{R}^n$, we write $\mathbf{a} \geq \mathbf{b}$ if and only if $a_i \geq b_i$, for all i. Also, for $\mathbf{e} \in \mathbb{R}^n$, we write $\mathbf{e} > \mathbf{0}$ if and only if all $e_i > 0$. A vector $\mathbf{e} > \mathbf{0}$ induces the scaled maximum norm for vectors in $\mathbb{R}^n$ ($|x_i|$ denotes absolute value):

$$\|\mathbf{x}\|_e := \max_{1 \leq i \leq n} \frac{|x_i|}{e_i}, \qquad \mathbf{x} \in \mathbb{R}^n.$$

For vectors $\mathbf{c},\mathbf{d} \in \mathbb{R}^m$ we denote the usual Euclidean scalar product and induced norm as

$$\langle \mathbf{c},\mathbf{d} \rangle := \sum_{i=1}^{m} c_i d_i, \qquad |\mathbf{c}| := \langle \mathbf{c},\mathbf{c} \rangle^{1/2}.$$

For vectors

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}^1 \\ \vdots \\ \mathbf{x}^n \end{pmatrix} \in \mathbb{R}^{nm}, \quad \mathbf{x}^j \in \mathbb{R}^m, \quad \text{and} \quad \mathbf{e} \in \mathbb{R}^n, \quad \mathbf{e} > \mathbf{0},$$

we define the scaled maximum two-norm:

$$\|\mathbf{x}\|_e := \max_{1 \leq j \leq n} \frac{|\mathbf{x}^j|}{e_j},$$

and if $e_j = 1$, for all j, we denote this norm simply as $\|\mathbf{x}\|$. Induced matrix norms are defined in the usual way.

2. Block M-matrices.

Nonsingular M-matrices can be characterized in many ways. In [BP] and [P], 40 equivalent conditions are listed. Here, we will generalize the following *semipositivity* characterization (it is K33 or K35 of [P]).

DEFINITION 2.1. A matrix $B = (b_{ij}) \in \mathbb{R}^{n \times n}$ is an M-matrix if

$$b_{ij} \leq 0, \qquad i \neq j \tag{2.1a}$$

and if there exists a vector $\mathbf{e} \in \mathbb{R}^n$ such that

$$\text{(2.1b)} \qquad \mathbf{e} > 0 \quad \text{and} \quad B\mathbf{e} > \mathbf{0}.$$

Several properties/characterizations of M-matrices are well known. For example, if B is an M-matrix, then B^{-1} exists (and $B^{-1} \geq 0$). We will extend this invertibility result in the stronger form given by the following lemma, of which we provide a simple proof for completeness.

LEMMA 2.2. *Let $B \in \mathbb{R}^{n \times n}$ be an M-matrix, and let $\mathbf{e} \in \mathbb{R}^n$, $\mathbf{e} > \mathbf{0}$, and $B\mathbf{e} > \mathbf{0}$. Let $\eta \in \mathbb{R}$, $\eta > 0$, such that $B\mathbf{e} \geq \eta\mathbf{e}$. Then we have*

$$\text{(2.2)} \qquad \|\mathbf{x}\|_e \leq \frac{1}{\eta}\|B\mathbf{x}\|_e \quad \forall\, \mathbf{x} \in \mathbb{R}^n.$$

Proof. Clearly we have

$$\pm\mathbf{r} \leq \|\mathbf{r}\|_e\mathbf{e} \quad \forall\, \mathbf{r} \in \mathbb{R}^n.$$

Since $B^{-1} \geq 0$, if we set $B\mathbf{x} =: \mathbf{r}$ we have

$$\pm\mathbf{x} = \pm B^{-1}\mathbf{r} \leq \|\mathbf{r}\|_e B^{-1}\mathbf{e} \leq \|\mathbf{r}\|_e \frac{1}{\eta}\mathbf{e}.$$

This implies (2.2), because

$$\|\mathbf{x}\|_e \leq \frac{1}{\eta}\|\mathbf{r}\|_e = \frac{1}{\eta}\|B\mathbf{x}\|_e. \qquad \square$$

We are ready to define block M-matrices.

DEFINITION 2.3. A matrix A as in (1.1) is called a *block M-matrix* if

$$\text{(2.3)} \qquad A_{ij} = \sigma_{ij}I, \quad \sigma_{ij} \in \mathbb{R}, \quad \sigma_{ij} \leq 0, \quad i \neq j,$$

where I is the identity matrix, and if there exists a vector $\mathbf{e} \in \mathbb{R}^n$, $\mathbf{e} > \mathbf{0}$, and a scalar $\sigma \in \mathbb{R}$, $\sigma > 0$, such that

$$\text{(2.4)} \qquad \sum_{j=1}^{n} e_j A_{ij} \geq_q \sigma I, \qquad i = 1, \cdots, n.$$

Remarks. (1) Conditions (2.3) and (2.4) generalize (2.1a) and (2.1b), respectively. If the blocks are scalar ($m = 1$) we obtain Definition 2.1.

(2) Since $\mathbf{e} > \mathbf{0}$ in the above definition, an immediate consequence of (2.4) is the following: there is some $\eta \in \mathbb{R}$, $\eta > 0$ such that

$$\text{(2.5)} \qquad \sum_{j=1}^{n} e_j A_{ij} \geq_q \eta e_i I, \qquad i = 1, \cdots, n.$$

The conditions (2.4) and (2.5) express a "scaled diagonal dominance" property. If $e_j = 1$ in (2.4), for all j, we obtain a condition equivalent in this context to that of a strictly block diagonally dominant block matrix.

We can now prove the following theorem.

THEOREM 2.4. *Let A be a block M-matrix. Let $\eta \in \mathbb{R}$, $\eta > 0$, such that* (2.5) *is satisfied. Then we have*

$$\|\mathbf{x}\|_e \le \frac{1}{\eta}\|A\mathbf{x}\|_e \quad \forall\, \mathbf{x} \in \mathbb{R}^{nm}. \tag{2.6}$$

Proof. (i) Let us first assume that (2.4) is satisfied with $e_j = 1$, $j = 1, \cdots, n$. Let $\mathbf{x},\mathbf{r} \in \mathbb{R}^{nm}$ such that $A\mathbf{x} = \mathbf{r}$. Determine an index k so that

$$\|\mathbf{x}\| = |\mathbf{x}^k| \ge |\mathbf{x}^j|, \qquad j = 1, \cdots, n.$$

Taking the scalar product of $A_{kk}\mathbf{x}^k + \sum_{j\ne k} A_{kj}\mathbf{x}^j = \mathbf{r}^k$ with $\mathbf{x}^k$, and using (2.3), we obtain

$$(\mathbf{x}^k)^T A_{kk}\mathbf{x}^k + \sum_{j\ne k} \sigma_{kj}(\mathbf{x}^k)^T\mathbf{x}^j = (\mathbf{x}^k)^T\mathbf{r}^k.$$

Since $\sigma_{kj} \le 0$ and $(\mathbf{x}^k)^T\mathbf{x}^j \le (\mathbf{x}^k)^T\mathbf{x}^k$ we have

$$(\mathbf{x}^k)^T A_{kk}\mathbf{x}^k + \sum_{j\ne k}(\mathbf{x}^k)^T A_{kj}\mathbf{x}^k \le (\mathbf{x}^k)^T\mathbf{r}^k.$$

Then, (2.4) with $i = k$ gives us

$$\sigma|\mathbf{x}^k|^2 \le (\mathbf{x}^k)^T\mathbf{r}^k \le |\mathbf{x}^k| \cdot |\mathbf{r}^k|,$$

and therefore (2.6) follows in this case, because

$$\|\mathbf{x}\| = |\mathbf{x}^k| \le \frac{1}{\sigma}|\mathbf{r}^k| \le \frac{1}{\sigma}\|\mathbf{r}\|.$$

(ii) In the general case, where $\mathbf{e} > \mathbf{0}$ is arbitrary, we define the invertible $nm \times nm$ diagonal matrix

$$\Lambda = \begin{pmatrix} e_1 I & & 0 \\ & \ddots & \\ 0 & & e_n I \end{pmatrix},$$

and then $\Lambda^{-1}A\Lambda =: B$ is a block M-matrix to which the first part of the proof applies with $\sigma = \eta$. Now, $A\mathbf{x} = \mathbf{r}$ implies

$$B\Lambda^{-1}\mathbf{x} = \Lambda^{-1}\mathbf{r},$$

and therefore

$$\|\mathbf{x}\|_e = \|\Lambda^{-1}\mathbf{x}\| \le \frac{1}{\eta}\|\Lambda^{-1}\mathbf{r}\| = \frac{1}{\eta}\|\mathbf{r}\|_e$$

and the proof is completed. □

An immediate consequence of the theorem is Corollary 2.5.

COROLLARY 2.5. *Under the assumptions of Theorem* 2.4, *A is nonsingular and*

$$\|A^{-1}\|_e \le \frac{1}{\eta}. \tag{2.7}$$

□

Remark. It is possible to generalize other properties of standard M-matrices to block M-matrices. In this work we have only extended that property which is needed for the stability analysis in the next section. However, not all of the properties that characterize usual M-matrices can be extended to block M-matrices, as we have defined them. This aspect will be addressed in the future. Other extensions (not equivalent to ours) of standard M-matrices to the block case are in [EM] (and also here, not all standard M-matrix properties extend to the block case).

3. Computation of invariant tori.

3.1. Basic setting. In this paper, we restrict the setting to dynamical systems of the following form

$$\begin{aligned} \dot{\boldsymbol{\theta}} &= \mathbf{f}(\boldsymbol{\theta}), \\ \dot{\mathbf{r}} &= -C(\boldsymbol{\theta})\mathbf{r} + \mathbf{g}(\boldsymbol{\theta}), \end{aligned} \tag{3.1}$$

where $\boldsymbol{\theta}(t) \in T^p$ and $\mathbf{r}(t) \in \mathrm{I\!R}^q$; here $T^p = \mathrm{I\!R}^p(\bmod \mathbb{Z}^p)$ denotes the standard p-torus. The functions

$$\mathbf{f} : T^p \to \mathrm{I\!R}^p, \quad C : T^p \to \mathrm{I\!R}^{q\times q}, \quad \mathbf{g} : T^p \to \mathrm{I\!R}^q$$

are assumed to be in C^r, where $r \geq 1$ will be specified below. Then, for all initial data

$$(\boldsymbol{\theta}(0), \mathbf{r}(0)) = (\boldsymbol{\theta}_0, \mathbf{r}_0) \in T^p \times \mathrm{I\!R}^q, \tag{3.2}$$

the system (3.1) has a unique solution existing for all times. We are interested in determining a manifold $\mathcal{M}$, diffeomorphic to T^p, of the form

$$\mathcal{M} = \{(\boldsymbol{\theta}, \mathbf{R}(\boldsymbol{\theta})) : \boldsymbol{\theta} \in T^p\}, \qquad \mathbf{R} : T^p \to \mathrm{I\!R}^q, \tag{3.3}$$

which is invariant under the flow of (3.1). This means that if $(\boldsymbol{\theta}_0, \mathbf{r}_0) \in \mathcal{M}$, then the whole trajectory determined by the initial data (3.2) lies in $\mathcal{M}$.

The following well-known result (e.g., see [K]) is at the core of our computational technique.

LEMMA 3.1. *Suppose* $\mathbf{R} : T^p \to \mathrm{I\!R}^q$ *is a* C^1*-function. Then* $\mathcal{M}$ *in* (3.3) *is invariant under the flow of* (3.1) *if and only if* $\mathbf{R}$ *satisfies*

$$\sum_{\nu=1}^{p} f_\nu(\boldsymbol{\theta}) \frac{\partial \mathbf{R}}{\partial \theta_\nu}(\boldsymbol{\theta}) + C(\boldsymbol{\theta})\mathbf{R}(\boldsymbol{\theta}) = \mathbf{g}(\boldsymbol{\theta}), \qquad \boldsymbol{\theta} \in T^p, \tag{3.4}$$

where f_ν *and* θ_ν, $\nu = 1, \cdots, p$, *denote the components of* $\mathbf{f}$ *and* $\boldsymbol{\theta}$, *respectively.* □

Remark. Note that (3.4) is a vector equation for the q components of $\mathbf{R}$. The *principal part* of each equation reads

$$\sum_{\nu=1}^{p} f_\nu \frac{\partial R_\mu}{\partial \theta_\nu}, \qquad \mu = 1, \cdots, q,$$

i.e., the coefficients are the same for each $\mu = 1, \cdots, q$. This is why we can force the outer diagonal blocks of the discretization matrix to become multiples of the identity, and we are able to satisfy the condition (2.3).

To motivate condition (3.6) below, let us assume temporarily that the matrices $C(\boldsymbol{\theta})$ are all positive definite. Then we can expect that the influence of initial data $\mathbf{r}(0) = \mathbf{r}_0$ becomes negligible as $t \to \infty$, and the $\mathbf{r}$-vector becomes a function of $\boldsymbol{\theta}(t)$ only, $\mathbf{r}(t) \sim \mathbf{R}(\boldsymbol{\theta}(t))$. Thus we expect existence of an attracting manifold (3.3), which is positively invariant. (For the compact manifold $\mathcal{M}$, invariance for $t \to \infty$ implies invariance for $t \to -\infty$ as well.)

However, precise existence and smoothness statements are more involved. The basic difficulty is well understood, and it is given by the fact that (3.4) does not generally have a smooth solution $\mathbf{R}$ even if $\mathbf{f}(\boldsymbol{\theta})$, $C(\boldsymbol{\theta})$, and $\mathbf{g}(\boldsymbol{\theta})$ are in C^∞. General existence and smoothness results for invariant manifolds can be obtained from Fenichel [F]. For our purposes, Theorem 3.2 below suffices. To state the result, we need some further definitions.

Consider the two initial value problems

$$\dot{\boldsymbol{\theta}} = \mathbf{f}(\boldsymbol{\theta}),$$
$$\boldsymbol{\theta}(0) = \boldsymbol{\theta}_0,$$

with solution $\boldsymbol{\theta}(t) = S^t(\boldsymbol{\theta}_0)$, and

$$\dot{\mathbf{r}} = -C(S^t(\boldsymbol{\theta}_0))\mathbf{r},$$
$$\mathbf{r}(0) = \mathbf{r}_0,$$

with solution $\mathbf{r}(t) = M(t, \boldsymbol{\theta}_0)\mathbf{r}_0$, $M(t, \boldsymbol{\theta}_0) \in \mathbb{R}^{q \times q}$. We let $DS^t(\boldsymbol{\theta}_0) \in \mathbb{R}^{p \times p}$ denote the derivative of S^t with respect to $\boldsymbol{\theta}_0$. Then we define the generalized Lyapunov-type numbers (denoted by ν and σ, respectively, in [F])

$$\rho(\boldsymbol{\theta}_0) = \inf\{a \in (0, \infty) : |M(t, S^{-t}(\boldsymbol{\theta}_0))|/a^t \to 0, \text{ as } t \to \infty\}, \tag{3.5a}$$

$$\mu(\boldsymbol{\theta}_0) = \inf\{s \in (0, \infty) : |DS^{-t}(\boldsymbol{\theta}_0)| \cdot |M(t, S^{-t}(\boldsymbol{\theta}_0))|^s \to 0, \text{ as } t \to \infty\}. \tag{3.5b}$$

(The expressions do not depend on the chosen matrix norm $|\cdot|$, but in our discussion below we shall assume that $|\cdot|$ corresponds to the Euclidean vector norm.)

THEOREM 3.2. *Let* $\mathbf{f}, C, \mathbf{g} \in C^r$, $r \in \mathbb{N}$, *in* (3.4). *Let* $\rho(\boldsymbol{\theta}_0) < 1$, $\mu(\boldsymbol{\theta}_0) < 1/r$, *for all* $\boldsymbol{\theta}_0 \in T^p$. *Then* (3.4) *has a solution* $\mathbf{R} \in C^r(T^p, \mathbb{R}^q)$. *The solution is unique in* $C^1(T^p, \mathbb{R}^q)$.

Remark. The existence result follows from Theorem 1 of [F] if we consider (3.1) with $\mathbf{g} \equiv \mathbf{0}$ as the unperturbed case with the invariant torus

$$\mathcal{M}_0 = \{(\boldsymbol{\theta}, 0) \ : \ \boldsymbol{\theta} \in T^p\}.$$

Then a perturbation $\epsilon\mathbf{g}(\boldsymbol{\theta})$ is introduced and the result of [F] applied; because of the linearity of the equation, the size of ϵ is irrelevant. Uniqueness of $\mathbf{R}$ follows from the proof in [F], which is based on a contraction argument.

To illustrate Theorem 3.2, let us assume:

(3.6) There exists a $\sigma \in \mathbb{R}$, $\sigma > 0$ such that $C(\boldsymbol{\theta}) \geq_q \sigma I \quad \forall\, \boldsymbol{\theta} \in T^p$.

Then we obtain $|M(t, \boldsymbol{\theta})| \leq e^{-\sigma t}$ for all $t \geq 0$ and all $\boldsymbol{\theta} \in T^p$, and therefore

$$\rho(\boldsymbol{\theta}_0) \leq e^{-\sigma} < 1 \quad \text{for all } \boldsymbol{\theta}_0 \in T^p.$$

Thus, if $\mathbf{g} \equiv \mathbf{0}$ in (3.1), $\mathcal{M}_0$ attracts all trajectories at the exponential rate $e^{-\sigma t}$, for $t \geq 0$. Furthermore, let us assume that

$$(3.7) \qquad |D\mathbf{f}(\boldsymbol{\theta})| \leq \epsilon \quad \forall \boldsymbol{\theta} \in T^p, \quad \text{where } 0 < \epsilon \ll 1,$$

i.e., if $\mathbf{g} \equiv \mathbf{0}$ in (3.1), then the flow on $\mathcal{M}_0$ is almost parallel. From (3.7) it follows that $|DS^{-t}(\boldsymbol{\theta}_0)| \leq e^{\epsilon t}$ for all $\boldsymbol{\theta}_0 \in T^p$, $t \geq 0$, and therefore we have $|DS^{-t}(\boldsymbol{\theta}_0)| \cdot |M(t, S^{-t}(\boldsymbol{\theta}_0))|^s \leq e^{(\epsilon - \sigma s)t}$. In (3.5b) we obtain $\mu(\boldsymbol{\theta}_0) \leq \frac{\epsilon}{\sigma}$, for all $\boldsymbol{\theta}_0 \in T^p$. Thus, Theorem 3.2 guarantees a C^r solution $\mathbf{R}$ of (3.4) as long as $\mathbf{f}, C, \mathbf{g} \in C^r$ and $r\epsilon < \sigma$. The invariant manifold (3.3) still attracts trajectories at the exponential rate $e^{-\sigma t}$ for $t \to \infty$. This follows easily from (3.6) using linearity of (3.1) in $\mathbf{r}$. To summarize the discussion, the larger $\sigma > 0$ in (3.6) (that is, the stronger the attractivity), and the smaller $\epsilon > 0$ in (3.7) (that is, the closer the flow on $\mathcal{M}_0$ is to being parallel), the more derivatives of the solution $\mathbf{R}$ can be guaranteed, and the faster the solution trajectories approach $\mathcal{M}$.

Next, we propose a discretization scheme for (3.4), which we want to investigate. Assumption (3.6) will enable us to satisfy the condition (2.5) for the discretization matrix.

3.2. Numerical discretization. To discretize (3.4), we use equispaced grid points in each of the angular coordinates. We let $h_\nu = 2\pi/N_\nu$, $\nu = 1, \cdots, p$, and $\mathbf{h} = (h_1, \cdots, h_p)$, and replace the torus T^p by its discrete analog

$$(3.8) \qquad T_h^p = \{\boldsymbol{\theta}_h = (j_1 h_1, \cdots, j_p h_p),\ j_\nu \in \mathbb{Z} \bmod N_\nu,\ \nu = 1, \cdots, p\}.$$

Then T_h^p denotes a periodic grid with $N := N_1 \cdots N_p$ mesh points. The unknown grid function will be denoted by $\mathbf{R}_h, \mathbf{R}_h : T_h^p \to \mathbb{R}^q$.

In the usual way, we define the shift operators E_ν, $\nu = 1, \cdots, p$, as

$$(E_\nu \mathbf{R}_h)(\boldsymbol{\theta}_h) := \mathbf{R}_h(\cdots, (j_\nu + 1)h_\nu, \cdots)$$

(and the integer $j_\nu + 1$ is computed modulo N_ν). We also define the forward, backward, and centered divided difference operators as usual:

$$D_{+\nu} = \frac{1}{h_\nu}(E_\nu - I), \qquad D_{-\nu} = \frac{1}{h_\nu}(I - E_\nu^{-1}), \qquad D_{0\nu} = \frac{1}{2}(D_{+\nu} + D_{-\nu}).$$

We replace (3.4) by the following periodic difference equations of *upwinding type*

$$(3.9\text{a}) \qquad \sum_{\nu=1}^{p} \left[f_\nu(\boldsymbol{\theta}_h) D_{0\nu} \mathbf{R}_h - \frac{1}{2} h_\nu \phi(f_\nu(\boldsymbol{\theta}_h)) D_{+\nu} D_{-\nu} \mathbf{R}_h \right] + C(\boldsymbol{\theta}_h)\mathbf{R}_h = \mathbf{g}(\boldsymbol{\theta}_h), \qquad \boldsymbol{\theta}_h \in T_h^p,$$

where

$$(3.9\text{b}) \qquad \phi(\alpha) = (1 + \alpha^2)^{1/2}, \qquad D_{+\nu} D_{-\nu} = \frac{1}{h_\nu^2}(E_\nu - 2I + E_\nu^{-1}).$$

The "classical" upwind scheme consists in taking $\phi(\alpha) = |\alpha|$ in the above; the choice of a nonsmooth ϕ, however, does not allow the error expansion of §3.3. This motivates us to choose a smooth ϕ in (3.9b); the particular choice $\phi(\alpha) = (1 + \alpha^2)^{1/2}$ is considered in [St] for a convection-diffusion equation with dominating convection. Other choices are possible.

Remark. Our computational experience indicates that the "classical" ($\phi(\alpha) = |\alpha|$) and the "smooth" ($\phi(\alpha) = (1+\alpha^2)^{1/2}$) upwinding perform very similarly in accuracy terms. However, the classical choice leads to greater computational efficiency because the resulting matrix has a more exploitable structure. As we will see in the following, Theorems 3.3 and 3.4 also hold for the classical upwinding. For this reason, unless the smooth choice of $\phi(\alpha)$ is needed, as for the extrapolation procedure, we suggest the use of the classical upwinding scheme.

To obtain a matrix equation, we order the N grid points in a reverse lexicographic way, for example, and we keep the q components of $\mathbf{R}_h$ at each grid point together. So doing, we obtain a system matrix A_h, consisting of $(N \times N)$ blocks, each of size $(q \times q)$. The diagonal blocks have the form

$$C(\boldsymbol{\theta}_h) + \sum_{\nu=1}^{p} \frac{1}{h_\nu}\phi(f_\nu(\boldsymbol{\theta}_h)) \cdot I_q. \tag{3.10}$$

The outer diagonal blocks are either 0, or have the form

$$-\frac{1}{2h_\nu}(f_\nu(\boldsymbol{\theta}_h) + \phi(f_\nu(\boldsymbol{\theta}_h))) \cdot I_q \quad \text{and} \quad \frac{1}{2h_\nu}(f_\nu(\boldsymbol{\theta}_h) - \phi(f_\nu(\boldsymbol{\theta}_h))) \cdot I_q \tag{3.11}$$

(for the classical upwinding scheme, one of these two blocks in (3.11) is 0). Since $\phi(\alpha) \geq |\alpha|$, we satisfy condition (2.3) and because of assumption (3.6) we also satisfy (2.4) with all $e_j = 1$. Therefore, we can use Theorem 2.4 and Corollary 2.5, with all $e_j = 1$ and $\eta = \sigma$, to obtain Theorem 3.3.

THEOREM 3.3. *Assume that* (3.6) *holds and that the matrix A_h corresponding to* (3.9) *has been set up as described above. Then the difference equations* (3.9) *have a unique solution $\mathbf{R}_h$. Furthermore, the discretization is unconditionally stable in the sense that the following bound on A_h^{-1} holds uniformly in* $\mathbf{h}$

$$||A_h^{-1}|| \leq \frac{1}{\sigma}. \tag{3.12}$$

It is now standard to obtain error estimates and convergence for the scheme (3.9), if existence of a sufficiently smooth solution $\mathbf{R}$ of (3.4) is assumed. In fact, we can prove the following theorem.

THEOREM 3.4. *Assume that* (3.6) *holds and that* (3.4) *has a solution* $\mathbf{R} \in C^2(T^p, \mathbb{R}^q)$. *Let $\mathbf{R}|_h$ denote the restriction of $\mathbf{R}$ to T_h^p and let $\mathbf{R}_h$ denote the solution of the difference equation* (3.9). *Then we have*

$$||\mathbf{R}_h - \mathbf{R}|_h|| \leq \frac{1}{\sigma}|\mathbf{h}|_\infty \left(\frac{1}{2} + ||f||_\infty\right) ||D^2\mathbf{R}||, \tag{3.13}$$

where we have used the notations

$$|\mathbf{h}|_\infty = \max_{1\leq\nu\leq p} h_\nu,$$

$$||\mathbf{f}||_\infty = \max_{1\leq\nu\leq p} \max_{\boldsymbol{\theta}\in T^p} |f_\nu(\boldsymbol{\theta})|,$$

$$||D^2\mathbf{R}|| = \sum_{\nu=1}^{p} \max_{\boldsymbol{\theta}\in T^p} \left|\frac{\partial^2\mathbf{R}}{\partial\theta_\nu^2}(\boldsymbol{\theta})\right| = \sum_{\nu=1}^{p} \left|\left|\frac{\partial^2\mathbf{R}}{\partial\theta_\nu^2}\right|\right|$$

with $|\cdot|$ *the Euclidean norm in* $\mathbb{R}^q$.

Proof. Taylor's formula with remainder in integral form (or integration by parts) yields

$$\frac{\partial \mathbf{R}}{\partial \theta_\nu}(\theta) = D_{0\nu}\mathbf{R}(\theta) + \frac{h_\nu}{2}\mathbf{T}_\nu, \quad |\mathbf{T}_\nu| \le \left\|\frac{\partial^2 \mathbf{R}}{\partial \theta_\nu^2}\right\|, \quad |D_{+\nu}D_{-\nu}\mathbf{R}(\theta)| \le \left\|\frac{\partial^2 \mathbf{R}}{\partial \theta_\nu^2}\right\|.$$

We substitute $\mathbf{R}|_h$ into (3.9a), use (3.4), and subtract (3.9a) to obtain

$$\sum_{\nu=1}^{p} f_\nu D_{0\nu}(\mathbf{R}|_h - \mathbf{R}_h) - \frac{1}{2}\sum_{\nu=1}^{p} h_\nu \phi_\nu D_{+\nu}D_{-\nu}(\mathbf{R}|_h - \mathbf{R}_h) + C(\mathbf{R}|_h - \mathbf{R}_h) = \mathbf{E}_h$$

with

$$\phi_\nu(\boldsymbol{\theta}) = \phi(f_\nu(\boldsymbol{\theta}))$$

and

$$\mathbf{E}_h = -\frac{1}{2}\sum_{\nu=1}^{p} h_\nu f_\nu \mathbf{T}_\nu - \frac{1}{2}\sum_{\nu=1}^{p} h_\nu \phi_\nu D_{+\nu}D_{-\nu}\mathbf{R}.$$

Therefore,

$$|\mathbf{E}_h(\boldsymbol{\theta})| \le \tfrac{1}{2}|\mathbf{h}|_\infty \max_{1\le\nu\le p}(|f_\nu|_\infty + |\phi_\nu|_\infty)\, \|D^2\mathbf{R}\|,$$

and since we have $|\phi_\nu|_\infty \le 1 + |f_\nu|_\infty$ for our choice of ϕ, an application of the stability result Theorem 3.3 finishes the proof. □

Remarks. (1) Let J denote the union of the ranges of the functions $f_\nu, \nu = 1, \cdots, p$. Then Theorem 3.3 remains valid if we choose any function $\phi(\alpha)$ in (3.9a) with $\phi(\alpha) \ge |\alpha|$ for all $\alpha \in J$. Similarly, a result like Theorem 3.4 is valid for any ϕ with $|\alpha| \le \phi(\alpha) \le \text{const}\,(1+|\alpha|)$ for all $\alpha \in J$. In particular, these results are valid for classical upwinding corresponding to $\phi(\alpha) = |\alpha|$ and for schemes based on exponential fitting.

(2) The scheme (3.9) is only first order. In [DLR] we considered the leap-frog scheme, for which the local truncation error for a C^3-solution is second-order. A stability analysis based on block M-matrices seems to be restricted to first-order schemes, however, and different tools—like discrete L_2-estimates—will probably be necessary to analyze higher-order difference schemes. However, we can also resort to extrapolation (or deferred correction) techniques for the above first-order scheme to increase the order; see §3.3.

(3) To discuss the influence of the parameter σ in (3.6), let us assume that we keep the coefficients $f_\nu(\boldsymbol{\theta})$ and $\mathbf{g}(\boldsymbol{\theta})$ in (3.4) fixed whereas we choose $C(\boldsymbol{\theta}) = C(\boldsymbol{\theta}, \sigma) \ge_q \sigma I$ for different $\sigma > 0$. If σ is increased, we can, in general, expect increased smoothness of the solution $\mathbf{R}(\boldsymbol{\theta}, \sigma)$ of (3.4); see Theorem 3.2. Another consequence of increasing σ is a stronger attractivity of $\mathcal{M}$; see the discussion following Theorem 3.2. On the discrete level, we obtain from (3.12) an improved stability constant with increasing σ. Thus, for small σ, we must expect difficulties in the numerical calculations because of a possible lack of smoothness of $\mathbf{R}$ and because of a large stability constant of the discrete system. On the other hand, for large σ, the solution $\mathbf{R}$ will become smoother and the stability will become better. These occurrences are borne out in practice (see §4).

(4) Note that our results do not require any restriction on the mesh ratios h_ν/h_μ $(\nu \ne \mu)$. Thus, nothing like a CFL condition is required for stability, though one of the variables θ_ν in (3.1) might correspond to time.

(5) It is interesting to note that the discretization (3.9) can also be viewed as a discretization of the PDE

$$\sum_{\nu=1}^{p}\left[f_\nu(\boldsymbol{\theta})\frac{\partial \mathbf{R}}{\partial \theta_\nu}(\boldsymbol{\theta}) - \epsilon_\nu \frac{\partial^2 \mathbf{R}}{\partial \theta_\nu^2}(\boldsymbol{\theta})\right] + C(\boldsymbol{\theta})\mathbf{R}(\boldsymbol{\theta}) = \mathbf{g}(\boldsymbol{\theta}),$$

with $\epsilon_\nu > 0$ corresponding to $\frac{1}{2}h_\nu\phi(f_\nu)$. Thus the setting is closely related to "elliptic regularization," as also used, for example, by Sacker [Sa]. However, in Sacker's analysis the dissipation term is introduced to ensure smoothness of the solution of the modified continuous problem, whereas we add the artificial dissipation term to ensure the block M-matrix structure, which leads to stability.

3.3. Global error expansion and extrapolation. We again assume (3.6). Here we want to show that the scheme (3.9) allows an expansion of the error $\mathbf{R}|_h - \mathbf{R}_h$ in terms of powers of the stepsize. To prove such a result, we need to assume sufficient smoothness of the solution $\mathbf{R}$ of the continuous problem (3.4). In addition, to obtain the expansion, the PDE (3.4) must be solved with smooth right-hand sides different from $\mathbf{g}$, and we need existence and smoothness of the corresponding solutions. Sufficient conditions for these assumptions can be obtained by Theorem 3.2. For simplicity, we take all stepsizes h_ν as equal, $h_\nu = h$, $\nu = 1, \cdots, p$. Then we can show an expansion

$$\mathbf{R}|_h = \mathbf{R}_h + h\mathbf{z}_1|_h + h^2\mathbf{z}_2|_h + \cdots + h^j\mathbf{z}_j|_h + O(h^{j+1}), \tag{3.14}$$

where $\mathbf{z}_i : T^p \longrightarrow \mathbb{R}^q$ are smooth functions with restrictions $\mathbf{z}_i|_h$ on T_h^p. We give a precise formulation and a proof only for $j = 1$; a generalization to higher-order corrections follows along the same lines.

THEOREM 3.5. *Consider the discretization* (3.9) *for* (3.4), *and assume* $\mathbf{R} \in C^4(T^p, \mathbb{R}^q)$ *solves* (3.4). *Also, let* $\mathbf{f} \in C^2$, *and assume that the* PDE (3.4) *has a* C^2-*solution for any* C^2-*right-hand side. Then, there exists a function* $\mathbf{z}_1 \in C^2(T^p, \mathbb{R}^q)$ *such that*

$$\mathbf{R}|_h = \mathbf{R}_h + h\mathbf{z}_1|_h + O(h^2). \tag{3.15}$$

Proof. By Taylor's formula (for $\mathbf{R} \in C^3$),

$$\frac{\partial \mathbf{R}}{\partial \theta_\nu}(\theta) = D_{0\nu}\mathbf{R}(\theta) + \frac{h_\nu^2}{6}\mathbf{T}_\nu, \qquad |\mathbf{T}_\nu| \le \left\|\frac{\partial^3 \mathbf{R}}{\partial \theta_\nu^3}\right\|,$$
$$D_{+\nu}D_{-\nu}\mathbf{R}(\theta) = \frac{\partial^2 \mathbf{R}}{\partial \theta_\nu^2}(\theta) + \frac{h}{3}\mathbf{S}_\nu, \qquad |\mathbf{S}_\nu| \le \left\|\frac{\partial^3 \mathbf{R}}{\partial \theta_\nu^3}\right\|.$$

We substitute $\mathbf{R}|_h$ into (3.9a), use (3.4), and subtract (3.9a) to obtain

$$\sum_{\nu=1}^{p} f_\nu D_{0\nu}(\mathbf{R}|_h - \mathbf{R}_h) - \frac{1}{2}\sum_{\nu=1}^{p} h\phi_\nu D_{+\nu}D_{-\nu}(\mathbf{R}|_h - \mathbf{R}_h) + C(\mathbf{R}|_h - \mathbf{R}_h) = \mathbf{E}_h \tag{3.16}$$

with

$$\phi_\nu(\boldsymbol{\theta}) = \phi(f_\nu(\boldsymbol{\theta}))$$

and

$$\mathbf{E}_h = -\frac{h^2}{6}\sum_{\nu=1}^{p} f_\nu \mathbf{T}_\nu - \frac{h}{2}\sum_{\nu=1}^{p} \phi_\nu \frac{\partial^2 \mathbf{R}}{\partial \theta_\nu^2} - \frac{h^2}{6}\sum_{\nu=1}^{p} \phi_\nu \mathbf{S}_\nu.$$

Thus, the right-hand side of (3.16) has the form

$$\mathbf{E}_h(\boldsymbol{\theta}) = h\mathbf{r}_1(\boldsymbol{\theta}) + O(h^2)$$

where

$$\mathbf{r}_1(\boldsymbol{\theta}) = -\frac{1}{2}\sum_{\nu=1}^{p} \phi_\nu(\boldsymbol{\theta})\frac{\partial^2 \mathbf{R}}{\partial \theta_\nu^2}(\boldsymbol{\theta})$$

is a C^2-function. We define $\mathbf{z}_1 \in C^2$ to be the solution of

$$\sum_{\nu=1}^{p} f_\nu \frac{\partial \mathbf{z}_1}{\partial \theta_\nu} + C\mathbf{z}_1 = \mathbf{r}_1,$$

and we define $\mathbf{z}_{1,h}$ to be the solution of the discrete analogue,

$$A_h \mathbf{z}_{1,h} = \mathbf{r}_1|_h.$$

(Here we identify, as usual, vectors with grid functions, and difference equations with their matrix representations.) Then we have

$$\begin{aligned} A_h(\mathbf{R}|_h - \mathbf{R}_h) &= h\mathbf{r}_1|_h + O(h^2) \\ &= hA_h\mathbf{z}_{1,h} + O(h^2), \end{aligned}$$

and therefore (by Theorem 3.3)

$$\begin{aligned} \mathbf{R}|_h - \mathbf{R}_h &= h\mathbf{z}_{1,h} + O(h^2) \\ &= h\mathbf{z}_1|_h + O(h^2). \end{aligned}$$

To obtain the last equation, we have used the fact that

$$\mathbf{z}_1|_h = \mathbf{z}_{1,h} + O(h),$$

which follows from Theorem 3.4. This completes the proof of the theorem. □

4. Algorithm of solution and numerical examples. Here we present the results obtained for two sample problems, which highlight the considerations of the previous sections. We restrict our attention to two angular coordinates, the two-torus $T^2 = \{\boldsymbol{\theta} = (x, y) : x, y \in \mathrm{IR} \bmod 2\pi\}$, and two "radial" coordinates $\mathbf{r} = \binom{u}{v} \in \mathrm{IR}^2$. The PDE system (3.4) then assumes the form

$$(4.1) \qquad f_1(x,y)\begin{pmatrix} u \\ v \end{pmatrix}_x + f_2(x,y)\begin{pmatrix} u \\ v \end{pmatrix}_y + C(x,y)\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} g_1(x,y) \\ g_2(x,y) \end{pmatrix},$$

with $C(x, y) \in \mathrm{IR}^{2\times 2}$.

The discretized periodic square is $T_h^2 = \{(x_i, y_j) = (ih_1, jh_2), h_1 = 2\pi/N, h_2 = 2\pi/M, i \in \mathbb{Z} \bmod N, j \in \mathbb{Z} \bmod M\}$, and we order the grid points as $(x_1, y_1), \cdots,$

$(x_N, y_1), (x_1, y_2), \cdots, (x_N, y_M)$. Then, the discretization matrix assumes the *macro*-structure:

$$(4.2) \qquad A_h = \begin{bmatrix} B_{11} & B_{12} & & & B_{1M} \\ B_{21} & B_{22} & B_{23} & & \bigcirc \\ & \ddots & \ddots & \ddots & \\ \bigcirc & & B_{M-1,M-2} & B_{M-1,M-1} & B_{M-1,M} \\ B_{M1} & & & B_{M,M-1} & B_{MM} \end{bmatrix},$$

where each $B_{ij} \in \mathrm{IR}^{2N \times 2N}$. Each outer diagonal B_{ij} has the form

$$\begin{pmatrix} \square & & \bigcirc \\ & \square & \\ & & \ddots & \\ \bigcirc & & & \square \end{pmatrix}$$

with (2×2) diagonal matrices $\square$ along the diagonal of B_{ij}. Each B_{ii} has the form

$$\begin{pmatrix} \square & \square & & & \square \\ \square & \square & \square & & \\ & \ddots & \ddots & \ddots & \\ & & \square & \square & \square \\ \square & & & \square & \square \end{pmatrix}$$

where all blocks $\square$ are again of size (2×2); the (2×2) blocks along the diagonal of B_{ii} are full, in general, whereas the outer diagonal blocks of B_{ii} are (2×2) diagonal matrices. This structure is typical for many discretizations of (4.1) (e.g., see [DLR]).

For the scheme (3.9), the (2×2) diagonal blocks are given by (3.10) and the (2×2) outer diagonal blocks are given by (3.11). We would like to stress that the matrix A_h, seen as a block matrix with blocks B_{ij}, is *not* a block M-matrix according to Definition 2.3, in general. We must look at the *micro*structure of (2×2) blocks in order to use block M-matrix theory.

There are various possibilities for solving systems of the type $A_h \mathbf{x} = \mathbf{b}$ with A_h as in (4.2). In [DLR], we described an efficient compactification procedure. This compactification takes advantage of the sparsity of the blocks B_{ij}, and it can be opportunely generalized to block M-matrices as well. However, in this paper we favor a direct factorization of the matrix A_h. This approach does not impose restrictions on the discretization, and it has its main limitation on storage requirements.

We gained experience with two (related, but not equivalent) factorization strategies for A_h. The first factorization relies on the *bordered* LU algorithm, which orginates from attempting the following block decomposition, $A_h = LU$, with L block unit lower triangular (see also [Ma]):

$$(4.3a) \qquad L = \begin{pmatrix} I & & & & \\ L_2 & I & & & \bigcirc \\ & \ddots & \ddots & & \\ \bigcirc & & L_{M-1} & & \\ E_1 & \cdots & E_{M-2} & E_{M-1} & I \end{pmatrix},$$

$$U = \begin{pmatrix} U_1 & B_{12} & & & G_1 \\ & U_2 & \ddots & \bigcirc & \\ & & \ddots & B_{M-1,M-1} & \vdots \\ \bigcirc & & & U_{M-1} & G_{M-1} \\ & & & & U_M \end{pmatrix}. \tag{4.3b}$$

We have successfully used this factorization as implemented in the solver LU2-SOL2,[1] where pivoting is performed only on the tridiagonal part of A_h; this reduces fill-in while retaining numerical stability in practice.

The second factorization, to be described next, and already mentioned in [DLR], uses a reordering of blocks. For our examples, we found this strategy to be more efficient, and the numerical results that we present below rely on this second strategy.

LEMMA 4.1. *Consider the permutation vector*

$$\mathbf{P} = \begin{cases} (1, M, 2, M-1, \cdots, \frac{M}{2}, \frac{M}{2}+1)^T, & M \text{ even} \\ (1, M, 2, M-1, \cdots, \lfloor \frac{M}{2} \rfloor, \lceil \frac{M}{2} \rceil + 1, \lceil \frac{M}{2} \rceil)^T, & M \text{ odd,} \end{cases} \tag{4.4}$$

and consider the block permutation matrix $\mathbb{P}$, *associated with* $\mathbf{P}$. *That is,* $\mathbb{P}$ *is zero except for identity blocks of dimension* $2N$, *which are placed in locations* (j, P_j), $j = 1, \cdots, M$. *Then,* $\mathbb{P}$ *transforms* A_h *into block-pentadiagonal structure:*

$$\bar{A}_h := \mathbb{P} A_h \mathbb{P}^T = \begin{pmatrix} B_{11} & B_{1M} & B_{12} & 0 & & \\ B_{M1} & B_{MM} & 0 & B_{M,M-1} & & \\ B_{21} & 0 & B_{22} & 0 & B_{23} & \\ & B_{M-1,M} & 0 & B_{M-1,M-1} & 0 & B_{M-1,M-2} \\ & \ddots & \ddots & \ddots & \ddots & \ddots \end{pmatrix}. \tag{4.5}$$

Proof. The proof is by direct verification. □

If M is even, we might consider $\bar{A}_h$ as a block tridiagonal matrix with blocks of dimension $4N \times 4N$. However, the results in [V] cannot be used, in general, because block diagonal dominance is not ensured. In any case, the block tridiagonal factorization of [V] would require as much work and storage as the factorization based on (4.3).

In our strategy, we disregard any block structure of $\bar{A}_h$ and consider $\bar{A}_h$ as a band matrix with upper and lower bandwidth $4N$; this takes into account that B_{ij} is diagonal for $i \neq j$. To compute the factorization, we use the LINPACK implementation. The factorization requires as much work and storage as the above-mentioned block factorization algorithms if no pivoting is performed (see [V]), and slightly more work and additional $4MN^2$ memory locations if partial pivoting is performed. Partial pivoting is necessary to retain numerical stability.

We also tried general-purpose sparse direct factorization software (using Harwell's routines MA28), but the performance was not competitive with the other approaches on our computational environment.

All computations below have been performed on a SPARC workstation in single precision arithmetic (unit roundoff is approximately 10^{-7}).

[1] This code was kindly provided to us by B. Fornberg.

TABLE 1
Example 1.

N	M	α	β	cond	E_u	E_v
10	10	1	1	11	.6	.6
"	"	.1	.1	19	1.59	1.59
"	"	10^{-3}	10^{-3}	23	1.95	1.95
"	"	10^{-3}	10^{+3}	$5 * 10^5$	.83	.0016
"	"	10^{+3}	10^{-3}	$5 * 10^5$	.0016	.83
"	"	10	10	2	.08	.08
"	"	10^3	10^3	1	.0008	.0008
20	20	1	1	20	.36	.37
"	"	.1	.1	58	1.37	1.37
"	"	10^{-3}	10^{-3}	85	2	2
"	"	10^{-3}	10^{+3}	$5 * 10^5$	.43	.0008
"	"	10^{+3}	10^{-3}	$5 * 10^5$	.0008	.43
"	"	10	10	3	.04	.04
"	"	10^3	10^3	1	.0004	.0004

Example 1. This problem, with a known analytical solution, has been chosen to confirm our theoretical results on the order of convergence and extrapolation, and to illustrate the dependence of the numerical error on the attractivity of the tori. If σ in (3.6) is small, we experience numerical difficulties because A_h has a large inverse. (In general, a small σ might also lead to a nonsmooth solution, but in the present example the solution is C^∞.) We have the system

$$u_x + u_y + \alpha u - v = \alpha(\sin x + \cos y),$$
$$v_x + v_y + u + \beta v = \beta(\cos x - \sin y),$$

as usual, subject to periodicity conditions, which has the exact solution $u(x,y) = \sin x + \cos y$, $v(x,y) = \cos x - \sin y$. It is easy to see that the matrix

$$C = \begin{pmatrix} \alpha & -1 \\ 1 & \beta \end{pmatrix}$$

is positive definite for any positive value of α and β, and that (3.6) is satisfied with $\sigma = \min(\alpha, \beta)$, and that (3.7) is satisfied with $\epsilon = 0$. The discretized problem leads to a block M-matrix, but not an M-matrix in the usual sense. The original dynamical system is simply

$$\dot{x} = 1,$$
$$\dot{y} = 1,$$
$$\dot{u} = -\alpha u + v + \alpha(\sin x + \cos y),$$
$$\dot{v} = -u - \beta v + \beta(\cos x - \sin y),$$

and it is clear that u and v approach the invariant torus (become functions of x and y only) approximately as fast as $e^{-\alpha t}$ and $e^{-\beta t}$, respectively. By changing α and β we change this speed. In Tables 1 and 2 we report on some results for various grids and values of α and β. The entries E_u and E_v refer to the max-error on the grid for u and v, respectively, and cond measures (as from LINPACK) the condition number of A_h. Obviously, the scheme is only first order.

TABLE 2
Example 1.

N	M	α	β	cond	E_u	E_v
40	40	1	1	38	.2	.2
"	"	.1	.1	174	1.05	1.05
"	"	10^{-3}	10^{-3}	330	1.98	1.98
"	"	10	10	4.7	.022	.022
80	80	1	1	74	.105	.105
"	"	.1	.1	469	.714	.714
"	"	10^{-3}	10^{-3}	1293	1.96	1.96
"	"	10	10	8.3	.01	.01

TABLE 3
Extrapolation for Example 1.

Order	$N = M$	α	β	E_v	E_v
h^2	10	10	10	.0066	.0066
"	20	"	"	.0017	.0017
"	40	"	"	.0005	.0005
"	10	1	1	.119	.119
"	20	"	"	.038	.038
"	40	"	"	.011	.011
"	"	.1	.1	.376	.376
h^3	20	"	"	.26	.26

For this particular problem, by using (2.1) to obtain $\|A_h^{-1}\| \leq 1/\sigma$, we might expect

$$\operatorname{cond}(A_h) \leq \left[\max(\alpha, \beta) + 1 + \frac{4}{h}\right] \cdot \frac{1}{\min(\alpha, \beta)},$$

and from Table 1 we observe that such a bound is essentially achieved for $\alpha = 10^{\pm 3}$ and $\beta = 1/\alpha$. For $\alpha = 10^{\pm 4}$, $\beta = 1/\alpha$, A_h is singular to working precision (that is, $\operatorname{cond}(A_h) > 5 * 10^{+7}$). From Tables 1 and 2, we clearly see that for $\alpha = \beta$ the stronger (weaker) the attractivity, the smaller (larger) the error, as we expected from the error estimates (3.13). For strongly attractive tori (say, $\alpha = \beta = 10^3$), very few grid points ($N = M = 10$) suffice for four digits accuracy, but for weakly attractive tori (say, $\alpha = \beta = 10^{-1}$ or 10^{-3}) even with $N = M = 80$, we have no significant digits at all. A higher-order scheme is of some help in this situation. In Table 3 we report on some typical results obtained with Richardson's extrapolation. The entry "Order" refers to the theoretical order of the procedure. Now we get one digit accuracy when $\alpha = \beta = .1$. The procedure is clearly second order, as can be appreciated when $\alpha = \beta = 1$.

Finally, an interesting occurrence can be inferred from Table 1: strong attractivity in only one of the u or v coordinates results in a very good approximation for that component and it also helps the approximation of the other one. The error estimates do not account for this behavior. This fact suggests that different (and more detailed) error estimates could be achieved.

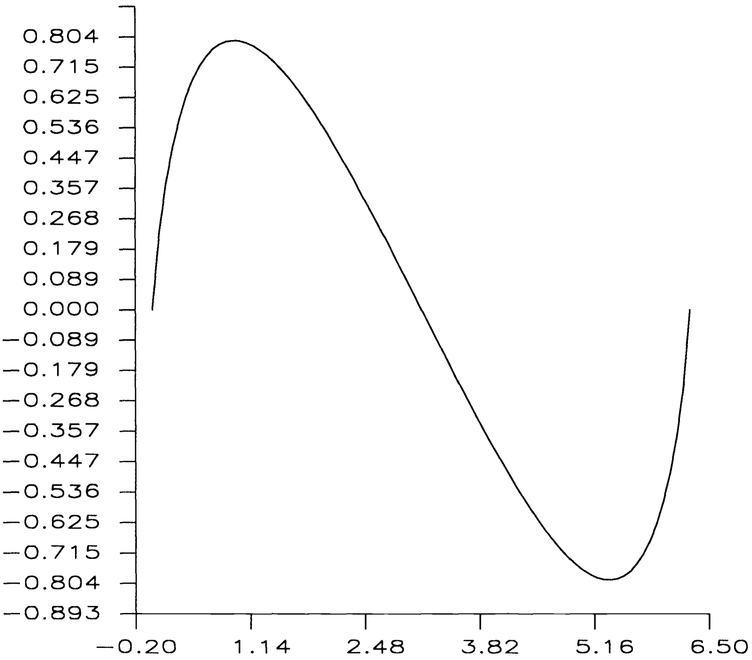


FIG. 1. *Example* 2, $b = 1$.

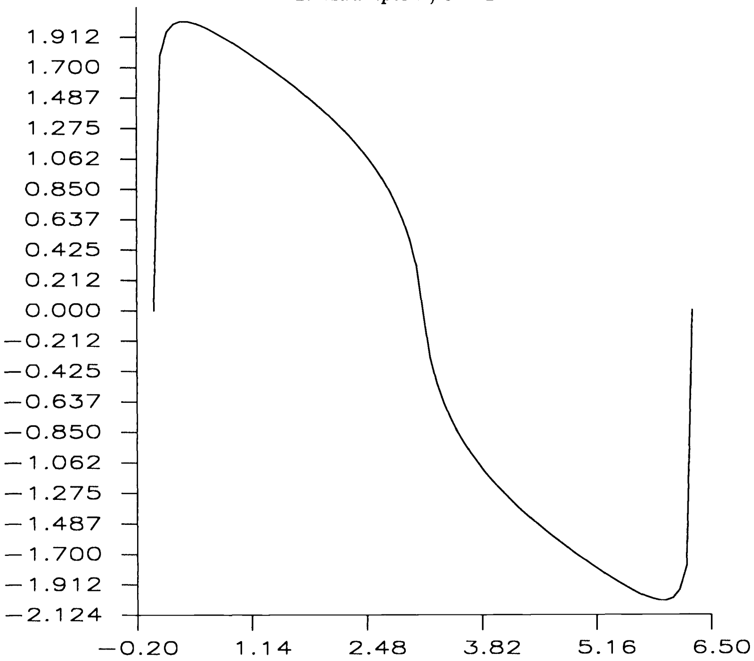


FIG. 2. *Example* 2, $b = \frac{1}{3}$.

Example 2. In this second example the function $\mathbf{f}(\boldsymbol{\theta})$ changes with $\boldsymbol{\theta}$, and a smooth solution cannot be guaranteed; in fact, it does not exist in the case considered. We have chosen this example to demonstrate how well the upwinding scheme still works

TABLE 4
Example 2, $b = 1$.

N	M	cond	Error
10	10	12	.23
20	20	25	.13
40	40	51	.065
80	80	105	.03

TABLE 5
Extrapolation for Example 2, $b = 1$.

Order	$N = M$	Error
h^2	10	.0077
"	20	.0025
"	40	.00046
h^3	10	.0008
"	20	.0002
h^4	10	.0004

in this context. The problem is adapted from an example in [Mo]. We have the scalar PDE

$$-u_x \cos x - u_y \sin y + bu = \sin^b y,$$

with b chosen so that $\sin^b y$ makes sense and with the usual periodicity conditions. The exact solution is a function of y only,

$$u(x,y) = u(y) = -2^b \tan^b(y/2) \int_{\pi/2}^{y/2} \frac{(\cos t)^{2b-1}}{\sin t} dt.$$

To satisfy (3.6) we take b positive (yielding, in this case, a standard M-matrix after discretization). Here we consider the case of $b = 1$ and $b = \frac{1}{3}$. We observe that (3.7) is satisfied with $\epsilon = 1$, and thus the discussion following (3.7) only guarantees a solution in C^0. In fact, with a bit of calculus and patience, we can find explicitly $u(y)$ when $b = 1$ and $b = \frac{1}{3}$, e.g., when $b = 1$,

$$u(y) = -2 \tan \frac{y}{2} \ln \left| \sin \frac{y}{2} \right|,$$

and when $b = \frac{1}{3}$,

$$u(y) = -\left(2 \tan \frac{y}{2}\right)^{1/3} \cdot \frac{1}{2} \left\{ \ln \left(1 - \cos^{2/3} \frac{y}{2}\right) \left(\cos^{4/3} \frac{y}{2} + \cos^{2/3} \frac{y}{2} + 1\right)^{-1/2} \right.$$
$$\left. - \sqrt{3} \arctan \left(\frac{2 \cos^{2/3} \frac{y}{2} + 1}{\sqrt{3}} \right) + \frac{\sqrt{3}\pi}{6} \right\}.$$

It is possible to check that, in both cases, u is a continuous function as $y \in [0, 2\pi)$ (in particular, $u(0) = u(\pi) = 0$), but the first derivative blows up at $y = 0$. When $b = \frac{1}{3}$, the first derivative of u is not even in L^2. The assumptions of Theorems 3.4 and 3.5 do not hold. Nonetheless, the upwinding scheme (3.9) works quite satisfactorily and the error behaves like $O(h)$ when $b = 1$. In Figs. 1 and 2 we show a plot of the solution $u(y)$ computed with 80 grid points for $b = 1$ and $b = \frac{1}{3}$, respectively. Of particular interest is that the computed approximation, even for $b = \frac{1}{3}$, does not crinkle (oscillate) and is qualitatively correct; that is, it reproduces the correct solution profile. However, for $b = \frac{1}{3}$ the solution has only one digit of accuracy near $y = 0$, when $N = M = 80$. In Tables 4 and 5, we have summarized the results of a few typical runs and of the extrapolation procedure for $b = 1$. The ∞-norm of the error has been computed at 10 equispaced points. For completeness, we point out that the classical upwinding scheme (see the discussion after (3.9)) performs better than the smooth upwinding for this problem, reducing the error by a factor of 2 in all runs. As already remarked, it is also computationally less expensive (much less). In any case, the good performance of the upwinding scheme is probably due to the very localized character of the singularity of the solution and to the smoothing ability of the upwinding scheme; there are similarities to the behavior of upwinding for shock calculations of hyperbolic PDEs. We plan to study these aspects, in our context, in the near future.

5. Conclusions. In this paper we have proposed a generalization of nonsingular M-matrices to block M-matrices. Such matrices arise when discretizing a linear first-order system of PDEs where each equation has the same principal part. PDEs with this structure arise in the study of invariant tori, and the setting we have studied in this paper allows us to numerically approximate a class of attracting invariant tori. The proposed discretization scheme leads to a block M-matrix under reasonable assumptions. We have proven unconditional stability and first-order convergence. Under smoothness assumptions, we have also shown an error expansion in powers of h (the mesh size), and have used this fact to achieve higher-order accuracy by Richardson's extrapolation. Finally, we have discussed some of the numerical linear algebra aspects of this and similar discretizations, and we have given numerical examples to highlight the theory and the interplay between attractivity and smoothness of the invariant tori, and accuracy of the numerical approximation. The extension to the nonlinear case will be the subject of future work.

REFERENCES

[BP] A. Berman and R. J. Plemmons (1979), *Nonnegative Matrices in the Mathematical Sciences*, Academic Press, New York.

[DLR] L. Dieci, J. Lorenz, and R. D. Russell (1991), *Numerical calculation of invariant tori*, SIAM J. Sci. Statist. Comput., 12, pp. 607–647.

[EM] L. Elsner and V. Mehrmann (1990), *Convergence of block iterative methods for linear systems arising in the numerical solution of Euler equations*, preprint 90-036, SFB 343, Universität Bielefeld, Bielefeld, Germany.

[K] E. Kamke (1950), *Differentialgleichungen lösungsmethoden und lösungen*, Akademische Verlagsgesellschaft, Leipzig.

[F] N. Fenichel (1971), *Persistence and smoothness of invariant manifolds for flows*, Indiana Univ. Math. J., 21, pp. 193–226.

[Ma] R. M. M. Mattheij (1984), *Stability of block* LU*-decompositions of matrices arising from* BVP, SIAM J. Algebraic Discrete Meth., 5, pp. 314–331.

[Mo] J. Moser (1966), *A rapidly convergent iteration method and non-linear partial differential* *equations*—I, Ann. Scuola Normale Superiore Pisa Serie 3, XX, pp. 265–315.

[P] R. J. PLEMMONS (1977), *M-matrix characterizations.* I: *Nonsingular M-matrices*, Linear Algebra Appl., 18, pp. 175–188.

[Sa] R. SACKER (1965), *A new approach to the perturbation theory of invariant surfaces*, Comm. Pure Appl. Math., 18, pp. 717–732.

[St] G. STOYAN (1979), *Monotone difference schemes for diffusion-convection problems*, ZAMM, 59, pp. 361–372.

[V] J. M. VARAH (1972), *On the solution of block-tridiagonal systems arising from certain finite-difference equations*, Math. Comp., 26, pp. 859–868.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 4, pp. 904–922, July 1992

ANALYSIS OF INITIAL TRANSIENT DELETION FOR PARALLEL STEADY-STATE SIMULATIONS*

PETER W. GLYNN† AND PHILIP HEIDELBERGER‡

Abstract. This paper investigates theoretical properties of a simple method for using parallel processors in discrete event simulations: running independent replications, in parallel, on multiple processors and averaging the results at the end of the runs. Specifically, the problem of estimating steady-state parameters from such an experiment is considered. Sampling plans are considered in which the replication lengths are given by limits on either simulated or computer time, and in which the beginning portion of each run may be deleted for the purpose of controlling initialization bias. The critical relative growth rates for the number of processors, the length of each replication, and the length of the deletion period that are required in order to produce valid confidence intervals for steady-state parameters are determined. When the replication length is determined by computer time, the straightforward estimator with deletion may not work for a large number of processors. In this case, the deletion is essentially useless due to an additional bias term that arises because the simulated time at the end of a replication is random. In this case, a new estimator can be used to remove this source of bias.

Key words. simulation, parallel processing, steady-state, initial transient

AMS(MOS) subject classifications. 68U20, 65C05, 60F05, 60K05, 60K20, 60K30

1. Introduction.

A simple way to exploit the power of parallel processing in computationally intensive discrete event simulations is to run multiple independent replications, in parallel, on the processors and to appropriately average the results at the end of the runs. At first glance, this method produces a p-fold speedup, i.e., reduction in completion time, over a sequential (one processor) simulation having the same variance where p is the number of processors. We call this the parallel replications approach.

An alternative approach is distributed simulation, in which all p processors cooperate on a single realization of the simulation. While significant speedups have been achieved using distributed simulation in specific problem domains (see, e.g., Fujimoto (1989, 1990); Goli, Heidelberger, Towsley, and Yu (1990); Lubachevsky (1989); Nicol (1988); Unger and Fujimoto (1989); Unger and Jefferson (1988); and Yu, Towsley, and Heidelberger (1989)), in our opinion, distributed simulation has not yet been demonstrated to be a robust and effective general purpose technique for dealing with the types of complex models arising in manufacturing, computer, and communications systems.

In contrast, parallel replications are conceptually and practically simple to apply and are almost universally applicable. The widespread applicability stems from the fact that a major reason why many models must run for a long time is the slow rate at which a simulation estimate's variance decreases. Essentially, parallel replications are inappropriate only when either

*Received by the editors December 20, 1989; accepted for publication (in revised form) March 11, 1991. This research was supported by the IBM Corporation under Shared University Research contract 12480042, by U.S. Army Research Office contract DAAL-03-88-K-0063, and by a grant from the Natural Sciences and Engineering Research Council of Canada.

†Department of Operations Research, Stanford University, Stanford, California 94305.

‡IBM Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598.

1. The model is such that a single replication cannot be completed on a single processor within a reasonable amount of time. This may be due to either an exceptionally large model, which, e.g., may not fit into the memory of a single processor, or, in steady-state simulations, to a model with a very slowly dissipating initial transient.

2. The variance from each replication is very small, in which case the output is nearly deterministic and having a large number of replications is merely a waste of computing resources.

A simple analytic model for comparing the statistical efficiencies of distributed simulation and parallel replications that justifies the above conclusion was developed in Heidelberger (1986).

However, there are some potentially serious statistical problems associated with the parallel replications approach, especially for a large number of processors. In the case of estimating expected values of so-called transient quantities, these problems have been studied in Heidelberger (1988) and Glynn and Heidelberger (1991a). The source of the problem can be illustrated as follows. Suppose that replications are run on each of p processors and that one sets a completion time constraint of t units of (computer) time per processor. The number of replications completed on each processor by time t is a random variable. While there are a number of different ways the output can be averaged, there is some sampling bias due to the fixed completion time t and the associated random number of replications. If t remains fixed, then as $p \to \infty$, the estimates can converge to the wrong quantity. A variety of estimators that alleviate this situation can be devised, but they have the property that some or all of the replications in progress at time t must be completed before the estimator can be formed. Thus one must pay a completion time penalty in order to obtain the correct convergence behavior. Even in the case of a single processor, some care needs to be taken in order to best handle the bias associated with stopping the simulation at time t; see Meketon and Heidelberger (1982), Glynn (1989b), and Glynn and Heidelberger (1990). In our discussion, the above results are described primarily in terms of estimating transient quantities, but they also apply to steady-state estimation in regenerative simulations (see Smith (1955) or Crane and Iglehart (1975)) since, in this case, steady-state performance measures can be expressed as a ratio of expected values of "transient" quantities. Bhavsar and Isaac (1987) discuss other properties associated with parallel replication schemes for transient quantities.

In this paper, we consider the parallel replications approach to the steady-state estimation problem in more generality. Specifically, we consider sampling schemes that delete some initial part of each run in order to reduce "initialization bias," i.e., bias due to the fact that the model cannot typically be started in its steady-state distribution (otherwise, there would be no need to simulate). We determine the critical relative growth rates for

1. the number of processors (replications),
2. the length of each replication, and
3. the length of the deletion period

that are required in order for the method of parallel replications with initial transient deletion to obey a usable central limit theorem. By a usable central limit theorem, we mean one that is centered about the unknown steady-state parameter and upon which confidence intervals for the steady-state parameter can be based.

Determining the length of each replication is a basic issue in such an experiment. We consider two approaches: the first, based on simulated time, and the second, based on computer time. The approach chosen makes a difference, in terms of both completion time and estimator bias. Deletion helps to reduce bias in the first approach;

however, the completion time is a random variable. On the other hand, when the replication length is based on computer time, the completion time is deterministic. However, an additional source of estimator bias is introduced in this case. Furthermore, this additional source of bias is not eliminated by deletion, and thus initial transient deletion is essentially useless in this case. This bias, which is of order one over the replication length, is due to the fact that the estimate from each processor takes the form of a ratio of two random variables. The denominator in this ratio is the (random) length of the replication in simulated time (minus the initialization period). When the run is based on simulated time, the denominator is deterministic and this additional bias is not introduced. A new estimator (also based on computer time) that gets around this problem is proposed and analyzed. This estimator has a deterministic stopping time and benefits from the initial transient deletion.

Some comments about the analysis techniques used in this paper are appropriate. First, we state limit theorems in a triangular array setting, i.e., we simultaneously let $p \to \infty$ and $t(p) \to \infty$ where p is the number of processors and $t(p)$ is the length of each replication. Since in practice only a fixed, finite number of processors are available, these results should be interpreted as determining (qualitatively) appropriate values for $t(p)$ in order to obtain proper convergence behavior for large values of p. Second, many of the results are established under regenerative assumptions that would appear to limit the applicability of the results. However, this is done mainly as a mathematical convenience and does not impose significant restrictions since

1. Many systems have (hidden) regenerative structure. For example, Glynn (1989a) shows that many finite state space generalized semi-Markov processes (GSMPs) are regenerative. Essentially all discrete event simulation models can be described as GSMPs.

2. The estimators involved do not make any specific use of the regenerative structure, i.e., one need not identify regeneration points and group the data by regenerative cycles. To further amplify this point, Glynn (1989b) derives bias expansions for certain integrals of regenerative processes and then proposes a bias-reducing technique based on the form of these expansions. However, to employ that method in practice requires identifying the regeneration times and estimating expectations of random variables defined over the regenerative cycles. No such requirement is made here.

Section 4 contains further discussion of these points.

In order to present a complete and self-contained description of the relevant results, certain theorems are restated from other papers, specifically, Glynn (1987, 1989b) and Glynn and Heidelberger (1991b). The rest of the paper is organized as follows. In §2 we describe the formal mathematical framework, and in §3 we consider estimators based on a fixed amount of simulated time, both with and without deletion. Section 4 treats estimators based on computer time but without deletion, while §5 treats estimators based on computer time with deletion. The proofs are contained in §6. Finally, the results are summarized in §7.

The focus of this paper is theoretical. However, in Glynn and Heidelberger (1992) we describe experiments with these estimators on simple queueing network models of computer systems. The experimental results reported in that paper confirm the theoretical results developed in this paper.

2. The framework. Suppose that our goal is to estimate the steady-state mean of $X = (X(t) : t \geq 0)$, where X is a real-valued stochastic process. We let $C = (C(t) : t \geq 0)$ be the associated *cumulative computer time* process, so that $C(t)$ represents the

random amount of computer time required to generate X over the interval $[0,t]$. Let $\Rightarrow$ denote weak convergence, or convergence in distribution (see Billingsley (1968)). We assume that (X, C) satisfies the following set of hypotheses:
(2.1)

(i) C is a nondecreasing process.

(ii) There exist (deterministic) constants α, λ^{-1} $(0 < \lambda^{-1} < \infty)$, and a 2×2 matrix G such that

$$Z_\varepsilon(\cdot) \Rightarrow GB(\cdot)$$

as $\varepsilon \downarrow 0$ in $D[0,\infty)$ (the Skorohod space of functions $x : [0,\infty) \to I\!R^2$ that are right continuous and have left limits), where B is a two-dimensional standard Brownian motion and

$$Z_\varepsilon(t) = \varepsilon^{-1}\left(\varepsilon^2 \int_0^{t/\varepsilon^2} X(s)ds - \alpha t,\ \varepsilon^2 C(t/\varepsilon^2) - \lambda^{-1}t\right).$$

Assumption (2.1) (i) is reasonable in view of the interpretation of $C(t)$. As for (2.1) (ii), we note that one consequence of the assumption is that X and C satisfy a joint central limit theorem (CLT):

$$T^{1/2}\left(\frac{1}{T}\int_0^T X(s)ds - \alpha,\ \frac{C(T)}{T} - \lambda^{-1}\right) \Rightarrow N(0,H) \tag{2.2}$$

as $T \to \infty$, where $H = GG^t$. (In this case note that $GN(0,I)$ has the same distribution as $N(0,H)$ where I is the identity matrix. In one dimension, $\sigma N(0,1)$ has the same distribution as $N(0,\sigma^2)$.) Thus, assumption (2.1) (ii) is best viewed as a strengthened version of the ordinary CLT. Because of the fact that (2.1) (ii) deals with weak convergence of stochastic processes (as opposed to ordinary random variables), it is typically termed a functional central limit theorem (FCLT) hypothesis. See Billingsley (1968) for further discussion of FCLTs.

It turns out that a great variety of stochastic processes exhibit FCLT behavior. For example, suppose that $C(\cdot)$ has a positive derivative so that it can be represented as

$$C(t) = \int_0^t \chi(s)ds, \quad \text{where} \quad \chi(t) > 0 \quad \text{a.s. for} \quad t \geq 0. \tag{2.3}$$

Under (2.3), $\chi(s)$ is the rate at which computer time is consumed at simulated time s. If the process $((X(t), \chi(t)) : t \geq 0)$ is regenerative and satisfies certain moment conditions, then (2.1) (ii) is known to be valid (see Glynn and Whitt (1987)). Regenerative structure is present in many of the stochastic models that are commonly simulated; see, for example, Glynn (1982). In addition, (2.1) (ii) holds if $((X(t), \chi(t)) : t \geq 0)$ is a martingale process or mixing process satisfying certain regularity hypotheses (see Ethier and Kurtz (1986)) or if it is an associated sequence (see Newman and Wright (1981)). Also, (2.1) (ii) is known to be valid for a large class of Markov processes (see Maigret (1978) and Nummelin (1984)). Because of the broad validity of (2.1) (ii), we view this condition as a mild regularity hypothesis that is satisfied by virtually all "real world" simulations.

One consequence of the joint CLT (2.2) is that

$$\frac{1}{T}\int_0^T X(s)ds \Rightarrow \alpha, \tag{2.4}$$

$$\frac{1}{T}C(T) \Rightarrow \lambda^{-1}, \tag{2.5}$$

as $T \to \infty$. The law of large numbers (2.4) states that the process X "settles down," on average, to the constant α; the parameter α is known as the *steady-state mean* of X. The goal of the steady-state simulation algorithms to be described in this paper is the efficient estimation of α. Also, (2.5) states that λ^{-1} may be interpreted as the long-run rate at which computer time is expended per unit of simulated time. Equivalently, λ is the long-run rate at which simulated time is generated per unit of computer time.

Two different (reasonable) strategies for estimating α can be employed by the simulation analyst. The first possibility is to generate a fixed amount of simulated time on each of the p processors, and to obtain an estimator for α by averaging the resulting observations over each of the p processors. This class of estimators, together with related initial transient deletion strategies, is discussed in §3. The second approach is to fix the amount of computer time available on each of the p processors, and to obtain an estimator for α by averaging over the random amount of simulated time generated on each of the p processors within the computational budget constraint. For $c \geq 0$, let $T_i(c) = \sup\{t \geq 0 : C_i(t) \leq c\}$ be the inverse process to $C_i(\cdot)$. The random variable (r.v.) $T_i(c)$ can be interpreted as the amount of simulated time that is generated on processor i in the first c units of computer time; we refer to $T_i(c)$ as the *cumulative simulated time* process associated with processor i. Hence, the second estimator involves averaging the process X_i over the random interval $[0, T_i(c)]$ and all p processors. Section 4 is devoted to studying this class of estimators when no initial transient deletion is applied, whereas §5 considers these estimators when initial transient deletion is applied.

3. Steady-state estimation using simulated time. Suppose that the process X is simulated up to (deterministic) time t on each of the p available processors. If the first $\beta(t)$ simulated time units are deleted from the initial segment of each copy X_i, we obtain the estimator

$$\alpha_s(p,t) = \frac{1}{p}\sum_{i=1}^{p} \frac{1}{t-\beta(t)} \int_{\beta(t)}^{t} X_i(s)ds$$

(the subscript s stands for simulated time).

In our limit theorems, we shall be interested in determining how large the time horizon t needs to be for the parallel estimation algorithm to work efficiently. We shall therefore view t as a deterministic function $t = t(p)$ of the number of processors. Then $\beta(t) = \beta(t(p))$ is also a deterministic function of p. The limit theorems will include growth conditions on $t(p)$ and $\beta(t(p))$. To simplify the notation, we shall write $\alpha_s(p)$ and $\beta_s(p)$ as shorthand for $\alpha_s(p, t(p))$ and $\beta(t(p))$, respectively.

A consequence of (2.2) (and hence (2.1)) is that

$$\begin{aligned} T\left(\frac{1}{T}\int_0^T X(s)ds - \alpha\right)^2 &\Rightarrow \sigma_1^2 N(0,1)^2, \\ T\left(\frac{C(T)}{T} - \lambda^{-1}\right)^2 &\Rightarrow \sigma_2^2 N(0,1)^2, \end{aligned} \tag{3.1}$$

as $T \to \infty$, where $\sigma_i^2 = H_{ii}$ $(i = 1, 2)$. In order to carry out certain arguments, we will need to assume that the expectation operator can be passed through (3.1):

$$TE\left(\frac{1}{T}\int_0^T X(s)ds - \alpha\right)^2 \to \sigma_1^2, \tag{3.2}$$
$$TE\left(\frac{C(T)}{T} - \lambda^{-1}\right)^2 \to \sigma_2^2,$$

as $T \to \infty$. A variety of stochastic processes obey (3.2), including regenerative processes (Smith (1955)), mixing and martingale process sequences (Ethier and Kurtz (1986)), and associated sequences (Newman and Wright (1981)) under appropriate moment conditions. Condition (3.2) is equivalent to asserting that $\{T(T^{-1}\int_0^T X(s)ds - \alpha)^2 : T > t_0\}$ and $\{T(C(T)/T - \lambda^{-1})^2 : T > t_0\}$ are uniformly integrable for some (finite) t_0 (see Chung (1974, p. 97)).

We will also need an assumption that controls the extent to which the initial transient biases the observations of the simulation. To be precise, let $b(t) = EX(t) - \alpha$. We assume that

$$\int_0^\infty |b(s)|ds < \infty. \tag{3.3}$$

Assumption (3.2) holds whenever $EX(t) \to \alpha$ exponentially fast (i.e., there exist constants $A, \lambda > 0$ such that $|b(t)| \leq Ae^{-\lambda t}$). This exponential rate of convergence is typical of most "real world" simulations. For example, finite-state irreducible aperiodic discrete-time Markov chains and finite-state irreducible continuous-time Markov chains both exhibit exponential convergence to their steady-state values (see Karlin and Taylor (1975)). Also, Nummelin and Tuominen (1982) prove exponential convergence rate results in a general state space Markov chain setting.

Our first theorem considers the case in which no initial bias deletion is performed, so that $\beta(t) \equiv 0$. Let $b = \int_0^\infty b(s)ds$.

THEOREM 1. *Assume* (2.1), (3.2), *and* (3.3) *are in force and that* $\beta(t) \equiv 0$. *Then,*

(i) *if* $p/t(p) \to \infty$, $t(p) \to \infty$, *and* $b \neq 0$, *then* $\sqrt{pt(p)}\ |\alpha_s(p) - \alpha| \Rightarrow \infty$ *as* $p \to \infty$,

(ii) *if* $p/t(p) \to m$ $(0 < m < \infty)$, *then* $\sqrt{pt(p)}\ (\alpha_s(p) - \alpha) \Rightarrow \sigma_1 N(0,1) + b\sqrt{m}$ *as* $p \to \infty$,

(iii) *if* $p/t(p) \to 0$, *then* $\sqrt{pt(p)}\ (\alpha_s(p) - \alpha) \Rightarrow \sigma_1 N(0,1)$ *as* $p \to \infty$.

The proof of this result appears in Glynn (1987). Note that if no truncation is used, $\alpha_s(1,t) = t^{-1}\int_0^t X_1(s)ds$. Then, (2.2) asserts that

$$\alpha_s(1,t) \overset{\mathcal{D}}{\approx} \frac{\sigma_1}{\sqrt{t}} N(0,1) \tag{3.4}$$

for large t ($\overset{\mathcal{D}}{\approx}$ denotes "approximately equal in distribution"). On the other hand, Theorem 1 (iii) states that

$$\alpha_s(p, t(p)) \overset{\mathcal{D}}{\approx} \frac{\sigma_1}{\sqrt{pt(p)}} N(0,1) \tag{3.5}$$

under the conditions stated there. Comparing (3.5) to (3.4), we see that (3.5) implies a p-fold speedup in the algorithm over that achieved with a single processor, i.e.,

$$\alpha_s(1, pt(p)) \overset{\mathcal{D}}{\approx} \alpha_s(p, t(p)).$$

Of course, a p-fold speedup is the best possible rate increase that we can expect in a parallel processing environment. Hence, Theorem 1 can be interpreted as stating that the time horizon $t = t(p)$ to be simulated on each of the p processors should satisfy $t >> p$, in order that the parallel algorithm achieve optimal efficiency.

Our next theorem considers the situation in which initial transient deletion is implemented. We note that $\beta_s(p)/t(p)$ is the fraction of the total simulated time that is deleted. The strategies that are considered here delete an asymptotically negligible fraction of the total observation set that is simulated.

THEOREM 2. *Assume* (2.1) *and* (3.2) *are in force and that* $b(t) \to 0$ *exponentially fast. Suppose* $\beta_s(p)/t(p) \to 0$ *as* $p \to \infty$. *If*

(i) $p/t(p) \to \infty$ *and* $\beta_s(p)/\log p \to \infty$, *or*

(ii) $p/t(p) \to m$ $(0 < m < \infty)$ *and* $\beta_s(p) \to \infty$, *or*

(iii) $p/t(p) \to 0$

as $p \to \infty$, *then*

$$\sqrt{pt(p)}(\alpha_s(p) - \alpha) \Rightarrow \sigma_1 N(0,1)$$

as $p \to \infty$.

For a proof, see Glynn and Heidelberger (1991b). Theorem 2 shows that with a modest amount of initial transient deletion, one can reduce the length of the time horizon significantly without affecting the p-fold speedup factor. In particular, if $t(p) = p^r$ $(r > 0)$, the p-fold speedup is retained so long as $\beta_s(p) = p^{\varepsilon}$ $(0 < \varepsilon < r)$. Thus, when initial transient deletion is suitably implemented, p-fold increases in efficiency ensue from time horizons that grow essentially arbitrarily slowly in p. A result similar to Theorem 2 can also be derived when the bias function $b(\cdot)$ decays polynomially fast (i.e., there exists $A, r > 0$, such that $|b(t)| \le At^{-r}$ for $t \ge 0$); see Glynn and Heidelberger (1991b) for further details.

In our next theorem, we consider the case in which the fraction of the total simulation time that is deleted is fixed, so that the fraction deleted is no longer negligible. As might be expected, this rule, although reasonable from an implementation viewpoint, has a cost in terms of (greater) asymptotic variability.

THEOREM 3. *Assume* (2.1) *and* (3.2) *are in force and that* $b(t) \to 0$ *exponentially fast. If* $\beta_s(p) = \beta t(p)$ $(0 < \beta < 1)$ *and* $t(p) = p^r$ $(r > 0)$, *then*

$$\sqrt{pt(p)}(\alpha_s(p) - \alpha) \Rightarrow (1 - \beta)^{-(1/2)}\sigma_1 N(0,1)$$

as $p \to \infty$.

For a proof, see Glynn and Heidelberger (1991b). Theorem 3, like Theorem 2, asserts that the time horizon over which we achieve a p-fold increase in efficiency is broadened considerably by using initial bias deletion. However, because of the fact that one deletes a fixed fraction of the total observation set, one pays a cost in the sense that the asymptotic variance is inflated by a factor of $(1 - \beta)^{-1}$. On the other hand, for $\beta = 0.1$, the increase in the variance is only about 11 percent. Thus, the statistical cost incurred in using such a procedure is quite modest.

We conclude this section with a discussion of the computational cost associated with using estimators based on simulated time. In particular, assuming that the machine is (temporarily) dedicated to the estimation of α, the question of the algorithm's completion time is of significant importance. Recalling that $C_i(t)$ is the time at which the ith processor completes the simulation of X over $[0,t]$, the completion time for a simulated time horizon of t is given by

$$C(p,t) = \max_{1\leq i\leq p} C_i(t).$$

An additional relevant performance characteristic is the total idle time cumulated over all p processors, assuming that each processor must remain idle until all p processors have completed their assigned tasks. The idle time quantity is defined as

$$I(p,t) = \sum_{i=1}^{p} [C(p,t) - C_i(t)].$$

Set $C_s(p) = C(p,t(p))$ and $I_s(p) = I(p,t(p))$. Our final theorem of this section examines the behavior of $C_s(p)$ and $I_s(p)$ when the number of processors p is large. We will need one further assumption. Note that one consequence of (2.2) is that if $\sigma_2^2 > 0$, then

$$F_t(x) \rightarrow \Phi(x) \tag{3.6}$$

as $t \rightarrow \infty$, where $F_t(x) = P\{t^{-(1/2)}(C(t)-\lambda^{-1}t)/\sigma_2 \leq x\}$ and $\Phi(x) = P\{N(0,1) \leq x\}$. We wish to strengthen (3.6) to

$$\sup_{-\infty<x<\infty} |F_t(x) - \Phi(x)| = 0(t^{-(1/2)}) \tag{3.7}$$

as $t \rightarrow \infty$. This condition is known, in the probability literature, as a Berry–Esséen condition. The convergence result (3.7) is usually valid for stochastic processes satisfying (2.1). For example, regenerative processes (Bolthausen (1980)) and general state space Markov chains (Bolthausen (1982)) are known to satisfy (3.7) under suitable regularity hypotheses.

THEOREM 4. *Assume that* (2.1) *is in force with* $\sigma_2^2 > 0$ *and that* (3.2) *and* (3.7) *hold. If* $t(p)/p^2 \rightarrow \infty$, *then*

(i)
$$\frac{C_s(p) - \lambda^{-1}t(p)}{\sigma_2\sqrt{2t(p)\log p}} \Rightarrow 1,$$

(ii)
$$\frac{I_s(p)}{\sigma_2 p\sqrt{2t(p)\log p}} \Rightarrow 1$$

as $p \rightarrow \infty$.

According to Theorem 4, if $t(p) >> p$, then we may approximate $C_s(p)$ and $I_s(p)$ as

$$\begin{aligned} C_s(p) &\overset{\mathcal{D}}{\approx} \lambda^{-1}t(p) + \sigma_2\sqrt{2t(p)\log p}, \\ I_s(p) &\overset{\mathcal{D}}{\approx} \sigma_2 p\sqrt{2t(p)\log p}. \end{aligned} \tag{3.8}$$

Since λ^{-1} and σ_2^2 are typically unknown, the magnitude of the completion time $C_s(p)$ is, to some extent, a priori unpredictable. This is clearly undesirable. Furthermore, (3.8) shows that the total idle time can potentially be quite large. As a consequence, the machine may be significantly underutilized with estimators of the type considered in this section (assuming that processors are not freed until time $C_s(p)$). For these reasons, the remainder of this paper explores estimators in which the completion time is fixed.

4. Steady-state estimation using computer time: No initial transient deletion. In this section, we suppose that each processor simulates X for a fixed (deterministic) amount of computer time c. At the completion time c, processor i will have generated X_i up to time $T_i(c)$. Given that no initial transient deletion is employed, two different estimators immediately suggest themselves:

$$\alpha_1(p,c) = \frac{1}{p}\sum_{i=1}^{p} \frac{1}{T_i(c)} \int_0^{T_i(c)} X_i(s)ds,$$

$$\alpha_2(p,c) = \frac{\sum_{i=1}^{p} \int_0^{T_i(c)} X_i(s)ds}{\sum_{i=1}^{p} T_i(c)}.$$

Note that $\alpha_2(p,c)$ bears a strong resemblance to the usual ratio estimator for steady-state parameters in regenerative simulations (identify $T_i(c)$ as the length of the ith regenerative cycle).

As in §3, we shall be interested in limit theorems that describe how large the (computer) time horizon c should be, relative to the number of processors, in order that a p-fold speedup ensues. We shall therefore take c as a deterministic function $c = c(p)$ of the number of processors p. The limit theorems will then provide appropriate growth conditions on $c(p)$. As a shorthand notation, we will write $\alpha_i(p) = \alpha_i(p, c(p))(i = 1, 2)$.

Our first task is to understand the behavior of the sample means obtained from the individual processors.

THEOREM 5. *Assume that* (2.1) *and* (2.3) *hold. Then,*

(i) $\tilde{Z}_\varepsilon(\cdot) \Rightarrow \tilde{G}B(\cdot)$ *as* $\varepsilon \downarrow 0$ *in* $D[0,\infty)$, *where*

$$\tilde{Z}_\varepsilon(c) = \varepsilon^{-1}\left(\varepsilon^2 \int_0^{T(c/\varepsilon^2)} X(s)ds - \alpha\varepsilon^2 T(c/\varepsilon^2),\ \varepsilon^2 T(c/\varepsilon^2) - \lambda c\right),$$

$$\tilde{G}\tilde{G}^t = \tilde{H} \quad \textit{and} \quad \tilde{H}_{11} = \lambda H_{11}, \quad \tilde{H}_{12} = \tilde{H}_{21} = -\lambda^2 H_{12}, \quad \tilde{H}_{22} = \lambda^3 H_{22}.$$

(ii)

$$c^{1/2}\left(\frac{\int_0^{T(c)} X(s)ds}{T(c)} - \alpha,\ \frac{T(c)}{c} - \lambda\right) \Rightarrow N(0, \underset{\sim}{H})$$

as $c \to \infty$, *where* $\underset{\sim}{H}_{11} = \lambda^{-1}H_{11}$, $\underset{\sim}{H}_{21} = -\lambda H_{12}$, *and* $\underset{\sim}{H}_{22} = \lambda^3 H_{22}$.

As in §3, we shall require that the cumulative processes defined on the time scale of computer time be appropriately uniformly integrable. Specifically, we shall need to assume that

$$cE\left(\frac{1}{T(c)}\int_0^{T(c)} X(s)ds - \alpha\right)^2 \to \lambda^{-1}\sigma_1^2,$$

$$c^{-1}E\left(\int_0^{T(c)} [X(s) - \alpha]ds\right)^2 \to \lambda\sigma_1^2, \tag{4.1}$$

$$cE\left(\frac{T(c)}{c} - \lambda\right)^2 \to \lambda^3\sigma_2^2,$$

as $c \to \infty$ (recall that $H_{ii} = \sigma_i^2$).

Theorem 5 asserts that the central limit behavior of X is preserved, in a qualitative sense, when the time scale is changed from that of simulated time to computer time. Since initial transient bias plays a critical role in the study of the estimators considered in this paper, it is incumbent upon us to also consider the extent to which the bias characteristics of X are altered by a change in the time scale.

We start by noting that if $E\int_0^{T(c)} |X(s)|ds < \infty$ and if (2.3) holds where $\chi(t) > 0$ almost surely for $t \geq 0$, a change-of-variables formula can be applied to obtain

$$\int_0^{T(c)} X(s)ds = \int_0^c Y(s)ds \quad \text{a.s.},$$

where $Y(s) = X(T(s))/\chi(T(s))$. The process $Y = (Y(t) : t \geq 0)$ is, in some sense, the original process X with its time scale transformed from simulated time to computer time. It turns out that the transformation that sends X into Y preserves any regenerative structure that may be present on the time scale of simulated time.

PROPOSITION 1. *Assume that* (2.3) *holds. If* $((X(t), \chi(t)) : t \geq 0)$ *is regenerative with respect to the random times* $(\tau_n : n \geq 1)$, *then* $(Y(t) : t \geq 0)$ *is regenerative with respect to the sequence* $(\eta_n : n \geq 1)$, *where* $\eta_n = C(\tau_n)$.

As indicated earlier in this paper, regenerative structure is to be found in many of the stochastic processes X that are simulated in practice. Given that X is regenerative, it is reasonable to further assume that the pair (X, χ) is also regenerative. Thus, we view regenerative hypotheses on the pair (X, χ) as a fairly mild restriction on the class of processes to be analyzed here. It is worth noting that while the proof techniques that appear in the remainder of this paper demand, to some extent, regenerative structure, our estimation strategies will be completely independent of the nature of the regenerations. As a consequence, our estimators will not require explicit identification of the regeneration points during the course of the simulation. The regenerative hypotheses will appear solely as mathematical regularity conditions and not as a vital component of the methodology itself.

The following hypotheses help to simplify certain proofs; they are not necessary to the development, however, and can be relaxed significantly, at the cost of greater mathematical complexity. Because we do not require that the simulation be started in the regeneration state, we need to make a distinction between the first (atypical) regeneration time τ_1 and the subsequent regeneration times. Let $\tilde{\eta} = \eta_2 - \eta_1$ and $\tilde{\tau} = \tau_2 - \tau_1$.

(4.2) There exists a constant $0 < x_1 < \infty$ such that $\chi(s) \geq x_1$ a.s.,

$$\begin{gathered} E[\eta_2] < \infty, \quad E[\tau_2] < \infty, \quad E[\eta_2\tau_2] < \infty, \\ E\left[\int_0^{\tau_2} |\chi(s)|ds\right] < \infty \quad \text{and} \quad E\left[\eta_2 \int_0^{\tau_2} |X(s)|ds\right] < \infty, \end{gathered} \tag{4.3}$$

Both η_1 and $\tilde{\eta}$ have probability density functions.

Assumption (4.2) merely states that there is a minimum rate at which computer time is consumed.

We are now ready to describe the behavior of the bias of the estimators $\alpha_1(p)$ and $\alpha_2(p)$.

THEOREM 6. *Assume that* (2.1), (2.3), (4.1), (4.2), *and* (4.3) *hold. If, in addition,* $c(p) \to \infty$ *as* $p \to \infty$, *then*

$$E\alpha_i(p) = \alpha + \frac{b_i}{c(p)} + o\left(\frac{1}{c(p)}\right)$$

as $p \to \infty$ $(i = 1, 2)$, *where* $b_1 = a + H_{12}$, $b_2 = a$, *and*

$$a = E\left[\int_0^{\tau_1} (X(s) - \alpha)ds\right] / \lambda - E\left[\int_0^{\tilde{\tau}} \int_0^s \chi(\tau_1 + u)du(X(\tau_1 + s) - \alpha)ds\right] / E\tilde{\tau}.$$

Roughly speaking, the bias in $\alpha_1(p)$ is due to two factors. First, the estimator $\alpha_1(p)$ is a ratio estimator, i.e., it is expressible as the ratio of two r.v.'s. The nonlinearity inherent in ratio estimators gives rise to the term $H_{12}/c(p)$. In addition, the expectation of the centered numerator r.v. $\int_0^{T(c)} [X(s) - \alpha]ds$ is nonzero, and this gives rise to the additional bias term $a/c(p)$. As for the bias of $\alpha_2(p)$, the ratio estimator bias is reduced by a factor of p because of the p-fold increase in the sample size of the numerator and denominator r.v.'s that appear in $\alpha_2(p)$.

Suppose that $\chi(s) = \lambda^{-1}$ almost surely. In this case, computer time is proportional to simulated time, i.e., $C(T) = T/\lambda$. Thus, $C(T)/T$ is deterministic, in which case $\sigma_2^2 = 0$ and $H_{12} = 0$. Therefore, $b_1 = b_2$ and the additional bias due to the randomness in $C(T)$ disappears.

The following theorem provides the analogue of Theorem 1 in the current setting, in which estimation occurs on the time scale of computer time rather than simulated time.

THEOREM 7. *Under the same hypotheses as in Theorem* 6, *the following results hold:*

(i) *If* $p/c(p) \to \infty$, $c(p) \to \infty$, *and* $b_i \neq 0$, *then* $\sqrt{pc(p)}|\alpha_i(p) - \alpha| \Rightarrow \infty$ *as* $p \to \infty$ $(i = 1, 2)$.

(ii) *If* $p/c(p) \to m$ $(0 < m < \infty)$, *then* $\sqrt{pc(p)}(\alpha_i(p) - \alpha) \Rightarrow \lambda^{-(1/2)}\sigma N(0, 1) + b_i m^{(1/2)}$ *as* $p \to \infty$ $(i = 1, 2)$.

(iii) *If* $p/c(p) \to 0$ *as* $p \to \infty$, *then* $\sqrt{pc(p)}(\alpha_i(p) - \alpha) \Rightarrow \lambda^{-(1/2)}\sigma N(0, 1)$ *as* $p \to \infty$ $(i = 1, 2)$.

According to Theorem 7, we typically need to choose $c(p) >> p$ when no initial bias deletion is used in order to achieve the desired p-fold increase in efficiency.

5. Steady-state estimation using computer time: Initial transient deletion. This section is devoted to analyzing modified versions of the estimators introduced in §4. Specifically, we shall modify the two estimators so that an initial segment is deleted from the observations generated by each processor. To precisely define the modified estimators, we let $\kappa(c) \leq c$ be a deterministic deletion point specified on the time scale of computer time. In other words, all observations generated in the first $\kappa(c)$ units of computer time are discarded. Of course, this just amounts to throwing away the initial segment $(X_i(t) : 0 \leq t \leq \gamma_i(c))$, where $\gamma_i(c) = T_i(\kappa(c))$. The modified versions of the two estimators studied in §4 are then defined as

$$\alpha_3(p, c) = \frac{1}{p} \sum_{i=1}^{p} \frac{1}{T_i(c) - \gamma_i(c)} \int_{\gamma_i(c)}^{T_i(c)} X_i(s)ds,$$

$$\alpha_4(p, c) = \frac{\sum_{i=1}^{p} \int_{\gamma_i(c)}^{T_i(c)} X_i(s)ds}{\sum_{i=1}^{p} [T_i(c) - \gamma_i(c)]}.$$

Once again, to simplify notation, we set $\alpha_i(p) = \alpha_i(p, c(p))$ $(i = 3, 4)$ and $\kappa_c(p) = \kappa(c(p))$. We note that the assumption $\kappa_c(p)/c(p) \rightarrow 0$ as $p \rightarrow \infty$ is just a statement that the fraction of computer time devoted to observations that will eventually be deleted tends to zero in the limit.

Just as Theorem 2 required exponentially decreasing bias (on the simulated time scale), we will need some additional hypotheses to generate exponentially decreasing bias on the computer time scale.

$$E \exp t\eta_2 < \infty \quad \text{for some } t > 0, \tag{5.1}$$

$$X \text{ is a bounded process, i.e., } \sup\{|X(t,w)| : t \geq 0,\ w \epsilon \Omega\} \overset{\Delta}{=} \|X\| < \infty. \tag{5.2}$$

Note that (5.1) is true if $E \exp t_2\tau_2 < \infty$ for some $t_2 > 0$ and $\chi(s) \leq M < \infty$ almost surely for all $s \geq 0$ (since $\eta_2 \leq M\tau_2$ in this case). Thus if X is a bounded process, exponential tail behavior of η_2 is inherited from that of τ_2.

THEOREM 8. *Suppose that* $\kappa_c(p)/c(p) \rightarrow 0$ *as* $p \rightarrow \infty$. *If, in addition to the hypotheses of Theorem* 6, (5.1) *and* (5.2) *hold, then*

(i) *if* $p/c(p) \rightarrow \infty$, $c(p) \rightarrow \infty$, *and* $H_{12} \neq 0$, *then* $\sqrt{pc(p)}|\alpha_3(p) - \alpha| \Rightarrow \infty$ *as* $p \rightarrow \infty$;

(ii) *if* $p/c(p) \rightarrow m$ $(0 < m < \infty)$, *then* $\sqrt{pc(p)}(\alpha_3(p) - \alpha) \Rightarrow \lambda^{-(1/2)}\sigma N(0,1) + H_{12}m^{(1/2)}$ *as* $p \rightarrow \infty$;

(iii) *if* $p/c(p) \rightarrow 0$, *then* $\sqrt{pc(p)}(\alpha_3(p) - \alpha) \Rightarrow \lambda^{-(1/2)}\sigma N(0,1)$ *as* $p \rightarrow \infty$.

Theorem 8 shows that even in the presence of initial bias deletion, the estimator $\alpha_3(p)$ does not effectively achieve a p-fold increase in efficiency unless the computer time $c(p)$ assigned to each processor is large (i.e., $c(p) >> p$). Thus, the estimator $\alpha_3(p)$ has essentially the same behavior as $\alpha_1(p)$ (see Theorem 7). In other words, $\alpha_3(p)$ does not benefit from the initial bias deletion present in the estimator. The basic problem is that deleting the initial segment from each processor's observations does not deal with the bias introduced by the nonlinearity of the ratio estimator obtained from each processor. This is reflected in Theorem 8 through the fact that the bias term that appears in part (ii) depends only on H_{12} and not also on the constant a that appears in Theorem 6.

Our next theorem describes the behavior of $\alpha_4(p)$.

THEOREM 9. *Suppose that* $\kappa_c(p)/c(p) \rightarrow 0$ *as* $p \rightarrow \infty$ *and that the same hypotheses as in Theorem* 8 *hold. If*

(i) $p/c(p) \rightarrow \infty$ *and* $\kappa_c(p)/\log p \rightarrow \infty$, *or*

(ii) $p/c(p) \rightarrow m$ $(0 < m < \infty)$ *and* $\kappa_c(p) \rightarrow \infty$, *or*

(iii) $p/c(p) \rightarrow 0$,

as $p \rightarrow \infty$, *then*

$$\sqrt{pc(p)}(\alpha_4(p) - \alpha) \Rightarrow \lambda^{-(1/2)}\sigma N(0,1)$$

as $p \rightarrow \infty$.

According to Theorem 9, initial bias deletion has a significant positive impact on the estimator $\alpha_4(p)$. With a modest amount of initial bias deletion from the observations associated with each processor, a p-fold speedup in efficiency can be obtained with computer time horizons that are significantly shorter than those associated with no initial transient deletion. Since the estimator $\alpha_4(p)$ incurs none of the completion time and idle time costs associated with the "simulated time" estimators of §3, this result suggests that the estimator $\alpha_4(p)$ is preferable to all the other estimators considered in this paper.

Our final theorem describes the behavior of $\alpha_4(p)$ when $\kappa(c) = \kappa c$ for $0 < \kappa < 1$, $c \geq 0$, i.e., when a proportion κ of all the observations are deleted before forming the estimator $\alpha_4(p)$.

THEOREM 10. *Assume the same hypotheses as in Theorem* 6. *If* $\kappa_c(p) = \kappa c(p)$ $(0 < \kappa < 1)$ *and* $c(p) = p^r$ $(r < 0)$, *then*

$$\sqrt{pc(p)}(\alpha_4(p) - \alpha) \Rightarrow (1-\kappa)^{-1/2}\lambda^{-1/2}\sigma N(0,1)$$

as $p \to \infty$.

The proof of Theorem 10 is similar to that of Theorem 9 and is therefore omitted. As in Theorem 3, deleting a positive fraction of all the observations leads to an asymptotic increase in the variability of the estimator of $(1-\kappa)^{-1}$. Of course, as pointed out in §3, this increase is quite modest if we choose κ small (say $\kappa = 0.1$).

6. Proofs.

Proof of Theorem 4. Let

$$\bar{\Phi}(x) = 1 - \Phi(x) \quad \text{and} \quad W_i(p) = (C_i(t(p)) - \lambda^{-1}t(p))/t(p)^{(1/2)}\sigma_2.$$

Note that the independence of the $W_i(p)$'s yields

$$\begin{aligned} P&\left\{C_s(p) \leq x\sigma_2\sqrt{2t(p)\log p} + \lambda^{-1}t(p)\right\} \\ &= P\left\{W_i(p) \leq x\sqrt{2\log p},\ 1 \leq i \leq p\right\} \\ &= F_{t(p)}(x\sqrt{2\log p})^p \\ &= \left(1 - \bar{\Phi}(x\sqrt{2\log p}) + O(1/\sqrt{t(p)})\right)^p. \end{aligned}$$

The quantity $\bar{\Phi}(x\sqrt{2\log p})$ may be estimated by using Lemma 2 of Feller (1968, p. 179). Noting that $O(1/\sqrt{t(p)}) = o(1/p)$, it is then straightforward to show that

$$P\{C_s(p) \leq x\sigma_2\sqrt{2t(p)\log p} + \lambda^{-1}t(p)\} \to \begin{cases} 0, & x < 1, \\ 1, & x > 1, \end{cases}$$

proving part (i). For part (ii), note that it is sufficient to prove that

$$\sum_{i=1}^{p} W_i(p)/p\sqrt{\log p} \Rightarrow 0$$

as $p \to \infty$. Fix $\varepsilon > 0$ and use Markov's inequality to obtain

$$P\left\{\left|\frac{1}{p}\sum_{i=1}^{p} W_i(p)\right| > \varepsilon\sqrt{\log p}\right\} \leq \frac{E|W_1(p)|}{\varepsilon\sqrt{\log p}}.$$

Assumption (3.2) states that $\{W(p)^2 : p > p_0\}$ is uniformly integrable for some (finite) p_0, from which one may conclude that $E|W_1(p)|$ is bounded in p. Hence, the right-hand side converges to zero as $p \to \infty$, proving (ii).

Proof of Theorem 5. We first note that (2.1) implies that

$$C_\varepsilon \Rightarrow \lambda^{-1}e$$

as $\varepsilon \downarrow 0$, where $C_\varepsilon(t) = \varepsilon^2 C(t/\varepsilon^2)$ and $e(t) = t$. Since χ is positive, it is evident that $C_\varepsilon^{-1}(\cdot)$ is continuous and satisfies $C_\varepsilon^{-1} \circ C_\varepsilon = e$. We may then apply Theorem 3.3 of Whitt (1980) to conclude that $C_\varepsilon^{-1} \Rightarrow \lambda e$ as $\varepsilon \downarrow 0$. But $C_\varepsilon^{-1} = T_\varepsilon$, where $T_\varepsilon(c) = \varepsilon^2 T(c/\varepsilon^2)$. Since composition is continuous as a mapping on the space of continuous functions (see Billingsley (1968, §17)), we obtain $Z_\varepsilon \circ T_\varepsilon \Rightarrow GB(\lambda e)$ as $\varepsilon \downarrow 0$. As a consequence, the continuous mapping principle implies that $h(Z_\varepsilon \circ T_\varepsilon) \Rightarrow h(GB(\lambda e))$, where $h(x, y) = (x, \lambda y)$; this proves part (i).

For part (ii), we let $\hat{X}(s) = X(s) - \alpha$. We note that part (i) implies that

$$c^{-(1/2)}\left(\int_0^{T(c)} \hat{X}(s)ds,\ T(c) - \lambda c\right) \Rightarrow N(0, \tilde{H})$$

as $c \to \infty$. Therefore, the continuous mapping principle shows that

$$\begin{aligned}\lambda^{-1}c^{-(1/2)} &\int_0^{T(c)} \hat{X}(s)ds - c^{(1/2)} \int_0^{T(c)} \hat{X}(s)ds/T(c) \\ &= \left(\lambda - \frac{c}{T(c)}\right) c^{-(1/2)} \int_0^{T(c)} \hat{X}(s)ds \Rightarrow 0\end{aligned}$$

as $c \to \infty$. The proof of part (ii) is complete, if we note that

$$c^{-(1/2)}\left(\lambda^{-1}\int_0^{T(c)} \hat{X}(s)ds,\ T(c) - \lambda c\right) \Rightarrow N(0, \underset{\sim}{H})$$

as $c \to \infty$, and apply a converging-together argument.

Proof of Proposition 1. We first note that $\sigma(Y(t) : t \leq \eta_n) \subseteq \sigma((X(t), \chi(t)) : t \leq \tau_n)$, so it suffices to show that

$$P\{Y(\eta_n + \cdot)\varepsilon A | X(t),\ \chi(t) : t \leq \tau_n\} = P\{Y(\eta_1 + \cdot)\varepsilon A\} \tag{6.1}$$

for arbitrary (measurable) sets A. Now, $Y(\eta_n + t) = V(T(\eta_n + t))$, where $V(t) = X(t)/\chi(t)$, and $T(\eta_n + t) = T(\eta_n) + \Delta T_n(t) = \tau_n + \Delta T_n(t)$, where $\Delta T_n(t) = T(\eta_n + t) - T(\eta_n)$. We now observe that $\Delta T_n(\cdot)$ is the inverse to the process $\Delta C_n(t) = C(\tau_n + t) - C(\tau_n)$, in the sense that $\Delta C_n \circ \Delta T_n = e$. As a consequence, ΔT_n can be represented as a function $g(\tilde{V}_n)$, where $\tilde{V}_n(t) = (V(\tau_n + t),\ \chi(\tau_n + t))$. Hence, $Y(\eta_n + t) = k(\tilde{V}_n(t))$, where $k(\tilde{x}) = x_1((g \circ \tilde{x})(t))$ and $\tilde{x} = (x_1, x_2)$. Thus, $Y(\eta_n + \cdot)$ is a functional of the "shifted path" $\tilde{V}_n$ and (7.1) is trivially satisfied.

Proof of Theorem 6. The proof for $\alpha_1(p)$ can be found in Glynn (1989b) (the assumptions 4.3 are heavily used there). For part (ii), we let $S_i(c) = \int_0^{T_i(c)}[X_i(s) - \alpha]ds$. Then,

$$\alpha_2(p) - \alpha = \frac{\sum_{i=1}^p S_i(c(p))}{\sum_{i=1}^p T_i(c(p))}.$$

We note that for $\varepsilon > 0$,

$$P\left\{\left|\sum_{i=1}^p \frac{T_i(c(p))}{pc(p)} - \lambda\right| > \varepsilon\right\} \leq \frac{1}{pc(p)\varepsilon^2} \text{ var}\left[\frac{T_i(c(p)) - \lambda c(p)}{\sqrt{c(p)}}\right] = O\left(\frac{1}{pc(p)}\right) \to 0 \tag{6.2}$$

as $p \to \infty$, where the uniform integrability condition (4.1) was used to guarantee that the variance term was bounded in p. Hence, $\sum_{i=1}^{p} T_i(c(p))/pc(p) \Rightarrow \lambda$ as $p \to \infty$ (this result will be used in Theorem 7). Expand $pc(p)/\sum_{i=1}^{p} T_i(c(p))$ in a first-order Taylor expansion about λ^{-1}, to obtain

$$\frac{pc(p)}{\sum_{i=1}^{p} T_i(c(p))} = \lambda^{-1} - \frac{1}{\xi(p)^2}\left(\sum_{i=1}^{p} \frac{T_i(c(p))}{pc(p)} - \lambda\right), \tag{6.3}$$

where $\xi(p)$ lies between λ and $\sum_{i=1}^{p} T_i(c(p))/pc(p)$.

Since $\chi(s)$ is bounded from below, there exists a finite constant M such that $|1/\xi(p)^2| \le M$. By using the Taylor expansion and taking expectations we have

$$\begin{aligned} E[(\alpha_2(p) - \alpha)] &= E\left[\sum_{i=1}^{p} \frac{S_i(c(p))}{\lambda pc(p)}\right] \\ &\quad - E\left[\sum_{i=1}^{p} \frac{S_i(c(p))}{pc(p)} \sum_{i=1}^{p} \frac{(T_i(c(p)) - \lambda c(p))}{\xi(p)^2 pc(p)}\right]. \end{aligned} \tag{6.4}$$

To handle the first term on the right-hand side of (6.4), we use Glynn (1989b) to show that

$$\frac{1}{\lambda pc(p)} E\left[\sum_{i=1}^{p} S_i(c(p))\right] = \frac{a}{c(p)} + o\left(\frac{1}{c(p)}\right) \tag{6.5}$$

as $p \to \infty$.

To deal with the second term on the right-hand side of (6.4), we bound it by

$$2E\left[\sum_{i=1}^{p} \frac{S_i(c(p))}{pc(p)}\right]^2 + 2E\left[\sum_{i=1}^{p} \frac{(T_i(c(p)) - \lambda c(p))}{\xi(p)^2 pc(p)}\right]^2. \tag{6.6}$$

The second term in (6.6) is dominated by

$$\begin{aligned} &2M^2 E\left[\sum_{i=1}^{p} \frac{(T_i(c(p)) - \lambda c(p))}{pc(p)}\right]^2 \\ &= \frac{2M^2}{c(p)^2}(E[T_1(c(p) - \lambda c(p))])^2 + \frac{2M^2}{pc(p)^2}\operatorname{var}(T_1(c(p)) - \lambda c(p)) \\ &= o(1/c(p)) + O(1/pc(p)), \end{aligned} \tag{6.7}$$

using (4.1)'s uniform integrability. The first term in (6.5) can be handled similarly, yielding an estimate of $O(1/c(p))$. Combining (6.4) through (6.6) yields the result for $\alpha_2(p)$.

Proof of Theorem 7. We first consider $\alpha_1(p)$. We note that Theorem 6 implies that

$$\begin{aligned} &\sqrt{pc(p)}(\alpha_1(p) - \alpha) \\ &= p^{-(1/2)} \sum_{i=1}^{p} V_i(p) + \sqrt{pc(p)}(E\alpha_1(p) - \alpha) \\ &= p^{-(1/2)} \sum_{i=1}^{p} V_i(p) + b_1\sqrt{\frac{p}{c(p)}} + \sqrt{p}o\left(\frac{1}{c(p)}\right), \end{aligned}$$

where

$$V_i(p) = \sqrt{c(p)}\frac{\int_0^{T_i(c(p))} X_i(s)ds}{T_i(c(p)) - E\alpha_1(p)}.$$

By condition (4.1), $\{V_i(p)^2 : p > p_0\}$ is uniformly integrable for some (finite) p_0. We can therefore apply the Lindeberg–Feller theorem (see Chung (1974, p. 205)) to conclude that

$$p^{-(1/2)}\sum_{i=1}^{p} V_i(p) \Rightarrow \lambda^{-(1/2)}\sigma N(0,1)$$

as $p \to \infty$; this proves all three implications for the estimator $\alpha_1(p)$.

The second estimator is handled similarly. We note that

$$\sqrt{pc(p)}(\alpha_2(p) - \alpha) = \frac{pc(p)}{\sum_{i=1}^{p} T_i(c(p))} \cdot \left[p^{-(1/2)}\sum_{i=1}^{p} \tilde{S}_i(p) - \sqrt{\frac{p}{c(p)}}E\tilde{S}_1(p)\right]$$

where $\tilde{S}_i(p) = c(p)^{-(1/2)}[S_i(c(p)) - ES_i(c(p))]$. From (7.2), it is evident that

$$pc(p)/\sum_{i=1}^{p} T_i(c(p)) \Rightarrow \lambda^{-1} \quad \text{as } p \to \infty.$$

A uniform integrability argument similar to that for $\alpha_1(p)$ then shows that

$$p^{-(1/2)}\sum_{i=1}^{p} \tilde{S}_i(p) \Rightarrow \lambda^{(1/2)}\sigma N(0,1).$$

To complete the proof, we refer to Glynn (1989b), where it is shown that $ES_i(c(p)) = \lambda b_2/c(p) + o(1/c(p))$.

Proof of Theorem 8. The argument largely mimics the proof of Theorem 7. The first step is to obtain an expression for the bias of $\alpha_3(p)$. We claim that the bias takes the form

$$E\alpha_c(p) = \alpha + \frac{H_{12}}{c(p)} + o\left(\frac{1}{c(p)}\right) \tag{6.8}$$

as $p \to \infty$. A careful study of Glynn (1989b) shows that the proof there needs to be modified in two respects. First, one needs to argue that

$$\left\{c^{-1}\left(\int_{\gamma(c)}^{T(c)}[X(s) - \alpha]ds\right)^2 : c > c_0\right\}$$

is uniformly integrable for some $c_0 < \infty$. This follows from the observation that

$$\int_{\gamma(c)}^{T(c)}[X(s) - \alpha]^2 ds \leq 2\left(\int_0^{T(\chi(c))}[X(s) - \alpha]ds\right)^2 + 2\left(\int_0^{T(c)}[X(s) - \alpha]ds\right)^2;$$

each of the processes on the right-hand side is uniformly integrable as a result of (4.1) and hence the left-hand side is uniformly integrable (Chung (1974, p. 100)). Second, we note that

$$E\int_{\gamma(c)}^{T(c)}[X(s) - \alpha]ds = \int_{\chi(c)}^{c} E\hat{Y}(s)ds,$$

where $\hat{Y}(c) = [X(T(c)) - \alpha]/\chi(T(c))$. Since $\|X\| < \infty$ and χ is bounded away from zero, it follows that $\hat{Y}$ is a bounded process. Hence, the results of Nummelin and Tuominen (1982) apply, showing that $E\hat{Y}(t) \to 0$ exponentially fast. It is then evident that

$$\int_{\chi(c)}^{c} E\hat{Y}(s)ds = o(1)$$

as $c \to \infty$. As a consequence, the term that contributes the quantity $a/c(p)$ to the asymptotic bias of $\alpha_1(p)$ is $o(1/c(p))$ in the current setting. This yields (6.7).

The proof of Theorem 8 is completed in much the same way as Theorem 7. The uniform integrability established above allows us to apply the Lindeberg–Feller theorem once again, finishing the proof.

Proof of Theorem 9. The analogue of (6.8) for the estimator $\alpha_4(p)$ is

$$E\alpha_4(p) = \alpha + o\left(1/c(p)\right),$$

and is established in basically the same way. The result of the proof goes through as in Theorem 8.

7. Summary. This paper has investigated theoretical properties of an attractive method for using parallel processors in discrete event simulations: running independent replications, in parallel, on multiple processors and averaging the results at the end of the runs. In previous papers, we considered the problem of estimating transient quantities, while in this paper we consider the steady-state estimation procedure. While the method of replications with initial transient deletion is conceptually simple to apply, some care needs to be taken in order to obtain estimators with the proper convergence behavior. Specifically, the growth rates for the number of processors (replications), the length of the replications, and the length of the deletion period need to be controlled in order to produce valid confidence intervals for steady-state parameters. In the parallel processing setting, a sampling plan in which the replication lengths are given by limits on computer time is particularly attractive since the completion time of the experiment is deterministic (assuming the machine is dedicated to running the simulation experiment). However, in this case, the leading term in the bias expansion of the straightforward estimator without deletion, $\alpha_1(p)$, is $(a + H_{12})/c(p)$ where $c(p)$ is the computer time per replication, a is due to initialization bias, and H_{12} is due to the fact that the denominator in the ratio estimate is random. Deleting an appropriate portion of each replication removes the initialization bias $a/c(p)$, but does not remove the ratio bias $H_{12}/c(p)$. Thus, when this estimator is used and the replication length is determined by computer time, deletion is essentially useless. On the other hand, the bias expansion of a new estimator, $\alpha_2(p)$, has leading term $a/c(p)$, which is removed entirely by appropriate deletion. Therefore, in practice, we recommend use of the new estimator with deletion, $\alpha_4(p)$.

Experimental results concerning the performance of these estimators in simulations of simple queueing network models are reported in Glynn and Heidelberger (1992). Those experimental results confirm the theoretical results presented here and reinforce our recommendation to use $\alpha_4(p)$ rather than $\alpha_3(p)$. Our experiments showed that $\alpha_4(p)$ outperforms $\alpha_3(p)$, in terms of exhibiting less bias and truer confidence interval coverage, when the number of processors is large and the amount of time per processor is relatively small.

While we have described the results in terms of a computer time constraint on the replication lengths, they remain valid for essentially any other measure of replication length. Examples include computing charges (which may involve costs for CPU, memory, and I/O use), real time (i.e., wall-clock time, which may differ in multiprogrammed environments), the total number of events processed, and the total number of events of a certain type processed (such as departures from a network). In addition, the results are applicable to simulation experiments on a single processor if the replication lengths are determined in the above fashion.

REFERENCES

B. C. BHAVSAR, AND J. R. ISAAC (1987), *Design and analysis of parallel Monte Carlo algorithms*, SIAM J. Sci. Statist. Comput., 8, pp. 73–95.

P. BILLINGSLEY (1968), *Convergence of Probability Measures*, John Wiley, New York.

E. BOLTHAUSEN (1980), *The Berry–Esseén theorem for functionals of discrete Markov chains*, Z. Wahrsch. verw. Gebiete, 54, pp. 59–73.

——— (1982), *The Berry–Esseén theorem for strongly mixing Harris recurrent Markov chains*, Z. Wahrsch. verw. Gebiete, 60, pp. 283–289.

K. L. CHUNG (1974), *A Course in Probability Theory*, Academic Press, New York.

M. A. CRANE AND D. L. IGLEHART (1975), *Simulating stable stochastic systems*, III: *Regenerative processes and discrete event simulations*, Oper. Res., 23, pp. 33–45.

S. N. ETHIER AND T. G. KURTZ (1986), *Markov Processes: Characterization and Convergence*, John Wiley, New York.

W. FELLER (1968), *An Introduction to Probability Theory and Its Applications*, John Wiley, New York.

R. M. FUJIMOTO (1989), *Time warp on a shared memory multiprocessor*, in Proc. 1989 Internat. Conf. Parallel Processing, Vol. III., F. Ris and P. M. Kogge, eds., The Pennsylvania State University Press, State College, PA, pp. 242–249.

——— (1990), *Parallel discrete event simulation*, Comm. ACM, 33, pp. 31–53.

P. W. GLYNN (1982), *Regenerative aspects of the steady-state simulation problem for Markov chains*, Tech. Report 17, Department of Operations Research, Stanford University, Stanford, CA.

——— (1987), *Limit theorems for the method of replication*, Stochastic Models, 4, pp. 344–350.

——— (1989a), *A* GSMP *formalism for discrete event systems*, Proc. IEEE, 77, pp. 14–23.

——— (1989b), *A low bias steady-state estimator for equilibrium processes*, Tech. Report 47, Department of Operations Research, Stanford University, Stanford, CA.

P. W. GLYNN AND P. HEIDELBERGER (1990), *Bias properties of budget constrained simulations*, Oper. Res., 38, pp. 801–814.

——— (1991a), *Analysis of parallel replicated simulations under a completion time constraint*, ACM Trans. Modeling and Computer Simulation, 1, pp. 3–23.

——— (1991b), *Analysis of initial transient deletion for replicated steady-state simulations*, Oper. Res. Lett., 10, pp. 437–443.

——— (1992), *Experiments with initial transient deletion for parallel, replicated steady-state simulations*, Management Sci., 38, pp. 400–418.

P. W. GLYNN AND W. WHITT (1987), *Sufficient conditions for functional-limit-theorem versions of $L = \lambda W$*, Queueing Systems, Theory, Appl., 1, pp. 279–287.

P. GOLI, P. HEIDELBERGER, D. TOWSLEY, AND Q. YU (1990), *Processor assignment and synchronization in parallel simulation of multistage interconnection networks*, in Distributed Simulation, D. Nicol, ed., The Society for Computer Simulation International, San Diego, CA, pp. 181–187.

P. HEIDELBERGER (1986), *Statistical analysis of parallel simulations*, in 1986 Winter Simulation Conference Proceedings, J. Wilson and J. Henriksen, eds., IEEE Press, Piscataway, NJ, pp. 290–295.

——— (1988), *Discrete event simulations and parallel processing: Statistical properties*, SIAM J. Sci. Statist. Comput., 9, pp. 1114–1132.

S. KARLIN AND H. M. TAYLOR (1975), *A First Course in Stochastic Processes*, Academic Press, New York.

B. D. LUBACHEVSKY (1989), *Efficient distributed event-driven simulations of multiple-loop networks*, Comm. ACM, 32, pp. 111–123.

N. MAIGRET (1978), *Thé orè me de limite centrale fonctionnel pour une chaîne de Markov recurrente au sens de Harris et positive*, Ann. Inst. Henri Poincaré, 14, pp. 425–440.

M. S. MEKETON AND P. HEIDELBERGER (1982), *A renewal theoretic approach to bias reduction in regenerative simulations*, Management Sci., 28, pp. 173–181.

C. M. NEWMAN AND A. L. WRIGHT (1981), *An invariance principle for certain dependent sequences*, Ann. Probab., 9, pp. 671–675.

D. M. NICOL (1988), *Parallel discrete-event simulation of* FCFS *stochastic queueing networks*, in Proc. ACM/SIGPLAN PPEALS 1988, Parallel Programming: Experience with Applications, Languages and Systems, ACM Press, New York, pp. 124–137.

E. NUMMELIN (1984), *General Irreducible Markov Chains and Non-negative Operators*, Cambridge University Press, Cambridge, U.K.

E. NUMMELIN AND P. TUOMINEN (1982), *Geometric ergodicity of Harris recurrent Markov chains with applications to renewal theory*, Stochastic Process. Appl., 12, pp. 187–202.

W. L. SMITH (1955), *Regenerative stochastic processes*, Proc. Roy. Soc. London Ser. A., 232, pp. 6–31.

B. UNGER AND D. JEFFERSON, EDS. (1988), *Distributed Simulation*, 1988, Simulation Series 19, No. 3, The Society for Computer Simulation International, San Diego, CA.

B. UNGER AND R. FUJIMOTO, EDS. (1989), *Distributed Simulation*, 1989, Simulation Series 21, No. 2, The Society for Computer Simulation International, San Diego, CA.

W. WHITT (1980), *Some useful functions for functional limit theorems*, Math. Oper. Res., 5, pp. 67–85.

Q. YU, D. TOWSLEY, AND P. HEIDELBERGER (1989), *Time-driven parallel simulation of multistage interconnection networks*, in Distributed Simulation, 1989, B. Unger and R. Fujimoto, eds., The Society for Computer Simulation International, San Diego, CA, pp. 191–196.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 4, pp. 923–947, July 1992

AN IMPLEMENTATION OF THE FAST MULTIPOLE METHOD WITHOUT MULTIPOLES*

CHRISTOPHER R. ANDERSON†

Abstract. An implementation is presented of the fast multipole method, which uses approximations based on Poisson's formula. Details for the implementation in both two and three dimensions are given. Also discussed is how the multigrid aspect of the fast multipole method can be exploited to yield efficient programming procedures. The issue of the selection of an appropriate refinement level for the method is addressed. Computational results are given that show the importance of good level selection. An efficient technique that can be used to determine an optimal level to choose for the method is presented.

Key words. Poisson equation, fast summation, point sources

AMS(MOS) subject classifications. 65C99, 35J05, 34B27

1. Introduction.

The purpose of this paper is three-fold. First we will present a method for computing N-body interactions that is similar to the fast multipole method (FMM) as developed by Greengard and Rokhlin [5], [6] and Van Dommelen and Rundensteiner [15], but one that does not use complex power series in two dimensions or spherical harmonic expansions in three dimensions. Our procedure will be based on the use of Poisson's formula for representing solutions of Laplace's equation. While the accuracy and operation count of the resulting method is almost identical to the fast multipole method, the method does offer some advantages. One advantage is that the component operations of the multipole method, such as shifting and combining multipoles, are very easy to formulate for approximations based on Poisson's formula. Another advantage is that the difference between the two- and three-dimensional methods is very slight, and so programming a three-dimensional method is relatively straightforward once a two-dimensional method has been programmed. The second aspect this paper discusses is how multigrid programming strategy can be used to facilitate the programming of our method and others like it (such as the original fast multipole method). Third, we wish to discuss the issue of parameter selection when using these "fast" methods. Essentially, the computational efficiency of these methods depends critically upon the choice of a level of refinement of physical space. A wrong choice can lead to a very inefficient algorithm. We shall present a procedure for obtaining an optimal choice of the refinement level.

The problem of calculating N-body interactions occurs in a wide variety of computational problems—discrete vortex calculations, galaxy simulations, plasma simulations, etc. For each of these computational problems the calculation takes on a slightly different form, but each shares the common feature that the interaction is determined via solutions of Laplace's equation. So, rather than address each different application, we will discuss the following N-body model problem: Given N charged particles at locations x_i with strengths κ_i the goal is to calculate the potential $\phi(x_i)$,

* Received by the editors July 23, 1990; accepted for publication (in revised form) May 16, 1991.

† Department of Mathematics, University of California, Los Angeles, California, 90024. This research was supported by Office of Naval Research contract N00014-86-K-0691, National Science Foundation grant DM586-57663, and IBM fellowship D880908.

where ϕ is a solution of

$$\Delta\phi = \sum_{i=1}^{N} \delta(x - x_i)\kappa_i. \tag{1}$$

Here $\delta(x)$ is Dirac's delta function and Δ is the Laplacian. Since the key ideas behind the method presented here do not change much when one goes from two to three dimensions, we will primarily discuss the two-dimensional case. Those aspects that do change with dimension will be specifically addressed.

Often in simulations, one uses smoothed delta functions (or "blobs"), and the model problem in this case is identical to (1) but we solve

$$\Delta\phi = \sum_{i=1}^{N} \Psi_\epsilon(x, x_i)\kappa_i \tag{2}$$

where $\Psi_\epsilon(x, x_i)$ is a smoothed delta function whose support is contained within a disk of radius ϵ about x_i. The choice of Ψ and ϵ is important for accuracy, but not particularly important for the methods used to accelerate the computation of (2). We do make the assumption that the blobs have support contained within disks or spheres of radius ϵ, and so the blob functions cannot be completely general.

The solution ϕ of (1) is given by

$$\phi(x) = \sum_{i=1}^{N} \frac{\kappa_i}{2\pi} \log(|x - x_i|), \tag{3}$$

while for (2),

$$\phi(x) = \sum_{i=1}^{N} \frac{\kappa_i}{2\pi} \Phi_\epsilon(x - x_i) \tag{4}$$

where Φ_ϵ is the potential induced by a single blob. (This can often be calculated explicitly by solving $\Delta\Phi = \Psi_\epsilon(x)$ using the method of separation of variables.)

From (3) or (4) it is clear that if we evaluate the solution ϕ at each point $x_i, i = 1, \cdots N$, then this computation requires $O(N^2)$ operations. For particle simulations, the larger the value of N the better, and so there is great interest in reducing this operation count. The fast multipole method is a technique for reducing the operation count of this problem to $O(N)$ operations.

There are two basic ingredients to the fast multipole technique. One ingredient is the process of combining large numbers of particles into single computational elements. When a cluster of particles is "far away" from a particular point, then the potential of the cluster is approximated by the potential induced by a single computational element located inside the cluster. (How far "far away" is must be determined, of course.) In the fast multipole method the computational element is a multipole expansion located at the center of a disk containing the cluster of particles. If we identify the complex plane with R^2, then one can represent the potential induced by a collection of particles of strength α_i located at the points z_i by $\mathrm{Re}\left(\sum_{i=1}^{N_c} \alpha_i \log(z - z_i)\right)$. The approximation used is the following:

$$\phi(z) = \mathrm{Re}\left(\sum_{i=1}^{N_c} \alpha_i \log(z - z_i)\right) \approx \mathrm{Re}\left(a_0 \log(z - z_0) + \sum_{k=1}^{p} \frac{a_k}{(z - z_0)^k}\right). \tag{5}$$

Here z_0 is the center of a disk enclosing the particles z_i, p is the order of the multipole, and a_k are coefficients chosen so that the multipole is an accurate approximation of the potential. The efficiency of the method comes about because the evaluation of the potential of this single computational element, $O(p)$ operations, is typically much less work than that of computing the corresponding potential of the whole collection of particles, $O(N_c)$ operations. In the three-dimensional implementation of the method, the complex multipole is replaced by an expansion in spherical harmonics [6].

The second ingredient of the fast multipole method has to do with organizing the computations so that the application of the technique of combining particles is efficient and does not lead to inaccuracies. For example, when one combines particles into single elements, the more widely distributed in space the particles of a given cluster are, the greater the inaccuracy the approximation (5) becomes (for a fixed value of p and a fixed point of evaluation). However, if the particles are fixed and the evaluation position z is moved away from the center of the expansion z_0, then the accuracy of the potential approximation at z improves. The net effect is that if a fixed degree of accuracy is desired, one approximates the potential by using a hierarchy of approximations of the form (5). Close to an evaluation point one combines particles over small regions to form multipole approximations. Particles further away are combined over larger regions to form multipole approximations. (Essentially the size of the region over which particles are combined is inversely proportional to their distance to the evaluation point.) In the fast multipole method, this aspect is organized by decomposing the region containing all of the particles into boxes of different sizes. The particles in each of the boxes are combined and their potential is replaced by an approximation of the form (5). The potential at a particular point is then the sum of these approximations and the potential induced by the direct contribution of very close particles. In Fig. 1 we show a test particle and the surrounding regions in which the particles are combined into multipole expansions. The potential induced by particles in the region immediately surrounding the test particle is computed using the exact interaction formula (3) or (4).

In the approximation of the potential due to a cluster of particles by a single computational element, one is not forced to use multipoles. There is a possibility for other types of computational elements to be used and in this paper we shall discuss one alternative computational element. The basic strategy is to use Poisson's formula. In two dimensions we have that for points outside of a disk of radius a containing the particles, the potential can be represented by

$$\phi(r,\theta) = \kappa \log(r) + \frac{1}{2\pi}\int_0^{2\pi} \tilde{\phi}(a,s)\left[\frac{1-(\frac{a}{r})^2}{1-2(\frac{a}{r})\cos((\theta-s))+(\frac{a}{r})^2}\right] ds, \tag{6}$$

where the function $\tilde{\phi}(a,s)$ and the value of κ are determined from the values of ϕ on the circumference of the disk. (r,θ) is the position in polar coordinates of the evaluation point from the center of the disk. In the numerical method, the integral in Poisson's formula is converted into a sum of K quantities (where K is the number of points in an integration rule for (6)). Thus, the potential induced by a cluster of N_c particles can be reduced to the evaluation of a sum of K terms. The idea of using a numerical approximation of Poisson's formula to represent the potential is similar in spirit to an aspect of the method introduced by Rokhlin [12] for solving the equations of scattering theory. While the idea of using Poisson's formula is straightforward, getting Poisson's formula to work numerically proved quite troublesome. The problem is that

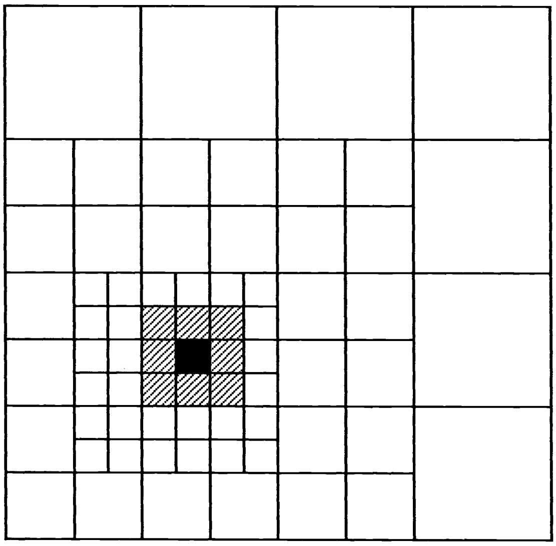


FIG. 1. *The hierarchical clustering of particles used to create a multipole approximation to the potential at a point. The evaluation point is within the darkened box. The potential induced by particles in unshaded boxes are combined into multipole approximations. The potential induced by particles in the lined boxes is computed using the direct interaction formula.*

the numerical evaluation of Poisson's kernel is difficult because of the singularity in the kernel. As the evaluation point approaches the ring where the integration is performed, the kernel becomes more and more singular, and the accuracy deteriorates rapidly. In the first section we discuss how this problem can be eliminated. In the first section we also present the necessary details (for both two and three dimensions) that facilitate the incorporation of this type of computational element into a fast multipole scheme. In order to distinguish this type of computational element based on Poisson's formula from a multipole element, we refer to them as "outer ring approximations" (two dimensions) or "outer sphere approximations" (three dimensions). (We shall also have use for these approximations inside rings or spheres and these will be "inner ring" and "inner sphere" approximations.) In Fig. 2, a schematic of the use of this element is presented. The computational particles (represented by the small circles) induce a potential at locations on the circumference of a ring that surrounds the particles. The potential outside the ring is evaluated by integrating these values against a Poisson kernel.

There are other possible choices for the computational elements, for example, Nowak [11] uses charge distributions over panels. While each different computational element will give rise to a different method, the structure of the algorithm is relatively independent of the type of element used. We shall therefore refer to the class of methods that uses the basic computational structure of the fast multipole method (but different elements) as "hierarchical element" methods.

As discussed earlier, in the multipole method one forms an approximation to the potential at a point by summing the potential induced by multipole approximations for different size clusters of particles. The same type of construction will be used with the computational elements based on Poisson's kernel. In both our method and the

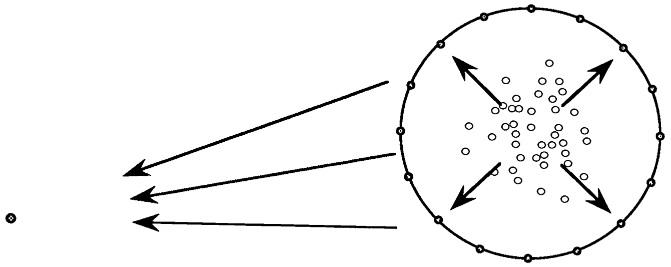


FIG. 2. *Schematic of an outer ring approximation. The computational particles (represented by the small circles) induce a potential at locations on the circumference of a ring that surrounds the particles. The potential outside the ring is evaluated by integrating these values against a (modified) Poisson kernel.*

original multipole method, when one chooses different evaluation points one sums a different collection of these approximations. The organization of the evaluation of the appropriate approximations presents a challenging computational problem. One important component of the fast multipole algorithm as presented by Greengard and Rokhlin [5] is an effective way to construct and evaluate this hierarchy of approximations. An observation that can be made about their method for carrying out this computation is that it is much like the multigrid method. The observation has some merit, and in the second section we show how this inherent multigrid structure can be exploited to yield an efficient programming strategy for the construction and evaluation of the hierarchical set of computational elements. (The identification of the multigrid, or tree structure, in the method is also useful in analyzing the performance of such methods on parallel processors; see [9], for example.) We shall present this discussion in the context of using Poisson kernel computational elements, but the results apply directly to the original multipole method as well.

The efficiency of hierarchical element methods comes about from the ability to represent a cluster of many particles by an approximation (multipole or otherwise) that can be evaluated with little numerical work. However, there is the possibility that the work to evaluate such an approximation may be more than that of evaluating the field induced by the particles in the cluster directly. This problem will certainly arise if the clusters contain only a very small number of particles. Thus, one is left with the problem of determining what is the minimal region size that should be used in forming the clusters of particles. This can be determined if the points are uniformly distributed, but if they are not, the problem is much more difficult. Fortunately, an a priori estimate of the computational time for any level of clustering is readily available and the level that requires the smallest time can be identified and chosen. In the third section we shall discuss this issue and present the details of a method for obtaining timing estimates for each level of clustering. We will also give computational results that illustrate the importance of appropriate level selection.

Finally, we present our conclusions and discuss some of the advantages and disadvantages of the techniques presented in this paper. The programs described in this paper are available from the author.

2. Derivation of computational elements. A basic component of the fast multipole method is the ability to represent the potential induced by a large number of particles by a single computational element that is relatively inexpensive to compute (i.e., the multipole expansion). The use of the multipole (or spherical harmonic) expansion is not a necessity, and in this section we present a computational element that is based on Poisson's formula for a circle in two dimensions and a sphere in three dimensions.

Let $\Psi(r,\theta)$ be the potential in two dimensions induced by a collection of N particles at locations $\vec{x}_i = (r_i, \theta_i)$ and strengths κ_i that are contained within a disk of radius a centered at the origin. This potential is a harmonic function outside of the disk, so one can use Poisson's formula (with a log term to satisfy the circulation requirements) to represent it. Set $\kappa = \sum_{i=1}^{N}(\kappa_i/2\pi)$ and $\tilde{\Psi}(r,\theta) = \Psi(r,\theta) - \kappa\log(r)$; then if (r,θ) is a point in the plane outside the disk, we have

$$\Psi(r,\theta) = \kappa\log(r) + \frac{1}{2\pi}\int_0^{2\pi} \tilde{\Psi}(a,s)\left[\frac{1-(\frac{a}{r})^2}{1-2(\frac{a}{r})\cos((\theta-s))+(\frac{a}{r})^2}\right] ds. \tag{7}$$

Our first attempt to get a representation suitable for numerical computation was to use the trapezoidal rule to approximate the integral in (7). Let M be an integer and set $K = 2M+1$. If we set $h = 2\pi/K$ and $s_i = (a\cos(ih), a\sin(ih)), i = 1,\cdots,K$, then an approximate representation is given by

$$\Psi(r,\theta) \approx \kappa\log(r) + \frac{1}{2\pi}\sum_{i=1}^{K} \tilde{\Psi}(a,s_i)\left[\frac{1-(\frac{a}{r})^2}{1-2(\frac{a}{r})\cos((\theta-s_i))+(\frac{a}{r})^2}\right] h. \tag{8}$$

However, the approximation in (8) is very ill conditioned. As the evaluation point moves towards the ring of radius a, the approximation is exceedingly inaccurate. In Fig. 3 the dashed lines are the errors in the potential that are obtained using the approximation (8). The ring is of radius $a = 2$ and the potential is that induced by a single particle of unit strength located at $r = \sqrt{2}/2$ and $\theta = \pi/3$. The upper, middle, and lower dashed lines correspond to $K = 9$, $K = 17$, and $K = 25$ points, respectively, in the integration formula.

The remedy for this problem becomes apparent when one considers the derivation of Poisson's formula. Let ϕ be a solution of Laplace's equation exterior to the disk of radius a, which has circulation κ. If $f(\theta)$ is the value of $\phi(r,\theta) - \kappa\log(r)$ at $r = a$ we have, using the method of separation of variables,

$$\phi = \kappa\log(r) + \sum_{k=-\infty}^{k=\infty} c_k \left(\frac{a}{r}\right)^{|k|} e^{ik\theta}, \tag{9}$$

where the coefficients c_k are the Fourier coefficients of the function $f(\theta)$,

$$c_k = \frac{1}{2\pi}\int_0^{2\pi} f(s)e^{-iks}\, ds. \tag{10}$$

If one combines (9) and (10) one obtains

$$\phi = \kappa\log(r) + \sum_{k=-\infty}^{\infty}\left[\frac{1}{2\pi}\int_0^{2\pi} f(s)e^{-iks}\, ds\right]\left(\frac{a}{r}\right)^{|k|} e^{ik\theta}. \tag{11}$$

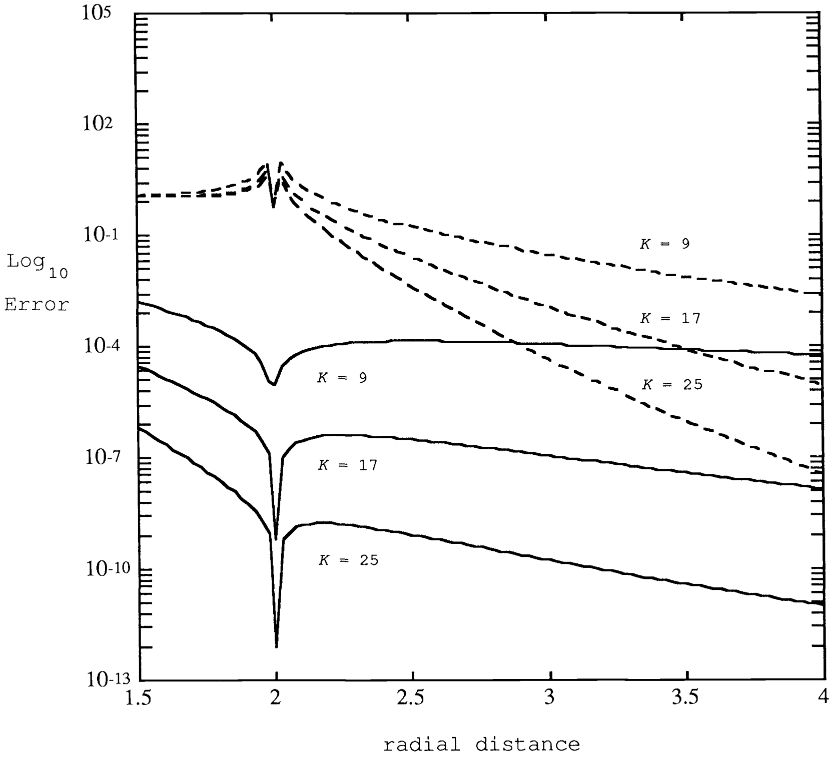


FIG. 3. *Error in the potential when outer ring approximations are used. The dashed lines correspond to an unmodified Poisson kernel, and the solid lines to a modified Poisson kernel. The ring radius was $a = 2$ and the value of K refers to the number of points in the integration formula.*

Poisson's formula results when one interchanges summation and integration in (11):

$$\begin{aligned}\phi &= \kappa\log(r) + \frac{1}{2\pi}\int_0^{2\pi} f(s)\left[\sum_{k=-\infty}^{\infty} \mathrm{e}^{-ik(\theta-s)}\left(\frac{a}{r}\right)^{|k|}\right] ds\\ &= \kappa\log(r) + \frac{1}{2\pi}\int_0^{2\pi} f(s)\left[\frac{1-(\frac{a}{r})^2}{1-2(\frac{a}{r})\cos((\theta-s))+(\frac{a}{r})^2}\right] ds.\end{aligned}$$

From this derivation one can see that using a trapezoidal rule approximation to the integral in (7) is equivalent to implicitly using the trapezoidal rule to approximate the Fourier coefficients (10). Clearly, if only $K = 2M + 1$ points are used in the quadrature rule, then one should not use any coefficients c_k with $|k| > M$, i.e., one should only use those Fourier modes in (9) that can be reliably estimated using K equispaced points. Thus we consider using (9) with only the first M modes,

$$\phi \approx \kappa\log(r) + \frac{1}{2\pi}\int_0^{2\pi}\left[\sum_{k=-M}^{M} \mathrm{e}^{-ik(\theta-s)}\left(\frac{a}{r}\right)^{|k|}\right] f(s)ds = \kappa\log(r) + \frac{1}{2\pi}\int_0^{2\pi} f(s)$$

$$\times\left[\frac{1-(\frac{a}{r})^2-2(\frac{a}{r})^{M+1}\cos((M+1)(\theta-s))+2(\frac{a}{r})^{M+2}\cos(M(\theta-s))}{1-2(\frac{a}{r})\cos((\theta-s))+(\frac{a}{r})^2}\right]ds.$$

Here, as in the original derivation of the Poisson kernel, the simplification arises from the formula for summing a geometric series. When this integral is approximated by the trapezoidal rule we obtain the numerical approximation

(12)

$$\phi(r,\theta)\approx\kappa\log(r)+\frac{1}{2\pi}\sum_{i=1}^{K}f(s_i)$$
$$\times\left[\frac{1-(a/r)^2-2(a/r)^{M+1}\cos((M+1)(\theta-s_i))+2(a/r)^{M+2}\cos(M(\theta-s_i))}{1-2(a/r)\cos((\theta-s_i))+(a/r)^2}\right]h$$

where the integration points s_i are equispaced on the ring with $h = 2\pi a/K$. By construction, this discrete approximation is precisely that which would be obtained if one formed an approximate solution of the exterior Laplace equation by combining solutions for the first M modes of the data $f(s)$. The Fourier coefficients used in the approximation are, however, those obtained from the discrete, rather than the continuous, Fourier decomposition of $f(s)$.

We will use (12) with $\kappa = \sum_{i=1}^{N}(\kappa_i/2\pi)$ and $f(s_i) = \Psi(a, s_i) - \kappa\log(a)$ to approximate the potential Ψ induced by N particles of strengths κ_i. The steps to create this approximation consist of summing the strengths of the particles in a region of space to obtain the strength of the log term in the approximation. This is then followed by the evaluation of the potential induced by the particles (minus the log term) at equispaced points on the ring of radius a that encompasses these points. To evaluate this approximation at some point outside the disk, one merely adds the contribution of the log term and the sum in (12). As before, if the number of particles N contained within the ring is large compared to K, a substantial savings in work is accomplished by using such an approximation. We shall refer to the approximation (12) as an outer ring approximation.

The improvement in accuracy when (12) is used is demonstrated by the solid lines in Fig. 3. In this figure, these lines indicate the error in the potential when (12) is used for the problem of a point charge located at $r = \sqrt{2}/2$ and $\theta = \pi/3$. The upper, middle, and lower solid lines correspond to $K = 9$, $K = 17$, and $K = 25$ integration points, respectively. As shown in the figure, the error remains sufficiently small as the evaluation point approaches the ring used in the approximation (a ring of radius 2). This behavior is in sharp contrast to the error in the approximation with the unmodified kernel (8), which gives $O(1)$ errors as the evaluation point approaches the ring.

Complete error estimates for the approximation have yet to be worked out, but the accuracy can be partially assessed using the results of Greengard and Rokhlin [5]. In particular, in the derivation of (12) one sees that the approximation is formed by combining the solutions of Laplace's equation corresponding to the first M Fourier modes associated with the data $f(s)$. If in this approximation we used the exact Fourier coefficients (10) instead of the discrete Fourier coefficients, then the approximation that would result would be identical to that of a multipole expansion with M terms. Hence, using the result of [5] we expect that if the particles are centered within a disk of radius α about the origin then the error in the potential at any test point with a radial distance r will be $O((\alpha/r)^M)$. It is important to note that this estimate of the error depends on the relative location of the evaluation point to the ring of radius α that surrounds the particles, and not on the relative location of the particle from the ring of radius a that is used in the approximation. For example, the solid lines in Fig. 3 indicate that the error in the approximation remains small even as the evaluation point moves inside the ring that is used for the approximation. This is expected, and in fact necessary, since in the complete algorithm we shall have reason to evaluate the approximation for points inside the ring. The ring radius that is used in the approximation (12) has an indirect effect on the accuracy. Essentially, the distance of the ring from the center of the disk that contains the particles affects the aliasing that occurs with the implicit use of discrete, rather than continuous, Fourier coefficients. If the ring is too close to the particles then large aliasing errors occur because of the higher harmonics that are present in the potential. Conversely, if one takes the ring radius too large, the coefficients that one is implicitly estimating are very small in magnitude and the effects of finite precision lead to accuracy problems. After some experimentation, we found that setting the ring radius $a = 2\alpha$ gives acceptable results.

In the complete method there is the need to represent the potential inside a given region. Following the same strategy for obtaining a numerically stable outer ring approximation we define an inner ring approximation by

$$\begin{aligned}(13)\qquad \phi(r,\theta) &\approx \frac{1}{2\pi}\sum_{i=1}^{K} f(s_i) \\ &\times \left[\frac{1-(\frac{r}{a})^2 - 2(\frac{r}{a})^{M+1}\cos((M+1)(\theta - s_i)) + 2(\frac{r}{a})^{M+2}\cos(M(\theta - s_i))}{1 - 2(\frac{r}{a})\cos((\theta - s_i)) + (\frac{r}{a})^2}\right] h,\end{aligned}$$

where (r,θ) is the evaluation point and $f(s_i)$ is the value of the potential induced by particles (or outer ring approximations) outside the ring of radius a. The evaluation point can be taken to be outside as well as inside the ring.

To make efficient use of approximations of the form (12), clustering is done on different levels, i.e., approximations are constructed for collections of particles in clusters of increasing size. In this construction it is necessary to combine several outer ring approximations into a single outer ring approximation. (In the multipole method, this is carried out by shifting the origin of the multipole expansions.) This operation of combining outer ring approximations is particularly simple to implement; one just evaluates the potential induced by the component outer ring approximations at the integration points of a single outer ring approximation under construction. (See Fig. 4.) The situation is similar for the other required combining operations in the method, i.e., forming inner ring approximations to represent the potential of outer ring approximations, etc., these are all carried out by just evaluating the component approximations at the set of points needed for the particular approximation. Due to the close relationship between these operations and those that are used in the original multipole method, the errors for ring approximations with $K = 2M + 1$ integration points can be expected to behave in a fashion similar to the errors in the multipole method when M terms in a multipole expansion are used. In particular, there is always concern that the hierarchical strategy of combining approximations may be unstable due to the accumulation of errors. While there is no proof of the stability for either the method based on ring approximations or on multipoles, all the computational results with which the author is familiar have indicated that the hierarchical

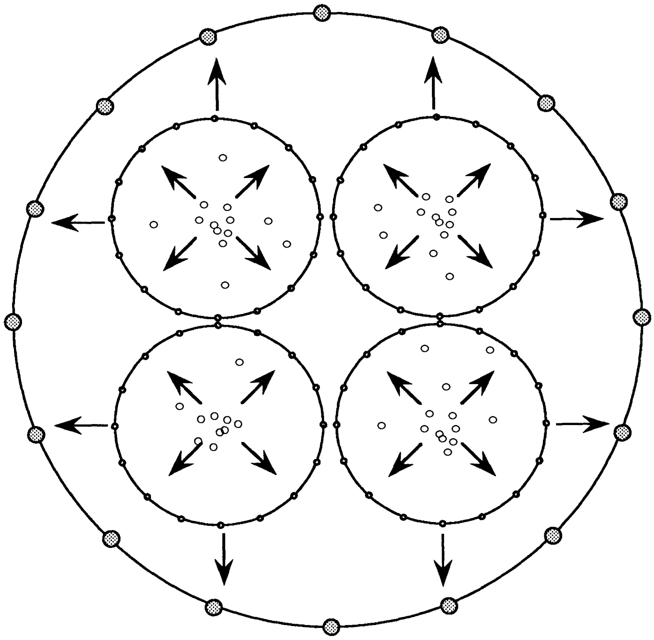


FIG. 4. *Schematic diagram of the operation of combining outer ring approximations. The coarse level outer ring approximation is obtained by evaluating the potential of the outer ring approximations contained within it.*

combination procedure is stable.

In three dimensions the construction is essentially the same. Let $g(x, y, z)$ denote values on a sphere of radius a and denote by Ψ the harmonic function external to the sphere with these boundary values. Given a point outside the sphere of radius a, $\vec{x}$ with spherical coordinates (r, θ, ϕ), let $\vec{x}_p = (\cos(\theta)\sin(\phi), \sin(\theta)\sin(\phi), \cos(\phi))$ be the point on the unit sphere that points in the direction of $\vec{x}$, then

$$\Psi(\vec{x}) = \frac{1}{4\pi}\int_{S^2}\left[\sum_{n=0}^{\infty}(2n+1)\left(\frac{a}{r}\right)^{n+1} \mathrm{P}_n(\vec{s}\cdot\vec{x}_p)\right] g(a\vec{s})\, ds, \tag{14}$$

where the integration is carried out over S^2, the surface of the unit sphere, and P_n is the nth Legendre function. (See [4, p. 513].)

As is the case with the ring approximations we shall refrain from using all of the terms in the kernel (14). Given a numerical formula for integrating functions on the surface of the sphere with K integration points $\vec{s}_i$ and weights w_i we use an approximation of the form

$$\Psi(\vec{x}) \approx \sum_{i=1}^{K}\left[\sum_{n=0}^{M}(2n+1)\left(\frac{a}{r}\right)^{n+1} P_n(\vec{s}_i\cdot\vec{x}_p)\right] g(a\vec{s}_i)\, w_i. \tag{15}$$

We call this approximation an outer sphere approximation.

A critical choice to be made is that of the appropriate number of terms M of the kernel in (15). If one uses an integration formula for the sphere of degree D (i.e., the formula integrates polynomials of up to and including degree D exactly) then an appropriate choice for M is $M \leq \frac{D}{2}$. The reasoning for this is as follows: Implicit in the use of Poisson's formula is the computation of an orthogonal expansion in spherical harmonics. If we keep $M+1$ terms in the kernel, then we are constructing an approximation by combining the potentials corresponding to the first $M+1$ terms of this expansion. In a numerical approximation of the type (15), we are implicitly using a discrete, and hence accuracy-limited, approximation to estimate the orthogonal expansion coefficients. We should therefore choose M on the basis of our ability to computationally estimate the coefficients of a spherical expansion with degree less than or equal to M. Given a function on the sphere comprised of spherical harmonics of degree m, then the spherical expansion coefficients of this function can be determined exactly by an integration formula with accuracy of degree $2m$. (Spherical harmonics of degree less than or equal to m can be expressed as the restriction of polynomials of at most degree m to a sphere, so a formula of degree $2*m$ can calculate inner products of two of these functions exactly.) Therefore, if we ignore the effects of aliasing, using $M \leq \frac{D}{2}$ allows us to determine the first $M+1$ coefficients in the orthogonal expansion accurately. The error in the resulting approximation can be expected to be on the order of the first neglected term in the expansion, i.e., if we keep $M+1$ terms, then the error in the potential induced by a collection of particles inside a sphere of radius α about the origin should behave like $O(\frac{\alpha}{r})^{M+2}$. (The highest power of r in the sum is $M+1$.)

Unlike two dimensions in which the trapezoidal rule furnishes us with the optimal integration formula for our integration, in three dimensions the choice of integration formula is more complicated. One obvious choice is the use of product integration formulas. The ones typically employed are those that use trapezoidal integration in the θ direction and then Gaussian quadrature in the ϕ direction. These are somewhat inefficient as the integration points are crowded near the poles, so we chose to use integration formulas from a nonproduct family, namely, those described in [13], which are taken from [10]. In Fig. 5 and Table 1 we present the results of computations obtained using (15). Each of the curves in Fig. 5 are the errors in the potential induced by a single unit strength particle located at $r = \sqrt{3}/2$, $\theta = \pi/3$, and $\phi = 0$. The test points are located along the x-axis ($\theta = 0, \phi = \pi/2$). This selection of points was chosen to be representative of the worst possible (most inaccurate) configuration that would occur when this element is incorporated into the complete method. The individual curves correspond to 5th, 7th, 9th, 11th, and 14th order integration formulas. These formulas required 12, 24, 32, 50, and 72 points, respectively. In Table 1 we present the rates of decay of the error as the radial distance to the evaluation point increases. As expected, the rate was approximately $M+2$. It is interesting to note that the 9th order formula has errors that are much better than expected—it appears that the integration formula is of higher order than advertised.

The approximation used to represent potentials inside a given region is

$$\Psi(\vec{x}) \approx \sum_{i=1}^{K} \left[\sum_{n=0}^{M} (2n+1) \left(\frac{r}{a}\right)^{n} \mathrm{P}_n(\vec{s}_i \cdot \vec{x}_p) \right] g(a\vec{s}_i)\, w_i, \tag{16}$$

where the notation is that used to define (15). (Note the change in the power of $\frac{r}{a}$ in this formula.) We shall refer to (16) as an inner sphere approximation.

The operations associated with forming and combining the outer and inner sphere approximations are essentially the same as those in two dimensions—one merely evaluates the component approximations at the integration points of the new approxi-

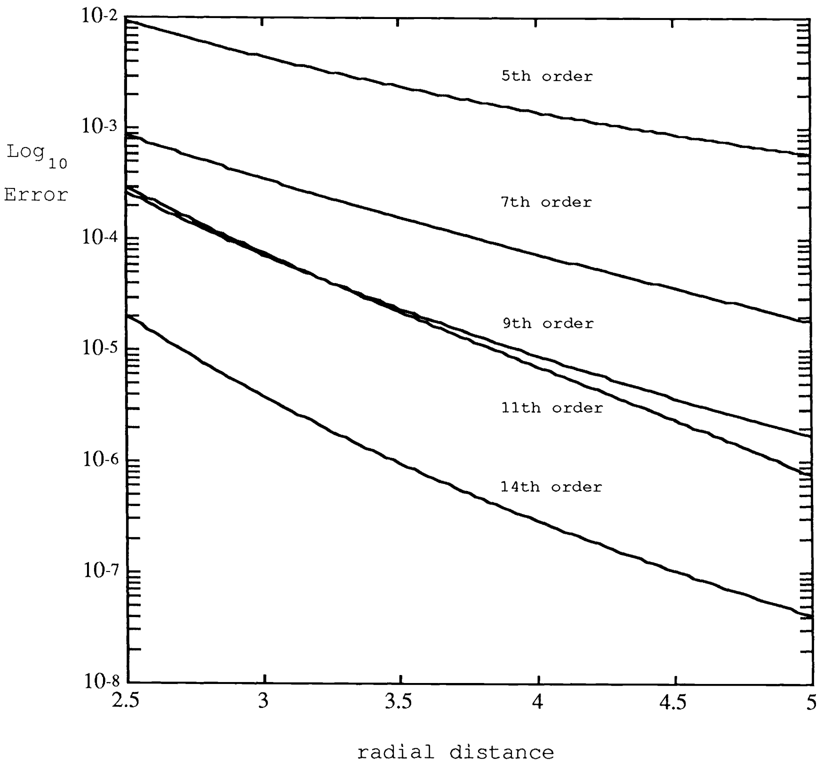


FIG. 5. *Error in the potential for an outer sphere approximation. Each curve represents the error for a given order of integration formula (and fixed number of terms in the Poisson kernel). The sphere was of radius* 3.

TABLE 1

Rate of the spatial decay of the error in the potential for an outer sphere approximation. The potential is assumed to decay like $O(\frac{1}{r})^\gamma$.

Order of Integration Method	Number of Terms in Kernel	Average Rate of Error Decay γ	Expected Rate of Error Decay γ
5	2	3.95	4
7	3	5.68	5
9	4	8.85	6
11	5	7.322	7
14	7	8.902	9

mation. If the integration points of the new approximation are sufficiently far away from the old approximation, then the accuracy degradation is minimal. One of the big advantages of using outer and inner sphere approximations is the simplicity of the process of combining them. This is in contrast to the complicated formulas that must be used if the approximations are based on spherical harmonics.

3. Hierarchical element implementation and multigrid. In this section we discuss an implementation strategy that uses the correspondence between the algorithmic structure of the fast multipole method [5] and the multigrid method. Although we shall discuss an implementation that uses computational elements based on the Poisson kernels (12) and (15), there is no explicit use of the particular features of this computational element, and so the results apply to programming with multipoles or any other type of element. Our goal here is to just outline the computational procedure that we have used (for more detailed information, one should see the programs in [1], [2]).

The basic algorithm structure used is that presented in [5]. The computation is broken down into two sweeps: the first sweep consists of constructing outer ring approximations for a hierarchical clustering of the computational particles. The second sweep consists of evaluating (in an efficient manner) the appropriate members of this hierarchy for each of the required evaluation points. There is no need for the evaluation points to be located at the same positions as the computational particles. In both sweeps the computational particles and evaluation points are assumed to lie in a bounded region Ω. A square box is then chosen which encloses Ω. The box is then recursively divided into smaller boxes. At the 0th level we have the original box, at the next level 4 boxes, the next 16, etc. Associated with each level n we have a uniform grid dividing up the original box into 4^n boxes. At the beginning of the calculation one chooses a highest (finest) level of discretization. We shall call this level n_f.

In the first sweep, outer ring approximations are constructed to represent the potential induced by the computational particles in each of the boxes at every level. This construction is performed recursively. Starting at the finest level, outer ring approximations are constructed for the computational particles in each of the boxes at that level. The center of the outer ring approximations are located at the centers of the respective boxes. The radius for the ring approximation is taken to be twice the size of the box. One then proceeds to the next lower (coarser) level and forms outer ring approximations for each of the particles contained within these coarser boxes. Instead of forming an outer ring approximation directly from the particles contained within a given box, one combines the four outer ring approximations associated with the finer boxes contained within the particular coarse box. The process of combining outer ring approximations is accomplished by evaluating the finer level outer ring approximations at the integration points of the coarse level outer ring approximation. Again, the centers of the outer ring approximations at this coarser level are located at the centers of their respective boxes and their radius is twice the coarse box size. This procedure is then repeated for all successively coarser levels. When the sweep is completed, outer ring approximations are available for the particles within each box at every level.

Two issues to be concerned with in implementing this sweep are avoiding unnecessary work and avoiding unnecessary storage requirements. A straightforward implementation would suggest that at each level one constructs an outer ring approximation for each box. If one uses K nodes in the approximation (12), then for n_f levels the storage required will be approximately $K \times (4^{(n_f+1)}/3)$ real numbers. Also, the work to construct all of these approximations for N uniformly distributed particles will be on the order of $K \times N + K^2\,(4^{(n_f+1)}/3)$. While this amount of work and storage is necessary if the particles are uniformly distributed, this amount is certainly not needed if they are not. If a given box (at any level) has no computational particles, then one need not compute an outer approximation for that box. Therefore, for nonuniform distributions (which are perhaps more prevalent in applications) one can save on storage and time by accounting for this fact.

The construction carried out in this first sweep is very similar to the computation that is performed in one sweep of a multigrid method. Both methods progress through a nested set of grids or boxes, and the operation of forming an outer approximation on a given level from the four outer approximations at the previous finer level is analogous to implementing the restriction operator (the forming of a coarse grid approximation from a fine grid approximation) in a multigrid method. However, in a hierarchical element method one is dealing with outer approximations associated with the grid boxes rather than solution values, so the programming strategy is to implement the equivalent of a multigrid restriction operator, but with this operator defined to act on approximations *pointed to* by address values in the boxes rather than acting on solution values. To carry out this computation a multigrid pointer structure F_n consisting of a nested set of integer arrays is first constructed. At a given level n, F_n is an integer array of size $2^n \times 2^n$. The (i,j)th element of F_n is the starting address in a storage array of the values used in the outer ring approximation associated with the (i,j)th box of the nth level partition of the computational domain. If there is no outer ring approximation associated with that box, then a negative integer is stored in the pointer array.

The computation is now performed as follows. The particles are sorted according to which box at the finest level (level n_f) they reside in. Particles in the same box are link-listed together, and the address of the first element of the link list for any given box is stored in a $2^{n_f} \times 2^{n_f}$ integer pointer array I_{n_f} associated with the boxes at the finest level. (By link-listing we mean that we associate with each particle an integer; the value of this integer gives the address of the next particle in the same box.) If there are no particles in a given box, then the value stored in I_{n_f} is a negative integer. Outer approximations are constructed for each of the boxes at the finest level. This is accomplished by looping over the array I_{n_f}. If there is a nonnegative element in I_{n_f}, then there are particles in the corresponding box and the potential induced by these particles are evaluated to obtain the values necessary for an outer ring approximation. These outer ring potential values are put into a storage array and the beginning address of these values is stored in the array F_{n_f}. Next, outer ring approximations are constructed for each coarser level. If we are at level k, then the pointer array corresponding to the finer level F_{k+1} is looped over. If there is an outer ring approximation associated with a given box at the finer level (indicated by a nonnegative entry in the pointer array), then the box at the coarser level which contains this finer level box (its parent box) will have an outer ring approximation associated with it. The potential induced by the finer level outer ring approximation is evaluated to construct the coarse box outer ring approximation. These new outer ring approximation values are put into a storage array and the beginning address of the values is stored in the F_k. A pictorial diagram of this process is depicted in Fig. 6. This process of constructing outer ring approximations is repeated until the second level is reached. (No outer ring approximations are needed for the first or zeroth levels.)

The second sweep, that in which the evaluation of the outer ring approximations is organized, utilizes the concept of being "well separated" [5]. Assume that the boxes on a given level are ordered by elements of $\mathbf{Z} \times \mathbf{Z}$; then, for a given level of refinement, one box is "well separated with distance L" from another box, if the maximum of the difference between their indices is greater than or equal to L (i.e., the boxes are at least L boxes apart). For a given level, the second sweep consists of constructing inner ring approximations for each box. The inner ring approximations are used to represent the potential from two sources. The first source is the inner ring approximation associated with the parent box at the previous coarser level. The second source is

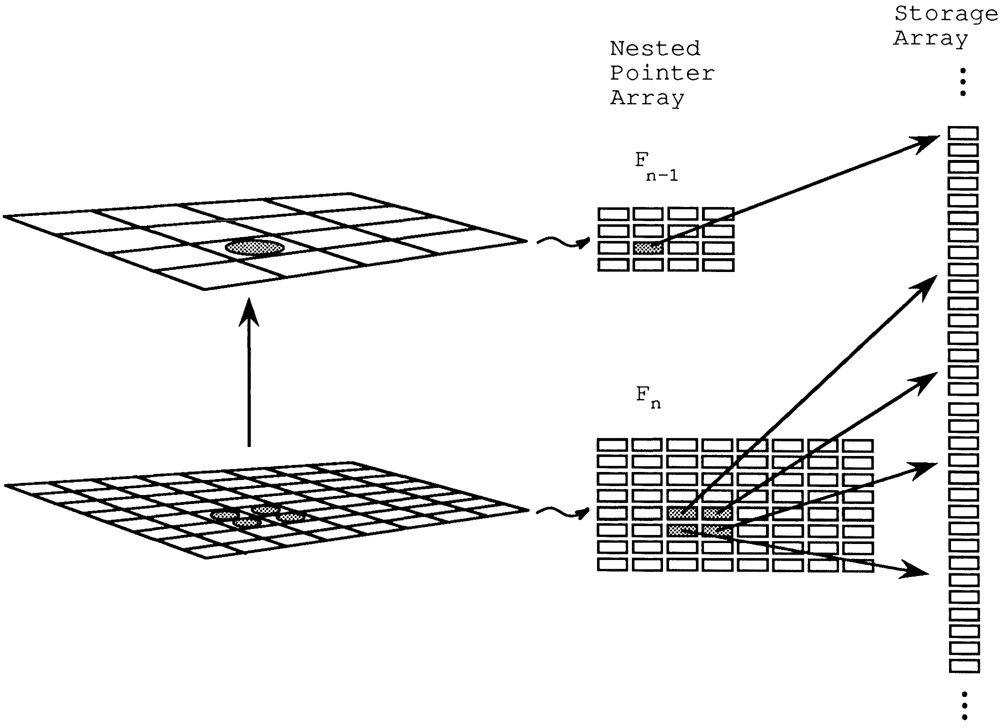


FIG. 6. *Schematic of outer ring construction. Formation of the coarse level outer ring approximation is performed by combing the four finer outer ring approximations beneath it. The storage array addresses for the outer ring approximation values are located in the nested arrays F_n.*

from all of the outer ring approximations from boxes that are well separated from the given box (with distance 1 in two dimensions and distance 2 in three dimensions) and are contained within boxes at the previous coarser level that are not well separated from the parent box. A graphical representation of this step is given in Fig. 7. This procedure is carried out for all levels except the finest. For this level, the inner ring approximations are only constructed from the inner ring approximations of the parents. At every level, the centers of the inner ring approximations are coincident with the centers of the boxes with which they are associated. The radius of the ring used in the approximation is one-half the box size. With the completion of this sweep, the potential at any given evaluation point is obtained by computing the potential of the inner ring approximation associated with the finest level box the point resides in. This potential is then added to the potential induced by particles in the nearby finest level boxes that are not well separated from the given evaluation points box. The reason for choosing a separation distance $L = 1$ in two dimensions and $L = 2$ in three dimensions is to ensure that the decay of the error in the approximations behaves like $O(\sqrt{2}/3)^\gamma$ where γ is determined by the number of modes kept in the kernel M. (See formula (12).)

As is the case with the first sweep, one must avoid using unnecessary storage and doing unnecessary work. Here, at any given level, if there is no evaluation point within a box, then one need not compute an inner ring approximation for that box. Work and storage space can be reduced by accounting for this fact.

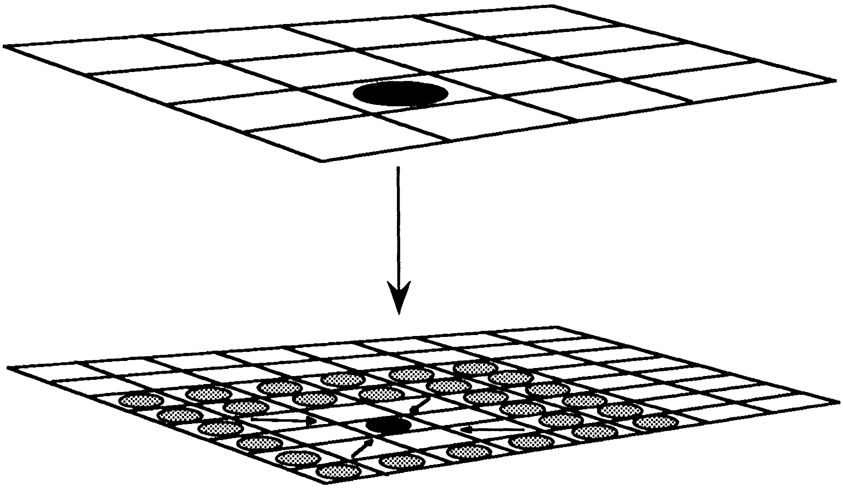


FIG. 7. *Formation of inner ring approximations from parent inner ring approximation and well-separated outer ring approximations.*

The construction carried out in this second sweep is also very similar to the computations that are performed in one sweep of a multigrid method. The process of creating an inner approximation from potential values induced by an inner ring approximation associated with a parent box is much like the prolongation operation of multigrid, i.e., the construction of a fine grid solution from a coarse grid solution. The process of combining the outer ring approximations from well-separated boxes is analogous to performing relaxation. Again, in this sweep we are working with approximations associated with the grid boxes rather than solution values, and so the programming strategy will be to implement multigrid-type operators, but with the operators defined to act on the approximations pointed to by address values in a given box rather than solution values. To implement this sweep, we again use a nested pointer array G_n similar to F_n. At a given level n the size of G_n is $2^n \times 2^n$ and the value at the (i,j)th element of this array is the starting address (in a storage array) of the inner ring approximation values associated with the (i,j)th box of the nth level partition. If there is no inner approximation associated with that box, then a negative integer is stored in the pointer array.

The first task in the actual computation is to identify which boxes will have inner ring approximations associated with them. The general rule is that if a box at any level has an evaluation point in it, then an inner ring approximation will be necessary for that box. The evaluation points are first sorted according to which box at the finest level (level n_f) they reside in. Particles in the same box are link-listed together, and the address of the first element of the link list for any given box is stored in an integer pointer array J_{n_f} associated with the finest level boxes. Next, the pointer structure G_n is swept through, starting at the finest level. At each level, a zero is stored in the pointer array if the corresponding box in the computational space with which it is associated contains at least one evaluation point. (This can be checked by considering the presence of zeros in the pointer array locations of the next finer level.) Once this initial sweep is performed, then any of the boxes whose pointer is zero indicates that an inner ring approximation will have to be formed for that box.

Now, starting at the coarsest level, each level of the nested pointer array G_n is swept through. If there is a zero in the pointer array location at a given level, then an inner ring approximation is constructed from a parent inner ring approximation and from the appropriate well-separated outer ring approximations. These approximations are combined by merely evaluating the potential they induce at the nodes of the inner ring approximation under construction. The addresses of the values needed to evaluate the parent inner ring approximation and the outer ring approximations are obtained from the pointer arrays G_n and F_n at the appropriate level. The values for this inner ring approximation being constructed are put in a storage array and the beginning address of these values is stored in the pointer array G_n. At the last (finest) level, inner ring approximations are formed from the parent inner ring approximations. The potential at the evaluation points is then obtained from these finest level inner ring approximations and direct evaluation of the potential from particles in nearby (not well separated) boxes. The particles in nearby boxes are accessed via the initially constructed link list whose starting addresses are pointed to by the entries in J_{n_f}.

The entire procedure consists of these two sweeps. The first sweep goes from a fine discretization of physical space to a coarse discretization and results in the construction of a hierarchy of outer ring approximations. The second sweep goes from a coarse discretization to a fine one and results in a hierarchy of inner ring expansions. If one considers the operations being performed on the pointer arrays, then the method is algorithmically similar to a multigrid V-cycle. In fact, the first version of the code was obtained by modifying an existing multigrid code. The modification consisted of changing the subroutines used in multigrid (like a relaxation routine) so that, instead of working with the values at the cell centers, the routine worked with the approximations that were pointed to by the values at cell centers. It proved to be very productive to use the programming techniques developed for multigrid, specifically those embodied in the sample code given in [14].

It is clear that if the computational particles and the evaluation points are distributed in a region that is not well represented by a square, then there will be a great number of elements in the pointer arrays that will be associated with no outer or inner approximations. In order to reduce storage space and shorten loops over the pointer structure, one can allow for regions that are nonsquare. This is most easily accomplished if the bounding region is constructed to be a rectangular box whose sides have an aspect ratio that is a power of two. To implement this, a rectangular box is constructed that contains the computational particles and the evaluation points. The sides of this box are then enlarged to create a rectangle that has an appropriate aspect ratio. The refinement of this rectangle proceeds by successively dividing the larger side by powers of two until the box side is equal to the length of the smaller side. From this point on, the refinement is accomplished by refining the boxes in both directions. (See Fig. 8, which depicts the refinement of a box with aspect ratio four to one.) The corresponding nested set of pointer arrays F_n, G_n, I_{n_f}, and J_{n_f} will be nonsquare and contain a number of elements corresponding to the number of boxes at each level of the decomposition of the original rectangle.

In three dimensions we found that using an outer sphere radius that was three times the box size was more accurate than using one that was only twice the box size. Other than this difference and the changes in the number of indices in the pointer arrays and the form of the kernel in Poisson's formula, the implementation of

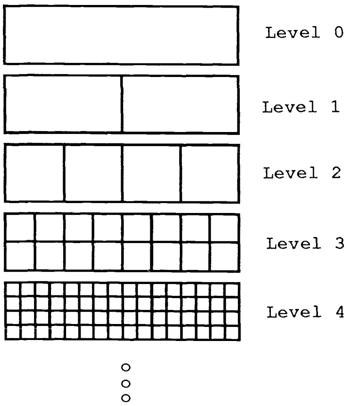


FIG. 8. *Refinement of a rectangle with an aspect ratio of* 4 *to* 1.

the three-dimensional method is identical to the two-dimensional method described above.

The particular implementation strategy just discussed works best when a large number of levels is not required in the computation. This will typically be the case when one requires a modest or high amount of accuracy. (The computations in the next section, performed with a tolerance of 1×10^{-4}, demonstrate this fact.) When one is interested in computations with a less severe accuracy criterion, then a larger number of levels is called for, and the multigrid pointer structure may be inefficient. In such cases other implementation strategies, such as those that use a tree-based data structure, should be considered. See, for example, the methods described in Barnes and Hut [3] or Hernquist [8].

4. Timing and level selection. As discussed in the introduction, the primary component of a hierarchical element method, which leads to a computational savings, is the representation of the potential induced by a large number of particles by a single computational element that requires less work to evaluate. On the other hand, it is clear that if one uses a computational element to represent the potential induced by only a small number of particles, then this may lead to more, rather than less, computational work. In the implementation of the method presented here, the computational domain is broken up into boxes, and as one increases the level of refinement, the size of these boxes decreases. Since the number of particles per box will ultimately decrease, and thus the number of particles per computational element will decrease, there is some level for which it is more expensive to use a hierarchical element method than to use the direct method. This is illustrated by the results in Figs. 9 and 10. (All of the computations where carried out on a SUN Sparcstation 1. There were 25 points in the ring approximations—a number of points that provides an accuracy of 10^{-4}.) The figures give the time it takes to compute the potential induced by all of the particles at the locations of all the other particles for different numbers of particles. (These times are estimated by the procedure described below.) For Fig. 9 the particles were uniformly distributed in a square and for Fig. 10 the particles were uniformly distributed in a rectangle with aspect ratio of ten to one. Each of the separate lines in the figures correspond to different finest levels of domain discretization. As is evident from the figures the level one should choose depends on

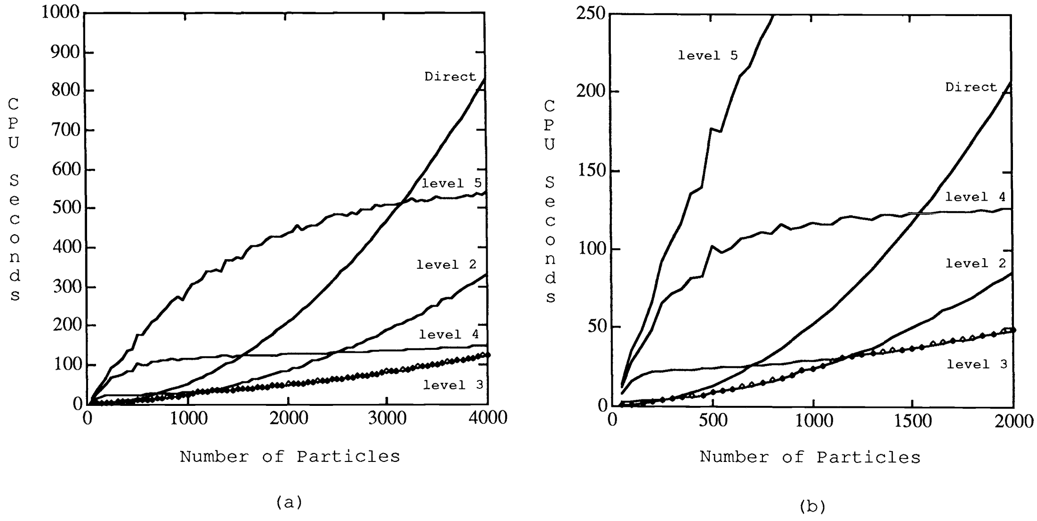


FIG. 9. CPU *time for a hierarchical element method based on ring approximations. The particles are distributed uniformly in a unit square. Each of the solid curves represents the estimated* CPU *time used by the method for a given level of spatial refinement. The circles represent the actual time taken when the minimal estimate is chosen.* (b) *is a magnified version of* (a).

the number of particles. The more particles there are, the higher the level of refinement. The choice of level also depends on the distribution of particles. The fact that the timings vary greatly as the level is changed indicate that it is crucial to choose the appropriate level. If the particles are always uniformly distributed in some particular geometric arrangement, then one could work out estimates of the appropriate level of refinement one should choose. If the particles are not distributed in such a way, then the construction of a scheme for identifying the appropriate level seems difficult. However, a computationally efficient and optimal level selection scheme can be constructed.

The key observation is that the amount of computational work at each level in the hierarchical element method is completely determined (and hence one can estimate it) by the number of computational particles and evaluation points in any given box at any given level. In essence, the selection scheme is to "dry run" the hierarchical element method, and, instead of performing all the required operations, just increment counters. The counter increments are based on the population density of the particles in each box and models of the computational time necessary to carry out specific computational tasks. Timings are then estimated for the work required for different levels of refinement, and the level selected is the one with the least amount of estimated time. It is not necessary to perform a different timing computation for every level of refinement—the recursive nature of the method allows one to compute the amount of work for each level up to some ultimate level in one computation.

Before presenting the details of the implementation of this selection procedure, we present some results. In Table 2 we present the CPU time estimate (in seconds) from our selection code and the actual CPU time taken for a computation of the potential induced by 2000 points distributed in a unit square. As can be seen, one can get rather good estimates of the time it takes for any given level. The cost of performing

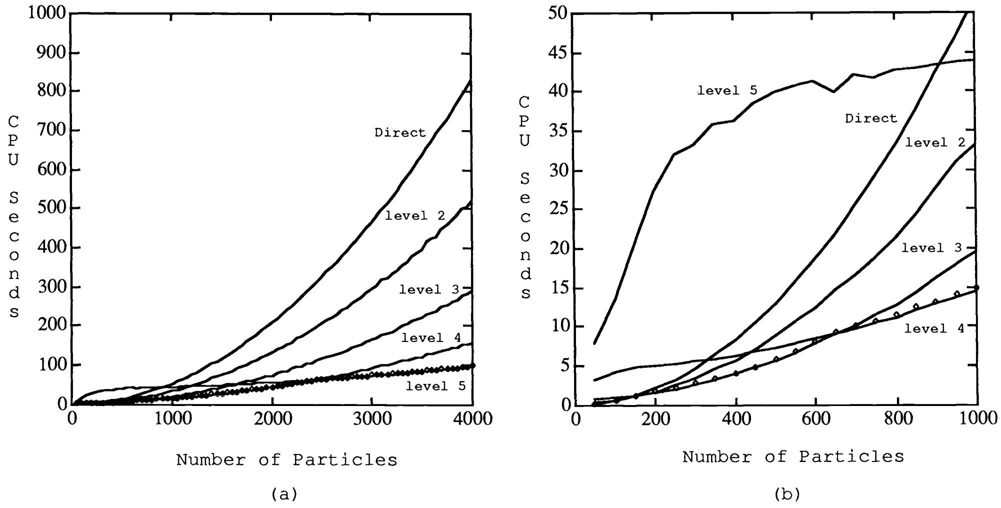


FIG. 10. CPU *time for a hierarchical element method based on ring approximations. The particles are distributed uniformly in a rectangle with an aspect ratio of* 10 *to* 1. *Each of the solid curves represents the estimated* CPU *time used by the method for a given level of spatial refinement. The circles represent the actual time taken when the minimal estimate is chosen.* (b) *is a magnified version of* (a).

TABLE 2

Estimated and actual CPU *time for a hierarchical element method based on ring approximations. The timings are for* 2000 *particles distributed uniformly in a unit square.*

Level	Estimated CPU Time (Secs.)	Actual CPU Time (Secs.)
Direct	207.3	206.8
2	85.5	86.0
3	48.1	48.4
4	126.4	123.5
5	443.2	430.9

this estimation is very small. For 2000 points it takes .6 CPU seconds to estimate the running time for six levels of refinement. This time is about 1.25 percent of the time it takes to actually perform the calculation. Figures 9 and 10 also illustrate the accuracy of the timing estimates. The circles in these figures correspond to the actual CPU time taken to perform the computation when the level with the least amount of estimated time is selected. As desired, these circles coincide with a lower envelope for all of the timing curves in the figure.

In the implementation of the method for approximating the computational time we use a nested set of integer arrays set up like F_n and G_n of §3 (a nested set of arrays whose individual elements at any level are associated with individual squares in the decomposition of physical space at that same level of refinement). We have one nested set of arrays S_n associated with the computational particles and one nested set of arrays T_n associated with the evaluation points. At each level of these nested arrays we store in the (i,j)th element the number of particles (or evaluation points) contained within the (i,j)th box in computational space at that level. To construct these population values, a finest level of refinement is chosen and the number of particles in each box is stored in the corresponding array element of S_n associated with that level. Next, the other levels of the array S_n are swept through, and the number of particles in each box is determined by summing the number of particles in each of the boxes at the finer level which is contained within the given box. The same procedure is carried out for the evaluation points and the nested array T_n.

In order to use the population density information in the arrays S_n and T_n, we need models of the computational time necessary to carry out particular operations of the method. Assume there are n computational particles and m evaluation points. In two dimensions, if we are using an approximation of the type (12) with K points, then the models used are:

(0) *direct interaction* : $c_0 \times n \times m \ + b_0$,
(1) *formation of outer approximations* : $c_1 \times n \times K \ + b_1$,
(2) *combining; formation of outer approximations*
from outer approximations, etc. : $c_2 \times K \times K \ + b_2$,
(3) *evaluation of finest level inner approximations* : $c_3 \times m \times K \ + b_3$.

These models depend upon values of the constants $b_0, c_0, b_1, \cdots, b_3, c_3$. We call these formulas models because they are not exact—peculiarities in specific computational hardware or compilers may mean that the constants are not truly constants. However, on the machines the author has worked with, these formulas capture rather well the actual amount of computational time for a wide range of the parameters n, m, and K. In order to estimate the values for these constants, a small program was written which times the specific operations. Using the results of these timing estimates for several values of n, m, etc., the constants in the models were determined by a least squares procedure.

In three dimensions the models are slightly different because the finite sum in the integral kernel in Poisson's formula is not represented by a single closed formula. Assuming n computational particles, m evaluation points, K integration points, and M terms in the Poisson kernel, we use the following computational models:

(0) *direct interaction* : $c_0 \times n \times m \ + b_0$,
(1) *formation of outer approximations* : $c_1 \times n \times K \ + b_1$,
(2) *combining; formation of outer approximations*
from outer approximations, etc. : $c_2 \times K \times K \times M \ + b_2$,
(3) *evaluation of finest level inner approximations* : $c_3 \times m \times K \times M \ + b_3$.

Here, as before, values for these constants are obtained by using a least squares fit to the output of the program which times the particular operations.

With the computational model constants determined, complete timing estimates are obtained in two sweeps. In the first sweep the amount of computational time needed to construct the hierarchy of outer ring approximations is estimated. Let n_f denote a finest level of discretization chosen for the timing. (This level should always be greater than the finest level chosen for the actual computation.) Starting at this finest level, the time it takes to construct the outer ring approximation is estimated by using the model (1) and information in the array S_n at the finest level. If there is a nonzero value in any element of this array, then an outer ring approximation will be constructed for the particles associated with the corresponding box. The work to construct this outer approximation is estimated using (1), with the value of n being the value in that particular array element of S_n (i.e., the particle count for the box). Each of these estimates is added together to get a complete timing estimate for the construction of the outer approximations at the finest level. One then proceeds to the next coarser level. At this level two computational time estimates are computed. The first estimate is that of the time necessary to construct all of the outer approximations assuming that this particular level is the starting level—the same procedure as for the finest level. The second estimate is that of the time necessary for combining outer ring approximations from the finer level into the outer ring approximations at the current level. In this second estimate, the current level of S_n is swept through and if any element is nonzero, then an outer ring approximation must be constructed for the corresponding box. The time to compute this construction is obtained by using the model (2) and determining the number of finer level outer ring approximations that must be used in the construction. (This information can be obtained from the presence of a nonzero value in the next finer level elements of S_n.) One then proceeds to successively coarser levels, forming these same two estimates, until level one is reached. At this point one has information about the time necessary to form outer ring approximations starting from any level, as well as the time to create outer ring approximations from outer ring approximations at previous levels. By summing appropriate combinations of these timings, one can compute the time necessary to construct the complete hierarchy of outer ring approximations assuming one starts at any level. As a specific example, assume that 4000 points are uniformly distributed in a unit square. Then if the finest level chosen is $n_f = 5$, one obtains timing estimates such as those given in the second column of Table 3. Each entry in this column is the time it takes to compute the complete hierarchy of outer ring approximations starting from the given level.

In the second sweep, estimates of the required time are made for the construction of the hierarchy of inner ring approximations, as well as the time for direct local interaction. Starting at level one, the amount of work required for a direct calculation is estimated. This is obtained using model (0) and computing n and m by summing the values in the elements of the array S_1 and T_1 associated with level 1. One then proceeds to higher (finer) levels. At each finer level two estimates are computed. The first is an estimate of the time necessary to form an inner ring approximation from nearby outer ring approximations at the same level and the inner ring approximation at the previous coarser level. This time can be estimated by sweeping through the array T_n and determining whether or not an inner expansion will be constructed for any of the boxes in the computational domain. A construction for any box will be necessary if there is a nonzero value in the corresponding element in this array. One then employs the model (2) and determines the number of outer ring approximations and coarse inner ring approximations used in this construction from values in S_n at current level and values in T_n at the previous coarser level. Secondly, an estimate of the time necessary if one stops at the current level (i.e., the time to evaluate the inner expansions at this level as well as the local direct computation time) is formed. This time can be computed using the models (0) and (3) and population information for the evaluation points contained in the array T_n. Both of these timing estimates are performed for successively finer and finer levels. The computation is stopped at the finest level n_f. After this computation, we have, for every level, the time it takes to construct the inner ring approximations and the time it takes to use that level for the evaluation of the potential from the inner ring approximations and local direct calculations. By summing the appropriate values of these timings, it is possible to compute the time it takes to construct and evaluate the hierarchical set of inner approximations for each level from the first to the finest. In the example above

TABLE 3

Estimated CPU *time for a hierarchical element method based on ring approximations. The timings are for* 4000 *particles distributed uniformly in a unit square.*

Level	Outer Ring Const. Time (Secs.)	Inner Ring Const. and Eval. Time (Secs.)	Total Time (Secs.)
Direct	0.0	829.0	829.0
2	1.6	331.5	333.1
3	2.6	122.1	124.7
4	6.6	142.2	148.8
5	22.5	509.4	531.9

with 4000 points, the timings for this second step are presented in the third column of Table 3.

The complete estimate of the time it takes for the whole method consists of adding the amount of work to construct the outer ring approximations and the amount of work to construct and evaluate the inner ring approximations. The level one chooses for the actual computation is the level for which this total is the least. For example, an estimate of the total time is obtained by summing the elements (across the row) in the second and third column of Table 3. These times are presented in the fourth column of that table. One should choose level 3 for this computation.

5. Discussion and conclusions. The model problem discussed in this paper has been the computation of the potential induced by N charged particles. In many calculations one is interested in calculating the derivatives of this potential and not the potential itself. There are two ways to use a hierarchical element method to determine the derivatives of the potential. One way is to first create the complete hierarchy of inner ring and outer ring approximations and then, at the last step, to compute the derivatives at an evaluation point by summing the derivatives of the potential induced by nearby particles (particles that are not contained in well-separated boxes) and the derivative of the potential induced by the inner ring expansion associated with the finest level box the evaluation point resides in. The local interaction is computed by using the analytic derivative of (3) or (4). The derivative of the inner ring expansion is computed using the derivative (either numerical or analytical) of the approximation (13).

The other way to evaluate derivatives consists of computing each derivative by a separate hierarchical element computation. Each derivative of the potential induced by N charged particles is identical to a potential induced by N dipoles. Since the hierarchical element method requires only that the interaction be governed by solutions to Laplace's equation, we can use the method to compute the potential defined by a distribution of N dipoles. Each component of the derivative is therefore obtained by a separate computation and the formulas used for the initial ring construction and local direct evaluation are the analytic expressions for the derivatives of the potential.

In two dimensions, it seems best to use the first technique, i.e., compute the potential and then differentiate the result. While there is a loss of one order of accuracy in this procedure, this can be compensated for by increasing the accuracy with which the potential is calculated. (This is accomplished by increasing the number of integration points in the ring approximations, or by using more terms in a multipole approximation.) The extra cost to preserve accuracy is very small compared to performing completely separate computations for each derivative. The first approach is also likely to be better in three dimensions if one just wants the derivatives of a scalar potential. However, if one is interested in computing the velocity induced by a distribution of vorticity in three dimensions, then the second approach is more efficient. In this case, the velocity field is determined by the curl of a vector potential. To implement the first method, one must determine the three separate components of the vector potential by three different computations and then evaluate the velocity by taking the curl of the resulting function. In the second approach, each component of the velocity is determined directly via separate computations and there is no need to differentiate the result. Thus, one saves on both work and accuracy by using the second approach.

One is of course interested in a comparison between the multipole method and the method based on the use of Poisson's formula. Since the method that uses Poisson's formula is in some sense the discrete transform of the multipole method, one cannot expect there to be any great computational advantages in using one method over the other, i.e., they basically give the same results. The reason for considering using a method based on Poisson's formula is that it is perhaps more convenient to program and use. Since the operations for combining and constructing the hierarchical elements involve only function evaluation, they are easy to formulate and to program. Furthermore, these operations are essentially the same in two and three dimensions and this makes the implementation of the method in three dimensions very easy. A hierarchical element method based on Poisson's formula can also easily treat the problem of determining the potential induced by collections of sources which are more general than point charges. For example, one might be interested in computing the potential induced by collections of segments with some given charge distribution on them. The difficulty with using the multipole method for this computation occurs in the first step. In this step one needs to form a multipole approximation induced by collections of segments. While for point charges this initial multipole approximation can be analytically determined from the locations of the charges and their strengths, this may not be possible (or if possible, then it may be rather complicated) for segments. To get an initial ring approximation for such elements one need only evaluate the potential induced by these distributions at the integration points of the ring approximation. One disadvantage of using a Poisson formula representation is that techniques [7] for further reducing the computation time of the fast multipole method in three dimensions do not appear to be directly applicable.

For either the method presented here, or the original multipole method, one can and should take advantage of the inherent multigrid structure. This inherent structure allows one to take advantage of the variety of techniques that have been formulated for programming multigrid. We have used the techniques for programming on single processor machines, but there is undoubtedly some benefit to using multigrid techniques for multiprocessor computers. Also, as seen in §3, the timing of the hierarchical element method is very important. From the results of that section it is easy to see why some early experimenters with the method (ourselves included), who naively chose a high level of refinement, were not particularly pleased with the hierarchical element method performance. Fortunately, the recursive nature of the method allows for efficient and accurate timing estimates that can be used to select the appropriate level of refinement. The three contributions of the paper, the use of Poisson's formula, the use of multigrid programming strategy, and the use of timing estimates to obtain good level selection, certainly do not exhaust the set of tasks that could be done to improve on the hierarchical element method. Other improvements could come by making the method more adaptive, or improving the speed at which the component operations can be carried out. Such procedures will hopefully lead to even more efficient algorithms.

REFERENCES

[1] C. R. Anderson, HELM2: *A hierarchical element method in two dimensions*, UCLA Report CAM 90-15, Department of Mathematics, University of California, Los Angeles, CA, 1990.

[2] ———, HELM3: *A hierarchical element method in three dimensions*, UCLA Report CAM 90-16, Department of Mathematics, University of California, Los Angeles, CA, 1990.

[3] J. Barnes and P. Hut, *A hierarchical $O(N \log N)$ force-calculation algorithm*, Nature, 324 (1986), pp. 446–449.

[4] R. Courant and D. Hilbert, *Methods of Mathematical Physics*, Vol. I, Wiley Interscience, New York, 1953.

[5] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comp. Phys., 73 (1987), p. 325.

[6] ———, *Fast methods in three dimensions*, in Vortex Methods, Lecture Notes in Mathematics 1360, C. Anderson and C. Greengard, eds., Springer-Verlag, Berlin, New York, 1988.

[7] ———, *On the efficient implementation of the fast multiple algorithm*, Department of Computer Science Report 602, Yale University, New Haven, CT, 1988.

[8] L. Hernquist, TREESPH: *A Unification of* SPH *with The Hierarchical Tree Method*, Astrophysical Journal Supp., 64 (1987), p. 715.

[9] J. Katzenelson, *Computational structure of the N-body problem*, SIAM J. Sci. Statist. Comput., 4 (1989), pp. 787–815.

[10] A. D. McLaren, *Optimal numerical integration on a sphere*, Math. Comput., 17 (1963), pp. 361–383.

[11] Z. P. Nowak, *Panel clustering technique for lifting potential flows in the three space dimensions*, in Panel Methods in Mechanics, J. Ballman, R. Eppler, and W. Hackbusch, eds., Notes in Numerical Fluid Mechanics, 1360, Vieweg-Verlag, Braunschewig, Wiesbaden, 1987.

[12] V. Rokhlin, *Rapid solutions of integral equations of scattering theory in two dimensions*, J. Comp. Phys., 86 (1990), pp. 414–439.

[13] A. H. Stroud, *Approximate Calculation of Multiple Integrals*, Prentice-Hall, Englewood Cliffs, NJ, 1971.

[14] K. Stuben and U. Trottenberg, *Multigrid methods: Fundamental algorithms, model problem analysis and applications*, in Multigrid Methods, W. Hackbusch and U. Trottenberg, eds., Lecture Notes in Mathematics 960, Springer-Verlag, Berlin, New York, 1982.

[15] L. Van Dommelen and E. Rundensteiner, *Fast adaptive summation of point forces in two-dimensional Poisson equations*, J. Comp. Phys., 83 (1989), pp. 126–147.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 4, pp. 948–966, July 1992

ON THE SPECTRUM OF A FAMILY OF PRECONDITIONED BLOCK TOEPLITZ MATRICES*

TAKANG KU† AND C.-C. JAY KUO†

Abstract. Research on preconditioning Toeplitz matrices with circulant matrices has been active recently. The preconditioning technique can be easily generalized to block Toeplitz matrices. That is, for a block Toeplitz matrix T consisting of $N \times N$ blocks with $M \times M$ elements per block, a block circulant matrix R is used with the same block structure as its preconditioner. In this research, the spectral clustering property of the preconditioned matrix $R^{-1}T$ with T generated by two-dimensional rational functions $T(z_x, z_y)$ of order (p_x, q_x, p_y, q_y) is examined. It is shown that the eigenvalues of $R^{-1}T$ are clustered around unity except at most $O(M\gamma_y + N\gamma_x)$ outliers, where $\gamma_x = \max(p_x, q_x)$ and $\gamma_y = \max(p_y, q_y)$. Furthermore, if T is separable, the outliers are clustered together such that $R^{-1}T$ has at most $(2\gamma_x + 1)(2\gamma_y + 1)$ asymptotic distinct eigenvalues. The superior convergence behavior of the preconditioned conjugate gradient (PCG) method over the conjugate gradient (CG) method is explained by a smaller condition number and a better clustering property of the spectrum of the preconditioned matrix $R^{-1}T$.

Key words. block Toeplitz matrix, preconditioned conjugate gradient method

AMS(MOS) subject classifications. 65F10, 65F15

1. Introduction.

The systems of linear equations associated with block Toeplitz matrices arise in many two-dimensional digital signal processing applications, such as linear prediction and estimation [9], [12], [13], image restoration [7], and the discretization of constant-coefficient partial differential equations. To solve the block Toeplitz system $T\mathbf{u} = \mathbf{b}$, where T is an $N \times N$ matrix with $M \times M$ blocks, by direct methods, such as Levinson-type algorithms, requires $O(M^3N^2)$ operations [2], [14], [17]. Recently, there has been active research on the application of iterative methods such as the preconditioned conjugate gradient (PCG) method to the solution of Toeplitz systems. To accelerate the convergence rate, various preconditioners have been proposed for symmetric positive definite (SPD) Toeplitz matrices [6], [8], [10], [15]. The proposed preconditioning techniques can be easily generalized to block Toeplitz matrices. Simply speaking, we construct the preconditioner with a block circulant matrix R that has the same block structure as T. Since both $R^{-1}\mathbf{w}$ and $T\mathbf{w}$, where $\mathbf{w}$ denotes an arbitrary vector of length MN, can be performed with $O(MN \log MN)$ operations via two-dimensional fast Fourier transform, the computational complexity per PCG iteration is $O(MN \log MN)$ only. The PCG method can be much more attractive than direct methods for solving block Toeplitz systems if it converges fast.

The convergence rate of the PCG method depends on the eigenvalue distribution of the preconditioned matrix $R^{-1}T$ [1]. Generally speaking, the PCG method converges faster if $R^{-1}T$ has clustered eigenvalues and/or a small condition number. The spectral properties of preconditioned point Toeplitz matrices have been extensively studied. Chan and Strang [3], [5] have proved that, for a Toeplitz matrix with a positive generating function in the Wiener class, the spectrum of the preconditioned matrix has eigenvalues clustered around unity except for a finite number of outliers. If the Toeplitz matrix is generated by a positive *rational* function

* Received by the editors November 14, 1990; accepted for publication (in revised form) June 3, 1991. This work was supported by the University of Southern California Faculty Research and Innovation Fund and by a National Science Foundation Research Initiation Award (ASC-9009323).

† Signal and Image Processing Institute and Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, California 90089-2564 (tkku@sipi.usc.edu and cckuo@sipi.usc.edu).

$A(z)/B(z) + A(z^{-1})/B(z^{-1})$ in the Wiener class, an even stronger result has been derived by Trefethen [16] and Ku and Kuo [11]. That is, if $A(z)$ and $B(z)$ are polynominals in z of orders p and q without common roots, the number of outliers is equal to $2\max(p,q)$ and the PCG method converges in at most $\max(p,q)+1$ iterations asymptotically (see [11] and the discussion in §4 below). Therefore, an $N \times N$ preconditioned rational Toeplitz system can be solved with $\max(p,q) \times O(N \log N)$ operations.

The spectral properties of preconditioned block Toeplitz matrices have not yet been very well understood. In this research, we analyze the spectral *clustering* property for a class of preconditioned block Toeplitz matrices. The block Toeplitz matrix under consideration has a two-dimensional quadrantally-symmetric generating sequence generated by a rational function in the Wiener class (see the definition in §3). We divide our discussion into two cases depending on whether or not the generating sequence is separable. When the block Toeplitz matrix has a separable generating sequence, the spectrum of the preconditioned block Toeplitz can be easily derived by using the preconditioned point Toeplitz result as given in [11]. However, we derive the preconditioned point Toeplitz result from a new viewpoint in this paper so that the same approach can be used for both separable and nonseparable cases. With this viewpoint, we interpret the operation $T\mathbf{w}$, where T is an $MN \times MN$ block Toeplitz matrix, as a two-dimensional constant-coefficient mask operating on a certain two-dimensional sequence construction based on $\mathbf{w}$.

Our main results can be summarized as follows. Let T be an $MN \times MN$ doubly symmetric block Toeplitz matrix generated by a rational function of order (p_x, q_x, p_y, q_y), $\gamma_x = \max(p_x, q_x)$ and $\gamma_y = \max(p_y, q_y)$. For the separable generating sequence case, the eigenvalues of $R^{-1}T$ are clustered together such that it has asymptotically $(2\gamma_x + 1)(2\gamma_y + 1)$ distinct eigenvalues. The PCG method converges asymptotically in at most $\gamma_x\gamma_y + 1$ iterations, and the complexity of the PCG method is therefore $O(MN \log MN)$. For the nonseparable generating sequence case, the eigenvalues of $R^{-1}T$ are clustered around unity except for at most $O(M\gamma_y + N\gamma_x)$ outliers. Since the number of outliers is proportional to M and N, rather than being $O(1)$ as in the point Toeplitz case, the convergence rate of the PCG method cannot be completely characterized by the number of outliers. The condition number $\kappa(R^{-1}T)$ should also be taken into account. For this case, the superior performance of the PCG method over the CG method is explained by a better spectral clustering property as well as a smaller condition number of the preconditioned matrix $R^{-1}T$.

The outline of this paper is as follows. The construction of the block circulant preconditioner R for block Toeplitz matrices T is presented in §2. In §3, we study the spectral clustering property of preconditioned block Toeplitz matrices $R^{-1}T$. Toeplitz matrices with separable and nonseparable generating sequences are examined, respectively, in §§3.1 and 3.2. Numerical results are given in §4 to assess the performance of the PCG method.

2. Construction of block Toeplitz preconditioners. Let T be a block Toeplitz matrix consisting of $N \times N$ blocks with $M \times M$ elements per block, which can be expressed as

$$T = \begin{bmatrix} T_0 & T_{-1} & \cdot & T_{2-N} & T_{1-N} \\ T_1 & T_0 & T_{-1} & \cdot & T_{2-N} \\ \cdot & T_1 & T_0 & \cdot & \cdot \\ T_{N-2} & \cdot & \cdot & \cdot & T_{-1} \\ T_{N-1} & T_{N-2} & \cdot & T_1 & T_0 \end{bmatrix}, \tag{2.1}$$

where T_n with $|n| \leq N-1$ are $M \times M$ Toeplitz matrices with elements

$$[T_n]_{i,j} = t_{i-j,n} \quad \text{where} \quad 1 \leq i, \quad j \leq M.$$

Note that T is also known as the doubly Toeplitz matrix. The $MN \times MN$ block Toeplitz matrix T is completely characterized by the two-dimensional sequence

$$t_{m,n} \quad \text{where} \quad |m| \leq M-1, \ |n| \leq N-1, \tag{2.2}$$

known as the generating sequence of T. To construct the preconditioner for T, we generalize the idea in [5], [10], and [15], and consider an $MN \times MN$ block circulant matrix of the form

$$R = \begin{bmatrix} R_0 & R_{N-1} & \cdot & R_2 & R_1 \\ R_1 & R_0 & R_{N-1} & \cdot & R_2 \\ \cdot & R_1 & R_0 & \cdot & \cdot \\ R_{N-2} & \cdot & \cdot & \cdot & R_{N-1} \\ R_{N-1} & R_{N-2} & \cdot & R_1 & R_0 \end{bmatrix}, \tag{2.3}$$

where R_n with $0 \leq n \leq N-1$ are $M \times M$ circulant matrices with elements

$$[R_n]_{i,j} = r_{(i-j) \bmod M,n} \quad \text{where} \quad 1 \leq i,j \leq M.$$

Thus, the block circulant matrix R is completely characterized by the two-dimensional sequence

$$r_{m,n} \quad \text{where} \quad 0 \leq m \leq M-1, \ 0 \leq n \leq N-1. \tag{2.4}$$

The construction of R based on T is described below.

In (2.1) and (2.3), linear operators T and R are expressed in matrix form. Block Toeplitz (or circulant) systems are in fact just one way to describe linear (or circular) convolutions between two two-dimensional sequences. In the current context, it is more convenient to characterize T (or R) in terms of the relationship between input and output vectors. Consider two arbitrary MN-dimensional vectors $\mathbf{w}$ and $\mathbf{v}$ related via $\mathbf{v} = T\mathbf{w}$. By using the natural rowwise ordering, we can rearrange elements of these vectors into two-dimensional sequences

$$w_{m,n} \text{ and } v_{m,n} \quad \text{where} \quad 0 \leq m \leq M-1, \ 0 \leq n \leq N-1. \tag{2.5}$$

Then, the block Toeplitz system $\mathbf{v} = T\mathbf{w}$ can be interpreted as a linear operator characterized by the two-dimensional mask

(2.6)
$$\begin{bmatrix} t_{M-1,-N+1} & t_{M-2,-N+1} & \cdots & t_{0,-N+1} & \cdots & t_{-M+2,-N+1} & t_{-M+1,-N+1} \\ t_{M-1,-N+2} & t_{M-2,-N+2} & \cdots & t_{0,-N+2} & \cdots & t_{-M+2,-N+2} & t_{-M+1,-N+2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ t_{M-1,0} & t_{M-2,0} & \cdots & t_{0,0} & \cdots & t_{-M+2,0} & t_{-M+1,0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ t_{M-1,N-2} & t_{M-2,N-2} & \cdots & t_{0,N-2} & \cdots & t_{-M+2,N-2} & t_{-M+1,N-2} \\ t_{M-1,N-1} & t_{M-2,N-1} & \cdots & t_{0,N-1} & \cdots & t_{-M+2,N-1} & t_{-M+1,N-1} \end{bmatrix}$$

operating on an extended sequence

$$\bar{w}_{m,n} = \begin{cases} w_{m,n}, & 0 \le m \le M-1, \quad 0 \le n \le N-1, \\ 0, & \text{otherwise.} \end{cases} \tag{2.7}$$

To compute the output element v_{m_0,n_0}, $0 \le m_0 \le M-1$, $0 \le n_0 \le N-1$, we put the center of the mask (i.e., $t_{0,0}$) on $\bar{w}_{m_0,n_0}$, multiply $\bar{w}_{m,n}$ with the corresponding coefficients t_{m_0-m,n_0-n}, and sum the resulting products. Now let us use the mask (2.6) to operate on a periodic sequence

$$\tilde{w}_{m,n} = w_{m \bmod M, n \bmod N}, \qquad -\infty < m < \infty, \quad -\infty < n < \infty. \tag{2.8}$$

This defines a block circulant matrix-vector product $R\mathbf{w}$, which is close to the operation $T\mathbf{w}$. Since $R^{-1}\mathbf{v}$ can be computed efficiently with two-dimensional fast Fourier transform, it is natural to use R as a preconditioner for T.

The characterization of a block Toeplitz or circulant matrix by a two-dimensional operator mask is not new. It is basically the same as the stencil form used in the finite-difference discretization of the constant-coefficient partial differential operator. For example, the five-point stencil discretization of the Poisson equation can be interpreted as the mask

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

operating on a two-dimensional sequence. By assuming the Dirichlet and periodic boundary conditions, we obtain block Toeplitz and circulant matrices, respectively.

The preconditioner constructed above can be described in matrix notation. First, for every point Toeplitz matrix T_n, $|n| \le N-1$, we construct a circulant preconditioner $\mathcal{R}_n$ with

$$t_{0,n}, \; t_{-1,n} + t_{M-1,n}, \; t_{-2,n} + t_{M-2,n}, \; \cdots, \; t_{1-M,n} + t_{1,n} \tag{2.9}$$

as the first row [10]. Then, we use $\mathcal{R}_n$ to construct R_n according to the linear combination

$$R_n = \begin{cases} \mathcal{R}_n, & n = 0, \\ \mathcal{R}_n + \mathcal{R}_{N-n}, & 1 \le n \le N-1, \end{cases} \tag{2.10}$$

which is used in (2.3) to define the block circulant preconditioner R.

It is worthwhile to point out that it is possible to design different preconditioners by considering different periodic extensions to form $\tilde{w}_{m,n}$. For readers interested in the design of preconditioners, we refer to [10].

3. The spectral clustering property of preconditioned block Toeplitz matrices. Let us consider a family of block Toeplitz matrices T whose generating sequences $t_{m,n}$ are quadrantally-symmetric,

$$t_{m,n} = t_{|m|,|n|}, \quad |m| \le M-1, \quad |n| \le N-1, \tag{3.1}$$

and absolutely summable (i.e., T is in the Wiener class),

$$\sum_{-\infty}^{\infty} \sum_{-\infty}^{\infty} |t_{m,n}| \le K < \infty, \tag{3.2}$$

and whose generating functions are of the form

$$T(z_x, z_y) \equiv \sum_{-\infty}^{\infty}\sum_{-\infty}^{\infty} t_{i,j} z_x^{-i} z_y^{-j}$$

$$= \frac{A(z_x, z_y)}{B(z_x, z_y)} + \frac{A(z_x^{-1}, z_y)}{B(z_x^{-1}, z_y)} + \frac{A(z_x, z_y^{-1})}{B(z_x, z_y^{-1})} + \frac{A(z_x^{-1}, z_y^{-1})}{B(z_x^{-1}, z_y^{-1})}, \tag{3.3a}$$

where

$$A(z_x, z_y) = \sum_{i=0}^{p_x}\sum_{j=0}^{p_y} a_{i,j} z_x^{-i} z_y^{-j}, \qquad B(z_x, z_y) = \sum_{i=0}^{q_x}\sum_{j=0}^{q_y} b_{i,j} z_x^{-i} z_y^{-j}. \tag{3.3b}$$

Note that the quadrantally-symmetric property of $t_{m,n}$ implies that T is doubly symmetric, i.e., $T_n = T_n^T$ and $T_n = T_{-n}$. We also assume that T has a nonsingular preconditioner R so that $R^{-1}T$ is also nonsingular. We call T satisfying (3.1)–(3.3) the $MN \times MN$ block Toeplitz matrix generated by a quadrantally-symmetric rational function of order (p_x, q_x, p_y, q_y). For convenience, we use the notation

$$\gamma_x = \max(p_x, q_x), \qquad \gamma_y = \max(p_y, q_y).$$

The following discussion focuses on the spectral clustering property of the preconditioned matrix $R^{-1}T$, namely, a bound on the number of eigenvalues clustered around unity. Note that the following spectral analysis does not depend on the positive-definiteness of T or R.

3.1. Separable generating sequences. One special case of block Toeplitz matrices described by (3.1)–(3.3) is that $T(z_x, z_y)$ is separable, i.e.,

$$T(z_x, z_y) = T_x(z_x) T_y(z_y), \tag{3.4a}$$

where

$$T_x(z_x) = \frac{A_x(z_x)}{B_x(z_x)} + \frac{A_x(z_x^{-1})}{B_x(z_x^{-1})}, \qquad T_y(z_y) = \frac{A_y(z_y)}{B_y(z_y)} + \frac{A_y(z_y^{-1})}{B_y(z_y^{-1})}, \tag{3.4b}$$

and where

(3.4c)

$$A_x(z_x) = \sum_{i=0}^{p_x} a_i z_x^{-i}, \quad B_x(z_x) = \sum_{i=0}^{q_x} b_i z_x^{-i}, \quad A_y(z_y) = \sum_{i=0}^{p_y} c_i z_y^{-i}, \quad B_y(z_y) = \sum_{i=0}^{q_y} d_i z_y^{-i}.$$

Note that the separability of $T(z_x, z_y)$ implies the separability of the generating sequence $t_{m,n}$, i.e.,

$$t_{m,n} = t_{x,m} t_{y,n}.$$

Based on $t_{x,m}$ (or $t_{y,n}$), we can construct Toeplitz matrix T_x (or T_y) and the corresponding preconditioner R_x (or R_y), where T_x and R_x (or T_y and R_y) are of dimension $M \times M$ (or $N \times N$). It is easy to see that the preconditioner R is also separable, and the eigenvalues of $R^{-1}T$ are the products of the eigenvalues of $R_x^{-1}T_x$ and $R_y^{-1}T_y$.

Thus, to understand the spectral properties of $R^{-1}T$, we only have to examine those of preconditioned (point) Toeplitz matrices $R_x^{-1}T_x$ and $R_y^{-1}T_y$.

According to the construction (2.9) and the symmetric property of $t_{x,m}$, we know that R_x is a circulant matrix with the first row

$$t_{x,0},\ t_{x,1}+t_{x,M-1},\ t_{x,2}+t_{x,M-2},\ \cdots,\ t_{x,M-1}+t_{x,1}.$$

When $B_x(z_x)=1$ (i.e., $q_x=0$), T_x is banded with bandwidth p_x, and R_x is almost the same as T_x except for the addition of elements in the northeast and southwest corners to make R_x circulant. Thus, the elements of $\triangle T_x = R_x - T_x$ are all zeros except the first and the last p_x rows. Consequently, $\triangle T_x$ has at least $M-2p_x$ $(=M-2\gamma_x)$ eigenvalues at zero and $R_x^{-1}T_x=(T_x+\triangle T_x)^{-1}T_x$ has at least $M-2p_x$ eigenvalues at one. This result can also be obtained by using the operator-mask interpretation. That is, the products $T_x\mathbf{w}$ and $R_x\mathbf{w}$, for arbitrary $\mathbf{w}=(w_0,\cdots,w_m,\cdots,w_{M-1})^T$, can be viewed as a linear operator characterized by the mask

$$[t_{x,p_x}\ \ t_{x,p_x-1}\ \ \cdots\ \ t_{x,1}\ \ t_{x,0}\ \ t_{x,1}\ \ \cdots\ \ t_{x,p_x-1}\ \ t_{x,p_x}]$$

operating, respectively, on two extended sequences

$$\bar{w}_m=\begin{cases} w_m, & 0\le m\le M-1,\\ 0, & \text{otherwise,}\end{cases} \quad \text{and} \quad \tilde{w}_m = w_{m \bmod M}.$$

It is clear that $T_x\mathbf{w}$ and $R_x\mathbf{w}$ give the same output elements if the center of the mask is located at $p_x\le m\le M-p_x-1$. There are $M-2p_x$ such elements and, as a consequence, the dimension of the null space of $\triangle T_x$ is at least $M-2p_x$. The operator-mask viewpoint will be generalized to the case of higher-dimensional generating sequences (see §3.2).

When $B_x(z_x)\ne 1$, we approximate the block matrix $\triangle T_x = R_x - T_x$ with an asymptotically equivalent block matrix $\triangle E_x$, and then use the recursive property of $t_{x,m}$ to show that $\triangle E_x$ has eigenvalues repeated at zero. The recursive property of $t_{x,m}$ is stated in the following lemma.

LEMMA 1. *The sequence $t_{x,m}$ generated by $T_x(z_x)$ in* (3.4b) *follows the recursion,*

(3.5)
$$t_{x,m+1}=-(b_1t_{x,m}+b_2t_{x,m-1}+\cdots+b_{q_x}t_{x,m-q_x+1})/b_0, \quad m\ge\gamma_x=\max(p_x,q_x).$$

Proof. The generating sequence associated with $A_x(z_x)/B_x(z_x)$ given by (3.4c) is

$$\tfrac{1}{2}t_{x,0},t_{x,1},t_{x,2},\cdots,t_{x,m},\cdots.$$

Thus, we have

$$\left(\frac{t_{x,0}}{2}+\sum_{m=1}^{\infty}t_{x,m}z^{-m}\right)(b_0+b_1z^{-1}+\cdots+b_{q_x}z^{-q_x})=a_0+a_1z^{-1}+\cdots+a_{p_x}z^{-p_x}.$$

The proof is completed by comparing the coefficients of the above equation. □

Consider the approximation of $\triangle T_x = R_x - T_x$ with $\triangle E_x = F_x + F_x^T$, where

$$F_x=\begin{bmatrix} t_M & t_{M-1} & \cdot & \cdot & t_2 & t_1\\ t_{M+1} & t_M & t_{M-1} & \cdot & \cdot & t_2\\ \cdot & t_{M+1} & t_M & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & t_{M-1} & \cdot\\ t_{2M-2} & \cdot & \cdot & t_{M+1} & t_M & t_{M-1}\\ t_{2M-1} & t_{2M-2} & \cdot & \cdot & t_{M+1} & t_M \end{bmatrix},$$

and where the subscript x is omitted for simplicity and t_m with $m > \gamma_x$ is recursively constructed from (3.5). Since elements $t_{x,m}$ in F_x satisfy (3.5), the rank of F_x or F_x^T is at most γ_x. Consequently, $\triangle E_x$ has at least $M - 2\gamma_x$ eigenvalues at zero.

Then, we examine the difference between $\triangle T_x$ and $\triangle E_x$. Consider

$$\text{(3.6)} \qquad \mathbf{v} = (\triangle T_x - \triangle E_x)\mathbf{w},$$

for nonzero $\mathbf{w}$. The mth, $1 \le m \le M$, elements of $\triangle T_x\mathbf{w}$ and $\triangle E_x\mathbf{w}$ can be written, respectively, as

$$\text{(3.7)} \quad [\triangle T_x\mathbf{w}]_m = [R_x\mathbf{w}]_m - [T_x\mathbf{w}]_m = \sum_{i=m+1}^{M} t_{x,M+m-i}w_i + \sum_{i=1}^{m-1} t_{x,M-m+i}w_i,$$

and

$$\text{(3.8)} \quad [\triangle E_x\mathbf{w}]_m = [F_x\mathbf{w}]_m + [F_x^T\mathbf{w}]_m = \sum_{i=1}^{M} t_{x,M+m-i}w_i + \sum_{i=1}^{M} t_{x,M-m+i}w_i.$$

Therefore, $\mathbf{v}$ is bounded above by

$$\begin{aligned} ||\mathbf{v}||_\infty = ||\triangle E_x\mathbf{w} - \triangle T_x\mathbf{w}||_\infty &\le \max_{1\le m\le M} \left| \sum_{i=1}^{m} t_{x,M+m-i}w_i + \sum_{i=m}^{M} t_{x,M-m+i}w_i \right| \\ &\le 2 \sum_{m=M}^{2M-1} |t_{x,m}| ||\mathbf{w}||_\infty, \end{aligned}$$

where the ∞-norm of the vector $\mathbf{v}$ (or $\mathbf{w}$) is the maximum absolute value of elements v_m (or w_m), $1 \le m \le M$. We have

$$\text{(3.9)} \qquad ||\triangle T_x - \triangle E_x||_\infty \equiv \max_{\mathbf{w}} \frac{||\mathbf{v}||_\infty}{||\mathbf{w}||_\infty} \le 2 \sum_{m=M}^{2M-1} |t_{x,m}|,$$

which converges to zero as M goes to infinity, since $\sum_{m=0}^{\infty} |t_{x,m}|$ converges and $S_m = \sum_{m'=0}^{m} |t_{x,m'}|$ is a Cauchy sequence. The matrix $\triangle T_x - \triangle E_x$ is symmetric so that

$$||\triangle T_x - \triangle E_x||_1 = ||\triangle T_x - \triangle E_x||_\infty$$

and

$$||\triangle T_x - \triangle E_x||_2 \le (||\triangle T_x - \triangle E_x||_1 ||\triangle T_x - \triangle E_x||_\infty)^{1/2} = ||\triangle T_x - \triangle E_x||_\infty.$$

Thus, $\triangle T_x$ is asymptotically equivalent to $\triangle E_x$. It also follows that $\triangle T_x$ has at least $M - 2\gamma_x$ eigenvalues asymptotically converging to 0, or $R_x^{-1}T_x = (T_x + \triangle T_x)^{-1}T_x$ has at least $M - 2\gamma_x$ eigenvalues asymptotically converging to 1.

The $\triangle T_x$ and $\triangle E_x$ above are amenable to the operator-mask interpretation (see Fig. 1 with $M = 8$). One can easily verify that $\triangle T_x\mathbf{w}$ and $\triangle E_x\mathbf{w}$ correspond, respectively, to the use of the two masks

$$\begin{aligned} &\triangle T_x : [t_{x,M-1}\ t_{x,M-2}\ \cdots\ t_{x,1}\ t_{x,0}\ t_{x,1}\ \cdots\ t_{x,M-2}\ t_{x,M-1}], \\ &\triangle E_x : [\cdots\ t_{x,M}\ t_{x,M-1}\ \cdots\ t_{x,1}\ t_{x,0}\ t_{x,1}\ \cdots\ t_{x,M-1}\ t_{x,M}\ \cdots], \end{aligned}$$

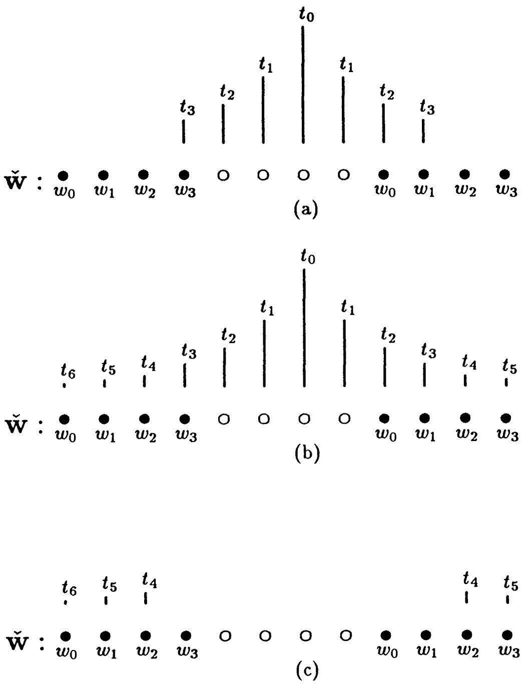


FIG. 1. *Operator-mask interpretations of* (a) $\triangle T_x \mathbf{w}$, (b) $\triangle E_x \mathbf{w}$, *and* (c) $(\triangle E_x - \triangle T_x)\mathbf{w}$.

operating on the same sequence

$$\check{w}_m = \begin{cases} w_{m \bmod M}, & -M \le m \le -1 \quad \text{or} \quad M \le m \le 2M-1, \\ 0, & \text{elsewhere.} \end{cases}$$

Note that the mask for $\triangle E_x$ is of infinite length. The corresponding mask for $\triangle E_x - \triangle T_x$ is

$$\triangle E_x - \triangle T_x : [\cdots \ t_{x,M+1} \ t_{x,M} \ 0 \ \cdots \ 0 \ 0 \ 0 \ \cdots \ 0 \ t_{x,M} \ t_{x,M+1} \ \cdots].$$

It is easy to derive (3.9) from the operator-mask viewpoint. Note that, for larger M, although there are more terms contributing to the ∞-norm of $\triangle E_x - \triangle T_x$, the weighting coefficient $t_{x,m}$, $m \ge M$, decays more rapidly. The resulting ∞-norm of $\triangle E_x - \triangle T_x$ asymptotically converges to zero. We conclude the above discussion as follows.

(a) When $B(z_x, z_y) = 1$, T_x (or T_y) is banded and $R_x^{-1}T_x$ (or $R_y^{-1}T_y$) has at most $2p_x + 1$ (or $2p_y + 1$) distinct eigenvalues, so that $R^{-1}T$ has at most $(2p_x+1)(2p_y+1)$ distinct eigenvalues.

(b) When $B(z_x, z_y) \ne 1$, $R_x^{-1}T_x$ (or $R_y^{-1}T_y$) has at most $2\gamma_x$ (or $2\gamma_y$) outliers not converging to unity and other eigenvalues are clustered between $(1-\epsilon_x, 1+\epsilon_x)$ (or $(1-\epsilon_y, 1+\epsilon_y)$). Thus, the eigenvalues of $R^{-1}T$ can be grouped into several clusters. The centers and clustering radii of these clusters and the numbers of eigenvalues contained are listed in Table 1, where $\lambda_{x,i}$ (or $\lambda_{y,j}$) denotes a typical outlier for

TABLE 1
Eigenvalues of $R^{-1}T$.

Center	$\lambda_{x,i}\lambda_{y,j}$	$\lambda_{x,i}$	$\lambda_{y,j}$	1
Radius	0	$O(\epsilon_y)$	$O(\epsilon_x)$	$O(\epsilon_x+\epsilon_y)$
Number	1	$N-2\gamma_y$	$M-2\gamma_x$	$(M-2\gamma_x)(N-2\gamma_y)$

$R_x^{-1}T_x$ (or $R_y^{-1}T_y$). Since ϵ_x and ϵ_y converge to zero as M and N become large, $R^{-1}T$ has asymptotically at most $(2\gamma_x+1)(2\gamma_y+1)$ distinct eigenvalues.

As a consequence, the PCG method converges in at most $(2\gamma_x+1)(2\gamma_y+1)$ iterations for positive definite T with sufficiently large M and N in both cases (a) and (b). This is confirmed numerically in §4.

3.2. Nonseparable generating sequences. For nonseparable generating functions $T(z_x,z_y)$ given by (3.3), we examine two typical cases, i.e., $B(z_x,z_y)=1$ and $B(z_x,z_y)\neq 1$ with $q_x>0$ and $q_y>0$.

When $B(z_x,z_y)=1$, we have a corresponding generating sequence of finite duration. As described in §2, the products $T\mathbf{w}$ and $R\mathbf{w}$ correspond to a linear operator characterized by the mask

$$\begin{bmatrix} t_{p_x,p_y} & t_{p_x-1,p_y} & \cdots & t_{0,p_y} & \cdots & t_{p_x-1,p_y} & t_{p_x,p_y} \\ t_{p_x,p_y-1} & t_{p_x-1,p_y-1} & \cdots & t_{0,p_y-1} & \cdots & t_{p_x-1,p_y-1} & t_{p_x,p_y-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ t_{p_x,0} & t_{p_x-1,0} & \cdots & t_{0,0} & \cdots & t_{p_x-1,0} & t_{p_x,0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ t_{p_x,p_y-1} & t_{p_x-1,p_y-1} & \cdots & t_{0,p_y-1} & \cdots & t_{p_x-1,p_y-1} & t_{p_x,p_y-1} \\ t_{p_x,p_y} & t_{p_x-1,p_y} & \cdots & t_{0,p_y} & \cdots & t_{p_x-1,p_y} & t_{p_x,p_y} \end{bmatrix}$$

operating, respectively, on $\bar{w}_{m,n}$ and $\tilde{w}_{m,n}$ given by (2.7) and (2.8). The output elements of $T\mathbf{w}$ and $R\mathbf{w}$ are identical if the center of the mask is located at

$$p_x\le m\le M-p_x-1, \qquad p_y\le n\le N-p_y-1.$$

The dimension of the null space of $\Delta T=R-T$ is at least $(M-2p_x)(N-2p_y)$. Consequently, $R^{-1}T$ has at least $(M-2p_x)(N-2p_y)$ eigenvalues repeated at 1 or, equivalently, there are at most $2(Mp_y+Np_x)-4p_xp_y$ outliers. This result is summarized in the following theorem.

THEOREM 1. *Let T be an $MN\times MN$ block Toeplitz matrix characterized by (3.1)–(3.3) with $B(z_x,z_y)=1$. The preconditioned matrix $R^{-1}T$ has at least $MN-2(Mp_y+Np_x)+4p_xp_y$ eigenvalues repeated at one.*

When $B(z_x,z_y)\neq 1$ with $q_x>0$ and $q_y>0$, the products $T\mathbf{w}$ and $R\mathbf{w}$ correspond to a linear operator characterized by the mask

$$\begin{bmatrix} t_{M-1,N-1} & t_{M-2,N-1} & \cdots & t_{0,N-1} & \cdots & t_{M-2,N-1} & t_{M-1,N-1} \\ t_{M-1,N-2} & t_{M-2,N-2} & \cdots & t_{0,N-2} & \cdots & t_{M-2,N-2} & t_{M-1,N-2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ t_{M-1,0} & t_{M-2,0} & \cdots & t_{0,0} & \cdots & t_{M-2,0} & t_{M-1,0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ t_{M-1,N-2} & t_{M-2,N-2} & \cdots & t_{0,N-2} & \cdots & t_{M-2,N-2} & t_{M-1,N-2} \\ t_{M-1,N-1} & t_{M-2,N-1} & \cdots & t_{0,N-1} & \cdots & t_{M-2,N-1} & t_{M-1,N-1} \end{bmatrix} \tag{3.10}$$

operating on $\bar{w}_{m,n}$ and $\tilde{w}_{m,n}$, respectively. Let us exploit the quadrantally-symmetric property (3.1) and decompose $t_{m,n}$ into four sequences

$$t_{m,n} = \sum_{k=1}^{4} t_{k,m,n},$$

where $t_{k,m,n}$ is called the kth quadrant-support sequence and defined as

$$\text{(3.11a)} \qquad t_{1,m,n} = \begin{cases} t_{0,0}/4, & m = n = 0, \\ t_{m,0}/2, & 1 \le m \le M-1, \quad n = 0, \\ t_{0,n}/2, & m = 0, \quad 1 \le n \le N-1, \\ t_{m,n}, & 1 \le m \le M-1 \quad \text{and} \quad 1 \le n \le N-1, \\ 0, & m < 0 \quad \text{or} \quad n < 0, \end{cases}$$

and

$$\text{(3.11b)} \qquad t_{2,m,n} = t_{1,-m,n}, \qquad t_{3,m,n} = t_{1,-m,-n}, \qquad t_{4,m,n} = t_{1,m,-n}.$$

The following lemma is on the recursive property of $t_{1,m,n}$.

LEMMA 2. *Let $t_{m,n}$ be a quadrantally-symmetric sequence generated by the two-dimensional rational function $T(z_x, z_y)$ given in* (3.3), *and $t_{1,m,n}$ is the first quadrant-support sequence defined by* (3.11a). *Then,*

$$\text{(3.12)} \qquad \sum_{i=0}^{q_x} \sum_{j=0}^{q_y} b_{i,j} t_{1,m-i,n-j} = 0 \qquad \text{for } m > \gamma_x \quad \text{or} \quad n > \gamma_y.$$

Proof. It is clear that $A(z_x, z_y)/B(z_x, z_y)$ is the generating function for $t_{1,m,n}$. Therefore,

$$\frac{A(z_x, z_y)}{B(z_x, z_y)} = \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} t_{1,i,j} z_x^{-i} z_y^{-j}.$$

We multiply both sides of the above equation with $B(z_x, z_y)$, substitute (3.3b) for $A(z_x, z_y)$ and $B(z_x, z_y)$, and compare the corresponding coefficients. This gives the desired equation (3.12). □

Thus, we can use (3.12) to recursively define $t_{1,m,n}$ with $m > \gamma_x$ or $n > \gamma_y$ in the first quadrant, and the corresponding $t_{k,m,n}$ with $k = 2, 3, 4$ can be obtained from $t_{1,m,n}$ through the symmetry (3.11b).

As a generalization of the one-dimensional case, we define

$$\breve{w}_{m,n} = \begin{cases} w_{m \bmod M, n \bmod N}, & (m,n) \in [\cup_{-1 \le i,j \le 1} Q_{i,j}] - Q_{0,0}, \\ 0, & \text{elsewhere}, \end{cases}$$

where

$$Q_{i,j} = \{(m,n) : \quad iM \le m \le (i+1)M - 1, \quad jN \le n \le (j+1)N - 1\}.$$

Then, the operation $\triangle T\mathbf{w} = (R - T)\mathbf{w}$ corresponds to the mask (3.10) operating on $\breve{w}_{m,n}$. We choose the approximation $\triangle E\mathbf{w}$ to be an extended infinite mask, with recursively defined $t_{m,n}$ via (3.12), operating on $\breve{w}_{m,n}$. This is illustrated in Fig. 2, where we only show the first quadrant of the mask for $\triangle E$. For the rest of this

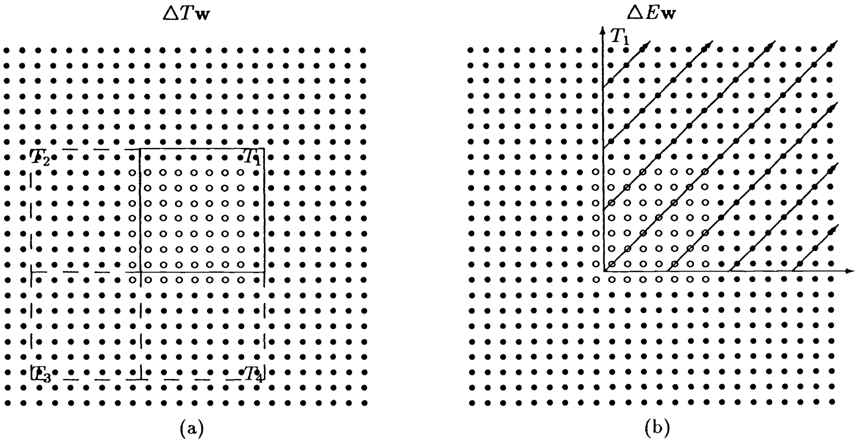


FIG. 2. *The matrix vector products* $\triangle T\mathbf{w}$ *and* $\triangle E\mathbf{w}$ *interpreted as* (a) *the operator-mask and* (b) *the extended operator-mask operating on* $\breve{w}$.

subsection, we are concerned with two issues: (i) the asymptotic equivalence of $\triangle T$ and $\triangle E$ and (ii) the number of eigenvalues of $\triangle E$ repeated at zero.

The operation $(\triangle E - \triangle T)\mathbf{w}$ corresponds to the difference between the extended mask and the original mask (3.10) operating on $\breve{w}_{m,n}$. The ∞-norm of the first quadrant of the difference mask $\triangle E - \triangle T$ operating on $\breve{w}_{m,n}$ in regions $Q_{1,0}$, $Q_{0,1}$, and $Q_{1,1}$ are bounded, respectively, by

$$K_{1,1,0} = \sum_{m=M}^{2M-1} \sum_{n=0}^{N-1} |t_{1,m,n}|, \quad K_{1,0,1} = \sum_{m=0}^{M-1} \sum_{n=N}^{2N-1} |t_{1,m,n}|, \quad K_{1,1,1} = \sum_{m=M}^{2M-1} \sum_{n=N}^{2N-1} |t_{1,m,n}|.$$

By exploiting the symmetry, we have the bound for $\triangle E - \triangle T$,

$$||\triangle E - \triangle T||_\infty \leq 4(K_{1,1,0} + K_{1,0,1} + K_{1,1,1}) = K.$$

With (3.2), we can order $t_{m,n}$ appropriately to be a Cauchy sequence and argue that K converges to zero for asymptotically large M and N. This establishes the asymptotic equivalence of $\triangle E$ and $\triangle T$.

The operator $\triangle E$ can be expressed as a superposition of 12 operators,

$$\begin{aligned} \triangle E &= F_{1,1,0} + F_{1,1,1} + F_{1,0,1} + F_{2,0,1} + F_{2,-1,1} + F_{2,-1,0}, \\ &\quad + F_{3,-1,0} + F_{3,-1,-1} + F_{3,0,-1} + F_{4,0,-1} + F_{4,1,-1} + F_{4,1,0}, \end{aligned} \tag{3.13}$$

where $F_{k,i,j}$ denotes the kth quadrant of the extended mask operating on sequences defined on $Q_{i,j}$. Consider operators $F_{1,1,0}$, $F_{1,0,1}$, and $F_{1,1,1}$. Their operations on $\mathbf{w}$ can be written as

$$F_{1,1,0} \ : \ v_{1,i,j} = \sum_{m=0}^{M-1} \sum_{n=j}^{N-1} t_{1,M+m-i,n-j} w_{m,n},$$

$$F_{1,0,1} \ : \ v_{2,i,j} = \sum_{m=i}^{M-1} \sum_{n=0}^{N-1} t_{1,m-i,N+n-j} w_{m,n},$$

$$F_{1,1,1}\ :\ v_{3,i,j}=\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}t_{1,M+m-i,N+n-j}w_{m,n}.$$

With (3.12), we have

$$\sum_{i=i_0}^{i_0+q_x}\sum_{j=j_0}^{j_0+q_y}b_{i-i_0,j-j_0}v_{1,i,j}=\sum_{m=0}^{M-1}\sum_{n=j+j_0}^{N-1}\left[\sum_{i=0}^{q_x}\sum_{j=0}^{q_y}b_{i,j}t_{1,M+m-i_0-i,n-j_0-j}\right]w_{m,n}=0,$$

for $0\le i_0\le M-1-\gamma_x$;

$$\sum_{i=i_0}^{i_0+q_x}\sum_{j=j_0}^{j_0+q_y}b_{i-i_0,j-j_0}v_{2,i,j}=\sum_{m=i+i_0}^{M-1}\sum_{n=0}^{N-1}\left[\sum_{i=0}^{q_x}\sum_{j=0}^{q_y}b_{i,j}t_{1,m-i_0-i,N+n-j_0-j}\right]w_{m,n}=0,$$

for $0\le j_0\le N-1-\gamma_y$; and

$$\sum_{i=i_0}^{i_0+q_x}\sum_{j=j_0}^{j_0+q_y}b_{i-i_0,j-j_0}v_{3,i,j}=\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left[\sum_{i=0}^{q_x}\sum_{j=0}^{q_y}b_{i,j}t_{1,M+m-i_0-i,N+n-j_0-j}\right]w_{m,n}=0,$$

for $0\le i_0\le M-1-\gamma_x$ or $0\le j_0\le N-1-\gamma_y$.

By combining the above three equations, we have

$$\sum_{i=i_0}^{i_0+q_x}\sum_{j=j_0}^{j_0+q_y}b_{i-i_0,j-j_0}(v_{1,i,j}+v_{2,i,j}+v_{3,i,j})=0,$$

for $0\le i_0\le M-1-\gamma_x$ and $0\le j_0\le N-1-\gamma_y$. Therefore, the rank of $F_1=F_{1,1,0}+F_{1,0,1}+F_{1,1,1}$ is at most $M\gamma_y+N\gamma_x-\gamma_x\gamma_y$. By using the symmetry, we can argue that the rank of $F_k=\sum_{i,j}F_{k,i,j}$, $k=2,3,4$, is also at most $M\gamma_y+N\gamma_x-\gamma_x\gamma_y$. Consequently, the rank of $\triangle E$ is at most $4(M\gamma_y+N\gamma_x-\gamma_x\gamma_y)$ or, equivalently, $\triangle E$ has at least $MN-4(M\gamma_y+N\gamma_x-\gamma_x\gamma_y)$ eigenvalues repeated at 0.

To conclude this section, we have the following theorem.

THEOREM 2. *Let T be an $MN\times MN$ block Toeplitz matrix satisfying* (3.1), (3.2), *and* (3.3) *with $q_x>0$ and $q_y>0$. Then, the preconditioned matrix $R^{-1}T$ has at least $MN-4(M\gamma_x+N\gamma_y-\gamma_x\gamma_y)$ eigenvalues asymptotically converging to one.*

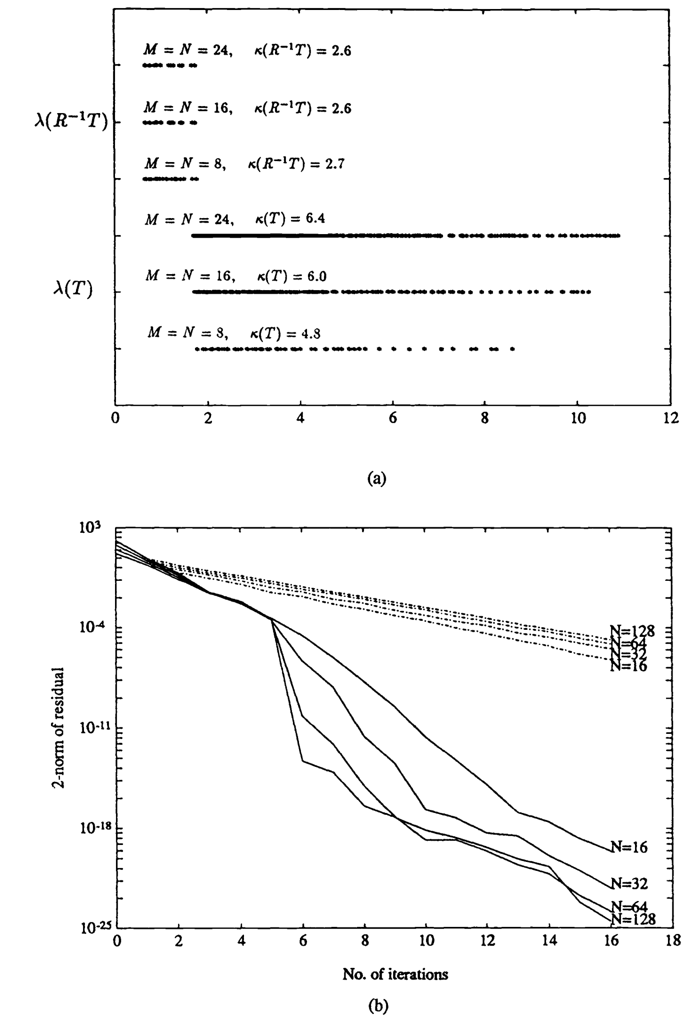
4. Numerical experiments.

4. Numerical experiments. Numerical experiments are performed to illustrate the spectra of T and $R^{-1}T$ and the convergence behavior of the CG and PCG methods. Note that the spectral clustering property derived in §3 does not require T to be positive definite. However, we focus on positive-definite T in our experiments so that the CG and PCG methods can be conveniently applied. For all test problems below, we choose $M=N$ and use $(0,\cdots,0)^T$ and $(1,\cdots,1)^T$ as the initial and right-hand-side vectors, respectively.

The first three test problems have positive rational generating functions in the Wiener class.

Example 1. Rational separable Toeplitz with $(p_x,q_x,p_y,q_y)=(0,2,0,2)$. The $T(z_x,z_y)$ is of the form (3.4) with $A_x(z_x)=A_y(z_y)=1$,

$$B_x(z_x)=(1+0.8z_x^{-1})(1-0.7z_x^{-1})\quad\text{and}\quad B_y(z_y)=(1+0.9z_y^{-1})(1-0.6z_y^{-1}).$$



(a)

(b)

FIG. 3. (a) *Eigenvalue distribution of* T *and* $R^{-1}T$ *and* (b) *convergence history for Example* 1.

Example 2. Rational Toeplitz with $(p_x, q_x, p_y, q_y) = (2, 0, 2, 0)$. The $T(z_x, z_y)$ is of the form (3.3) with $B(z_x, z_y) = 1$ and

$$A(z_x, z_y) = 0.25 - 0.02(z_x^{-1} + z_y^{-1}) + 0.015(z_x^{-2} + z_y^{-2}) + 0.03z_x^{-1}z_y^{-1}$$
$$-0.02z_x^{-1}z_y^{-1}(z_x^{-1} + z_y^{-1}) - 0.01z_x^{-2}z_y^{-2}.$$

Example 3. Rational Toeplitz with $(p_x, q_x, p_y, q_y) = (0, 2, 0, 1)$. The $T(z_x, z_y)$ is of the form (3.3) with $A(z_x, z_y) = 1$ and

$$B(z_x, z_y) = 1 + 0.5z_x^{-1} - 0.3z_y^{-1} - 0.2z_x^{-2} - 0.1z_x^{-1}z_y^{-1} + 0.2z_x^{-2}z_y^{-1}.$$

We plot the corresponding spectra of T and $R^{-1}T$, their condition numbers, and the convergence history of the CG (dashed lines) and PCG (solid lines) methods in Figs. 3(a),(b)–5(a),(b). Since the eigenvalues of T for Examples 1–3 all satisfy

$$0 < \delta_1 \leq \lambda(T) \leq \delta_2 < \infty,$$

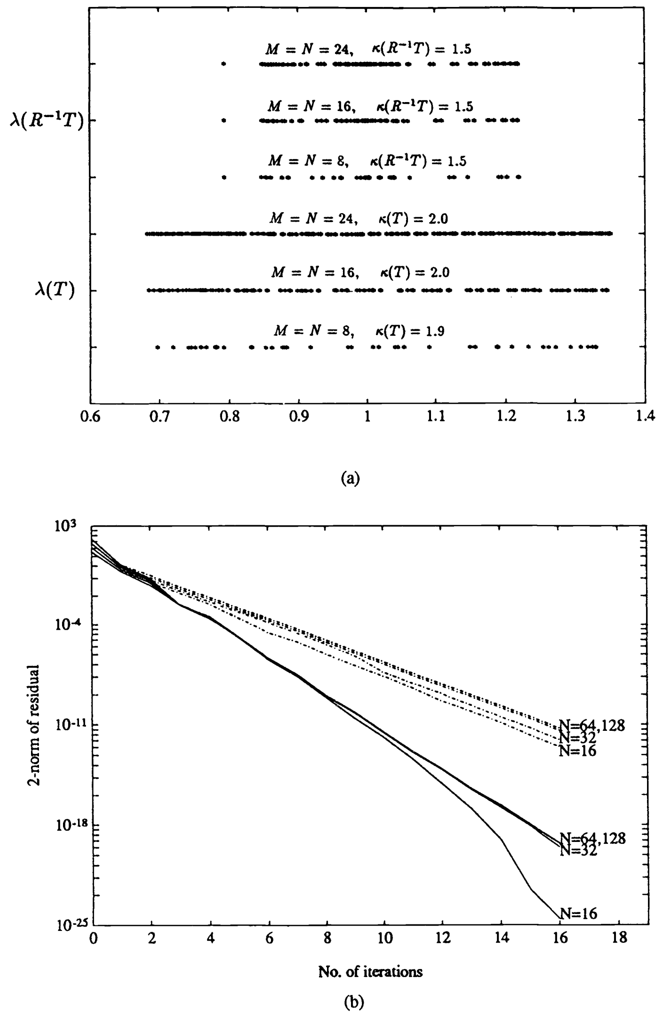
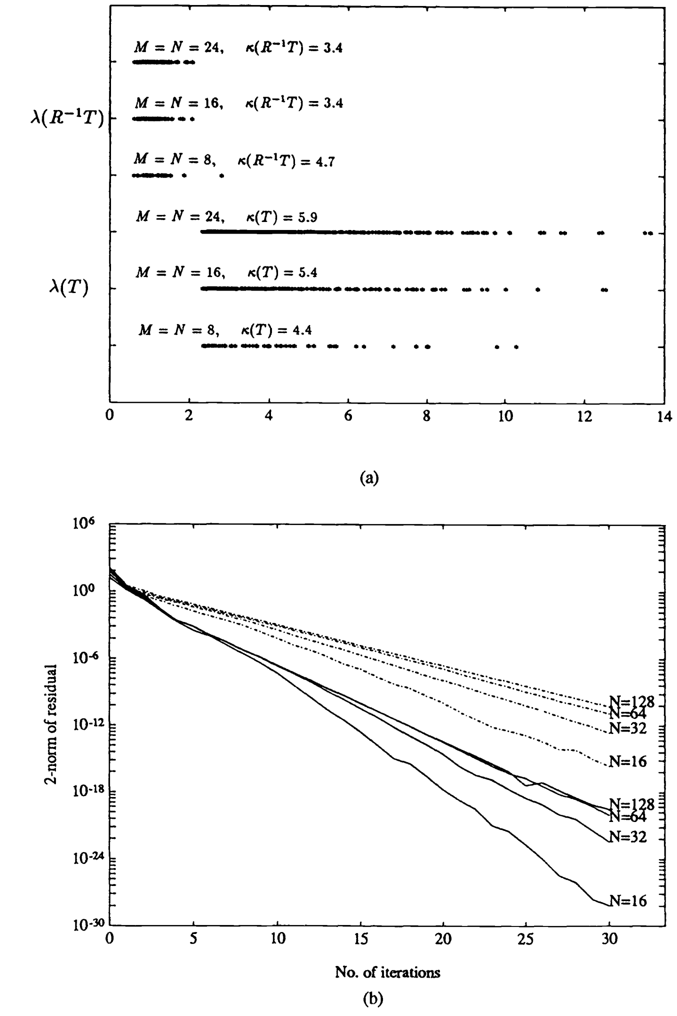


FIG. 4. (a) *Eigenvalue distribution of* T *and* $R^{-1}T$ *and* (b) *convergence history for Example* 2.

where δ_1 and δ_2 are constants independent of the dimensions M and N of the given block matrix, the condition number $\kappa(T)$ is bounded by $\delta_2/\delta_1 = O(1)$. Clearly, $R^{-1}T$ has a smaller condition number and a better clustering feature than T. Consequently, the PCG method performs better than the CG method.

One important difference between the separable and nonseparable cases is that, as N becomes larger, the PCG method converges faster for the separable case but more slowly for the nonseparable case. This can be easily explained by the analysis given in §3. When T is separable, the number of clusters is fixed and the clustering radius ϵ becomes smaller as N becomes larger. According to the analysis, $R^{-1}T$ has asymptotically 25 $(= (2\gamma_x + 1)(2\gamma_y + 1))$ distinct eigenvalues, including isolated outliers, clustered outliers, and clustered eigenvalues converging to 1. However, the 2-norm of the residual decreases rapidly in five iterations as given in Fig. 3(b). We observe an empirical formula for the separable case, namely, the PCG method converges



(a)

(b)

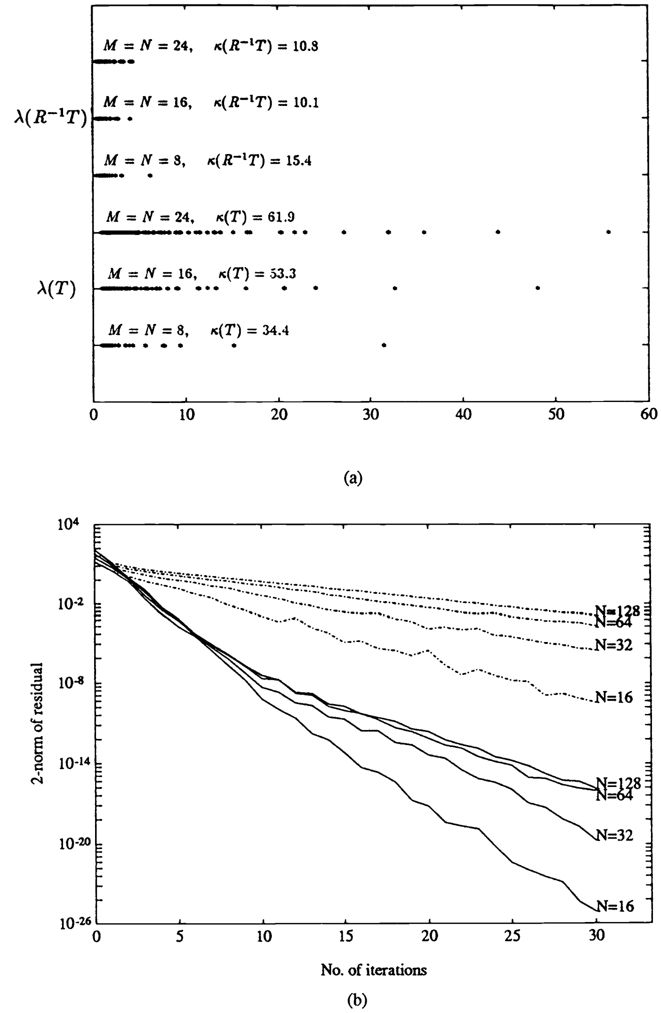
FIG. 5. (a) *Eigenvalue distribution of* T *and* $R^{-1}T$ *and* (b) *convergence history for Example* 3.

asymptotically in $\gamma_x\gamma_y + 1$ iterations. This phenomenon is closely related to the point Toeplitz result [10], where we found that although there are asymptotically $2\gamma_x + 1$ distinct eigenvalues, the PCG method converges asymptotically in $\gamma_x + 1$ iterations. When T is nonseparable, the number of outliers increases with N. Although the PCG method converges more slowly for larger N, the effect is not obvious until the 2-norm of the residual is very small. Besides, the convergence curves are getting closer for larger N. This indicates that the number of PCG (or CG) iterations required is $O(1)$, which is determined by the condition number rather than the number of outliers.

A block Toeplitz with a nonrational generating function is given in Example 4.

Example 4. Nonrational Toeplitz. The block Toeplitz matrix is generated by a spherically-symmetric sequence

$$t_{m,n} = 0.7^{\sqrt{m^2+n^2}} + 0.5^{\sqrt{m^2+n^2}} + 0.3^{\sqrt{m^2+n^2}}.$$



(a)

(b)

FIG. 6. (a) *Eigenvalue distribution of* T *and* $R^{-1}T$ *and* (b) *convergence history for Example* 4.

The spectra of T and $R^{-1}T$ and the convergence history of the CG and PCG methods are plotted in Fig. 6. Although our analysis in §3 is restricted to the rational generating function case, it appears that this case does not differ much from the rational case. The PCG method converges faster than the CG method due to a smaller condition number and a better spectral clustering property of $R^{-1}T$. The condition numbers of T and $R^{-1}T$ are again $O(1)$ so that the number of PCG (or CG) iterations required is $O(1)$, which is consistent with the observation that the convergence history curves are getting closer for larger N.

For the above four problems, T is well conditioned, i.e., $\kappa(T) = O(1)$, so that the condition number reduction through preconditioning is just a constant factor. To see a more dramatic condition number improvement, let us consider an ill-conditioned block Toeplitz below.

Example 5. Ill-conditioned Toeplitz with $(p_x, q_x, p_y, q_y) = (2, 0, 2, 0)$. The block

Toeplitz is characterized by the two-dimensional mask:

$$(-1)\times\begin{bmatrix} 0.01 & 0.02 & 0.04 & 0.02 & 0.01 \\ 0.02 & 0.04 & 0.12 & 0.04 & 0.02 \\ 0.04 & 0.12 & -1 & 0.12 & 0.04 \\ 0.02 & 0.04 & 0.12 & 0.04 & 0.02 \\ 0.01 & 0.02 & 0.04 & 0.02 & 0.01 \end{bmatrix}.$$

Note that the sum of the coefficients of the mask is zero. Masks of this nature arise in the discretization of constant-coefficient elliptic partial differential equations. However, the preconditioning block circulant matrix R is singular for this problem, since $R\mathbf{w} = 0$ for $\mathbf{w} = (1, 1, \cdots, 1, 1)^T$. In order to perform the preconditioning properly, we modify the preconditioner R slightly by replacing the zero eigenvalue with the smallest nonzero eigenvalue of R in our experiment.

The spectra of T and $R^{-1}T$, the convergence history of the CG and PCG methods, and the number of iterations required for the 2-norm of the relative residual less than 10^{-12} are plotted in Fig. 7 (a)–(c). As shown in Fig. 7(a), the condition numbers of T and $R^{-1}T$ increase at the rates of $O(N^2)$ and $O(N)$, respectively. Thus, the preconditioning provides an order of condition number improvement. A detailed analysis for the improvement of conditioned number from $O(N^2)$ to $O(N)$ is given in [4]. We see from Fig. 7(c) that PCG and CG methods converge in $O(\sqrt{N})$ and $O(N)$ iterations, respectively.

As far as the computational complexity is concerned, both the PCG and CG methods require $O(N^2 \log N)$ operations for Examples 1–4, where the condition numbers of T and $R^{-1}T$ are $O(1)$. For Example 5, the PCG and CG methods require, respectively, $O(N^{5/2} \log N)$ and $O(N^3 \log N)$ operations. For all above test problems, the PCG and CG methods require much lower computational complexity than the direct method, which requires $O(N^5)$ operations.

5. Conclusion. In this research, we extended the preconditioning technique from point Toeplitz matrices to block Toeplitz matrices. We interpreted the block Toeplitz matrix-vector $T\mathbf{w}$ in terms of a two-dimensional constant-coefficient mask operating on a certain two-dimensional sequence construction based on $\mathbf{w}$. This viewpoint provides a natural way to analyze the spectral clustering property of $R^{-1}T$. For block Toeplitz matrices T generated by two-dimensional rational functions $T(z_x, z_y)$ of order (p_x, q_x, p_y, q_y), we showed that the eigenvalues of $R^{-1}T$ are clustered around unity except at most $O(M\gamma_y + N\gamma_x)$ outliers, where $\gamma_x = \max(p_x, q_x)$ and $\gamma_y = \max(p_y, q_y)$. Furthermore, if T is separable, the outliers are clustered together such that $R^{-1}T$ has at most $(2\gamma_x + 1)(2\gamma_y + 1)$ asymptotic distinct eigenvalues. Thus, $R^{-1}T$ has a better spectral clustering property than T. Additionally, it was shown numerically that $R^{-1}T$ generally has a smaller condition number than T. These two spectral properties explain the superior convergence behavior of the PCG method over the CG method.

For point rational Toeplitz matrices, the number of outliers is often small ($= 2\max(p, q)$) and independent of the size of the problem so that it can be used to characterize the convergence rate of the PCG method. However, for block rational Toeplitz matrices, the number of outliers is proportional to the size of the problem, and is often too large to be useful for characterizing the convergence behavior of the PCG method. Hence, we have to examine both the condition number improvement and the spectral clustering effect. More research on the adaptation of the preconditioning technique to more general classes of block Toeplitz matrices, such as indefinite or

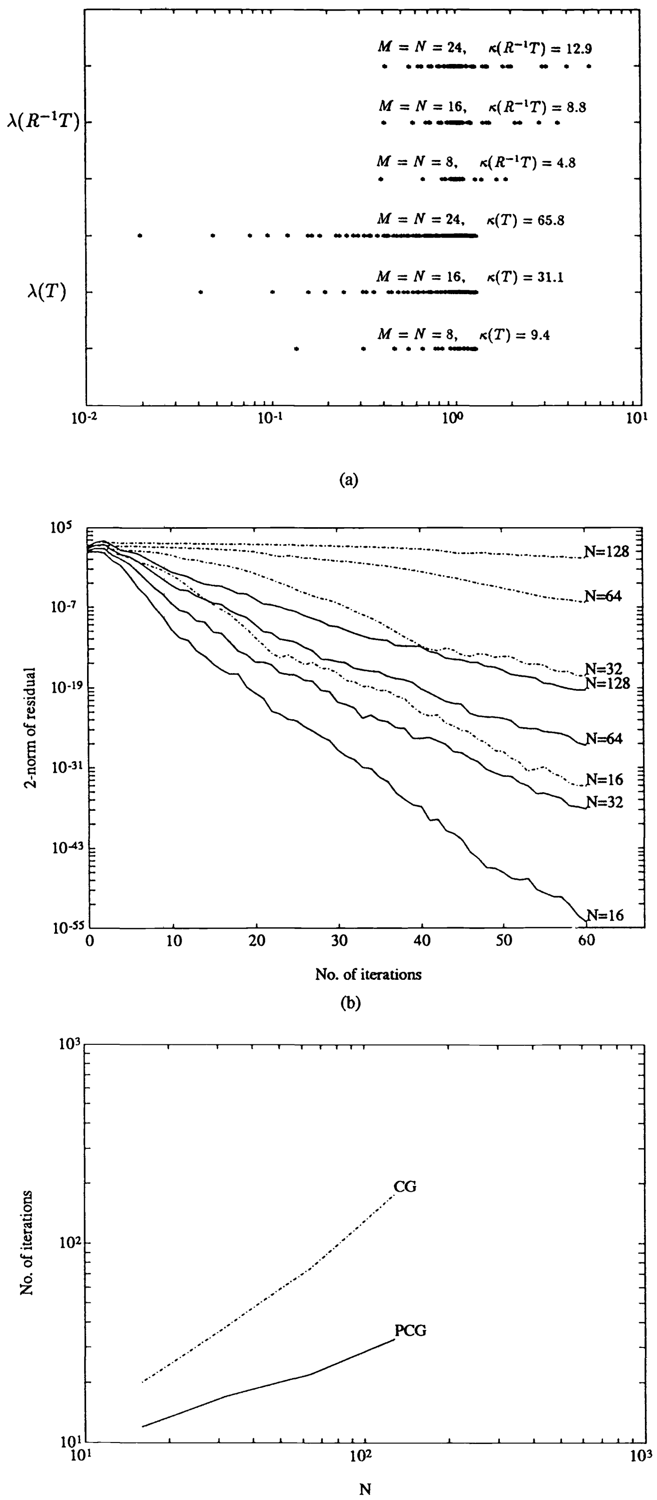


FIG. 7. (a) *Eigenvalue distribution of* T *and* $R^{-1}T$ *and* (b) *convergence history and* (c) *convergence rate of* CG *and* PCG *method for Example* 5.

nonsymmetric problems and the spectral analysis of the preconditioned matrices, is expected in the future.

REFERENCES

[1] O. Axelsson and G. Lindskog, *On the rate of convergence of the preconditioned conjugate gradient method*, Numer. Math., 48 (1986), pp. 499–523.

[2] E. H. Bareiss, *Numerical solution of linear equations with Toeplitz and vector Toeplitz matrices*, Numer. Math., 13 (1969), pp. 404–424.

[3] R. H. Chan, *Circulant preconditioners for Hermitian Toeplitz system*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 542–550.

[4] R. H. Chan and T. F. Chan, *Circulant preconditioners for elliptic problems*, Tech. Report, Department of Mathematics, University of California, Los Angeles, CA, Dec. 1990.

[5] R. H. Chan and G. Strang, *Toeplitz equations by conjugate gradients with circulant preconditioner*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 104–119.

[6] T. F. Chan, *An optimal circulant preconditioner for Toeplitz systems*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 766–771.

[7] L. F. Chaparro and E. I. Jury, *Rational approximation of 2-D linear discrete systems*, IEEE Trans. Acoust., Speech Signal Process., 30 (1982), pp. 780–787.

[8] T. Huckle, *Circulant and skew-circulant matrices for solving Toeplitz matrices problems*, in Cooper Mountain Conference on Iterative Methods, Cooper Mountain, Colorado, 1990.

[9] J. H. Justice, *A Levinson-type algorithm for two-dimensional Wiener filtering using bivariate Szegö polynomials*, Proc. IEEE, 65 (1977), pp. 882–886.

[10] T. K. Ku and C. J. Kuo, *Design and analysis of Toeplitz preconditioners*, Tech. Report 155, Signal and Image Processing Institute, University of Southern California, May 1990; IEEE Trans. Signal Processing, 40 (1992), pp. 129–141.

[11] ———, *Spectral properties of preconditioned rational Toeplitz matrices*, Tech. Report 163, Signal and Image Processing Institute, University of Southern California, Sept. 1990; SIAM J. Matrix Anal. Appl., 13 (1992), to appear.

[12] B. C. Levy, M. B. Adams, and A. S. Willsky, *Solution and linear estimation of 2-D nearest-neighbor models*, Proc. IEEE, 78 (1990), pp. 627–641.

[13] T. L. Marzetta, *Two-dimensional linear prediction: Autocorrelation arrays, minimum-phase prediction error filters, and reflection coefficient arrays*, IEEE Trans. Acoust., Speech, Signal Process., 28 (1980), pp. 725–733.

[14] J. Rissanen, *Algorithm for triangular decomposition of block Hankel and Toeplitz matrices with application to factoring positive matrix polynomials*, Math. Comp., 27 (1973), pp. 147–154.

[15] G. Strang, *A proposal for Toeplitz matrix calculations*, Stud. Appl. Math., 74 (1986), pp. 171–176.

[16] L. N. Trefethen, *Approximation theory and numerical linear algebra*, in Algorithms for Approximation II, M. Cox and J. C. Mason, eds., Chapman and Hall, London, 1988.

[17] G. A. Watson, *An algorithm for the inversion of block matrices of Toeplitz form*, J. Comput. Mach., 20 (1973), pp. 409–415.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 4, pp. 967–993, July 1992

008

DOMAIN DECOMPOSITION WITH LOCAL MESH REFINEMENT*

WILLIAM D. GROPP† AND DAVID E. KEYES‡

Abstract. A preconditioned Krylov iterative algorithm based on domain decomposition for linear systems arising from implicit finite-difference or finite-element discretizations of partial differential equation problems requiring local mesh refinement is described. To keep data structures as simple as possible for parallel computing applications, the fundamental computational unit in the algorithm is defined as a subregion of the domain spanned by a locally uniform tensor-product grid, called a tile. In the tile-based domain decomposition approach, two levels of discretization are considered at each point of the domain: a global coarse grid defined by tile vertices only, and a local fine grid where the degree of resolution can vary from tile to tile. One global level and one local level provide the flexibility required to adaptively discretize a diverse collection of problems on irregular regions and solve them at convergence rates that deteriorate only logarithmically in the finest mesh parameter, with the coarse tessellation held fixed. A logarithmic departure from optimality seems to be a reasonable compromise for the simplicity of the composite grid data structure and concomitant regular data exchange patterns in a multiprocessor environment. Some experiments with up to 1024 tiles are reported, and the evolution of the algorithm is commented on and contrasted with optimal nonrefining two-level algorithms and optimal refining multilevel algorithms. Computational comparisons with some other popular methods are presented.

Key words. domain decomposition, elliptic problems, parallel algorithms, mesh refinement

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

1. Introduction.

The combination of domain decomposition with preconditioned iterative methods provides a framework that extends the usefulness of numerical techniques for certain special partial differential equation (PDE) problems to those of more general structure. Nonsmooth features, nonseparable geometries, or massive sizes of practical problems limit the application of many "standard" numerical techniques. Direct methods are rapidly defeated by problem size. "Fast" methods that take advantage of special coefficient and grid structure often do not apply globally. Iterative methods often depend, for efficient implementation, on regular grids that, if global in extent, are inconsistent with accurate and economical resolution of the physics of the problem. However, the domains of problems with these features can often be decomposed into smaller subdomains of simpler structure, increasing the utility of extant software libraries, particularly as components of preconditioners. Moreover, the domain decomposition can be made to produce a convenient mapping of many problems onto medium-scale parallel computers. Our primary focus in this paper is the incorporation of spatially varying mesh refinement requirements into a domain decomposition algorithm based on finite differences. We illustrate the convergence behavior of the algorithm on a variety of two-dimensional elliptic PDE problems,

$$\mathcal{L}u = f \text{ on } \Omega, \quad \text{with } au + bu_n = g \quad \text{on } \mathrm{d}\Omega, \tag{1}$$

* Received by the editors September 13, 1989; accepted for publication (in revised form) April 23, 1991.

† Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois 60439-4844. The work of this author was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy contract W-31-109-Eng-38.

‡ Department of Mechanical Engineering, Yale University, New Haven, Connecticut 06520-2159. The work of this author was supported by the National Science Foundation under contracts EET-8707109 and ECS-8957475. This work was performed in part while the author was visiting the Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois 60439-4844.

including nonselfadjoint, nonseparable geometry cases. We also point out features of the method that are relevant to a parallel implementation. We defer most discussion that would be distinctly architecture- and performance-related to a companion paper [27].

Many PDE problems that are "large" in the discrete sense are so because the continuous problems from which they are generated require resolution of several different length scales for the production of a meaningful solution. The value of compromising between the extremes of globally uniform refinement (which leads to simple and usually vectorizable algorithms but wastes time and memory) and pointwise adaptive refinement (which minimizes the discrete problem size for a given accuracy requirement but leads to complicated data structures) has been recognized for some time and described in contexts too numerous to acknowledge fairly. Locally uniform mesh refinement (LUMR) characterizes one such class of discretizations, based on composites of highly structured subgrids. Many treatments of LUMR in the literature pertain to explicit methods for transient problems, a class with its own advantages (see [4] and references therein) and limitations [45], which is somewhat distinct from ours. Implicit treatments of locally regular refinement for elliptic problems include approaches arising out of classical multigrid ([8]; see [38] for a concise state-of-the-art treatment), a nonconforming spectral technique [35], and methods rooted in iterative substructuring for finite-element problems [6], [16].

Computationally practical locally uniform grids are usually expressible as the union of a coarse uniform tensor-product grid covering the entire domain with one or more refined tensor-product grids defined over subregions, including the possibility of multiple, nested levels. Generalizations of these grids within the LUMR framework include allowing the grids at any particular level of refinement to themselves be the union of tensor-product subgrids, and reinterpreting "uniform" as "quasi-uniform" to allow general curvilinear coordinates for custom body- or solution-fitting. Few *parallel* implementations of schemes of this generality have been reported thus far. Selected for consideration here is a structurally restricted form of LUMR in which refinement occurs exclusively within complete cells of a quasi-uniform coarse grid, as described in §3 below.

The goal of the present contribution is an LUMR methodology with starkly simple data structures for efficient portability to a variety of parallel machines. Current implementations on two distributed- and two shared-memory parallel machines share approximately 98 percent common code measured by line count, inclusive of comments, exclusive of standard libraries. The methodology borrows from the mesh refinement and domain decomposition literature and from the authors' own experience in these areas and in parallel computation [23], [24], [34]. The serial arithmetic complexity bows somewhat to modularity, portability, and overall parallel performance, in which we include both efficiency and total execution time. For example, by refining only in units of full coarse-grid cells, we often impose a tendency towards refinement in subregions where it would be unnecessary from the viewpoint of truncation error alone. As another example, the convergence rate of many domain decomposition algorithms is mildly dependent upon a coarse-grid resolution which may be chosen with criteria beyond convergence rate, such as the balance of work among multiple processors. Our algorithm therefore does not scale (with constant problem size) to indefinitely large numbers of processors, but it does sit comfortably on today's MIMD supercomputers. We comment in the final section about a hybridized two-level algorithm suitable for massive parallelism on an MIMD cluster of SIMD subclusters.

Prior to the discussion of LUMR, §2 describes a domain decomposition algorithm employing "nearly" parallel preconditioners in conjunction with generalized minimal residual (GMRES) iteration, a nonstationary method not dependent upon operator symmetry. In two dimensions, the preconditioner involves three phases: a global coarse-grid solve, independent solves along interfaces between subdomains, and independent solves in the subdomain interiors. The global coarse-grid solve is an essential feature, as it provides the only global exchange of information in the preconditioner itself. We introduce a simple "tangential" operator preconditioning for the subdomain interfaces that is preferable to the interface probe preconditioning advocated in our earlier work on convective-diffusive systems with stripwise decompositions [32]. We also prefer exact subdomain solves to incomplete factorizations. These "exact solves" can be performed by multigrid if the subdomains become too large for direct methods. For multicomponent problems in which source terms codominate with convection and diffusion, block incomplete factorization may also be an economical subdomain solver.

The main body of the paper (§4) is a collection of numerical experiments on two-dimensional elliptic boundary value problems (BVPs). The experiments include reentrant domains, nonselfadjoint operators, and mixed boundary conditions. Up to 1,024 coarse-grid elements, called *tiles*, are used. The last two subsections of §4 compare the boxwise decompositions used throughout the paper with stripwise decompositions exploiting physical anisotropies, as well as with some conventional undecomposed solvers.

Section 5 indicates some future directions for this methodology.

2. An iterative domain decomposition algorithm.

Preconditioned iterative methods and domain decomposition provide a framework that includes a wide class of algorithms. This framework comprises four elements:

1. a global operator arising from the discretization of the PDE (or system of PDEs);
2. an approximate inverse, or preconditioner, for the global operator;
3. an iterative method requiring only the application of the preconditioned operator; and
4. a geometry-based partition of the discrete unknowns so that size, locality, and uniformity can be exploited in forming the action of the preconditioned operator.

Since the numerical analysis literature contains many successful discretization schemes and iterative methods specialized for different operator properties, such as the presence or absence of definiteness and symmetry, the recent burgeoning effort in iterative domain decomposition algorithms has concentrated primarily (though not exclusively) on the interaction of the second and fourth of these elements. In the parallel context, this is a natural preoccupation because the bottleneck to parallelism usually (though not exclusively) lies in the requirement of the global transport of information in the preconditioner.

2.1. Iterative methods and operator structure.

Many of the numerical examples described in §4 rule out the use of iterative methods based on symmetry but permit the assumptions of definiteness and diagonal dominance. In particular, full or incomplete factorizations of preconditioner matrix blocks can be undertaken without pivoting. Because of its robustness, we adopt the parameter-free generalized minimal residual (GMRES) method [43] as the outer iteration. The main disadvantages of GMRES, its linear and quadratic (in iteration index) memory and execution time requirements, respectively, must be mitigated by scaling and preconditioning. For other acceleration schemes, such as Chebyshev, the memory and execution time requirements may be only constant and linear, respectively; however, GMRES dispenses with the difficulty of estimating parameters. The recently proposed, parameter-free, bounded recursion Bi-CGSTAB method [14] combines the above-mentioned advantages and deserves further study. In preliminary tests we have found it usually to be competitive in execution time with GMRES, but it can in some instances be substantially slower. Other methods for the iterative solution of nonsymmetric systems, such as QMR [21], also deserve broader investigation. In solving $Au = b$ each of these methods requires an initial iterate for x that it improves through repeated calls to a routine forming the product of A with a direction vector. For improved convergence we employ a change of variables, iteratively solving $(AB^{-1})y = b$ for y and then $Bx = y$ for x. Here, B is a right preconditioner matrix, whose inverse action should be convenient to compute and should cluster the eigenvalues of AB^{-1}. A arises from an FD, FE, or FV discretization, with a local stencil. The stencil is regarded as uniform in this section and generalized in §3.

The type of domain decomposition used here involves unit aspect ratio subdomains, as opposed to thin strips joining opposite sides of a domain; therefore, interior subdomain vertices are created. We denote all subdomain vertices "cross points" but distinguish between interior and boundary cross points. Ordering the interior points as well as the physical boundary points other than cross points first, the interfaces connecting the cross points plus the cross points on the boundary next, and the interior cross points last imposes the following outer tripartition on the global discrete operator A:

$$A \equiv \begin{pmatrix} A_I & A_{IB} & A_{IC} \\ A_{BI} & A_B & A_{BC} \\ A_{CI} & A_{CB} & A_C \end{pmatrix}. \tag{2}$$

Note that the partitions vary greatly in size. If H is a quasi-uniform subdomain diameter and h a quasi-uniform fine-mesh width, the discrete dimensions of A_I, A_B, and A_C are $\mathcal{O}(h^{-2})$, $\mathcal{O}(H^{-1}h^{-1})$, and $\mathcal{O}(H^{-2})$, respectively. The numerical experiments described below employ a five-point stencil (extended in [33] to second-order upwind differencing with a skew six- or seven-point stencil). No cross-derivative terms appear; therefore, there are no corner points in the stencil and blocks A_{IC} and A_{CI} may be set to zero.

The outer structure of our preconditioner B may be a conformally partitioned block upper triangular matrix:

$$B = \begin{pmatrix} \tilde{A}_I & \tilde{A}_{IB} & \tilde{A}_{IC} \\ 0 & B_B & \tilde{A}_{BC} \\ 0 & 0 & B_C \end{pmatrix}, \tag{3}$$

whose components are elucidated in the next subsection. The application of B^{-1} to a vector $v = (v_I, v_B, v_C)^T$ consists of solving $Bw = v$ for $w = (w_I, w_B, w_C)^T$. It begins with a cross-point solve with B_C for w_C. This updates through $\tilde{A}_{BC}$ the right-hand sides of a set of independent interface solves for subvectors of w_B. These, in turn, update the right-hand sides through $\tilde{A}_{IB}$ of a set of independent interior solves for subvectors of w_I. For a stencil with corner points, $\tilde{A}_{IC}$ would be nonzero, and the cross-point result would also update the interior block right-hand sides. Within the preconditioner, however, there is no dependence of the interface solution upon the result of the interior solution, or of the cross-point solution upon either. This fact distinguishes the method from [7] and [10] and means that the $\mathcal{O}(h^{-2})$-sized block of the preconditioner is visited only once per iteration.

A useful optimization is available when the tilde quantities in the top row are taken equal to their tilde-free counterparts. In this case, it is readily verified that the right-preconditioned form of the operator is

$$AB^{-1} = \begin{pmatrix} I & 0 & 0 \\ * & (A_B - A_{BI}A_I^{-1}A_{IB})B_B^{-1} & * \\ 0 & * & (A_C - A_{CB}B_B^{-1}\tilde{A}_{BC})B_C^{-1} \end{pmatrix}, \tag{4}$$

where the starred blocks are nonzero (and not necessarily small). The identity block row means that $\mathcal{O}(h^{-2})$ of the unknowns in the Krylov vectors can go untouched (except for scaling) throughout the entire solution process until the preconditioning is unwound in the final step, after the interface and cross-point values have converged. Since A_I^{-1} is needed to advance the solution on these separator sets, we cannot entirely escape solving subdomain problems, but substantial arithmetic work can be saved.

2.2. Components of the preconditioner. The derivation of the coefficients of the preconditioner blocks is as follows. The cross-point operator B_C is simply an H-scale coarse-grid discretization of the continuous PDE. To accommodate Neumann or Robin boundary conditions, we include physical boundary points lying at subdomain vertices in the cross-point system in this step. (Later the boundary cross-point values are overwritten with the results of more accurate h-scale data from the interface solve. This distinction is, of course, moot for Dirichlet data but important for boundary conditions involving spatial gradients.) The cross-point system right-hand side at interior points can be taken as the injected vertex values of the fine-grid right-hand side, though we remark on a better choice below. The current implementation supports a direct solve on the coarse-grid system. If strip decompositions are used, there is no cross-point system, and the lower right block of the preconditioner consists only of the interface system described next.

Unlike the coarse-grid and subdomain interior problems, which possess the full physical dimension of the domain of the underlying PDE and thus inherit a literature full of preconditionings, the lower-dimensional interfacial equations are properly derived from a pseudodifferential trace operator. (See [5, §3] for an informal theory.) The tangential interface preconditioner we employ below is the one-dimensional discretization of the terms of the underlying operator that remain when the derivatives normal to the interface are set to zero. It is equivalent to solving a two-point BVP with boundary conditions inherited from the interior cross-point values or from the physical boundary.

The subdomain interior equations consist of approximate fine-grid discretizations of the PDE over local regions, with physically appropriate boundary conditions along any true boundary segments and Dirichlet boundary conditions derived from the already available w_B at artificial interfaces. Only first-order interior differences are accommodated in the physical boundary conditions of the preconditioner, though first- *or* second-order boundary conditions may be elected in the operator A. The current implementation supports full LU Gaussian elimination with both banded and sparse data structures, fast cyclic reduction, incomplete LU decomposition, and modified incomplete LU decomposition. To maintain a reasonable scope, we concentrate on full elimination results here. Full elimination on the interiors yields the best iteration counts, although not always the best execution times (for large H/h). Like the interface solves, each subdomain interior solve may be performed independently.

2.2.1. A better variant of the cross-point system. In keeping with an exposition that is as independent as possible of particular discretization techniques, the right-hand side of the cross-point system was assumed above to be the injected vertex values of the fine grid weighted by the subdomain areas instead of the grid-cell areas. It is necessary, however, to rely on a finite-element discretization with a hierarchical basis to properly motivate the construction of a better cross-point system. In particular, we have obtained faster convergence by using the function space decomposition approach of [7], which yields essentially the same coefficient block B_C but replaces the simple injection of fine-grid values with ramp-weighted averages of interface values along all interfaces feeding a given cross point. Specifically, the element of the right-hand side v_C corresponding to an interior subdomain vertex is a discrete approximation to one-quarter of the sum of four line integrals of the form

$$\frac{2}{H}\int_0^H v(s)\left(1-\frac{s}{H}\right)\, ds\ , \tag{5}$$

where s parameterizes the interfaces leading from the vertex in question. This leads to the following sequence of steps to produce a preconditioned matrix-vector product u from input v, where $v = (v_I, v_B, v_C)^T$:

First, v_C is reweighted according to (5). The reweighting has the matrix representation $v' = Cv$, where

$$C = \begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & K & J \end{pmatrix},$$

J is diagonal with all positive elements, all elements of K are nonnegative, and the row sums of K and J together give unity. Then, as above, we solve for $w = B^{-1}v'$ and multiply by A to get $u = Aw$. Thus, the preconditioned matrix-vector product is $u = AB^{-1}Cv$. Treating everything apart from A itself as the effective preconditioner Q, we find that $Q^{-1} = B^{-1}C$, or equivalently, $Q = C^{-1}B$, which is straightforwardly seen to be

$$Q = \begin{pmatrix} \tilde{A}_I & \tilde{A}_{IB} & \tilde{A}_{IC} \\ 0 & B_B & \tilde{A}_{BC} \\ 0 & -J^{-1}KB_B & J^{-1}(B_C - K\tilde{A}_{BC}) \end{pmatrix}.$$

When tilde-free quantities are used in the first block row, AQ^{-1} has a first block row equal to the identity. Thus, the remark following (4) about not touching the upper portion of the Krylov vectors, except for scaling, remains valid. Although the preconditioner with ramp weighting of the right-hand side of the cross-point system is no longer strictly block triangular, it still requires only one solve with A_I per iteration.

2.3. Parallelism in the preconditioner. We note that the permutation into block matrix form described in this section is a purely formal one for notational convenience. The data structure used in a computer implementation is a local natural ordering of gridpoints within a natural ordering of tiles, as detailed in §3 below. The parallelism within $\tilde{A}_I$ and B_B is not visible at the level of blocking in (3), but the parallel bottleneck represented by communication-intensive B_C and the sequential use of that result in multiplications with $\tilde{A}_{BC}$ (and, generally, $\tilde{A}_{IC}$) blocks *is* evident. We mention variants of the algorithm that alleviate this bottleneck at the price of some extra local work and extra storage.

The solve with B_C itself can be performed in any of three ways: redundantly on each processor after broadcasting the required coefficient data, with single-threaded code between collecting the coefficients on a single processor and redistributing the results, or in a fully (or partially) distributed fashion. Determination of the most efficient technique is generally decomposition- and network-dependent, since problem size and computation-to-communication ratios enter the complexity estimate in nonisolable ways. Some global data exchange is necessary in this phase, so it may be desirable to prevent idling on a given multiprocessor to allow the remaining local exchange phases to proceed before the cross-point results are available.

The sequentiality of the cross-point solve can be broken by the following technique, which exploits the relatively small size of the cross-point system. Lumping the balance of the unknowns together, let (3) be condensed to

$$B = \begin{pmatrix} B_h & B_{hH} \\ 0 & B_H \end{pmatrix},$$

where B_h contains the upper 2×2 blocks of (3), and B_H is just another name for B_C. Consider the application of the preconditioner

$$B \begin{pmatrix} w_h \\ w_H \end{pmatrix} = \begin{pmatrix} v_h \\ v_H \end{pmatrix},$$

with the sequential solution

$$\begin{aligned} w_H &= B_H^{-1} v_H, \\ w_h &= B_h^{-1}(v_h - B_{hH} w_H). \end{aligned}$$

A preprocessing step can compute and store the vectors $g_k = (B_h^{-1})(B_{hH})e_k$, $k = 1, \cdots, K$, where there are K interior cross points and e_k is the kth unit vector in this K-dimensional space. Once $w_H = B_H^{-1} v_H$ and $w_h^{(1)} \equiv B_h^{-1} v_h$ are independently solved for, we can (through local computations) form $w_h = w_h^{(1)} - \sum_k (w_H)_k g_k$. By construction, the support of each g_k is limited to the four tiles sharing vertex k, and the cross points possess a four-coloring that allows the g_k to be computed in just four sets of independent subdomain solves (for a scalar PDE). This process was inspired by, and has an interpretation in terms of, function space decompositions. Indeed, the function space framework is critical in generalizations to multilevel preconditioners, but for a two-level preconditioner the algebraic description above is sufficient.

3. Mesh refinement by tiles. This section describes a simple mesh refinement philosophy based on a regular tessellation of two-dimensional domains into subdomain "tiles." A tile is a tensor-product of half-open intervals in each coordinate direction, except that a tile abutting a physical boundary along what would ordinarily be one of its open edges is closed along that edge. Each tile possesses its own interior, at least two of its four sides, and at least one of its four corners and is locally discretized on a tensor-product grid. Although the specific convention is arbitrary, we assume for definiteness that in its own local right-handed coordinate system, each tile contains its origin and its x and y axes (see Fig. 1).

We require that the cross points be embeddable in a tensor-product global quasi-uniform coarse grid, from which only points lying exterior to the (possibly multiply connected) boundary are missing. Irregular tiling patterns such as in Fig. 2(b) are ruled out for convenience in setting up the coarse-grid system and keeping the code

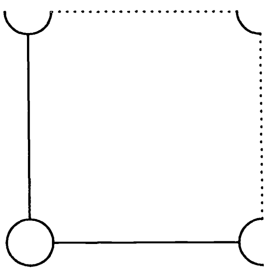


FIG. 1. *The anatomy of a tile. Unless closed by a physical boundary, a tile is open along its high-x and high-y perimeter.*

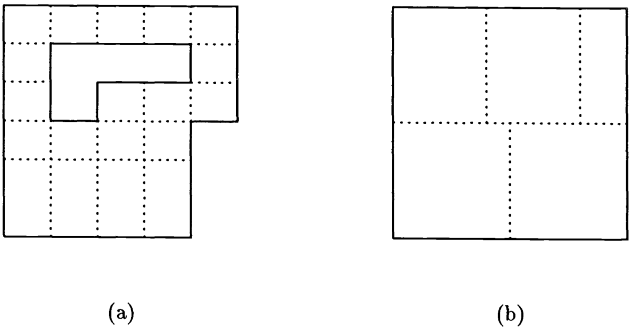


FIG. 2. *Sample tessellations:* (a) *is permissible,* (b) *is not.*

that manages the interfacial data exchanges short. However, there is no requirement that the domain itself be of tensor-product type; the decomposition in Fig. 2(a) is permissible. Without coordinate stretching and other body-fitted coordinate transformations, the embedding requirement would generally enslave the granularity of the decomposition to the geometric complexity of the domain, a situation that we wish to avoid since granularity has important implications on load balance and convergence rate. Although we have yet to fully implement them, domain-wide coordinate transformations represent a simple extension in principle. From an algebraic point of view, an orthogonal body-fitted coordinate transformation is indistinguishable from a perturbation to the operator coefficients. Preserving orthogonality should create less of a strain on a mesh generator acting over local regions than it does in much current practice using global mappings.

Associated with each tile is the data defined over a quasi-uniform grid covering its portion of the domain and a set of operators for executing its block-row portions of the preconditioner solve, as described in §2. In our object-oriented approach, these operators can vary widely from tile to tile. In our present examples, however, we assume that the grids covering individual tiles share a common parent uniform tile (of arbitrary discrete size) and are refined only in powers of two. We can therefore later indicate refinement levels using the graphical shorthand of Fig. 9, where the

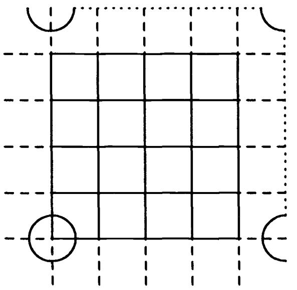


FIG. 3. *Sample tile, showing the computational buffer region (dashed extensions) required for the completion of standard five-point stencils centered at the points of the local grid.*

integer indicates the logarithm of the refinement ratio.

3.1. Tile-tile interfaces. To minimize restrictions on the structure of adjacent tiles (and to eliminate redundant communication between tiles in a multiprocessor implementation in which different tiles will generally be assigned to different processors), each tile stores and maintains, in addition to its own data, the data associated with a buffer region of phantom points equal in width to one-half of that of its associated discrete stencil. Figure 3 illustrates the buffer unknowns for a five-point stencil, superimposed on Fig. 1. With the exception of these redundant phantom points, each point of the domain is uniquely associated with a single tile.

Data at the phantom points is supplied in a manner dependent upon the internal structure and refinement ratios of the associated adjacent tiles. A finer tile obtains biquadratically interpolated data from its coarser neighbor. Since the problems studied here involve second-order operators, this allows the use of conventional finite-difference techniques in generating the difference equations at the subdomain interfaces. (Bilinear interpolation alone would limit the potential accuracy of a second-order differencing scheme, as observed in some preliminary experiments.) A coarser tile obtains its data by simple (unweighted) injection. That is, the value at the point in the finer neighboring tile that lies on the extended coarser tile stencil is scaled appropriately and used in the coarser grid.

We note that such a simple scheme neither guarantees discrete flux conservation nor delivers a symmetric A for a selfadjoint $\mathcal{L}$. However, the algebraic method does not depend on either property. The focus of this paper is on the solution of a consistent set of discrete equations. More careful attention has been given to the conservation properties of the discretization in the context of locally regular refinement in [19] and [38], for instance.

Each iteration of GMRES requires multiplying with A, which involves at most nearest-neighbor data exchanges between tiles to complete the local stencils, and solving with B, which likewise requires only nearest-neighbor data exchanges to form right-hand sides, *apart from* the globally cooperative task of solving with B_C.

The selection of refinement criteria is a much studied, yet still open, problem; see [30] and the collections [2] and [20] for representative work in this area. The refinement criteria, however, are orthogonal to the equation-solving aspect considered here, except to the extent that a part of the computational work required by one of these tasks may be a by-product of the other. In the examples, "good" refinement strategies can be done manually.

In general, tile interfaces can also be the site of changes in the discretization in addition to the refinement level. For instance, the discrete stencil can change order at interfaces. Even the form of the operators or the number of dependent variables can change at interfaces while still preserving the subdomain uniformity required for efficient subdomain solution algorithms. As a motivational example, a reacting flow problem frequently consists of large regions in which there is only transport of mass, momentum, and thermal energy but no reaction among stable constituents to all adequate orders of approximation. In other regions it is essential to retain a full set of composition variables, including trace radicals, and reaction terms must also be retained in the equations. To accommodate such generality, the routines that pack the buffer regions are responsible for providing the necessary mappings.

3.2. Physical boundaries. For generality, the equations for the physical boundaries are incorporated into the overall system matrix, including Dirichlet conditions. Our implementation allows inhomogeneous Robin boundary conditions at all boundary points, namely,

$$a(x,y)u + b(x,y)\frac{\partial u}{\partial n} = c(x,y).$$

Either first- or second-order one-sided difference approximations to the normal derivative term may be employed in the actual operator, but only first-order approximation is used in the preconditioners (to preserve uniformity of bandwidth). Although tempting in their simplicity, Dirichlet boundary conditions alone in the preconditioner were found to perform poorly in practice in mixed boundary condition (BC) problems, as expected. The hierarchical structure of the preconditioner renders the BC mismatch between the operator and preconditioner difficult to study theoretically. The theory in [36] and [40] reveals that spectral equivalence is generally lost in such BC mismatches, but only a small number of eigenvalues of the preconditioned operator may be responsible.

3.3. Comparison with other approaches. Before appealing to numerical experimention to illustrate the techniques presented above, we briefly compare them with other known techniques arising from similar motivations.

The field of locally uniform mesh refinement is spanned by a continuum of resolution strategies governed by clustering rules that control the size and shape of the refined subregions. Global refinement lies at one extreme and pointwise adaptive refinement at the other. As soon as the global tensor-product mesh is abandoned, a host of difficult practical decisions must be made about data structures and clustering algorithms. The logic required to handle the numerous types of subgrid-subgrid interactions that can arise and to ensure the consistency of the possibly distributed data structure can be a significant impediment to efficient parallelism. It is impractical to use domain-based "horizontal" decompositions to obtain distributed parallelism if refined subgrids are allowed to span the coarse grid in a general nested fashion. Instead, parallel decompositions of general, multilevel, locally uniform, composite grids should proceed by level, as argued and implemented in [37]. However, "horizontal" neighbor-neighbor interactions on a tensor-product grid of individually refined tiles are simple.

The tile algorithm requires only one grid that possesses connectivity with arbitrarily distant regions of the domain, namely, the grid of cross points. In the framework

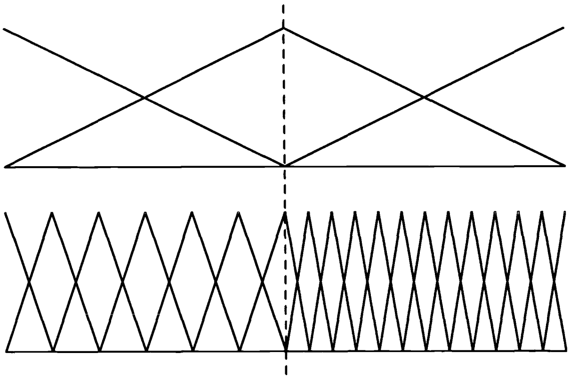


FIG. 4. *One-dimensional schematic of the tile basis functions.*

of the hierarchical basis function technique [3], [47], we have simply a two-level hierarchy, but the higher level may be different in different subregions. Figure 4 is a one-dimensional illustration. This represents a severe condensation of the range of intermediate scales present in multilevel local uniform refinement, on which the asymptotic convergence theory is based. Tiles are much closer to being the software equivalent of the "geometry-defining processors" (GDPs) of [1]. The tile algorithm shares the philosophy of commercial structural analysis packages offering libraries of elements that an engineer can assemble in composing a domain, although comparably transparent user interfaces have yet to be written. *Unlike* most structural analysis packages, no global linear system involving all of the degrees of freedom is formed, nor is an exact Schur complement derived through the expensive process of static condensation. Rather, an iterative path to parallelism is elected.

In the latter respect, the tile algorithm is similar to the original additive Schwarz method [15] and the techniques of [7]. All three rely upon a single, coarse-domain–spanning grid. The main differences between the techniques of [7] and [15] and the tile algorithm are in the treatment of the interfacial degrees of freedom. In the additive Schwarz technique, interior problems are solved on extended overlapped subdomains, so that the interfacial degrees of freedom of one subdomain are interior points of another and thus demand no special consideration. In [7], good preconditioners for the interfacial degrees of freedom of abutting subdomains are derived theoretically for selfadjoint operators. Near optimal algebraic convergence for the refined case has been proved for both classes of algorithms in [16] and [6], respectively, for selfadjoint systems. For nonselfadjoint systems, convergence proofs for the uniformly refined case have been given in [9] and [11] for overlapping and in [10] for abutting decompositions, respectively.

A disadvantage shared by all two-scale approaches is that the coarse grid—on which the optimal approaches perform an exact solve, and on which we also prefer one—cannot necessarily remain as coarse as one might like. In contrast, multilevel methods are not held hostage to a fine "coarse" grid. Even so, multilevel convergence estimates for nonselfadjoint operators are aided by sufficiently fine coarse grids, and complex domain geometry or "ragged" coefficients can also make a fine coarse grid

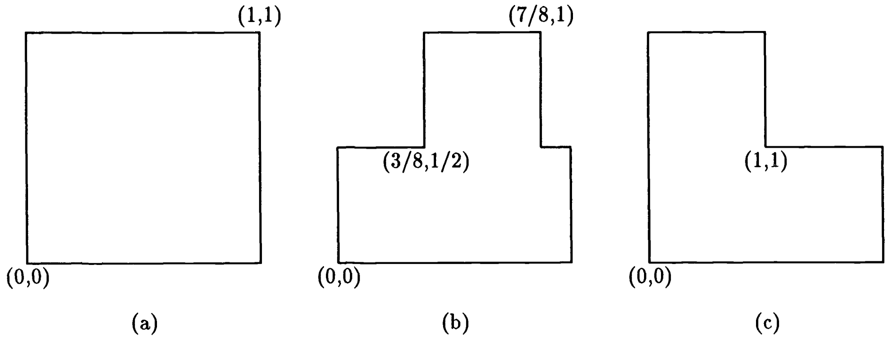


FIG. 5. *The three domains considered in this paper.*

desirable in practice.

General multilevel methods with a number of levels substantially larger than two maintain their optimal convergence rates at the price of increasingly complex data-dependency patterns with attendant degradation on multiprocessor architectures and intricacy of coding in practical problems. The additive or asynchronous methods [38] relieve most of the interlevel data traffic but do not obviate the need to collect data vertically across the levels at each iteration. The ability of a two-level approach to obtain convergence rates only a log factor worse than optimal is demonstrated in §4. Compelling overall superiority of approaches with a greater richness of scales has not been established in production-parallel software. In the course of establishing it, experience on parallel computers with a two-level algorithm will be beneficial and will aid in evaluating the complex tradeoffs.

We have too little experience with the full spectrum of methods discussed above to conjecture about the sizes of the relevant constants in asymptotic complexity analyses or to provide experimental comparisons (but see [10] for a comparison of the tile algorithm with additive Schwarz on a model scalar convection-diffusion problem). It is clear, however, that the limitations of the tile algorithm are shared to some degree by the optimal methods, while the simplicity of implementation and straightforwardness of generalization are not universally shared.

4. Numerical experiments.

To illustrate the effectiveness of the tile algorithm in terms of the convergence of the iterations, and the effectiveness of the locally uniform mesh refinement in terms of the convergence of the discretization, we consider a suite of experiments.

4.1. Model problems.

We present ten model problems, each containing a single dependent variable and two independent variables. Some of the problems below are selfadjoint and could be discretized in a symmetric manner and perhaps solved more cheaply with conjugate gradients than with GMRES. Our main interest, however, is in the more extensible formulation. In all the examples an exact solution of the continuous problem $\mathcal{L}u = f$ is specified. From this $u(x, y)$, all of the source terms f and boundary condition inhomogeneities g may be calculated. In cases where the expressions for f and g are sufficiently simple, they are written out along with the solution. The ten problems are defined over three different domains, pictured in Fig. 5.

The first two examples, with constant coefficients and an exact solution quadratic in each independent variable, are extremely simple and possess second-order finite difference representations free from truncation errors. They are identical except for the type of boundary conditions along one side of their square domain. These problems are not candidates for mesh refinement; rather, they were chosen to illustrate the deterioration in convergence rate caused when Dirichlet boundary conditions are replaced with Neumann, and to allow controlled experimentation on the effect of mismatched boundary conditions in the preconditioner. The poor convergence of Problem 2 using the preconditioner of Problem 1 originally forced the decision to expand the cross-point system to include physical boundary points in the general case.

PROBLEM 1. Pure isotropic diffusion with all Dirichlet boundaries.

$$\begin{gathered} \nabla^2 u = 4, \\ u(x,y) = x^2 + y^2, \\ \text{Dirichlet data on } \partial\Omega, \\ \Omega = \text{Unit square.} \end{gathered}$$

PROBLEM 2. Pure isotropic diffusion with a partial Neumann boundary.

$$\begin{gathered} \nabla^2 u = 4, \\ u(x,y) = x^2 + y^2, \\ \text{Dirichlet data on the three lower sides of } \partial\Omega, \\ \frac{\partial u}{\partial n}(x,1) = 2, \\ \Omega = \text{Unit square.} \end{gathered}$$

The next example is included to study orientation sensitivity of the substructuring resulting from anisotropic diffusion, for comparison with Problem 1, to which it is identical when $a = 1$. It is of further interest in that the order-of-magnitude ratio between the diffusion coefficients in the x and y directions is mathematically indistinguishable at the discrete level from an order-of-magnitude physical domain aspect ratio in an isotropic diffusion problem. Thus, the discretized version of Problem 3 covers two physical problem parameter extremes in one.

PROBLEM 3. Anisotropic diffusion.

$$\begin{gathered} \frac{\partial}{\partial x}\left(a\frac{\partial u}{\partial x}\right) + \frac{\partial^2 u}{\partial y^2} = 2(a+1), \\ u(x,y) = x^2 + y^2, \\ a = 10, \\ \text{Dirichlet data on } \partial\Omega, \\ \Omega = \text{Unit square.} \end{gathered}$$

The fourth example is a prototype convection-diffusion problem: a passive scalar in a plug flow that is well developed at the outflow. It is a companion problem to Problem 2 in the sense of possessing a smooth solution with one Neumann boundary, but it is asymmetric as a result of the convection term. In that its anisotropy comes from a first- rather than second-order operator, it also complements Problem 3.

PROBLEM 4. Plug-flow convection-diffusion with fully developed outflow boundary.

$$-\nabla^2 u + c\frac{\partial u}{\partial y} = f,$$

$$u(x,y) = \sin(\pi x)\sin\left(\frac{\pi y}{2}\right),$$
$$c = 10,$$
$$u = 0 \text{ on the three lower sides of } \partial\Omega,$$
$$\frac{\partial u}{\partial n}(x,1) = 0,$$
$$\Omega = \text{Unit square.}$$

The next two canonical examples (from the "population" of elliptic problems in [41] and [42]) bring in nonconstant coefficients, the latter in a nonselfadjoint way with Robin boundary conditions.[1]

PROBLEM 5. Selfadjoint, nonconstant coefficient, Dirichlet boundaries.

$$\frac{\partial}{\partial x}\left(e^{xy}\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(e^{-xy}\frac{\partial u}{\partial y}\right) - \frac{u}{1+x+y} = f,$$
$$u(x,y) = e^{xy}\sin(\pi x)\sin(\pi y),$$
$$u = 0 \text{ on } \partial\Omega,$$
$$\Omega = \text{Unit square.}$$

PROBLEM 6. Nonselfadjoint, nonconstant coefficient, Robin boundaries.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial}{\partial y}\left((1+y^2)\frac{\partial u}{\partial y}\right) - \frac{\partial u}{\partial x} - (1+2y+y^2)\frac{\partial u}{\partial y} = f,$$
$$u(x,y) = 0.135(e^{x+y} + (x^2-x)^2\log(1+y^2)),$$
$$u - \frac{\partial u}{\partial n} = g \text{ on } \partial\Omega,$$
$$\Omega = \text{Unit square.}$$

The derivative is the outward normal.

The seventh example, from [5] and [31], is on an irregularly shaped domain with reentrant corners, but possesses a smooth solution. It emphasizes how an irregular domain may force a minimum granularity upon a tessellation comprising congruent tiles. For the problem at hand, however, the minimum granularity is near the ideal one.

PROBLEM 7. T-shaped domain.

$$\nabla^2 u = 4 - 2\cos(y)e^x,$$
$$u(x,y) = x^2 + y^2 - xe^x\cos(y),$$
$$\text{Dirichlet data on } \partial\Omega,$$
$$\Omega = \text{T-shaped region.}$$

The last three examples are obtained by taking three different values of the convection—respectively, $c = 0$, $c = -1$, and $c = 10$—in the convection-diffusion problem below.

PROBLEMS 8–10. Cylindrically separable reentrant corner convection-diffusion problem.

$$-\nabla^2 u + \frac{c}{r}\frac{\partial u}{\partial r} = 0,$$

[1] The more widely available reference [29] contains an identical listing of Problem 5 and a similar but not identical version of Problem 6. A typographical error in the latter renders it ill posed.

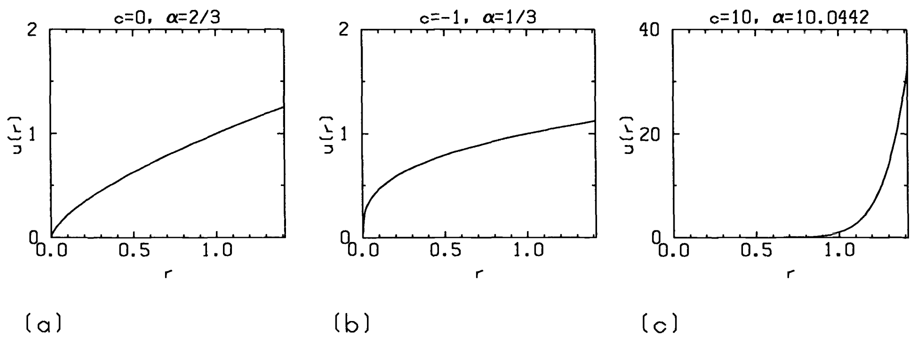


FIG. 6. *Cross section of $u(r)$ along the symmetry axis:* (a) *Problem* 8, *pure diffusion, nondifferentiable at $r = 0$;* (b) *Problem* 9, *convective inflow, strengthening the singularity;* (c) *Problem* 10, *convective outflow, eliminating the singularity.*

$$u(x,y) = r^{\alpha} \sin\left(\frac{2}{3}\left(\theta - \frac{1}{2}\pi\right)\right),$$
$$\text{where } r = \sqrt{(x-1)^2 + (y-1)^2},$$
$$\text{and } \theta = \arg((x-1) + i(y-1)),\ 0 \le \theta < 2\pi,$$
$$\text{Dirichlet data on } \partial\Omega,$$
$$\Omega = \text{L-shaped region.}$$

The first of these corresponds to pure diffusion, and the second and third to convection in towards the reentrant corner and away from it, respectively, at a rate inversely proportional to the radius. The respective values of the radial eigenfunction exponent α are $\frac{2}{3}$, $\frac{1}{3}$, and approximately 10.04, from the Euler equation formula $\alpha = \left[c + \sqrt{c^2 + (16/9)}\right]/2$. Figure 6 displays $u(r)$ along the ray $\theta = \frac{5\pi}{4}$, which is the symmetry axis of the three L-shaped problems. The first two solutions of this trio lack derivatives at the reentrant corner. The last is everywhere twice differentiable, but the solution is characterized by steep variation in the three *non*reentrant corner regions, where $r > 1$. Local mesh refinement is critical to improving the accuracy of a finite-difference solution. In [27] we show the complementary benefit of rediscretization of the tiles surrounding the reentrant corner in Problems 8 and 9 to fit the discrete solution to the known power-law radial dependence of the singular exact solution.

4.2. Parameters studied. Four categories of experiments are reported. First, a two-dimensional parameter space consisting of coarse-grid resolution and overall (uniform) resolution is explored by numerical experiment for each problem. The goal of these experiments is the evaluation of the convergence of the algorithm, in terms of iteration count and execution time, over a range of resolutions for comparison with a leading-term complexity analysis in §4.3 and related theory in §4.4. No adaptive refinement is performed.

Another set of experiments is performed on Problems 8–10 only with the goal of evaluating the economy of the locally uniform refinement technique. As previously shown in [27], LUMR is capable of significant CPU and memory savings with no sacrifice of accuracy relative to uniform refinement.

In a third set of experiments, the effect of orientation for nonunit-aspect ratio tiles is investigated. The limiting case of stripwise decompositions shows how physical anisotropies can be exploited in the decomposition for improved convergence.

Finally, we compare the domain-decomposed preconditioner of this paper with some popular global preconditioners and with the topologically related direct solve using a nested dissection ordering [22].

Additional studies, including modular replacement of $\tilde{A}_I$ or B_B with some of the alternatives listed in §2, are available in [25], upon which the present article is based. Use of two different orders of discretization in A and B is explored in [33]. (This approach loses the identity block in (4) but delivers higher-order upwinding while preserving monotonicity in the preconditioner.) In this study, we simply use $\tilde{A}_I = A_I$, $\tilde{A}_{IB} = A_{IB}$, and $\tilde{A}_{BC} = A_{BC}$ and derive B_B from the tangential terms of the differential operator.

The timings given below are from a SPARCserver 390 with 64-bit reals. The code was written in C except for low-level Fortran kernels, such as factoring or solving linear systems entirely resident on one processor. *Relative* comparisons of CPU times for alternative formulations of the same problem executed in the same hardware and software environment are an important part of our results. It should be borne in mind while studying the results that different organizations of the code and different compiler capabilities can account for large variations in execution times across architectures and software releases; therefore, *absolute* execution times are not very meaningful. We have run the same experiments (or representative subsets, to the extent supported by memory) in scalar mode on seven other Unix machines and find that even the *proportions* of time spent in factorization and solution phases may vary widely between machines. In spite of this, there are surprisingly few shifts in the overall performance rankings of alternative decompositions. In other words, while the timings in the tables are far from machine independent, the conclusions based thereon *are*, until parallelism enters the picture.

4.3. Convergence as a function of coarse-grid granularity . To test coarse-grid granularity over an interesting range, we fix the finest mesh spacing at $h^{-1} = 128$ (relative to the total length of the domain, whether that be 1 in the problems posed on the unit square or 2 in the problems on the L-shaped domain) and investigate the tradeoff between numbers of tiles and points per tile, as shown in Tables 1 and 2 and plotted in Fig. 7. The mesh is identical and uniform for all runs in these tables (with the obvious exception that pieces of the circumscribing square are missing from it in Problems 7–10, whose columns therefore lack entries at the coarsest tile subdivisions). The convergence criterion is a relative reduction in residual of five orders of magnitude. Throughout these studies we use an initial iterate of zero. Table 1 shows that the iteration count peaks in the middle of the granularity range, at four or eight tiles per side, and decreases to 1 in either degenerate limit of one tile per domain or one tile per point (not shown), where a global direct solve results.

Table 2 shows the deceptiveness of iteration count alone as a measure of overall performance. In execution time, the extreme runs, representing few-domain cases, suffer as a result of the high cost per iteration, even though the number of iterations required is very small. This table is a profound illustration of an earlier version of [12], entitled *Domain Decomposition Beneficial Even Sequentially.* The most favorable total *sequential* execution times are found for multidomain cases at 16 or 32 tiles per side.

The factorization of the banded matrix in the single subdomain case is the dom-

TABLE 1

Iteration count as a function of number of tiles per side of the circumscribing square, H^{-1}, and number of mesh points along a tile side, H/h, at constant refinement parameter, $h^{-1} = 128$, for a reduction in the initial residual of 10^{-5}.

H^{-1}	H/h	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
1	128	1	1	1	1	1	1	–	–	–	–
2	64	10	14	18	25	26	17	–	12	11	4
4	32	11	15	24	25	32	21	–	15	16	15
8	16	9	12	25	21	29	16	11	14	15	16
16	8	7	10	22	18	26	12	10	11	12	13
32	4	6	7	15	14	21	7	8	8	9	8

TABLE 2

Total execution time (sec), including both preconditioner factorization and GMRES *iteration, as a function of number of tiles per unit length, H^{-1}, and number of mesh points along a tile side, H/h, at constant $h^{-1} = 128$, for a reduction in the initial residual of* 10^{-5}.

H^{-1}	H/h	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
1	128	371.	376.	371.	373.	371.	375.	–	–	–	–
2	64	109.	118.	127.	144.	145.	125.	–	86.1	84.2	73.0
4	32	34.7	39.7	51.9	53.3	62.4	47.9	–	30.3	31.4	30.5
8	16	12.0	14.4	26.2	22.5	29.8	17.8	10.2	11.9	12.4	13.1
16	8	5.5	8.0	18.3	14.4	22.1	9.6	5.7	6.0	6.7	7.2
32	4	6.8	7.9	17.9	16.3	27.2	7.9	6.5	6.4	7.1	6.4

inant contribution to the overall time. In Problems 1–6, over six minutes are spent doing the factorization alone. A similar penalty would accrue in an attempt to do direct solves on a very fine "coarse" grid, in which each tile contains just one point. However, this second peak is not visible since the table is truncated below tile sizes of $H/h = 4$. Even in modest-sized two-dimensional problems, direct solves on the undecomposed domain are inefficient relative to decomposition-preconditioned GMRES. Of course, there are many alternatives to direct solves for solving a smooth elliptic equation discretized on a tensor-product grid problem on a uniprocessor, some of which are considered in §4.7, but most are not coded or parallelized as cleanly as domain-decomposed Krylov iteration.

The behavior in Table 2 can be understood with reference to leading-term complexity estimates for the solution and factorization operators of the preconditioner. We observe that there are $\mathcal{O}(H^{-2})$ cross points, interfaces, and interiors. Naturally ordered banded direct factorizations and solves require $\mathcal{O}(Nb^2)$ and $\mathcal{O}(Nb)$ operators, respectively, where N is the number of unknowns and b the bandwidth. For the cross-point system, $N \approx H^{-2}$ and $b \approx H^{-1}$; for the interfaces, $N = H/h$ and $b = 1$; and for the subdomain interiors, $N = (H/h)^2$ and $b = H/h$. Thus, the interface operation counts are always asymptotically subdominant and can be omitted in the following. From choosing the larger of the cross-point and interior complexities,we see that factorization costs $\max\{\mathcal{O}(H^{-4}), \mathcal{O}(H^2h^{-4})\}$ and solves cost $\max\{\mathcal{O}(H^{-3}), \mathcal{O}(Hh^{-3})\}$. The first term grows with H^{-1} and the second decays with it. To the resolution of the table the minima for both factorization and solve costs occur at or between $H^{-1} = 16$ and 32 when $h^{-1} = 128$. The tendency of buffer overhead, neglected in these estimates, is to favor a slightly smaller number of tiles per side than thus estimated. It is important to note that the memory requirements follow the solve complexities above. Thus, for a fixed memory size, an intermediate cross-point grid granularity accommodates the largest problem in core. Of course, these per iteration complexity

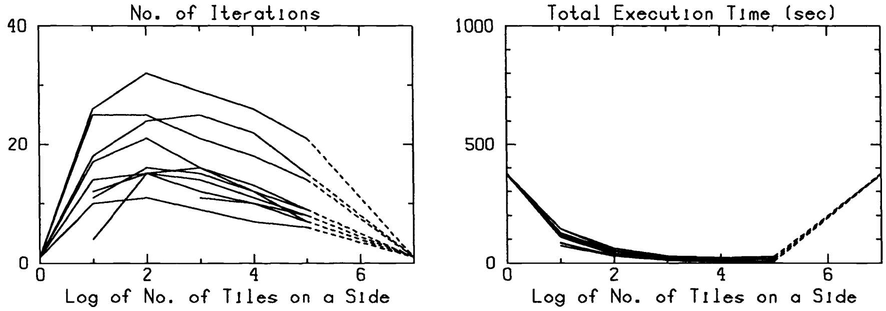


FIG. 7. *Plots of Tables* 1 *and* 2 (*Problems* 1–10 *superposed*), *illustrating that the minimum execution time of the serial algorithm occurs near* $H^{-1} = 16$ *tiles on a side.* (*The dashed portions of the curves are extrapolated beyond the data of the tables.*)

TABLE 3
Iteration count as a function of number of tiles per side of circumscribing square, H^{-1}, *and refinement parameter,* h^{-1}, *at constant number of mesh points along a tile side,* $H/h = 8$, *for a reduction in the initial residual of* 10^{-5}.

H^{-1}	h^{-1}	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
2	16	6	9	11	11	12	11	NA	6	6	3
4	32	9	12	17	15	19	17	NA	12	12	10
8	64	9	11	22	18	23	15	10	12	13	14
16	128	7	10	22	18	26	12	10	11	12	13

estimates shift when the preconditioner blocks are other than banded direct solves.

4.4. Convergence as a function of refinement. In contrast to the preceding section, we here investigate iteration count as a function of overall resolution, for a fixed number of subintervals per tile. The results are shown in Table 3. The global mesh grows in refinement from 16 to 128 while the number of points per tile remains constant at 8. Thus, the fine grid in the last row of Table 3 corresponds to the $H^{-1} = 16$ row of the earlier tables. In spite of the fact that the truncation error improves with h^{-2} in some of these problems, we impose a constant convergence tolerance of 10^{-5} on the tests in Table 3, in order to focus on the algebraic convergence alone.

With the minor exception of Problem 5, which has not quite reached its iteration maximum at 16 tiles per side, the experiments suggest that the iteration count is bounded as resolution increases at constant H/h. In over half of the cases, the finest mesh results are even relatively *better* than the immediately preceding coarser ones. This fact is not surprising since there is a price for this favorable iteration count when H/h is held constant and h^{-1} is increased, namely, a larger cross-point system. The theory for conjugate gradient iteration for selfadjoint problems [7] and for GMRES iteration for nonselfadjoint problems [10] contains similar results for abutting domains, namely, constant upper bounds on the iteration count for constant H/h.

As representative convergence histories, we present Fig. 8, which follows the residual reduction over five orders of magnitude, and the time versus iteration count history for Problems 1 and 2. In the latter plots, the quadratic term in the GMRES work estimate (which comes from the need to orthogonalize each iterate over a subspace

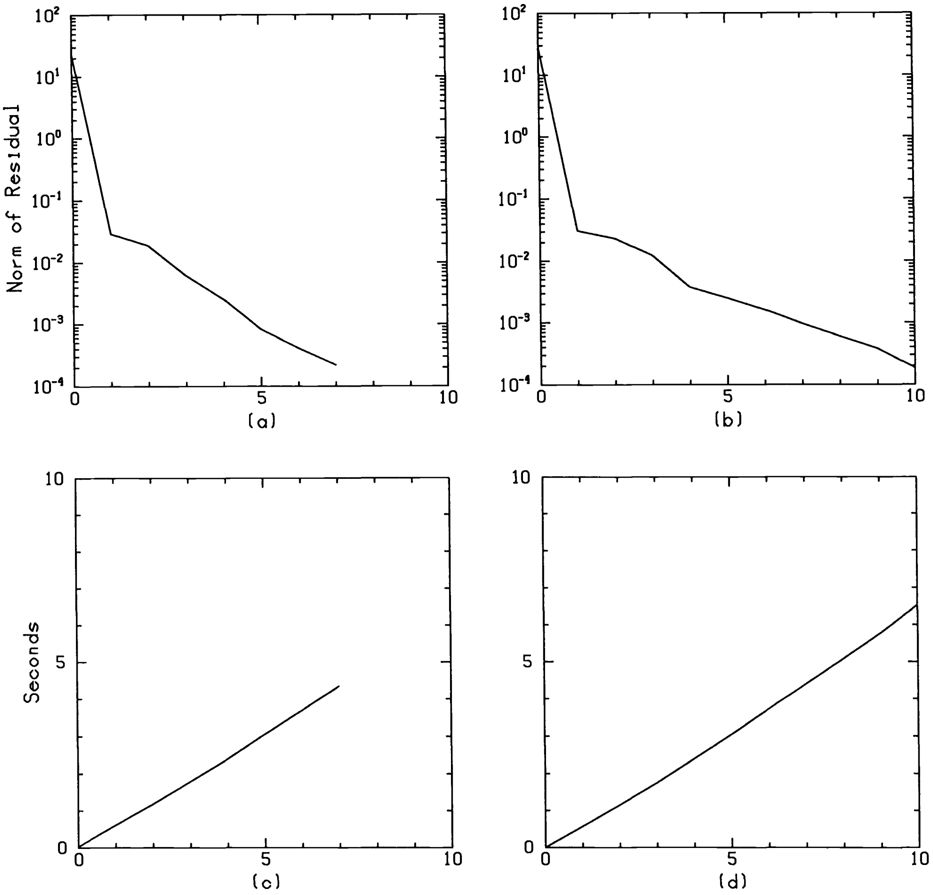


FIG. 8. *Convergence histories for Problems* 1 *and* 2, *for* $H^{-1} = 16$, $H/h = 8$, $h^{-1} = 128$. (a) *and* (b) *show the normalized Euclidean norm of the residual versus iteration count, and* (c) *and* (d) *show time versus iteration count.*

whose size grows linearly in iteration count) is almost invisible. This is due to the exploitation of the identity row in (4). This pair of figures also illustrates the poorer conditioning of Neumann problems, since the initial iterates and the solutions converged to are identical, and so are the operators, except for one Neumann boundary segment.

4.5. Economies of local mesh refinement. Problems 8–10 were used in [27] to illustrate the well-known benefits of locally uniform mesh refinement in elliptic problems: comparable accuracy in considerably fewer operations, compared with globally uniform refinement. These problems were solved at effective refinement levels of $h^{-1} = 32$, 64, and 128, based on the global grid using both global and local refinements for comparison. The tolerance for the relative reduction in the algebraic residual was set to 10^{-8} in these tests, well below the finite-difference discretization error, so that a clear demonstration could be made of the capability of the locally refined grid to provide the full discretization benefit of the globally refined grid. The choice of where

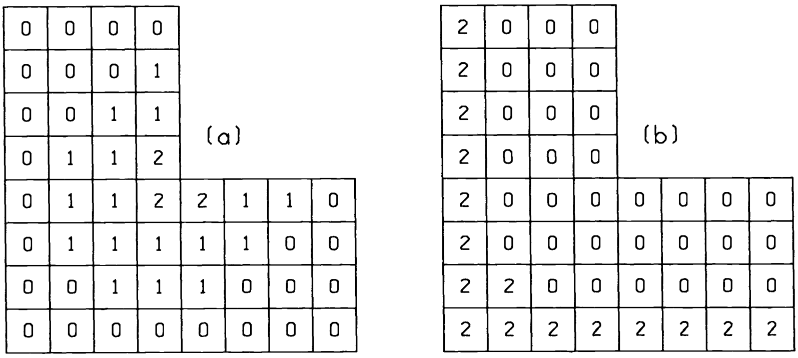


FIG. 9. *Refinement levels, indicated by the logarithm of the refinement ratio. The maximum (second level) local uniform refinements are shown:* (a) *Problems* 8 *and* 9, (b) *Problem* 10. *In first-level tests, all tiles showing* "2" *are set to* "1." *In zeroth-level refinement, all tiles are set to* "0," *which here corresponds to* $H/h = 4$ (*from* [27]).

TABLE 4

Number of unknowns for globally (N_G) *and locally* (N_L) *refined grids of the types shown in Fig.* 9, *iteration counts for a reduction in the initial algebraic residual of* 10^{-8} *for each grid* (I_G *and* I_L), *and ratios of globally to locally refined execution times, for Problems* 8–10.

				#8			#9			#10		
h^{-1}	N_G	$N_{L(a)}$	$N_{L(b)}$	I_G	I_L	T_G/T_L	I_G	I_L	T_G/T_L	I_G	I_L	T_G/T_L
32	833	833	833	18	18	1.00	18	18	1.00	19	19	1.00
64	3201	1817	1609	22	22	1.47	23	23	1.50	23	22	1.73
128	12545	2409	4697	26	23	6.08	28	25	5.76	29	27	2.77

to refine was made manually. The local refinement is as illustrated in Fig. 9.

Table 4 shows the number of discrete unknowns for the globally and locally refined problems and the iteration counts and execution time ratios for each refinement level. All entries share a constant value of $H^{-1} = 8$ in order to fix in space regions of enhanced refinement that do not shrink as h does. Therefore, the "global" iteration columns of Table 4 incidentally provide a constant-H traverse through convergence rate parameter space, complementary to Table 1 (a constant-h traverse) and Table 3 (a constant-H/h traverse).

The linear increases of iteration count with each doubling of global refinement in the selfadjoint problem (Problem 8) and the nearly selfadjoint problem (Problem 9) are consistent with a logarithmic growth in conditioning with h^{-1}, though we reiterate that we have no proof of such a bound for the algorithm generating the tables. The locally refined examples likewise worsen mildly in conditioning with h^{-1} when H is held constant, but the CPU time advantage, (T_G/T_L), of local refinement increases with h^{-1} overall. Comparing iteration counts of globally and locally refined problems at the same effective refinement shows that the often drastically smaller number of unknowns in the latter does not much affect convergence. This observation leads to the hypothesis that in the case of variously refined tiles, $(\mathrm{H}/h_{\mathrm{finest}})$ is the convergence-controlling parameter, with the details of the tile-size distribution important only in estimating the work per iteration.

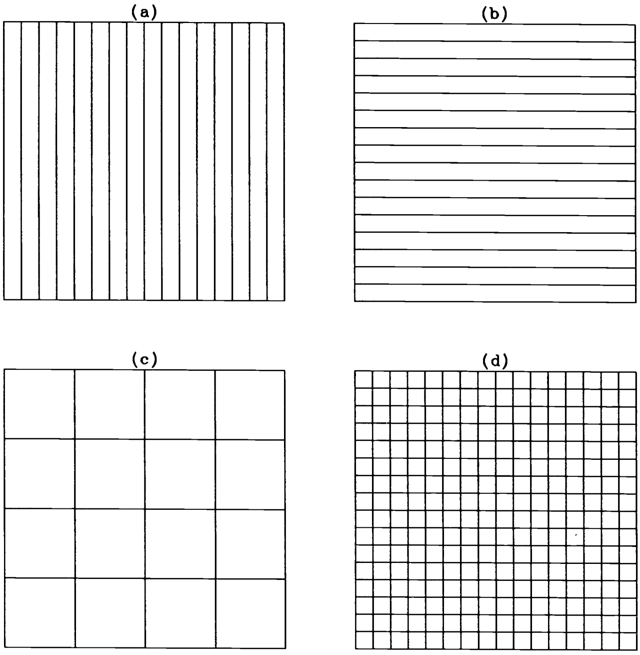


FIG. 10. *The four decompositions tested in Table* 5: (a) *vertical strips,* (b) *horizontal strips,* (c) *"large" boxes with the same granularity as the strips,* (d) *"small" boxes with the same bandwidth as the strips.*

4.6. Anisotropic decompositions. Throughout the foregoing, we have considered decompositions of the problem domain into uniform square tiles exclusively. More general decompositions are possible and should often be considered. Varying the aspect ratio and the orientation of tiles can lead to significant variations in convergence rate in anisotropic problems. Problems 1–4 of the test suite contain a sufficient variety of operators and boundary conditions to illustrate this point. Table 5, based on these four problems, provides a link between the boxwise decompositions studied in this report and the stripwise decompositions employed in many of our earlier studies, such as [34]. Four different decompositions are considered in this table: vertical strips, horizontal strips, a boxwise decomposition with the same number of tiles as the strip cases, and a boxwise decomposition whose tiles have the same bandwidth as that of the most compact natural ordering of the strip cases. These decompositions are shown in Fig. 10. In every case, the mesh spacing is held constant at $h^{-1} = 128$; thus these problems contain 16,641 discrete unknowns.

Among the two boxwise decompositions, the finer is always closer to the optimum found earlier in Table 1 for iteration count and in Table 2 for execution time.

Contrasting the boxwise and stripwise decompositions, we note that for isotropic Problems 1 and 2 the isotropic (boxwise) decompositions lead to significantly better iteration counts than the nonisotropic (stripwise) decompositions. The coarser boxwise decomposition nevertheless leads to poorer execution times than either stripwise decomposition because of the large bandwidth of its tiles, resulting in large factoriza-

TABLE 5

Iteration count I and total execution time T (sec) as a function of the tessellation parameters H_x^{-1} and H_y^{-1}, at constant mesh parameter $h^{-1} = 128$, for a reduction in the initial residual of 10^{-5}.

			#1		#2		#3		#4	
Case	H_x^{-1}	H_y^{-1}	I	T	I	T	I	T	I	T
a	16	1	20	13.7	29	20.5	52	40.5	4	3.5
b	1	16	20	13.7	23	15.9	12	8.4	16	10.9
c	4	4	11	34.7	15	39.7	24	51.9	25	53.3
d	16	16	7	5.5	10	8.0	22	18.3	18	14.4

tion costs in setting up the interior solves of the preconditioner. The finer boxwise decomposition yields the best execution times.

For the nonisotropic Problems 3 and 4, one or both of the nonisotropic decompositions is superior in both iteration count and execution time to both of the isotropic decompositions. In Problem 3, we note the major advantage of handling the strong x-directional diffusive coupling as implicitly as possible in the preconditioner when few (here 16) subdomains are employed. In Problem 4, where the diffusive part of the operator is isotropic, the decomposition that is aligned with the strong convection is superior. This is related to the relatively poorer performance of the tangential interface preconditioner in problems where the convection is normal to the interface rather than tangential to it [13]. In spite of the poor representation of the convection in the interface blocks for Problem 4, case (b), the "wrong" stripwise decomposition is still slightly superior to the best boxwise decomposition. Although the boxwise decomposition convergence rates are asymptotically superior to the stripwise decomposition convergence rates (see the theoretical arguments summarized in [31]), the crossover point is evidently strongly influenced by the physics of the problem.

4.7. Comparison with undecomposed preconditioners. Tables 1 and 2 produced the observation that among preconditioners employing direct banded factorizations for both the cross-point system and the subdomain interiors, a tessellation of intermediate granularity is much superior to one at either coarse or fine extremes. In other words, domain decomposition-preconditioned GMRES methods are superior to bandsolvers even on sequential computers and even in two dimensions. It is of interest to attempt to strengthen such a statement by comparing domain decomposition-preconditioned GMRES iteration with other candidate solvers in the sequential, two-dimensional context. For this purpose, a direct sparse matrix solver and three popular incomplete factorizations have been implemented as alternative subdomain interior preconditioners and compared with the domain-decomposed preconditioner on the first six problems of the test suite.

Table 6 lists the iteration counts and Table 7 the execution times for nine different solution algorithms callable from the same code used to generate all previous tables. (The global domain solvers contain just one tile.)

Six of these solvers are iterative methods based on GMRES and global preconditioners of the incomplete factorization type, tested in two sets of three each. In each set, we test ILU(0), ILU(1), and MILU(0) [17], [39], where the integer in parentheses denotes the number of diagonals of extra fill-in retained adjacent to the original five-diagonal structure of the discrete operator [46]. In the first set, the maximum size of the Krylov subspace used in GMRES is 90; in the second set, the maximum Krylov subspace has dimension 5. In a majority of cases, the globally preconditioned GM-

TABLE 6

Iteration counts for Problems 1–6 *for nine different algorithmic combinations at a mesh parameter* $h^{-1} = 128$, *for a reduction in the initial residual of* 10^{-5}. "$>$" *signifies more than* 500 *iterations.*

Method	#1	#2	#3	#4	#5	#6
GMRES(90)/ILU(0)	73	108	80	82	99	>
GMRES(90)/MILU(0)	22	77	19	147	39	>
GMRES(90)/ILU(1)	45	58	61	51	59	160
GMRES(5)/ILU(0)	351	312	227	>	>	>
GMRES(5)/MILU(0)	27	141	23	>	57	>
GMRES(5)/ILU(1)	139	150	140	213	244	>
Direct	1	1	1	1	1	1
GMRES(90)/DD	7	10	22	18	26	12
GMRES(5)/DD	8	10	28	25	39	12

TABLE 7

Execution times (sec) for Problems 1–6 *for nine different algorithmic combinations at a mesh parameter* $h^{-1} = 128$, *for a reduction in the initial residual of* 10^{-5}. *The best time in each column is italicized.*

Method	#1	#2	#3	#4	#5	#6
GMRES(90)/ILU(0)	121.	191.	142.	149.	183.	–
GMRES(90)/MILU(0)	16.7	133.	13.5	256.	41.2	–
GMRES(90)/ILU(1)	54.2	83.1	90.1	66.7	85.4	297.
GMRES(5)/ILU(0)	195.	173.	126.	–	–	–
GMRES(5)/MILU(0)	14.9	78.6	*12.6*	–	31.5	–
GMRES(5)/ILU(1)	82.8	90.7	84.3	128.	136.	–
Direct/Nest. Diss.	71.2	70.1	70.1	70.1	70.0	71.6
GMRES(90)/DD/Nest. Diss.	*7.1*	*9.9*	23.8	*18.7*	*29.7*	*12.0*
GMRES(5)/DD/Nest. Diss.	8.2	10.4	25.0	23.2	34.2	12.0

RES iteration converges in fewer than 90 iterations; thus, the first set consists mainly of full GMRES convergence results. In practical applications, restarted GMRES is often used to conserve memory or defeat the quadratic term in the GMRES work estimate that arises from orthogonalization over an ever-expanding Krylov subspace. GMRES(k) denotes a restart after k steps.

One of the solvers is a direct method, the Yale Sparse Matrix Package [18] using a global nested-dissection ordering (rather than the minimum degree ordering provided with YSMP), which naturally converges in one step.

Finally, we test two domain-decomposed GMRES algorithms based on a 16×16 array of 8×8 tiles. Both are slight variants of the algorithm used in preceding sections in which the bandsolver is replaced with the nested-dissection sparse solver. Full GMRES and GMRES(5) are considered.

Comparing first the convergence rates of the various global preconditioners, we observe that in the diffusively dominated problems with Dirichlet boundary conditions (Problems 1, 3, 5) the fill-capturing modified incomplete factorization MILU(0) is much superior to ILU(0) and ILU(1). The existence of a non-Dirichlet boundary segment weakens MILU (boundary conditions are the only difference between Problems 2 and 1), and the presence of convection weakens it substantially (Problems 4 and 6). As expected, ILU(1) uniformly requires fewer iterations than ILU(0) in these tests, and this convergence rate advantage translates into an execution time advantage even after the marginally higher cost of the ILU(1) preconditioner is taken into account. Experience with ILU(l) shows a law of diminishing returns as l increases beyond a fairly small problem-dependent value. The tests with GMRES(5) show how the higher iteration counts of a restarted method often translate into lower execution times for well-conditioned problems, but how poorly conditioned problems may fail to converge with too small a Krylov subspace.

Having noted the strong degree of problem dependence in the selection of the best global preconditioner, we note that this problem dependence extends to the relative ranking of globally preconditioned GMRES and the direct sparse nested dissection factorization. In terms of execution time, the nested dissection method loses out to the best global iterative method, GMRES(5)/MILU(0), in the odd-numbered problems, is close to the best global iterative method, GMRES(90)/ILU(1) in Problem 4, and beats all globally preconditioned methods in Problems 2 and 6.

Comparison of the nested dissection rows of Table 7 with the rows of Table 2 at corresponding granularity reveals, as expected, that the sparse direct subdomain solvers run faster than the banded direct subdomain solvers on large problems (approximately 70 sec versus approximately 370 sec on 128 × 128 tiles) and slower than bandsolvers on small ones (by approximately 20–30 percent on 8 × 8 tiles). The latter observation justifies our use of bandsolvers to perform the A_I^{-1} solves in the preconditioner throughout the majority of this report, where the focus is on relatively fine granularity.

Finally and most significantly, we observe that domain decomposition-preconditioned GMRES always beats the direct method, and it beats the best globally preconditioned method in all problems except for Problem 3, for which good preconditioners of both global and domain-decomposed varieties can be found. Overall, it is the fastest executing method and performs reliably and evenly over the range of problems considered. It is a compelling serial algorithm even apart from the virtues of modularity and adaptability.

5. Conclusions and future directions. Experiments on a variety of model problems demonstrate that a two-level domain decomposition algorithm with a single global coarse grid provides effective convergence and convenient refinement and permits a data structure amenable to parallel and vector implementations, as summarized in closing below. Although often motivated by parallelization, domain decomposition also yields runtime and memory use benefits as a sequential algorithm. Relative to traditional global preconditioners, domain-decomposed preconditioners can dramatically improve convergence rates. Furthermore, the simple structure of individual blocks of the domain-decomposed preconditioner means that new applications are found for the "standard solvers" in conventional software libraries. The traditional economies of local uniform mesh refinement can be incorporated into the domain decomposition framework at the small price of interface handlers with conditionals for refinement differences between adjacent subdomains. Because of the highly modular nature of a tile-oriented domain decomposition code, custom discretizations for certain classes of singularities may be archived into applications libraries for reuse. In short, software engineering is a major motivation for the restricted class of algorithms explored here.

The applicable problem class is greater than the present examples indicate; for instance, the tile algorithm has been extended to multiple-dependent variable cases. A two-independent-variable streamfunction-vorticity formulation of the incompressible Navier–Stokes equations is considered in [26] and [28]. The nonlinearity in this problem is handled by a Newton method wrapped around the domain-decomposed linear solver. The entire nonlinear code has been parallelized on shared- and distributed-memory machines, and the linear and nonlinear portions are comparable in their parallel efficiencies (which vary in the usual way from arbitrarily good to arbitrarily bad, depending upon problem size relative to number of processors).

Extension of the tile algorithm to a brick algorithm in three-dimensional problems is conceptually straightforward. The software engineering motivation for restriction to a tensor-product grid of substructure vertices is even more compelling in three dimensions than it is in two. One new feature is the presence of two-dimensional interfaces, upon which preconditioner blocks could be constructed by dropping normal derivative terms, by analogy with one-dimensional interfaces in the plane. The effectiveness of this straightforward extension is not demonstrated at present. For the theoretically endowed selfadjoint case it is known that the condition number of the hierarchically preconditioned system grows like the first power of (H/h), not merely like its logarithm. A discussion and some alternatives are presented in [44].

The tile algorithm is amenable to vectorization in either of two ways. The regular operation sequences on the tensor-product subgrid arrays are precisely the type for which vectorizing compilers were conceived. The vector lengths depend on the precise form of solvers used in the preconditioner but would tend to be rather small for the rows of individual 8×8 or 16×16 tiles found best in the two-dimensional applications above. An alternative form of vectorization can be realized by grouping together all tiles of a given (discrete) size and shape and operating in lock step on corresponding elements in each tile, assuming an identical solver is applied to each. A vector in this approach consists of the ith element from each of the subdomains. Thus, 8×8 arrays of tiles deliver optimal processing rates for machines with a vector length of 64.

Parallelization requires attention to the load balancer/mapper [27] and also to the coarse-grid solve in the preconditioner [24]. The main disadvantage of the two-level algorithm in the parallel context is that the choice of coarse-grid granularity is more of an "overdetermined" problem than in serial. Communication cost per iteration and convergence properties potentially inveigh against the lower bounds on the number of tiles imposed by domain geometry, solution and coefficient roughness, and parallel load balance. The key determination for future applications of the tile methodology will be whether this overdetermination is "consistent" in practice. Inasmuch as the early examples are representative of one or two dependent variable problems, and parallel communication costs generally comprise a *relatively* smaller proportion of the total work in coupled multicomponent problems, there are substantial grounds for optimism that this will be the case.

Acknowledgments. We express our deep appreciation to Dr. Xiao-Chuan Cai for his influence on the refinement of the tile algorithm through his mathematical insight and through his experience in using the code and in adapting it to additive Schwarz-type preconditioning. We are also indebted to two anonymous referees for constructively critical comments on an early version of this manuscript.

REFERENCES

[1] G. Anagnostou, D. Dewey, and A. T. Patera, *Geometry-defining processors for engineering design and analysis*, The Visual Computer, 5 (1989), pp. 304–315.

[2] I. Babuška, J. Chandra, and J. Flaherty, eds., *Adaptive Computational Methods for Partial Differential Equations*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.

[3] R. Bank and H. Yserentant, *Some remarks on the hierarchical basis multigrid method*, in Second International Symposium on Domain Decomposition Methods, T. F. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983, pp. 140–143.

[4] M. J. Berger and J. Oliger, *Adaptive mesh refinement for hyperbolic partial differential equations*, J. Comput. Phys., 53 (1984), pp. 484–512.

[5] P. E. Bjorstad and O. B. Widlund, *Iterative methods for the solution of elliptic problems on regions partitioned into substructures*, SIAM J. Numer. Anal., 23 (1986), pp. 1097–1120.

[6] J. H. Bramble, R. E. Ewing, J. E. Pasciak, and A. H. Schatz, *A preconditioning technique for the efficient solution of problems with local grid refinement*, Comput. Methods Appl. Mech. Engrg., 67 (1988), pp. 149–159.

[7] J. H. Bramble, J. E. Pasciak, and A. H. Schatz, *The construction of preconditioners for elliptic problems by substructuring,* I, Math. Comp., 47 (1986), pp. 103–134.

[8] A. Brandt, *Multi-level adaptive techniques* (MLAT) *for fast numerical solution to boundary-value problems*, in Proceedings of the Third International Conference on Numerical Methods in Fluid Mechanics, Lecture Notes in Physics 18, H. Cabannes and R. R. Temam, eds., Springer-Verlag, Berlin, New York, 1973, pp. 82–89.

[9] X.-C. Cai, *An additive Schwarz algorithm for nonselfadjoint elliptic equations*, in Third International Symposium on Domain Decomposition Methods, T. F. Chan, R. Glowinski, J. Périaux, and O. B. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990, pp. 232–244.

[10] X.-C. Cai, W. D. Gropp, and D. E. Keyes, *Convergence rate estimate for a domain decomposition method*, Numer. Math., 61 (1992), pp. 153–169.

[11] X.-C. Cai and O. B. Widlund, *Domain decomposition algorithms for indefinite elliptic problems*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 243–258.

[12] T. F. Chan and D. Goovaerts, *A note on the efficiency of domain decomposed incomplete factorizations*, SIAM J. Sci. Statist. Comp., 11 (1990), pp. 794–803.

[13] T. F. Chan and D. E. Keyes, *Interface preconditionings for domain-decomposed convection-diffusion operators*, in Third International Symposium on Domain Decomposition Methods, T. F. Chan, R. Glowinski, J. Périaux, and O. B. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990, pp. 245–262.

[14] H. A. van der Vorst, Bi-CGSTAB: *A fast and smoothly converging variant of* Bi-CG *for the solution of nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 631–644.

[15] M. Dryja, *An additive Schwarz algorithm for two- and three-dimensional finite-element elliptic problems*, in Second International Symposium on Domain Decomposition Methods, T. F. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989, pp. 168–172.

[16] M. Dryja and O. B. Widlund, *On the optimality of an additive refinement method*, in Proceedings of the Fourth Copper Mountain Conference on Multigrid Methods, J. Mandel, S. F. McCormick, J. E. Dendy, Jr., C. Farhat, G. Lonsdale, S. V. Parter, J. W. Ruge, and K. Stuben, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989, pp. 161–170.

[17] T. Dupont, R. Kendall, and H. H. Rachford, *An approximate factorization procedure for solving selfadjoint elliptic difference equations*, SIAM J. Numer. Anal., 5 (1968), pp. 559–573.

[18] S. C. Eisenstat, H. C. Elman, M. H. Schultz, and A. H. Sherman, *The (new) Yale sparse matrix package*, Tech. Report 265, Department of Computer Science, Yale University, New Haven, CT, April 1983.

[19] R. E. Ewing, R. D. Lazarov, and P. S. Vassilevski, *Local refinement techniques for elliptic problems on cell-centered grids*, I. *Error analysis*, Math. Comp., 56 (1991), pp. 437–461.

[20] J. E. Flaherty, P. J. Paslow, M. S. Shepard, and J. D. Vasilakis, eds., *Adaptive Methods for Partial Differential Equations*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989.

[21] R. W. Freund and N. M. Nachtigal, QMR*: A quasi-minimum residual method for non-Hermitian linear systems*, Tech. Report 90.51, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, CA, 1990.

[22] A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.

[23] W. D. Gropp, *Local uniform mesh refinement for elliptic partial differential equations*, Tech. Report YALE/DCS/RR-278, Department of Computer Science, Yale University, New Haven, CT, July 1983.

[24] W. D. Gropp and D. E. Keyes, *Domain decomposition on parallel computers*, Impact Comput. Sci. Engrg., 1 (1989), pp. 421–439.

[25] ———, *Domain decomposition with local mesh refinement*, Tech. Report RR-726, Department of Computer Science, Yale University, New Haven, CT, August 1989.

[26] W. D. GROPP AND D. E. KEYES, *Parallel domain decomposition and the solution of nonlinear systems of equations*, in Fourth International Symposium on Domain Decomposition Methods, R. Glowinski, Y. A. Kuznetsov, G. Meurant, J. Périaux, and O. B. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1991, pp. 373–381.

[27] ———, *Parallel performance of domain-decomposed preconditioned Krylov methods for* PDE*s with locally uniform refinement*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 128–145.

[28] ———, *Domain decomposition methods in computational fluid dynamics*, Internat. J. Numer. Methods Fluids, 14 (1992), pp. 147–165.

[29] E. N. HOUSTIS, R. E. LYNCH, J. R. RICE, AND T. S. PAPATHEODOROU, *Evaluation of numerical methods for elliptic partial differential equations*, J. Comput. Phys., 27 (1978), pp. 323–350.

[30] H. JARAUSCH, *On an adaptive grid refining technique for finite element approximations*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1105–1120.

[31] D. E. KEYES AND W. D. GROPP, *A comparison of domain decomposition techniques for elliptic partial differential equations and their parallel implementation*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s166–s202.

[32] ———, *Domain decomposition techniques for nonsymmetric systems of elliptic boundary value problems: Examples from* CFD, in Second International Symposium on Domain Decomposition Methods, T. F. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989, pp. 321–339.

[33] ———, *Domain-decomposable preconditioners for second-order upwind discretizations of multicomponent systems*, in Fourth International Symposium on Domain Decomposition Methods, R. Glowinski, Y. A. Kuznetsov, G. Meurant, J. Périaux, and O. B. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1991, pp. 129–139.

[34] ———, *Domain decomposition techniques for the parallel solution of nonsymmetric systems of elliptic boundary value problems*, Appl. Numer. Math., 6 (1990), pp. 281–301.

[35] Y. MADAY, C. MAVRIPLIS, AND A. T. PATERA, *Nonconforming mortar element methods: Application to spectral discretizations*, in Second International Symposium on Domain Decomposition Methods, T. F. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989, pp. 392–418.

[36] T. A. MANTEUFFEL AND S. V. PARTER, *Preconditioning and boundary conditions*, SIAM J. Numer. Anal., 27 (1990), pp. 656–694.

[37] S. MCCORMICK AND D. QUINLAN, *Asynchronous multilevel adaptive methods for solving partial differential equations on multiprocessors: Performance results*, Parallel Comput., 12 (1989), pp. 145–156.

[38] S. F. MCCORMICK, *Multilevel Adaptive Methods For Partial Differential Equations*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989.

[39] J. A. MEIERINK AND H. A. VAN DER VORST, *Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems*, J. Comput. Phys., 44 (1981), pp. 134–155.

[40] W. PROSKUROWSKI, *Remarks on the spectral equivalence of certain discrete operators*, in Second International Symposium on Domain Decomposition Methods, T. F. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989, pp. 103–113.

[41] 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, Tech. Report 2078, Mathematics Research Center, University of Wisconsin, Madison, WI, May 1980.

[42] ———, *A population of linear second order, elliptic partial differential equations on rectangular domains—Part* II, Tech. Report 2079, Mathematics Research Center, University of Wisconsin, Madison, WI, May 1980.

[43] Y. SAAD AND M. H. SCHULTZ, GMRES: *A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.

[44] B. F. SMITH, *Domain decomposition algorithms for the partial differential equations of linear elasticity*, Tech. Report 517, Courant Institute, New York, September 1990.

[45] B. SWARTZ, *Courant-like conditions limit reasonable mesh refinement to order h^2*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 924–933.

[46] J. W. WATTS, III, *A conjugate gradient-truncated direct method for the iterative solution of the reservoir simulation pressure equation*, Soc. Petrol. Engrg. J., 21 (1981), pp. 345–353.

[47] H. YSERENTANT, *On the multi-level splitting of finite element spaces for indefinite elliptic boundary value problems*, SIAM J. Numer. Anal., 23 (1986), pp. 581–595.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 4, pp. 994–1008, July 1992

009

AN $O(n^2 \log n)$ TIME ALGORITHM FOR THE MINMAX ANGLE TRIANGULATION*

HERBERT EDELSBRUNNER†, TIOW SENG TAN†‡, AND ROMAN WAUPOTITSCH†

Abstract. It is shown that a triangulation of a set of n points in the plane that minimizes the maximum angle can be computed in time $O(n^2 \log n)$ and space $O(n)$. The algorithm is fairly easy to implement and is based on the edge-insertion scheme that iteratively improves an arbitrary initial triangulation. It can be extended to the case where edges are prescribed, and, within the same time- and space-bounds, it can lexicographically minimize the sorted angle vector if the point set is in general position. Experimental results on the efficiency of the algorithm and the quality of the triangulations obtained are included.

Key words. computational geometry, two dimensions, triangulations, minmax angle criterion, iterative improvement, edge insertion

AMS(MOS) subject classifications. 68C05, 65M50

1. Introduction. Let S be a finite set of points in the Euclidean plane. A *triangulation* of S is a maximally connected straight-line plane graph whose vertices are the points of S. By maximality, each face is a triangle except for the exterior face, which is the complement of the convex hull of S. Occasionally, we will call a triangulation of a finite point set a *general triangulation* in order to distinguish it from a *constrained triangulation*, which is a triangulation of a finite point set where some edges are prescribed. A special case of a constrained triangulation is the so-called *polygon triangulation*, where S is the set of vertices of a simple polygon and the edges of the polygon are prescribed. In this paper only the triangles *inside* the polygon will be of interest.

For a given set of n points there are, in general, exponentially many triangulations. Among them one can choose those that satisfy certain requirements or optimize certain objective functions. Different properties are desirable for different applications in areas such as finite element analysis [1], [3], [23], computational geometry [21], and surface approximation [12], [18]. The following are some important types of triangulations that optimize certain objective functions.

(i) The *Delaunay triangulation* has the property that the circumcircle of any triangle does not enclose any vertex [5].

(ii) The *constrained Delaunay triangulation* has the same property except that visibility constraints depending on the enforced edges are introduced [13].

(iii) The *minimum weight triangulation* minimizes the total edge length over all possible triangulations of the same set of points and prescribed edges [10], [17].

It is known that the Delaunay triangulation maximizes the minimum angle over all triangulations of the same point set [22]. This result can be extended to a similar statement about the sorted angle vector of the Delaunay triangulation [6] and to the constrained case [13]. The Delaunay triangulation of n points in the plane can be constructed in time $O(n \log n)$ [6], [19], and even if some edges are prescribed, its constrained version can be constructed in the same amount of time [20]. There is no polynomial time algorithm known for the minimum weight triangulation if the input

* Received by the editors May 16, 1990; accepted for publication (in revised form) June 14, 1991.

† Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801. The research of the first author was supported by National Science Foundation grants CCR-8714565 and CCR-8921421.

‡ This author is on study leave from the National University of Singapore, Republic of Singapore.

is a finite point set, but dynamic programming leads to a cubic algorithm [10] if the input is a simple polygon.

In this paper, we study the problem of constructing a triangulation that minimizes the maximum angle, over all triangulations of a finite point set, with or without prescribed edges. We call such a triangulation a *minmax angle triangulation.* Although avoiding small angles is related to avoiding large angles, the Delaunay triangulation does not minimize the maximum angle—four points are sufficient to give an example to this effect. Triangulations that minimize the maximum angle have potential applications in the area of finite element and surface approximation [1], [2], [8]. Our main result is summarized in the following statement.

MAIN THEOREM. *A minmax angle triangulation of a set of* n *points in the plane, with or without prescribed edges, can be computed in time* $O(n^2 \log n)$ *and space* $O(n)$.

Curiously, our algorithm has the same complexity for point sets and for simple polygons. Prior to this paper no polynomial time algorithm for constructing a minmax angle triangulation for a finite point set was known. On the other hand, if the input is a simple n-gon, then a cubic time and quadratic space solution can be derived simply by substituting the angle criterion for the edge-length criterion in the dynamic programming algorithm of [10]. Thus, it seemed that the problem for simple polygons is much simpler than for point sets. Indeed, our attempts to apply popular techniques such as local edge-flipping [11], [9], divide-and-conquer [21] and plane-sweep [7] to construct a minmax angle triangulation for a point set were not successful; see also [15].

Instead, we solve the problem by an iterative improvement method based on what we call the *edge-insertion scheme*. An *edge-insertion* step adds some new edge qs to the current triangulation, deletes edges that cross qs, and retriangulates the resulting polygonal regions to the left and the right of qs. The difference from the simpler edge-flip operation is that qs can cross up to a fraction of the current edges, whereas an edge added in an edge-flip crosses only one edge. This difference turns out to be crucial in the case of minimizing the maximum angle: the edge-flip scheme can get stuck in a nonglobal optimum [15], whereas the edge-insertion scheme is powerful enough to always reach the optimum. A proof of the latter property is sufficient to design a polynomial time implementation of the edge-insertion scheme. Clever strategies to find an edge qs that leads to an improvement of the current triangulation and to retriangulate the created polygonal regions are needed to obtain the claimed $O(n^2 \log n)$ time bound.

Section 2 presents the algorithm to construct a minmax angle triangulation, and §3 proves the crucial piece needed to show that the algorithm is correct. Section 4 gives the algorithmic details that lead to an efficient implementation of the algorithm. Section 5 discusses the extensions to the constrained case and to the problem of lexicographically minimizing the sorted angle vector. Finally, §6 presents experimental results, and §7 mentions some related open problems.

2. The global algorithm.

In general, there is more than one minmax angle triangulation for a given set of points. Below we outline an algorithm that constructs one such triangulation for a set S of n points in the plane. The maximum angle of a given triangulation $\mathcal{A}$ is denoted by $\mu(\mathcal{A})$.

```
Construct an arbitrary triangulation $\mathcal{A}$ of $S$.
repeat
(M1) Find a largest angle $\angle pqr$ of $\mathcal{A}$.
```

```
(M2) Apply the ear-cutting procedure (§4) to modify $\mathcal{A}$ by
        adding a "suitable" edge $qs$ to $\mathcal{A}$, where $s \in S - \{p, q, r\}$ and $pr \cap qs \neq \emptyset$,
        removing edges that intersect $qs$ (this step creates polygons $P$ and $R$
            which have $qs$ as a common edge), and
        constructing triangulations $\mathcal{P}$ of $P$ and $\mathcal{R}$ of $R$ so that $\mu(\mathcal{P}), \mu(\mathcal{R}) < \angle pqr$.
until the ear-cutting procedure fails to find such a $qs$.
```

To show that this algorithm is correct, we need the following two lemmas and some forward references to the cake-cutting lemma of §3 and the ear-cutting procedure of §4. We define $\angle xsy = 0$ if any two of the three points are identical.

LEMMA 2.1. *If xy is an edge in a triangulation $\mathcal{A}$ of a point set S, then $\mu(\mathcal{A}) \geq \max_{s \in S} \angle xsy$.*

Proof. Let t be a point so that $\angle xty = \max_{s \in S} xsy$. Thus no points of S lie inside the triangle xty. Clearly, if xty is a triangle in $\mathcal{A}$ then there is nothing to be proved. Otherwise, there exists a triangle utv in $\mathcal{A}$ so that either $u = x$, $v \in S - \{y, t\}$, and uv intersects ty or $u, v \in S - \{x, y, t\}$, and uv intersects both xt and ty. In both cases, $\mu(\mathcal{A}) \geq \angle utv > \angle xty$. □

The proof of the next lemma makes use of the cake-cutting lemma to be presented in §3. We suggest that the reader read the statement of that lemma (Lemma 3.1) and then return to the current discussion leading to Lemma 2.2. We call a triangulation $\mathcal{B}$ of S an *improvement* of $\mathcal{A}$ if

(i) $\mu(\mathcal{B}) < \mu(\mathcal{A})$, or

(ii) $\mu(\mathcal{B}) = \mu(\mathcal{A})$, every triangle abc in $\mathcal{B}$ with $\angle abc = \mu(\mathcal{B})$ is also a triangle in $\mathcal{A}$, and $\mathcal{B}$ has at least one fewer maximum angle than $\mathcal{A}$.

The next lemma asserts that the algorithm makes progress as long as the current triangulation is not yet a minmax angle triangulation. It does this by proving that there is at least one suitable edge qs. In its current version, the algorithm can be thought of as trying all possible edges going out of q, so if there exist edges qs that lead to an improvement of $\mathcal{A}$, then the algorithm finds one such edge.

LEMMA 2.2. *Assume that $\mathcal{A}$ is not yet a minmax angle triangulation. Then an iteration of the repeat-loop constructs an improvement of $\mathcal{A}$.*

Proof. Step (M1) of the repeat-loop finds a triangle pqr in $\mathcal{A}$ so that $\angle pqr = \mu(\mathcal{A})$. The main observation is that there is some edge qs that intersects pr and belongs to a minmax angle triangulation $\mathcal{T}$ of S. This is because $\mu(\mathcal{T}) < \mu(\mathcal{A})$ implies that $\angle pqr$ cannot exist in $\mathcal{T}$, and consequently, $pr \notin \mathcal{T}$ (by the previous lemma). Therefore, there exists a point $s \in S - \{p, q, r\}$ such that $qs \cap pr \neq \emptyset$ and qs is an edge of $\mathcal{T}$. With this edge qs, the cake-cutting lemma (§3) ensures that there are polygon triangulations of P and R such that the largest angle of any triangle within P and R is still smaller than $\angle pqr$. Section 4 shows that the ear-cutting procedure of step (M2) indeed finds such a point s and produces triangulations $\mathcal{P}$ and $\mathcal{R}$ of P and R such that $\mu(\mathcal{P}), \mu(\mathcal{R}) < \angle pqr$. □

The above two lemmas can now be used to analyze the running time of the algorithm. First, we address the number of iterations of the repeat-loop, which is one plus the number of successful iterations of step (M2).

LEMMA 2.3. *The above algorithm reaches a minmax triangulation after at most $O(n^2)$ iterations of the repeat-loop.*

Proof. Each iteration produces a triangulation with a smaller maximum angle than before, or with fewer maximum angles of the same size. Since the number of different triangulations is finite, an optimum must be reached. To get an upper bound on the number of iterations, notice that the edge pr removed from $\mathcal{A}$ during some iteration will not reappear in the future. The claim follows because S allows only $\binom{n}{2}$ different edges. □

We are now ready to argue that the above algorithm runs in time $O(n^2 \log n)$ and space $O(n)$. There are two data structures needed for the algorithm. First, the quad-edge structure of Guibas and Stolfi [9] is used to represent $\mathcal{A}$; it permits common operations, such as removing an edge, adding an edge, and walking from one edge to the next, in constant time each. Second, the angles of $\mathcal{A}$ are stored in a priority queue that admits insertions, deletions, and finding the maximum. Standard implementations support each such operation in time $O(\log n)$; see, e.g., [4]. The space needed for both data structures is $O(n)$.

With these preliminaries we can give the analysis of the algorithm. By Lemma 2.3, the number of times the priority queue is consulted to get a largest angle is $O(n^2)$, which implies that step (M1) takes total time $O(n^2 \log n)$. Section 4 will show that the ear-cutting procedure performs only a total of $O(n^2)$ operations on the quad-edge structure, each in constant time, and only $O(n^2)$ insertions into and deletions from the priority queue, each in time $O(\log n)$. We conclude that the running time of the algorithm is $O(n^2 \log n)$, as claimed.

3. The cake-cutting lemma.

The result of this section is a technical lemma, which is nevertheless the heart of this paper. It ensures that for some edge qs the generated regions, P and R, can be triangulated without angles that are too large. We first discuss the shape of these regions and then state and prove the lemma.

The regions P and R are generated in step (M2) of the algorithm by adding an edge qs and removing all edges that intersect qs. It follows that P (and by symmetry R) is very similar to a simple polygon; that is, it is simply connected and bounded by straight-line edges. The only difference is that there can be edges surrounded by P on both sides; these are the edges contained in the interior of the closure of P (see Fig. 1). To simplify the forthcoming discussion (and also in the implementation of the algorithm) we treat each such edge as if it consisted of two edges, one for each side. Effectively, this means that we can talk about P and R as if they were simple polygons.

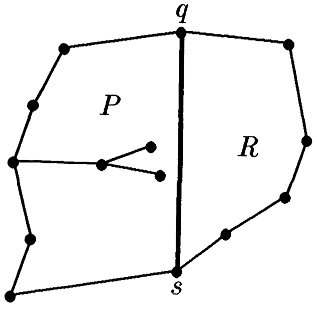


FIG. 1. *Regions P and R.*

With this note we now state and prove the cake-cutting lemma. The intuition behind the proof is that we look at a piece of an optimal triangulation $\mathcal{T}$ and argue about its edges. Keep in mind, however, that during the algorithm we have no way of knowing what $\mathcal{T}$ really is; we only know that it exists.

LEMMA 3.1. *Let $\mathcal{T}$ be a minmax angle triangulation of S, $\mathcal{A}$ a triangulation of S with $\mu(\mathcal{A}) > \mu(\mathcal{T})$, pqr a triangle in $\mathcal{A}$ so that $\angle pqr = \mu(\mathcal{A})$, and qs an edge in $\mathcal{T}$ that intersects pr. Let P and R be the polygons generated by adding qs to $\mathcal{A}$ and removing all edges that intersect qs. Then there are triangulations $\mathcal{P}$ and $\mathcal{R}$ of P and R so that $\mu(\mathcal{P}), \mu(\mathcal{R}) < \mu(\mathcal{A})$.*

Proof. We prove the claim for P; it follows for R by symmetry. Imagine we have $\mathcal{A}$ and $\mathcal{T}$ on separate pieces of transparent paper that we lay on top of each other so that the points match. Following step (M2) of the algorithm, we add qs to $\mathcal{A}$ and remove intersecting edges from $\mathcal{A}$, thus creating P and R. Next, we clip everything outside P. In $\mathcal{A}$ only P without intersecting edges is left, and in $\mathcal{T}$ there will generally be edges that cut through P. By assumption, qs is also in $\mathcal{T}$, which implies that none of these edges meets qs. We define a *clipped edge* as a connected component of such an edge of $\mathcal{T}$ intersected with P. Since P is not necessarily convex, some clipped edges can belong to the same edge of $\mathcal{T}$. Given a point x on the boundary of P, let the *path* from x to q (or x to s) be the part of the boundary between x and q (or x and s) that does not contain qs. We have four classes of clipped edges xy; see Fig. 2.

I. Both endpoints, x and y, are not vertices of P and thus lie on edges of P.
II. Both endpoints are vertices of P.
III. Endpoint x is a vertex of P, y is not, and y lies on the path from x to s.
IV. The same as class III except that y lies on the path from x to q.

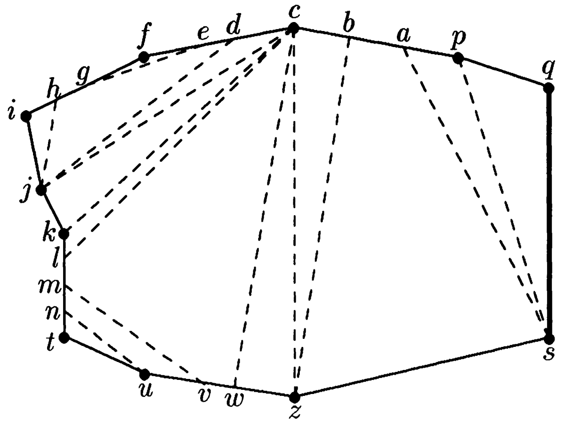


FIG. 2. *The class* I *edges in this example are eg and mv; the class* II *edges are cj, ck, cz, and sp; the class* III *edges are cl and cw; and the class* IV *edges are jh, jd, un, zb, and sa.*

At any vertex x of P, the clipped edges with one endpoint at x define angles at x that are all smaller than $\mu(\mathcal{A})$, because the clipped edges come from $\mathcal{T}$ and $\mu(\mathcal{T}) < \mu(\mathcal{A})$ holds by assumption. The only disadvantage of the partition of P defined by the clipped edges is that some of their endpoints lie on edges of P rather than at the vertices. We will now construct a triangulation of P based on the clipped edges. It proceeds step by step, where each step either removes or rotates a clipped edge or introduces a new edge.

1. All class I edges are removed. This does not harm any angle.
2. All class II edges remain where they are.
3. Let xy be a class III edge with y on the edge $\alpha\beta$ of P, where α precedes β

on the path from x to s. We replace xy by $x\beta$.

Note first that $x\beta$ is indeed a diagonal of P. Otherwise, it intersects the boundary of P, which implies that either x or β is not visible from qs. This is a contradiction to the way P is constructed. Note second that the angle at x that precedes xy in the counterclockwise order increases in step 3. Still, the angle formed by $x\beta$ is strictly contained in an angle at x in $\mathcal{A}$ because all edges of $\mathcal{A}$ that intersect P also intersect qs. It follows that the angle formed by $x\beta$ is smaller than $\mu(\mathcal{A})$. Another issue that comes up is that there can be class IV edges $x'y'$ with y' on the same edge $\alpha\beta$ of P—these edges now intersect $x\beta$. To remedy this situation we replace $x'y'$ by $x'x$. By the same argument as above, $x'x$ is a diagonal of P, and the angle at x' that precedes $x'y'$ in the clockwise order and that increases as we replace $x'y'$ by $x'x$ remains smaller than $\mu(\mathcal{A})$.

4. If xy is a class IV edge with y on the edge $\alpha\beta$ of P, where α precedes β on the path from x to q, then we replace xy by $x\beta$.

5. After steps 1–4 we have a partial triangulation of P, which we complete by adding edges arbitrarily. This finishes the construction of $\mathcal{P}$.

We have $\mu(\mathcal{P}) < \mu(\mathcal{A})$ since we started out with all angles smaller than $\mu(\mathcal{A})$; each time an angle increases it remains smaller than $\mu(\mathcal{A})$ as argued above, and step 5 decomposes angles, thus creating only smaller angles. □

Remark. Note that the only property of $\mathcal{T}$ used in the proof of the cake-cutting lemma is that $\mu(\mathcal{T}) < \mu(\mathcal{A})$. The lemma thus also holds if we replace $\mathcal{T}$ by an arbitrary triangulation $\mathcal{B}$ of S that satisfies $\mu(\mathcal{B}) < \mu(\mathcal{A})$. In fact, it suffices if $\mathcal{B}$ is an improvement of $\mathcal{A}$ and pqr is not a triangle in $\mathcal{B}$.

4. The ear-cutting procedure. The cake-cutting lemma in §3 shows that if $\mathcal{A}$ is not yet a minmax angle triangulation and qs is an edge in $\mathcal{T}$, chosen by the algorithm to improve $\mathcal{A}$, then there are triangulations of the generated polygons P and R with all angles smaller than $\angle pqr$. The two questions that remain are how to find such an edge qs and how to quickly triangulate P and R. One obvious way to find qs (not necessarily in $\mathcal{T}$ but in an improvement of $\mathcal{A}$) is to try all possible points s with $qs \cap pr \neq \emptyset$. For each such s we add qs to $\mathcal{A}$ and remove all edges that intersect qs. The thus-created polygons P and R are triangulated with minimum largest new angle using dynamic programming. If the largest new angle is smaller than $\angle pqr$ we have an improvement of $\mathcal{A}$ and thus a desired qs.

Apparently, the implementation of an iterative step sketched in the above paragraph is rather inefficient. We improve the performance with a more clever way to search for an appropriate point s and with a fast procedure for triangulating P and R. The two tasks are woven together to the extent that it is not advisable to look at them as separate steps. For a chosen point s we attempt to triangulate P and R with all angles smaller than $\angle pqr$. If this fails we get some guidance on where to look for a better point s. Following this guidance, a next point s is chosen so that we can reuse part of the work done during the unsuccessful triangulation attempt. The fundamental notion in all of this is that of an ear of a polygon triangulation.

4.1. Ears. An *ear* in a polygon triangulation is a triangle bounded by two polygon edges and one diagonal. It is easy to show that any triangulation of a simple polygon with more than three vertices has at least two ears [14].

In order to efficiently triangulate P and R, with all angles smaller than $\mu = \mu(\mathcal{A}) = \angle pqr$, we need two properties. The first guarantees that no expensive testing is necessary to recognize when an edge is a diagonal.

LEMMA 4.1. *Let P' be a polygon obtained from P by repeatedly removing ears not incident to qs. If a, b, c are three consecutive vertices of P' with $\{q, s\} \not\subseteq \{a, b, c\}$ and $\angle abc < \pi$, then ac is a diagonal of P'.*

Proof. By construction of P each of its vertices can be connected by a straight-line segment within P to a point on qs. This property is maintained whenever we remove an ear not incident to qs, so it also holds for P'. In particular, it holds for the vertices a, b, and c of P'. The edge ac can avoid being a diagonal only if it intersects the boundary of P' (it cannot lie outside P' because $\angle abc < \pi$). But this contradicts the above property for either a or c or for both. □

By symmetry, Lemma 4.1 also holds for R. It is now easy to identify ears because only one angle has to be checked. This is because the angles at a and c inside abc are always smaller than μ as they are properly contained in angles of $\mathcal{A}$. Thus, all three angles of abc are smaller than μ if and only if $\angle abc < \mu$.

The second property we need is that it does not matter which ears we remove, and in what sequence we remove them, as long as their angles are small enough. This property is implied by the following lemma whose proof is omitted because it is identical to that of the cake-cutting lemma.

LEMMA 4.2. *Let P' be a polygon obtained from P by repeatedly removing ears not incident to qs. If qs is an edge of $\mathcal{T}$ then there exists a triangulation of P' without angles larger than or equal to μ.*

The two lemmas suggest that we triangulate P and R simply by repeatedly finding consecutive vertices a, b, c, with $\angle abc < \mu$, and removing the ear abc. We remark that this strategy can also be used to get an inductive proof of the cake-cutting lemma. The next two subsections show how ear cutting and the search for an appropriate point s can be combined to yield an efficient implementation of an iterative step.

4.2. How to cut. The way we search for a point s (§4.3) guarantees a certain property of the polygons P and R that simplifies their triangulation by ear cutting. To be accurate we should mention that at the time we start the triangulation process for P and R, some ears will already have been removed as a result of earlier attempts to triangulate polygons generated for other points s. Consistent with our earlier notation, we therefore denote the two polygons that we attempt to triangulate by P' and R'. We state the mentioned property as an invariant of the algorithm after introducing some notation.

As justified above we pretend that P' and R' are simple polygons; by construction they share the edge qs. Let $k + 2$ be the number of vertices of P' and $m + 2$ the number of vertices of R', and label them consecutively as $q = p_0, p_1, \cdots, p_k, p_{k+1} = s$ and $q = r_0, r_1, \cdots, r_m, r_{m+1} = s$ (see Fig. 3). Define $\phi_i = \angle p_{i-1}p_ip_{i+1}$ for $1 \leq i \leq k$ and $\rho_j = \angle r_{j-1}r_jr_{j+1}$ for $1 \leq j \leq m$.

We can now state the property of P' and R'.

Invariant. $\phi_i \geq \mu$ for all $1 \leq i < k$ and $\rho_j \geq \mu$ for all $1 \leq j < m$.

This implies that p_{k-1}, p_k, s are the only three vertices that possibly define an ear of P' that is not incident to qs (provided $k > 1$) and has all three angles smaller than μ. Symmetrically, r_{m-1}, r_m, s are the only such three vertices of R'. If $\phi_k < \mu$ then $p_{k-1}p_ks$ is indeed such an ear and we can remove it from P'. This operation decreases ϕ_{k-1}, the angle at p_{k-1}, and leaves all other ϕ_i unchanged. Thus, P' still satisfies the invariant after setting $k := k - 1$. Similarly, the invariant is maintained if we remove $r_{m-1}r_ms$ from R' and set $m := m - 1$.

We now describe this process more formally as a procedure that alternates between removing an ear from P' and removing an ear from R'. It either completes its

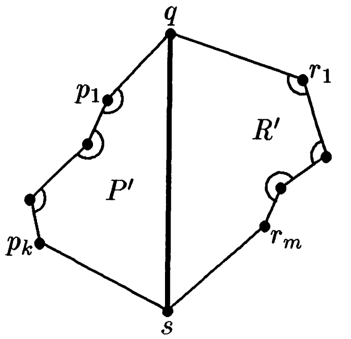


FIG. 3. *The circular arcs indicate angles that are known to be at least as large as μ.*

task of triangulating P' and R' or it stops because it encounters a situation where $\phi_k \geq \mu$ or $\rho_m \geq \mu$. To avoid repetition we separate out the code that tests an angle and removes an ear if the angle is small enough.

```
procedure CUTEARP'.
  if $\phi_k < \mu$ then
      if $k > 1$ then add the edge $p_{k-1}s$ to the triangulation endif;
      remove the triangle $p_{k-1}p_ks$ from $P'$ and set $k := k - 1$
    else set stop := true
  endif.
```

Similarly, we define a procedure CUTEARR$'$, which either removes $r_{m-1}r_ms$ from R' or raises the flag by setting *stop* := **true**. The attempt to triangulate P' and R' first alternates between the two polygons and, if one polygon is successfully triangulated, attempts to complete the polygon that remains.

```
stop := false;
while $k > 0$ and $m > 0$ and not stop do
  CUTEARP'; if not stop then CUTEARR' endif
endwhile;
while $k > 0$ and not stop do CUTEARP' endwhile;
while $m > 0$ and not stop do CUTEARR' endwhile.
```

If the procedure finishes without raising the flag (*stop* = **false**) then we must have $k = m = 0$ and the triangulation is complete. Otherwise, the flag is raised either while testing P' or while testing R' (so we should really have used two flags to be able to distinguish the two cases—and we pretend we did).

Assume the flag was raised because of $\phi_k \geq \mu$. Let $\vec{qs}$ be the half-line that starts at q and goes through s, and let p' be the point among $p_1, \cdots, p_k$ so that $\angle p'qs$ is a minimum. Note that p' is not necessarily equal to p_k, but $p' = p_k$ if P' is convex. We have the following lemma, which will be useful in searching for a new point s.

LEMMA 4.3. *Assuming $\phi_k \geq \mu$, there is no point $t \in S$ so that qt is an edge in a minmax angle triangulation $\mathcal{T}$ of S, $qt \cap p_ks \neq \emptyset$, and $qt \cap p's \neq \emptyset$.*

Proof. Suppose there is a point t that contradicts the assertion. Because $qt \cap p_ks \neq \emptyset$, this edge qt generates a polygon P'' so that $q = p_0, p_1, \cdots, p_k$ is a contiguous subsequence of its vertices (after removing appropriate ears). Let $p_{k+1}, \cdots, p_{k''}, p_{k''+1} = t$

be the other vertices of P''. By assumption we have $\angle p_{i-1}p_ip_{i+1} \geq \mu$ for $1 \leq i \leq k-1$. Furthermore, $\angle p_{k-1}p_kp_j \geq \mu$ for all $k+1 \leq j \leq k''+1$ because all these angles are larger than ϕ_k, the angle at p_k in P'. Hence, any attempt to triangulate P'' by removing ears (not incident to qs with angles all smaller than μ) must fail to cut off ears at p_i for all $1 \leq i \leq k$. □

Remark. As in the remark after the cake-cutting lemma, we can argue that Lemma 4.3 is also true if we replace $\mathcal{T}$ by an arbitrary triangulation that is an improvement of $\mathcal{A}$.

Lemma 4.3 suggests that the search for a new s continue between $\vec{qr'}$ and $\vec{qs}$ if the flag is raised while testing P', where r' is the counterpart of p' in R' and s is the old s. Thus, all ears removed from P' are safe and do not have to be considered again. However, all ears removed from R' have to be added back because they will intersect any future edge qs. Simultaneously, the value of m has to be adjusted. The amount of time needed to add these ears back in is proportional to the number of ears removed from P', because the ear cutting alternates between P' and R'. Symmetric actions are in order when the flag is raised while testing R'.

4.3. How to search. Let us go back to the triangulation $\mathcal{A}$ of S that is not yet a minmax angle triangulation, and as usual let p, q, r be the points so that pqr is a triangle in $\mathcal{A}$ and $\angle pqr = \mu = \mu(\mathcal{A})$. The first vertex s that we test is the third vertex of the other triangle of pr (if no such triangle exists, then pr is an edge of the convex hull of S and no appropriate point s exists). Thus we add qs and remove pr. If the new angles at p and r are both smaller than μ, then we are done. If $\angle qps < \mu$ and $\angle qrs \geq \mu$, then, by Lemma 4.3, the edges we should test must intersect ps. Symmetrically, if $\angle qps \geq \mu$ and $\angle qrs < \mu$, then we must search for edges that intersect sr. If both angles are at least μ, then no appropriate edge exists.

We now generalize and formalize this idea. For given polygons P' and R' we define vertices p' and r' as above, and we denote the open wedge between $\vec{qp'}$ and $\vec{qr'}$ by W. This wedge will get progressively smaller as we proceed with the search, and only points s within the wedge will be considered as endpoints of new edges qs. Initially, $p' = p$ and $r' = r$. We are now ready to describe the algorithm that searches for an appropriate point s.

Input. A triangulation $\mathcal{A}$ of S with maximum angle $\angle pqr = \mu = \mu(\mathcal{A})$.
Output. An improved triangulation or a message that the maximum angle cannot be decreased. In the latter case, the input triangulation is a minmax angle triangulation of S.
Define. THIRD(a, b) is the vertex c of the triangle abc so that q and c lie on opposite sides of the line through a and b. If such a vertex does not exist, which is the case if ab is an edge of the convex hull of S, then THIRD(a, b) is undefined. As before, W denotes the open wedge defined by p', q, and r'.

```
Initialize $k := 1$, $p_1 := p' := p$, $m := 1$, and $r_1 := r' := r$.
loop
 if THIRD$(p_k, r_m)$ is not defined then
   return the message that the maximum angle cannot be decreased and stop.
  else
   set $s :=$ THIRD$(p_k, r_m)$ and remove $p_kr_m$ from $\mathcal{A}$.
   if $s \in W$ then
     add $qs$ to $\mathcal{A}$ and attempt the triangulation of $P'$ and $R'$ as described in §4.2.
     case 1. The attempt succeeds. Return the new triangulation and stop.
```

```
      case 2. The flag was raised while testing $P'$. Set $k := k + 1$ & $p_k := p' := s$.
      case 3. The flag was raised while testing $R'$. Set $m := m + 1$ & $r_m := r' := s$.
    else (i.e., $s \notin W$)
      if $sr_m$ intersects $W$ then
        set stop := false; while not stop do CUTEARP' endwhile;
        set $k := k + 1$ and $p_k := s$.
      else (i.e., $sp_k$ intersects $W$)
        set stop := false; while not stop do CUTEARR' endwhile;
        set $m := m + 1$ and $r_m := s$.
      endif
    endif
  endif
forever.
```

We would like to point out a subtlety of the algorithm needed to prove its correctness. That is, the polygons P' and R' defined by any edge qs are obtained from $\mathcal{A}$ by removing *only* edges that intersect qs. Of course, some edges not in $\mathcal{A}$ have been added already to remove some ears. In other words, P' is the polygon P (as defined in §2) with some ears removed, and the same is true for R' and R.

4.4. The final analysis. The running time of an iterative step (the above algorithm) is proportional to the number of removed ears. Because of the alternation between removing an ear from P' and one from R', at most only one more than half of the removed ears are added back to the polygon. This is also true if one polygon is completely triangulated while ears are still removed from the other polygon, because in this case only the ears of the former polygons need to be added back in, and their number is smaller than the number of ears cut off from the other polygon. It follows that the total number of removed ears is $O(n)$. A single iteration therefore takes only $O(n)$ time. Together with Lemma 2.3, which states that there are only $O(n^2)$ iterations, this implies a cubic upper bound on the time complexity of our algorithm (if implemented without priority queue).

Below we argue that its running time is actually $O(n^2 \log n)$. To achieve this bound it is necessary to store the angles of the current triangulation in a priority queue, for otherwise finding all maximum angles costs time $\Omega(n^3)$. The crucial observation is that the time spent in an iterative step is proportional to the number of edges in the input triangulation that intersect the new edge qs. Each such edge has been removed and we argue that it will never be added again because every future triangulation will have an edge qt that intersects $p_k r_m$, the last edge before s. First note that every future triangulation is an improvement of $\mathcal{A}$. By Lemma 4.3 and the remark following it, every improvement of $\mathcal{A}$ has an edge qt in the final wedge W as maintained by the algorithm. Both p_k and r_m lie outside W (possibly on its boundary) and the edge $p_k r_m$ intersects W. The claim follows because all points of $W \cap S$ lie beyond $p_k r_m$ as seen from q. This implies the $O(n^2 \log n)$ bound because we have only $\binom{n}{2} = O(n^2)$ edges to work with. It should be noted that the maintenance of the priority queue storing the angles is the sole reason for the $\log n$ term in the $O(n^2 \log n)$ bound; all other operations take total time $O(n^2)$.

5. Extensions. We address two types of extensions of our algorithm for constructing minmax angle triangulations. The first extension is to the constrained case where the input consists of a set of n points plus some pairwise disjoint edges defined by the points that are required to be in the triangulation. The second extension discusses the optimization of the entire angle vector rather than just the maximum angle.

Only minor changes are necessary to adapt the algorithm presented in §§2 and 4 to the constrained case. The most important change is that no prescribed edge will be removed to give way to searching for a new point s. This modification takes no extra time, which implies the part of the Main Theorem that deals with prescribed edges.

Before we introduce angle vectors, note that for a given point set S all triangulations (whether constrained or not) have the same number of triangles and therefore the same number of angles. By Euler's formula for planar graphs the number of triangles is $t = 2n - h - 2$, where $n = |S|$ and h is the number of points of S that lie on the boundary of its convex hull. For any triangulation $\mathcal{A}$ of S we define its *angle vector* $V_{\mathcal{A}} = (\alpha_1, \alpha_2, \cdots, \alpha_{3t})$, with $\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_{3t}$ the $3t$ angles of the t triangles. If $\mathcal{B}$ is another triangulation of S with angle vector $V_{\mathcal{B}} = (\beta_1, \beta_2, \cdots, \beta_{3t})$ we define $V_{\mathcal{B}} < V_{\mathcal{A}}$ if there is an index $1 \leq j \leq 3t$ so that $\beta_i = \alpha_i$ for $1 \leq i < j$ and $\beta_j < \alpha_j$. For example, $V_{\mathcal{B}} < V_{\mathcal{A}}$ if $\mathcal{B}$ is an improvement of $\mathcal{A}$, but the reverse is not necessarily true.

The problem of finding a triangulation with minimum angle vector is at least as difficult as finding a minmax angle triangulation. If any two angles defined by three points of S each are different we can construct the minimum angle vector triangulation—which is unique in this case—as follows.

> First, construct a minmax angle triangulation $\mathcal{T}_1$ and declare the three edges of the triangle that contains the maximum angle as prescribed. Second, construct a minmax angle triangulation $\mathcal{T}_2$ for the thus constrained input and introduce new constraints to enforce the second largest angle in future triangulations. Continue this way and construct triangulations $\mathcal{T}_3$, $\mathcal{T}_4$, and so on until the prescribed edges add up to a triangulation themselves. This triangulation minimizes the angle vector.

An $O(n^3 \log n)$ time-bound for this algorithm is obvious because it just iterates the minmax angle triangulation algorithm a linear number of times. Even better, we have an $O(n^2 \log n)$ time-bound if we use $\mathcal{T}_i$ as the input triangulation for the construction of $\mathcal{T}_{i+1}$. The improved bound follows because an edge once removed cannot appear in any future triangulation. We thus get the following result by the same argument as in §4.4.

THEOREM 5.1. *Given a set of n points in the plane so that no angles defined by three points each are equally large, the triangulation that lexicographically minimizes the angle vector can be constructed in time $O(n^2 \log n)$ and space $O(n)$.*

Remark. In the presence of multiple angles it is not clear how to adapt the approach of this paper without requiring an exponential amount of time in the worst case. We pose the existence of a polynomial algorithm for minimizing the angle vector in the presence of multiple angles as an open problem. A case where multiple angles can be handled relatively easily is that of a simple polygon. The straightforward cubic time algorithm for minimizing the maximum angle, derived from the dynamic programming algorithm of Klincsek [10], can be extended to an $O(n^4)$ time algorithm for minimizing the angle vector as follows. Instead of characterizing a (partial) triangulation by its maximum angle we store its sorted angle vector. The best triangulation of a sequence of vertices is then selected on the basis of these vectors. The cubic time increases to $O(n^4)$ because comparing two angle vectors takes $O(n)$ time in the worst case, in contrast to constant time for comparing maximum angles.

6. Experimental results. To demonstrate that the results of the preceding sections, which we believe are of theoretical interest, are significant also from a practical viewpoint, we implement the algorithm along with a few other triangulation algorithms from the literature. Using these implementations, we perform a small-scale comparative study of the triangulations they produce. A more extensive study and complete description of the findings will soon be available as the master's thesis of Waupotitsch. The difference between two triangulations is expressed in terms of their angles and edges (as in [16]).

The experimental study is based on implementations of four different triangulation algorithms. Three work by iterative improvement, and to construct an initial triangulation we use a plane-sweep strategy (see, e.g., [6, §8.3.1]). Triangulations constructed by plane-sweep are denoted by PS. The implementation of the edge-insertion algorithm of this paper minimizes the angle vector as discussed in §5. Its triangulations are referred to as MV. To avoid the difficulty that arises when two angles are equally large (see the remark at the end of §5), we use a heuristic that breaks ties in a consistent manner. Delaunay triangulations, *DEL*, are constructed by flipping the diagonals of convex quadrilaterals as long as the smallest angle involved increases (see, e.g., [11]). The third incremental improvement algorithm flips the diagonal of a convex quadrilateral if the largest of the six involved angles decreases. As shown in [15], this heuristic typically gets stuck in a local optimum depending on the initial triangulation as well as on the way the flips are scheduled. We use this algorithm to construct triangulations *FPD, FPN, FDD,* and *FDN,* where the middle letter distinguishes between PS and *DEL* as the initial triangulation and the final letter distinguishes between deterministic (largest angle first) and "nondeterministic" (first in first serve) scheduling.

The point sets chosen for our experimental study are drawn uniformly either inside a square or near a circle (see Fig. 4). To allow for exact arithmetic all points are chosen on the integer grid. For each of various point set sizes, 30 experiments are carried out and average statistics are compiled.

Table 1 compares triangulations and their quality. More specifically, it compares each triangulation $X \in \{PL, DEL, FPD, FPN, FDD, FDN\}$ with MV, the optimum triangulation. The parameter Δe gives the number of edges in X that are not in MV. The angle vectors of X and MV are compared using parameters $\angle_{eq}$, $\angle_{sm}$, and $(\angle/\angle_{MV})$. This means that the $\angle_{eq}$ largest angles of X and MV are the same and that the next $\angle_{sm}$ largest angles are smaller for *MV.* $(\angle/\angle_{MV})$ is the ratio between the $\angle_{eq}+1$ largest angle of each triangulation. The statistics show that for points uniformly distributed in a square the edge-flip heuristic produces triangulations that come close to the optimum. Consistent with the findings reported in [16], *DEL* differs from MV by slightly less than 10 percent of its edges. This is in sharp contrast to the relative performance of the algorithms for points chosen on or close to a circle. In this case, *DEL* and MV share very few nonconvex hull edges. The edge-flip heuristic produces triangulations that are superior in terms of angles to *DEL*, but they hardly share any more edges with MV.

It is interesting to note that the amount of work needed to construct MV is far less for points in a square than for points near a circle. Table 2 shows the number of edges removed during the construction of MV. While the difference between the two point distributions is striking, the choice of the initial triangulation seems to have far less

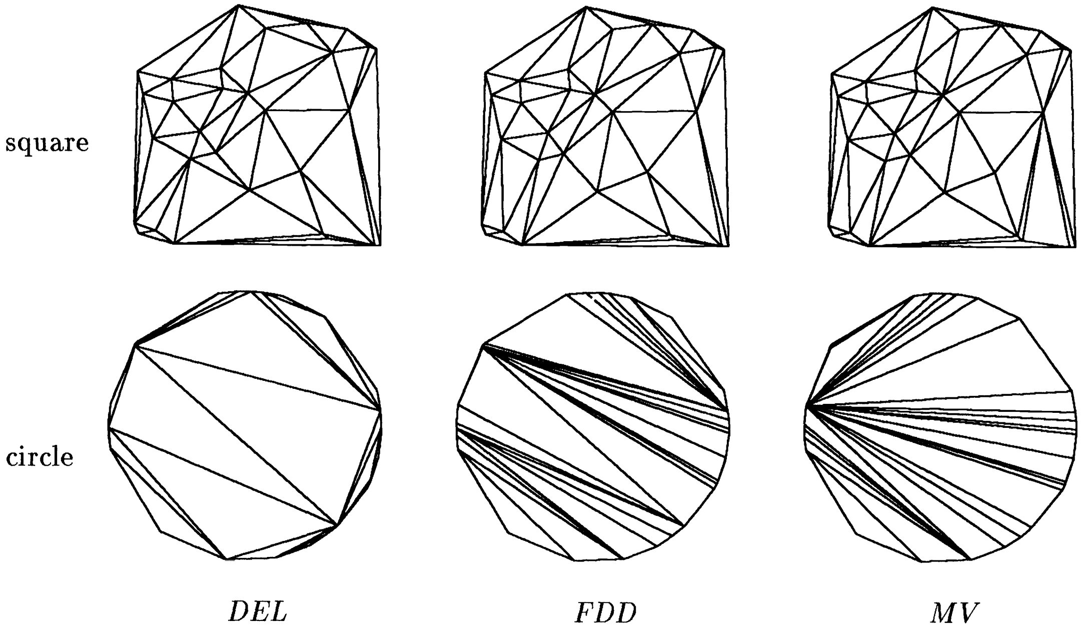


FIG. 4. *The Delaunay triangulation, DEL, a locally optimal triangulation, FDD, and the globally optimal triangulation, MV, for two small point sets.*

influence on the running time of the edge-insertion algorithm. In general, we observe that the edge-insertion algorithm is much faster on the average than expressed by the worst-case analysis in §4. We would also like to remark that there are no polynomial time-bounds known for the edge-flip heuristic used in our experimental study.

7. Conclusions. The main result of this paper is an $O(n^2 \log n)$ time algorithm for constructing a minmax angle triangulation of a set of n points in the plane, with or without prescribed edges. This seems fairly efficient considering that it is the first polynomial time algorithm for the problem and that it somehow avoids looking at all the $\binom{n}{3}$ angles defined by the n points. On the other hand, our algorithm is a factor n slower than the best algorithms for constructing Delaunay triangulations, at least in the worst case. We thus pose the question of whether a minmax angle triangulation can be constructed in $o(n^2 \log n)$ time.

In the nondegenerate case where no two angles defined by three points each are equal, the algorithm can be extended to compute the triangulation that lexicographically minimizes the sorted vector of angles. The running time is still $O(n^2 \log n)$ in the worst case, and our experiments indicate that the average run time is significantly less.

A problem related to minimizing the maximum angle is to construct a triangulation that minimizes the number of obtuse angles. It seems that the edge-insertion scheme does not work for this criterion. The problem thus remains open for point sets, although dynamic programming yields a cubic time algorithm if the input is a simple polygon. Still, the authors of this paper believe that the edge-insertion scheme is more generally applicable and plan to further investigate this paradigm.

TABLE 1

Comparison of MV with other triangulations.

	50 points (in square)				100 points (in square)				200 points (in square)			
	Δe (%)	$\angle_{eq}$	$\angle_{sm}$	$\frac{\angle}{\angle_{MV}}$	Δe (%)	$\angle_{eq}$	$\angle_{sm}$	$\frac{\angle}{\angle_{MV}}$	Δe (%)	$\angle_{eq}$	$\angle_{sm}$	$\frac{\angle}{\angle_{MV}}$
PS	51.9	0	91	1.020	62.5	0	190	1.012	70.8	0	391	1.006
DEL	7.3	7	29	1.033	8.0	8	53	1.018	8.8	11	66	1.008
FPD	1.5	97	5	1.011	3.1	36	42	1.022	3.3	24	66	1.011
FPN	1.6	89	6	1.012	3.2	35	32	1.022	3.2	26	68	1.012
FDD	2.4	54	7	1.015	2.2	74	13	1.017	2.5	28	30	1.010
FDN	2.6	29	7	1.017	2.5	91	15	1.022	2.9	28	34	1.010
	500 points (in square)				1000 points (in square)				50 points (near circle)			
PS	79.0	0	995	1.002	84.0	0	2001	1.001	45.7	0	33	1.058
DEL	8.8	17	185	1.005	9.1	21	231	1.002	46.5	0	47	1.072
FPD	3.3	40	195	1.005	3.0	55	438	1.003	43.1	1	26	1.046
FPN	3.3	41	151	1.006	3.2	57	454	1.003	43.7	1	26	1.045
FDD	2.7	45	92	1.009	2.7	62	200	1.007	42.0	2	24	1.039
FDN	3.0	43	91	1.008	3.0	60	201	1.006	40.6	2	22	1.035
	100 points (near circle)				200 points (near circle)				500 points (near circle)			
PS	43.5	4	79	1.003	47.1	7	104	1.001	44.5	2	176	1.00006
DEL	38.6	5	28	1.001	46.5	7	209	1.001	43.4	24	358	1.00003
FPD	39.4	16	34	1.005	46.4	15	94	1.004	43.4	77	215	1.00018
FPN	39.3	16	34	1.005	46.6	15	93	1.004	43.4	76	242	1.00018
FDD	38.2	18	28	1.004	46.1	16	88	1.004	43.3	80	335	1.00024
FDN	39.8	18	17	1.005	46.8	16	85	1.006	43.3	80	202	1.00011

TABLE 2

The number of edges removed by the edge-insertion algorithm when it computes MV from either PS or DEL.

	Square					Circle			
	50 pts	100 pts	200 pts	500 pts	1000 pts	50 pts	100 pts	200 pts	500 pts
PS	240	647	1607	5067	11847	1301	2340	17136	63003
DEL	153	390	946	2887	6658	1303	2276	16660	58934

Acknowledgment. The second author thanks Professor C. L. Liu for his constant support and encouragement.

REFERENCES

[1] I. BABUŠKA AND A. K. AZIZ, *On the angle condition in the finite element method*, SIAM J. Numer. Anal., 13 (1976), pp. 214–226.

[2] R. E. BARNHILL AND F. F. LITTLE, *Three- and four-dimensional surfaces*, Rocky Mountain J. Math., 14 (1984), pp. 77–102.

[3] J. CAVENDISH, *Automatic triangulation of arbitrary planar domains for the finite element method*, Internat. J. Numer. Methods Engrg., 8 (1974), pp. 679–696.

[4] T. H. CORMEN, C. E. LEISERSON, AND R. L. RIVEST, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

[5] B. DELAUNAY, *Sur la sphère vide*, Izv. Akad. Nauk SSSR, Otdel. Mat. i Estest. Nauk, 7 (1934), pp. 793–800.

[6] H. EDELSBRUNNER, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Heidelberg, Germany, 1987.

[7] S. J. FORTUNE, *A sweepline algorithm for Voronoi diagrams*, Algorithmica, 2 (1987), pp. 153–174.

[8] J. A. GREGORY, *Error bounds for linear interpolation on triangles*, in The Mathematics of Finite Element and Applications II, J. R. Whiteman, ed., Academic Press, New York, 1975, pp. 163–170.

[9] L. J. Guibas and J. Stolfi, *Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams*, ACM Trans. Graphics, 4 (1985), pp. 74–123.

[10] G. T. Klincsek, *Minimal triangulations of polygonal domains*, Ann. Discrete Math., 9 (1980), pp. 121–123.

[11] C. L. Lawson, *Generation of a triangular grid with applications to contour plotting*, Jet Propulsion Laboratory Tech. Memo. 299, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 1972.

[12] ———, *Software for C^1 surface interpolation*, in Mathematical Software III, J. R. Rice, ed., Academic Press, New York, 1977, pp. 161–194.

[13] D. T. Lee and A. K. Lin, *Generalized Delaunay triangulations for planar graphs*, Discrete & Comput. Geom., 1 (1986), pp. 201–217.

[14] G. H. Meisters, *Polygons have ears*, Amer. Math. Monthly, 82 (1975), pp. 648–651.

[15] G. M. Nielson, *An example with a local minimum for the minmax ordering of triangulations*, manuscript, Lawrence Livermore National Laboratory, Livermore, CA, 1987.

[16] G. M. Nielson and R. Franke, *Surface construction based upon triangulations*, in Surfaces in Computer Aided Geometric Design, R. E. Barnhill and W. Boehm, eds., North-Holland, Amsterdam, 1983, pp. 163–177.

[17] D. A. Plaisted and J. Hong, *A heuristic triangulation algorithm*, J. Algorithms, 8 (1987), pp. 405–437.

[18] M. J. D. Powell and M. A. Sabin, *Pairwise quadratic approximation on triangles*, ACM Trans. Math. Software, 3 (1977), pp. 316–325.

[19] F. P. Preparata and M. I. Shamos, *Computational Geometry—an Introduction*, Springer-Verlag, New York, 1985.

[20] R. Seidel, *Constrained Delaunay triangulations and Voronoi diagrams with obstacles*, in 1978–1988, 10-Years IIG, Report of the Institute for Information Processing, Technical University of Graz, Austria, 1988, pp. 178–191.

[21] M. I. Shamos and D. Hoey, *Closest point problems*, in Proc. 16th Annual IEEE Symposium on the Foundations of Computer Science, Berkeley, CA, 1975, pp. 151–162.

[22] R. Sibson, *Locally equiangular triangulations*, Comput. J., 21 (1978), pp. 243–245.

[23] G. Strang and G. Fix, *An Analysis of the Finite Element Method*, Prentice-Hall, Englewood Cliffs, NJ, 1973.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 4, pp. 1009–1024, July 1992

010

WHICH CUBIC SPLINE SHOULD ONE USE?*

R. K. BEATSON† AND E. CHACKO†

Abstract. The aim of this paper is to provide a quantitative comparison of eight different C^1 and C^2 cubic spline interpolation schemes. The C^1 schemes discussed are local while the C^2 ones are global.

In practice cubic splines are often used when the smoothness of the function being interpolated/approximated is unknown. Also, it is often necessary, or advantageous, to use a nonuniform mesh. Therefore performance over a variety of smoothness classes, using uniform and also several thousand random meshes, is compared. The performance criteria used are the *quantitative* ones of exact operator and derived operator norms and best possible pointwise error estimates.

Key words. cubic splines, end conditions, operator norms, error estimates

AMS(MOS) subject classifications. 65D07, 65D10, 41A15

1. Introduction.

Our aim is to find an interpolant that will give good results when used to interpolate functions of unknown smoothness on a possibly nonuniform mesh. We consider eight different C^1 or C^2 cubic spline interpolation schemes and compare their operator and derived operator norms as well as the pointwise error

$$\sup_{f\in\mathcal{F}} |(f-s)(x)|$$

for uniform knot distributions, several thousand random knot distributions, and several smoothness classes $\mathcal{F}$. It is worth emphasizing that we do not seek the best interpolation method for a fixed smoothness class $\mathcal{F}$—the problem of optimal interpolation.

All the schemes considered fit cubic splines with knots at the nodes of interpolation t_i

$$s(t_i) = f(t_i), \qquad i = 0, 1, \cdots, n.$$

1.1. C^2 methods—not strictly local.

In the first six methods the spline is chosen to be C^2, so that each scheme corresponds to a different choice of two *end conditions.* Such schemes are not suitable for certain applications as they are not *strictly local.* Thus data far away from x can influence the value of $s(x)$. However, they are *semilocal,* meaning that the influence of data at point t_i on $s(x)$ falls off geometrically with the number of knots between x and t_i (see the discussion in §2.1 below).

Method A:

$$s^{(3)}(t_1-) = s^{(3)}(t_1+) \quad\text{and}\quad s^{(3)}(t_{n-1}-) = s^{(3)}(t_{n-1}+), \tag{1}$$

the well-known *not-a-knot* end condition. (See Kershaw [7] and de Boor [2].) This end condition forces the restrictions of the spline to the first two and the last two intervals to be single cubics. The maximum convergence rate, meaning the rate for a general C^∞ function, is $\mathcal{O}(\delta^4)$, where δ is the mesh size. Of course the limiting rate is actually achieved for C^3 functions with Lipschitz third derivative.

* Received by the editors October 31, 1988; accepted for publication June 25, 1991.

† Mathematics Department, University of Canterbury, Private Bag, Christchurch 1, New Zealand (rkb@math.canterbury.ac.n z and pec@math.canterbury.ac.nz).

Method B:

$$s'(t_0) = c_l'(t_0) \quad \text{and} \quad s'(t_n) = c_r'(t_n), \tag{2}$$

where c_l and c_r are cubic polynomials with

$$c_l(t_i) = f(t_i) \quad \text{and} \quad c_r(t_{n-i}) = f(t_{n-i}), \qquad 0 \le i \le 3.$$

Thus this spline is chosen to have the same first derivative as the local cubic interpolant through the first (last) four knots, at the first (last) knot. The maximum convergence rate is $\mathcal{O}(\delta^4)$.

Method C:

$$s''(t_0) = c_l''(t_0) \quad \text{and} \quad s''(t_n) = c_r''(t_n) \tag{3}$$

with c_l and c_r as above. Thus this spline is chosen to have the same second derivative as the local cubic interpolant through the first (last) four knots, at the first (last) knot. The maximum convergence rate is $\mathcal{O}(\delta^4)$.

Method D:

$$d_1 = d_2 \quad \text{and} \quad d_{n-2} = d_{n-1}, \tag{4}$$

where

$$d_i = s^{(3)}(t_i+) - s^{(3)}(t_i-).$$

Here the jump discontinuities in the third derivative at the second and third knots are forced to be the same, and similarly for the second and third to last knots. This method is known to minimize $\|f - s\|_\infty$ when the knots are equispaced and f is a quartic polynomial. The maximum convergence rate is $\mathcal{O}(\delta^4)$, and the method is not recommended for use with nonuniform meshes.

Method E:

$$s''(t_0) = 0 \quad \text{and} \quad s''(t_n) = 0, \tag{5}$$

the so-called *natural cubic spline* end condition. This *should not generally be used* for approximation purposes, as it throws away any second derivative information in the data and, as a consequence, the order of approximation near the endpoints is restricted to $\mathcal{O}(\delta^2)$.

Method F:

$$s'(t_0) = q_l'(t_0) \quad \text{and} \quad s'(t_n) = q_r'(t_n), \tag{6}$$

where q_l and q_r are quadratic polynomials with

$$q_l(t_i) = f(t_i) \quad \text{and} \quad q_r(t_{n-i}) = f(t_{n-i}), \quad 0 \le i \le 2.$$

Thus, this spline is chosen to have the same first derivative as the local quadratic interpolant through the first (last) three knots, at the first (last) knot. The maximum convergence rate is $\mathcal{O}(\delta^3)$.

1.2. C^1 methods—strictly local. The last two schemes are simple *strictly local* methods. This property allows easy modification of previously obtained fits, and is essential for some applications, such as CAD. However, there is a cost, namely, that the error estimates for smooth functions away from the ends of the interval are generally somewhat worse. The local methods we consider are C^1 rather than C^2.

Method G: s' agrees with the derivative of a local quadratic at every knot. Thus defining q_i as the quadratic that interpolates to f at t_i, t_{i+1}, and t_{i+2},

$$s'(t_j) = \begin{cases} q_0'(t_0), & j = 0, \\ q_{j-1}'(t_j), & 0 < j < n, \\ q_{n-2}'(t_n), & j = n. \end{cases} \tag{7}$$

The maximum convergence rate is $\mathcal{O}(\delta^3)$.

Method H: s' agrees with the derivative of a local cubic at every knot. Thus defining c_i as the cubic that interpolates to f at t_i, t_{i+1}, t_{i+2}, and t_{i+3},

$$s'(t_j) = \begin{cases} c_0'(t_0), & j = 0, \\ c_{j-1}'(t_j), & 0 < j \le n/2, \\ c_{j-2}'(t_j), & n/2 < j < n, \\ c_{n-3}'(t_n), & j = n. \end{cases} \tag{8}$$

The maximum convergence rate is $\mathcal{O}(\delta^4)$.

1.3. Outline of criteria used. The present work is an extension of that in [1], where it was shown that, for methods A, B, and C,

$$\|f - s\|_\infty \le K\delta^j \omega(f^{(j)}, \delta), \qquad 1 \le j \le 3, \tag{9}$$

where $\omega(f^{(j)}, \delta)$ is the modulus of continuity for $f^{(j)}$. For $j = 2, 3$, K is an absolute constant independent of the mesh $\mathbf{t}$. For $j = 1$ it depends on the spacing of the first few and last few knots. Estimate (9) implies, in particular, $\mathcal{O}(\delta^4)$ approximation to C^4 functions. However, [1] does not give any grounds for choosing a particular interpolant from amongst those methods with $\mathcal{O}(\delta^4)$ error estimates, and no information about strictly local methods, or those methods with maximum convergence rates less than $\mathcal{O}(\delta^4)$.

Our present aim was therefore to make a quantitative comparison of various commonly used cubic spline interpolants. Comparisons are made using uniform meshes and also several thousand nonuniform meshes. For each of these cases we computed the operator norms of the spline projector itself, and also of the first derived projectors $L' : f' \to s'$. For the C^2 methods we also computed the norm of the second derived projector $L'' : f'' \to s''$. These norms give a quantitative measure of the tendency of the particular spline operator to introduce *extraneous bumps and wiggles.* We also compute the *pointwise error multiplier*

$$K(j,x) = \sup_{f \in W_{j,\infty} \,:\, \|f^{(j)}\|_\infty \le 1} |f(x) - s(x)|,$$

$1 \le j \le 4$. The utility of these functions lies in the associated "tight" bounds

$$|f(x) - s(x)| \le K(j,x)\|f^{(j)}\|_\infty$$

and

$$\|f - s\|_\infty \le C_j \|f^{(j)}\|_\infty,$$

where $C_j = \sup_x K(j,x)$.

1.4. Summary of results. The results are presented in detail in §2, and the mathematics underlying the calculations, in §3. While we do reach a conclusion and recommend one method, namely, Method B, for general purpose use, the reader can use the results to analyze the pros and cons of other choices. For example, the results show the cost of using the *strictly local* Method H, rather than a more conventional cubic spline interpolant, when approximating C^4 functions in the middle of the interval. For uniform meshes this is a worsening of the error bound by a factor of approximately 1.8, which may be quite acceptable. They also show that giving up fourth-order convergence to smooth enough functions can result in better convergence to functions with fewer derivatives.

When a simple *strictly local* method is required we would recommend instead Method H. Clearly the norm of the latter interpolation operator can be bounded in terms of the local mesh ratio, $m_n = \max\{h_i/h_j : |i-j| = 1\}$, and independently of the number of knots. Marsden [8] has shown that this property fails to hold for several of the C^2 cubic spline interpolants we consider. We remind the reader that in an adaptive curve fitting setting the local mesh ratio may get very large, and the norm of all of the interpolation operators discussed here, including the strictly local ones, gets large with it. However, the norm of the interpolation operator can be kept bounded if one chooses the nodes of interpolation after the knots of the spline, or replaces interpolation by quasi interpolation. (See de Boor [2, pp. 191–196; 208–213].)

Method B fits a C^2 function and has fourth-order convergence when the data comes from a sufficiently smooth function. Define

$$\begin{aligned} W_{j,\infty}[a,b] &= \left\{ f \in C^{j-1}[a,b] : f^{(j-1)} \text{ absolutely continuous and} f^{(j)} \in L_\infty[a,b] \right\} \\ &= \left\{ f \in C^{j-1}[a,b] \text{ with } f^{(j-1)} \text{ Lipshitz} \right\}. \end{aligned}$$

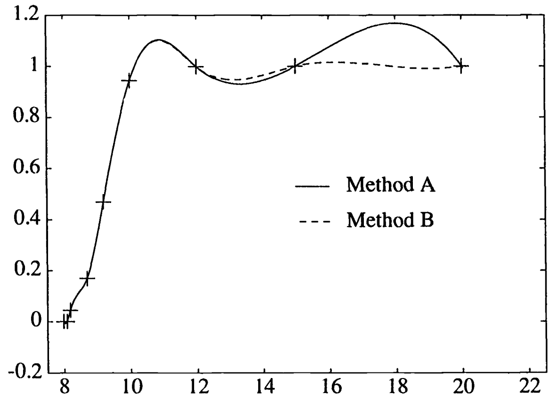


FIG. 1. *Different cubic spline fits to* RPN-14 *data.*

Method B's performance for $W_{4,\infty}$ functions is on average very slightly worse than the best of the other methods. However, it performs significantly better than the other C^2 fourth-order methods on functions of lower smoothness. Of course, the various C^2 cubic interpolatory splines will differ greatly only in the first and last few intervals. Figure 1 illustrates this. However, our contention is that the differences in these few intervals are important. This is particularly so since adapting a standard cubic spline code to end conditions of Method B is trivial, and the extra computational expense is at worst five or six *flops*.

2. Detailed results. In this section we present the results of our computations of exact error estimates and operator norms.

2.1. Differences between the C^2 and C^1 methods in the middle intervals. We wish to emphasize the point made at the end of the previous section, that the differences between the various C^2 interpolants are generally only significant in the first and last few intervals. For example, Kershaw [6] gives matrix estimates that, for a reasonably general set of end conditions, can be used to show the geometric decay of the influence of the end conditions, on the value of $s(x)$, as the number of knots between x and the endpoints grows.

To give a particular instance, if one considers the middle two intervals of a 30 subinterval uniform mesh, then the calculated least constant C_4 for which

$$\sup_{14h \le x \le 16h} |f(x) - s(x)| \le C_4 h^4 \|f^{(4)}\|_{[0,30h]},$$

for each of the C^2 methods, agrees to eight significant digits with the value 5/384 appropriate for cardinal interpolation on an infinite mesh. (For the infinite problem, see, for example, Powell [9].) Interestingly, 5/384 is also the optimal constant in the bound $\|f - s\| \le Ch^4 \|f^{(4)}\|$ for the error in complete cubic spline interpolation where one has extra first derivative information at the end points (see Hall [4] and Hall and Meyer [5]). In contrast, the strictly local methods can vary in their behavior in the middle intervals. Results are shown in Table 1 below. The values under the optimal column are values that can be achieved by a, possibly noncubic, spline interpolation method. That these values are lower bounds can be seen from the behavior of the Euler splines $\mathcal{E}_k$ (see Schoenberg [10]).

TABLE 1
Error constant in middle of a 30 *interval uniform mesh.*

Derivative	Optimal	C^2 methods	Method G	Method H
1	$1/2 = .5$	.7745	.6250	.6875
2	$1/8 = .125$	.1623	.1406	.1517
3	$1/24 \approx .0416$	.0431	.0468	.0468
4	$5/384 \approx .0130$	.0130	undefined	.0234

It is apparent that the C^2 methods do somewhat better than the strictly local methods when f is smooth.

2.2. Results for pseudorandom meshes. Fix for the moment the mesh $\mathbf{t}$ and the smoothness class $C^j[t_0, t_n]$, $1 \le j \le 4$. Then for each $x \in [t_0, t_n]$, and each method α that reproduces π_{j-1}, we can compute the *pointwise error multiplier* $K(\alpha; j, x)$, which is the smallest number for which the bound

$$|f(x) - s(x)| \le K(\alpha; j, x) \|f^{(j)}\|_\infty$$

holds for all $f \in C^j[t_0, t_n]$. For the interpolants considered here the same bound holds for $W_{j,\infty}[t_0, t_n]$. Then the *error constant* $C_{\alpha,j}$ is defined to be the smallest number for which the relationship

$$\|f - s\|_\infty \le C_{\alpha,j} \|f^{(j)}\|_\infty, \quad \forall f \in C^j[t_0, t_n]$$

holds. Clearly $C_{\alpha,j} = \sup_{x\in[t_0,t_n]} K(\alpha; j, x)$. We can compare interpolation methods by comparing the error constants $C_{\alpha,j}$. Using Method A as a standard, we compute for each fixed mesh, and method α, the *relative error constant*

$$e_{\alpha,j} = C_{A,j}/C_{\alpha,j}.$$

The performance of the interpolants on nonuniform meshes was compared by conducting 5000 pseudorandom trials. In each trial nine numbers were generated uniformly at random in $[0, 1]$, then sorted and scaled to obtain a mesh $\mathbf{t} : 0 = t_0 < t_1 < \cdots < t_8 = 1$. Then *relative error constants* were computed numerically. The resulting 5000 relative error constants were then sorted and the 1 percentile, mean, and 99 percentile points displayed in a bar graph. A logarithmic scale was used so that *relative error constants* of κ and $1/\kappa$ have the same visual impact.

2.2.1. Comparison of the overall error bounds. In the first two graphs of Fig. 2, we observe that the $\mathcal{O}(\delta^4)$ methods do not do as well as the lower-order methods for these not very smooth functions. This is perhaps a consequence of the reproduction of cubics. Method A does particularly badly. Intuitively, lacking knots at t_1 and t_{n-1}, it cannot be as flexible, or local, as the other methods.

In the last two graphs of Fig. 2, we note that the global methods do better than the strictly local methods for these smoother functions. Method A, the not-a-knot spline, does the best of any of the methods on $W_{4,\infty}$ functions.

2.2.2. Comparison of the first interval error bounds. In this section we compare the error bounds for the first interval only. Thus the first interval error constant $C'_{\alpha,j}$ is defined as the smallest number for which the bound

$$\sup_{[t_0,t_1]} |f(x) - s(x)| \le C'_{\alpha,j} \|f^{(j)}\|_\infty, \quad \forall f \in C^j[t_0, t_n],$$

holds. Clearly

$$C'_{\alpha,j} = \sup_{[t_0,t_1]} K(\alpha; j, x).$$

We first compute these first interval error constants, and then the corresponding relative error constant $e'_{\alpha,j} = C'_{A,j}/C'_{\alpha,j}$. The results from numerical experiments were graphed as in the last section. For the first and second derivative bounds the first interval results were very much like the overall results. These graphs were therefore omitted. The first interval results for third and fourth derivative bounds are more extreme than the overall results and appear in Fig. 3.

2.3. Comparison of operator norms. We also calculated the norm of L, and of the derived projector L', for each of the methods and eight random interval meshes. Bar graphs showing the 1 percentile, mean, and 99 percentile points over 20,000 trials are shown in Fig. 4. We remind the reader that the norm of L cannot be bounded independently of the mesh ratio (see de Boor [2, pp. 209–214] and the references therein); that the norm of L' (at least for Methods A, B, and C) can be bounded in terms of the mesh ratio in the first and last two intervals (see [1]); and that the norm of L'' can be bounded independently of the mesh. Table 2 shows the extreme values of $\|L''\|$ seen over 20,000 random trials with 28 interval meshes.

These norms represent a quantitative measure of the tendency of the various methods to introduce *spurious bumps and wiggles* in the fitted curve. We emphasize

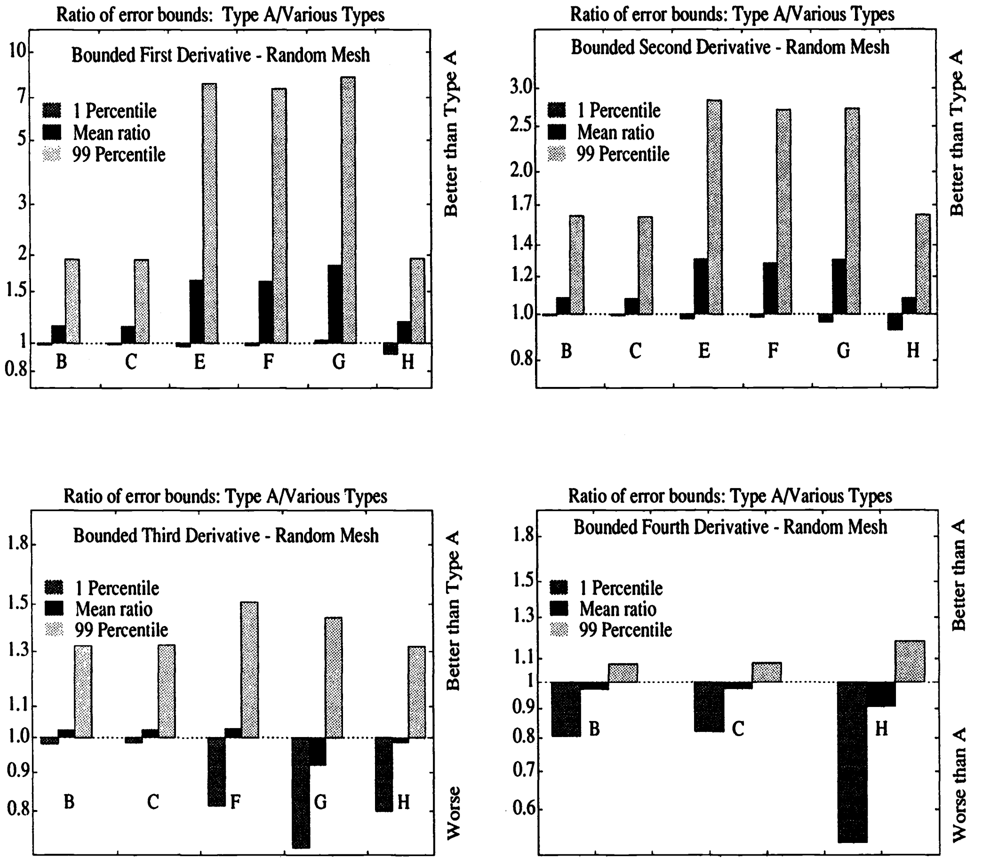


FIG. 2. *Error bounds: Type* A/*Various types; Overall.*

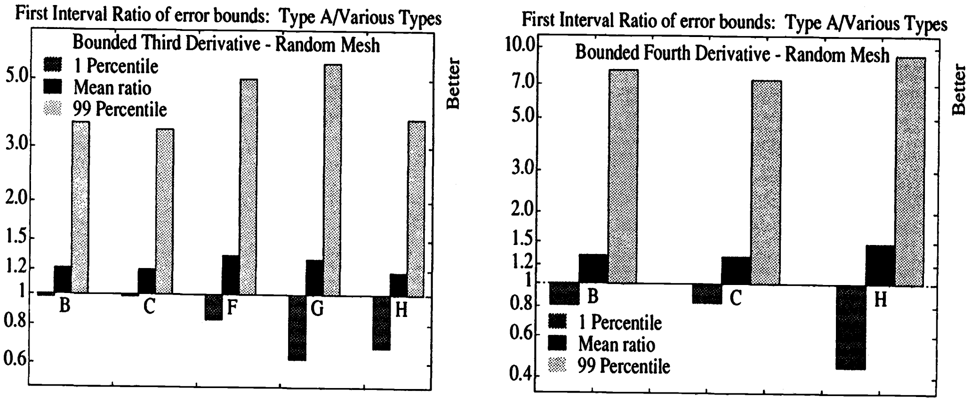


FIG. 3. *Error bounds: Type* A/*Various types; First interval.*

TABLE 2
Norm of second derived projector, random mesh. Maximum value observed over 20,000 *trials.*

Method A	Method B	Method C	Method E
7.8919	4.9699	4.9652	2.6599

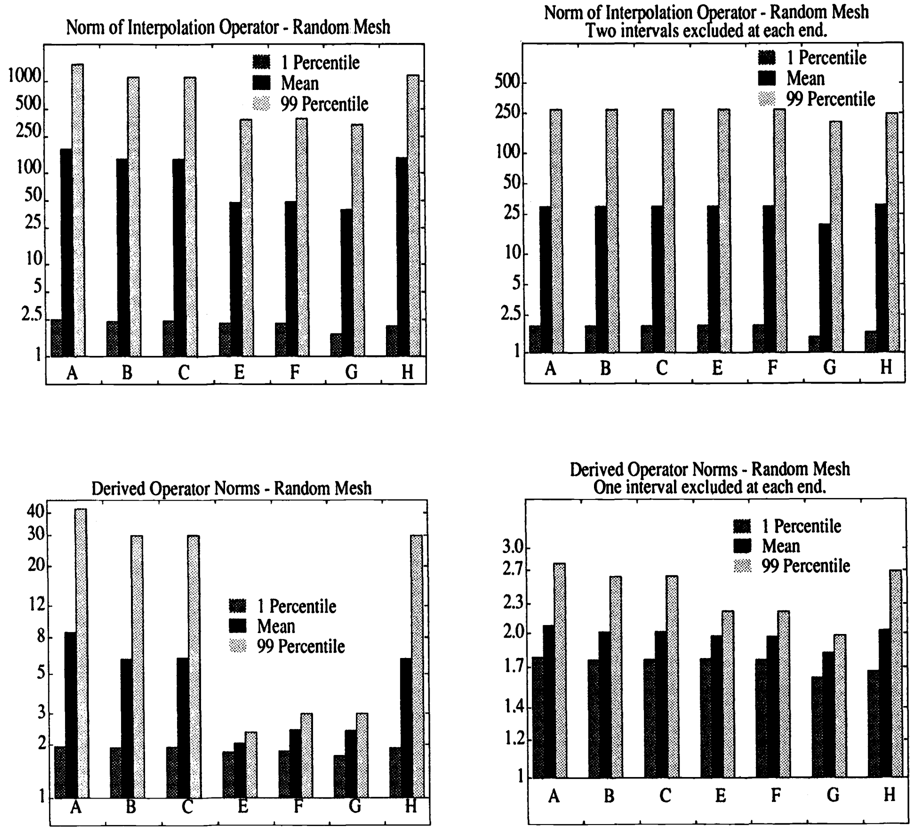


FIG. 4. *Operator norms; Random meshes.*

that, for the C^2 methods, the fitted curves will differ very little in the middle intervals, so that any large difference in operator norms corresponds to differing behavior in the first few or last few intervals.

2.4. Results for a uniform mesh. The graphs in Figs. 5 and 6 show the pointwise error multipliers $K(j, x)$ in the first three intervals of a uniform mesh of 30 intervals for each of the methods and the four smoothness classes. We do not show the plot for the interior subintervals where all the C^2 methods are practically indistinguishable. This is to be expected because of the semilocal nature of cubic splines.

We also calculated the norms of s and the derived projectors s' and s'' for uniform meshes of various sizes, with end intervals included and excluded. The results are shown in Tables 3–8.

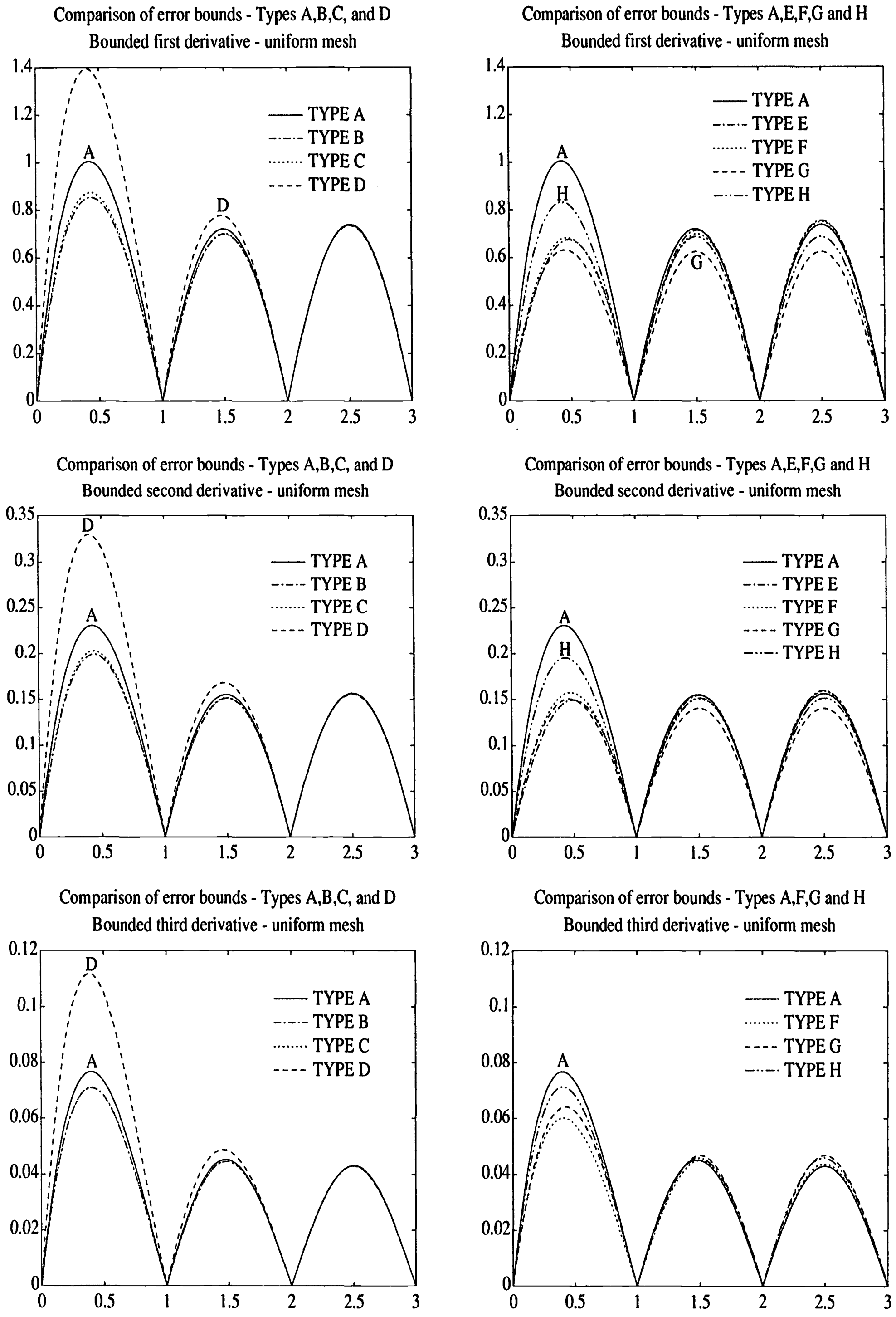


FIG. 5. *Pointwise error multipliers* $K(j, x)$.

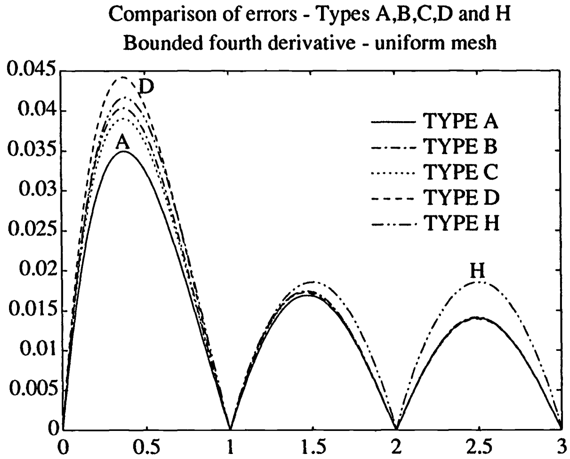


FIG. 6. *Pointwise error multipliers* $K(4,x)$.

TABLE 3
Norm of spline operator, end intervals included.

n	A	B	C	D	E	F	G	H
8	1.97098	1.67836	1.71712	2.72960	1.53579	1.53345	1.25000	1.63113
12	1.97164	1.67843	1.71725	2.73294	1.54808	1.54793	1.25000	1.63113
16	1.97164	1.67843	1.71725	2.73296	1.54897	1.54896	1.25000	1.63113
20	1.97164	1.67843	1.71725	2.73296	1.54903	1.54903	1.25000	1.63113

TABLE 4
Norm of spline operator, end intervals excluded.

n	A	B	C	D	E	F	G	H
8	1.51768	1.52316	1.52243	1.54745	1.53579	1.53345	1.25000	1.38490
12	1.54666	1.54719	1.54712	1.54903	1.54808	1.54793	1.25000	1.38490
16	1.54887	1.54890	1.54890	1.54904	1.54897	1.54896	1.25000	1.38490
20	1.54902	1.54903	1.54903	1.54904	1.54903	1.54903	1.25000	1.38490

TABLE 5
Norm of first derived projector, end intervals included.

n	A	B	C	D	E	F	G	H
8	4.30769	3.33333	3.46392	6.78788	1.73196	2.00000	2.00000	3.33333
12	4.30939	3.33333	3.46410	6.79738	1.73205	2.00000	2.00000	3.33333
16	4.30940	3.33333	3.46410	6.79743	1.73205	2.00000	2.00000	3.33333
20	4.30940	3.33333	3.46410	6.79743	1.73205	2.00000	2.00000	3.33333

TABLE 6

Norm of first derived projector, end intervals excluded.

n	A	B	C	D	E	F	G	H
8	1.73120	1.69759	1.69643	2.27352	1.71428	1.71134	1.50000	1.58333
12	1.73205	1.72958	1.72949	2.27669	1.73077	1.73057	1.50000	1.58333
16	1.73205	1.73187	1.73187	2.27671	1.73196	1.73194	1.50000	1.58333
20	1.73205	1.73204	1.73204	2.27671	1.73204	1.73204	1.50000	1.58333

TABLE 7

Norm of second derived projector, end intervals included.

n	Method A	Method B	Method C	Method D	Method E
8	3.05846	2.31680	2.33333	5.53712	1.99244
12	3.05920	2.31689	2.33333	5.54195	1.99946
16	3.05921	2.31689	2.33333	5.54198	1.99996
20	3.05921	2.31689	2.33333	5.54198	2.00000

TABLE 8

Norm of second derived projector, end intervals excluded.

n	Method A	Method B	Method C	Method D	Method E
8	1.97675	1.98322	1.98235	1.96058	1.99244
12	1.99838	1.99879	1.99874	1.99735	1.99946
16	1.99988	1.99991	1.99990	1.99981	1.99996
20	1.99999	1.99999	1.99999	1.99999	2.00000

The entries in the first of each pair of tables are the usual operator norms,

$$\|L\| = \sup_{f \neq 0} \|Lf\|_{[t_0,t_n]}/\|f\|_{[t_0,t_n]},$$

while for those in the second, the end intervals are excluded in the numerator. Thus,

$$\|L\|' = \sup_{f \neq 0} \|Lf\|_{[t_1,t_{n-1}]}/\|f\|_{[t_0,t_n]}.$$

Our interpretation of these results is as follows. For these uniform meshes the various C^2 methods differ very little when the end intervals are excluded.[1] Thus once again we see that the end conditions make a difference only near the end points. Also, as expected, the operator norms for a particular method hardly change as the number of intervals in the uniform mesh is increased beyond eight. The error curves for $W_{4,\infty}$ functions show the C^2 methods doing better than the strictly local Method H in the interior subintervals.

3. Mathematics underlying the computations.

3.1. Calculation of the optimal error bounds. Fix for the moment the mesh $\mathbf{t}$, the cubic spline interpolant L, and $\xi \in [t_0, t_n] = [a, b]$. Let

$$E(f,\xi) = f(\xi) - L(f,\xi).$$

[1] In fact the end intervals deleted operator norm rapidly approaches the value $(1 + 3\sqrt{3})/4 \approx 1.549038\cdots$ valid for C^2 cubic spline interpolation on an infinite uniform mesh (see Powell [9]).

Suppose that this error functional annihilates π_j for some $0 \le j \le 3$. The application of the Peano kernel theorem (Davis [3], Powell [9]) shows that if $f \in C^{j+1}[a,b]$, then

$$E(f,\xi) = \int_a^b f^{(j+1)}(t)K(t)dt \tag{10}$$

where

$$K(t) = \frac{1}{j!}E_x((x-t)_+^j,\xi) \tag{11}$$

and the notation $E_x((x-t)_+^j,\xi)$ means that the functional $E(\cdot,\xi)$ is applied to $(x-t)_+^j$ considered as a function of x. Because in the case we are considering, K will have only a finite number of sign changes, it follows easily from (10), (11), and smoothing arguments that

$$\sup_{\{f\in C^{j+1}[a,b]:\|f^{(j+1)}\|_\infty\le 1\}} |E(f,\xi)| = \|K\|_1,$$

or the equivalent: that the least value of C for which the relationship

$$|E(f,\xi)| \le C\|f^{(j+1)}\|_\infty$$

holds for all $f \in C^{j+1}[a,b]$ is $C = \|K\|_1$. These conclusions also hold when $C^{j+1}[a,b]$ is replaced by $W_{j+1,\infty}$.

It only remains to discuss how one can calculate K and $\|K\|_1$ numerically. First we let $\{\ell_i\}_{i=0}^n$ be the cardinal splines corresponding to the interpolant L. That is, ℓ_i is the cubic spline interpolant, $L(f)$, when $f(t_k) = \delta_{ik}$. Then for any f, ξ,

$$L(f,\xi) = \sum_{i=0}^{N} f(t_i)\ell_i(\xi)$$

and

$$E(f,\xi) = f(\xi) - \sum_{i=0}^{n} f(t_i)\ell_i(\xi).$$

Substituting from (11) it follows that

$$j!K(t) = (\xi - t)_+^j - \sum_{i=0}^{n}(t_i - t)_+^j\ell_i(\xi). \tag{12}$$

Hence for each $0 \le j \le 3$, $K(t)$ is a spline of degree j with possible knots at ξ and the t_i's, whose coefficients we can easily calculate. $\|K\|_1$ can then be calculated numerically by using a cubic root finder to find the zeroes of K and Simpson's rule to integrate $|K(t)|$ exactly over each subinterval within which it reduces to a cubic polynomial.

3.2. Computation of $\|L\|_\infty$ and $\|L'\|_\infty$. The methods of computing $\|L\|_\infty$ and $\|L'\|_\infty$ are essentially the same, so we will only give the details of the computation of $\|L'\|_\infty$.

Let Lf denote one of the cubic spline interpolants under consideration applied to the function f at the nodes $\mathbf{t} : t_0 < t_1 < \cdots < t_n$. Then assuming the map L is exact for polynomials of degree $j-1$, the derived projector $L^{(j)}$ given by

$$L^{(j)}(D^j f) = D^j(Lf)$$

is well defined. We define, as is usual, the operator norm

$$\|L^{(j)}\|_\infty = \sup_{g \in C[t_0,t_n] : \|g\|_\infty \leq 1} \|L^{(j)}(g)\|.$$

For convenience, denote $(Lf)(x)$ by $s(x)$. Assume L is exact for constants. Then considering the matrix system expressing the first derivatives of s at the knots in terms of the data, we see that the linear map from g to $L'(g)$ may be expressed as the composition of three linear maps

$$L' = E \circ P \circ S, \tag{13}$$

where $S : C[t_0, t_n] \rightarrow \mathbb{R}$ is the map taking $g = f'$ to the n-vector with ith component

$$f[t_i, t_{i+1}] = \frac{1}{t_{i+1} - t_i} \int_{t_i}^{t_{i+1}} g(t)dt.$$

$P : \mathbb{R}^n \rightarrow \mathbb{R}^{2n+1}$ is the map taking the vector $\boldsymbol{\alpha}$ to the $2n+1$ vector $\boldsymbol{\beta}$ with ith component

$$\beta_i = \begin{cases} \alpha_i, & i < n, \\ s'(t_{i-n}), & i \geq n, \end{cases}$$

and finally,

$$E : \mathbb{R}^{2n+1} \rightarrow C[t_0, t_n]$$

is the map taking $\boldsymbol{\beta}$ to the piecewise quadratic s', with specified end point and average values on each subinterval.

Because L' receives only the information about g given to it by the map S, we can rewrite (13) in a form more useful for computation. More precisely, as g ranges over

$$\{g \in C[t_0, t_n] : \|g\|_\infty \leq 1\},$$

$S(g)$ ranges over a set in $\mathbb{R}^n$ whose closure is the closed ℓ^∞ unit ball in $\mathbb{R}^n$. Hence (13) implies

$$\|L'\|_\infty = \sup_{\Omega = \{\alpha : \|\alpha\|_\infty \leq 1\}} \|(E \circ P)(\boldsymbol{\alpha})\|_\infty. \tag{14}$$

Finally, since $E \circ P(\cdot)$ is a linear function and $\|\cdot\|_\infty$ a convex function, the function whose supremum is taken on the right-hand side of (14) is convex. Hence it achieves its supremum at an extreme point of the closed, bounded, convex set Ω. This shows that for each fixed mesh $\mathbf{t}$, $\|L'\|_\infty$ can be found by a process of exhaustive search over the 2^n extreme points of Ω, computing for each extreme point the corresponding value of $\|s'\|_\infty$. Indeed, since the sign of the first slope can be fixed, without loss of generality, the search can be restricted to 2^{n-1} extreme points. We note that it is known [1] that if L reproduces cubics, then $\|L'\|_\infty$ cannot be bounded independently of $\mathbf{t}$.

The computation of $\|L\|_\infty$ is analogous to that of $\|L'\|_\infty$. In this case Ω is the ℓ^∞ unit ball in $\mathbb{R}^{n+1}$, extreme points of which correspond to function values $[f(t_0),\cdots,f(t_n)]^T$ to be interpolated. $\|L\|_\infty$ is computed by computing $\|s\|_\infty$ for 2^n of these 2^{n+1} extreme points.

3.3. Computation of $\|L''\|_\infty$. When L is exact for linear polynomials the map L'', from f'' to $L''(f'')$, is a well-defined linear map. It can be written as the composition of three linear maps

$$L'' = E \circ P \circ S.$$

Here S maps f'' to the $n-1$ vector of second divided differences $\mathbf{b}$ with

$$b_i = f[t_i, t_{i+1}, t_{i+2}] = \frac{1}{t_{i+2}-t_i}\int_{t_i}^{t_{i+2}} N_{i,2}(x) f''(x)dx.$$

P maps $\mathbf{b}$ to the vector of second derivatives $\boldsymbol{\sigma} = (s''(t_i))_{i=0}^n$ and therefore has the form $\boldsymbol{\sigma} = D\mathbf{b}$ for some $(n+1)\times(n-1)$ matrix D. Finally, E maps the vector of second derivative values $\boldsymbol{\sigma}$ to the piecewise linear interpolant s''.[2]

Since s'' is piecewise linear, $\|s''\|_\infty$ corresponds to a value of s'' at one of the knots. Hence

$$\|L''\|_\infty = \max_{0\le i\le n} \sup_{\{f: f\in L_\infty^2 \text{ and } \|f''\|_\infty \le 1\}} \left| \sum_j d_{ij} f[t_j, t_{j+1}, t_{j+2}] \right|.$$

Since values of f'' in one interval can affect two of the second differences b_i, we cannot simply choose the differences in a bang-bang way as we did in computing $\|L\|$ and $\|L'\|$. However, fixing i and writing e_j for $d_{i,j}$,

$$\begin{aligned}
s''(f;t_i) &= \sum_{j=0}^{n-2} e_j f[t_j, t_{j+1}, t_{j+2}] \\
&= \sum_{j=0}^{n-2} \frac{e_j}{t_{j+2}-t_j}\int_{t_j}^{t_{j+2}} N_{j,2}(x) f''(x)dx \\
&= \sum_{j=0}^{n-2} e_j \left[\left(\frac{t_{j+1}-t_j}{t_{j+2}-t_j}\right)\int_0^1 x f''(t_j + (t_{j+1}-t_j)x)dx \right. \\
&\qquad \left. + \left(\frac{t_{j+2}-t_{j+1}}{t_{j+2}-t_j}\right)\int_0^1 (1-x) f''(t_{j+1} + (t_{j+2}-t_{j+1})x)dx \right] \\
&= \omega_{0,2}\lambda_{0,2} + \sum_{j=1}^{n-2}(\omega_{j,1}\lambda_{j,1} + \omega_{j,2}\lambda_{j,2}) + \omega_{n-1,1}\lambda_{n-1,1},
\end{aligned}$$

[2] For the C^2 methods considered here the system for σ takes the form $A\sigma = Bb$, with A^{-1} and B bounded in the infinity norm (see, for example, [1, pp. 905–906] for explicit formulations). Hence $D = A^{-1}B$ is bounded and the relationship between second divided differences and second derivatives implies immediately that $\|L''\|_\infty \le \frac{1}{2}\|D\|_\infty < \infty$. This simple analysis provides, for example, the (not tight) upper bounds $\|L''_C\|_\infty \le 8$ and $\|L''_E\|_\infty \le 3$.

where

$$\omega_{j,2} = e_j \frac{(t_{j+1} - t_j)}{(t_{j+2} - t_j)}, \qquad \omega_{j,1} = e_{j-1} \frac{(t_{j+1} - t_j)}{(t_{j+1} - t_{j-1})},$$

and

$$\boldsymbol{\lambda}_j = \begin{bmatrix} \int_0^1 (1-x) f''(t_j + (t_{j+1} - t_j)x)dx \\ \int_0^1 x f''(t_j + (t_{j+1} - t_j)x)dx \end{bmatrix} = \begin{bmatrix} \int_0^1 (1-x) g_j(x)dx \\ \int_0^1 x g_j(x)dx \end{bmatrix}.$$

Hence defining Λ as the closed, convex set in $\mathbb{R}^2$,

$$\Lambda = \left\{ \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} \int_0^1 (1-x) g(x)dx \\ \int_0^1 x g(x)dx \end{bmatrix} : \|g\|_\infty \le 1 \right\},$$

the maximum of $s''(t_i)$ over functions f with $\|f''\| \le 1$ equals

$$\max_{\lambda_j \in \Lambda} \omega_{0,2}\lambda_{0,2} + \left(\sum_{j=1}^{n-2} (\omega_{j,1}\lambda_{j,1} + \omega_{j,2}\lambda_{j,2}) \right) + \omega_{n-1,1}\lambda_{n-1,1}. \tag{15}$$

Here the numbers $\omega_{j,k}$ depend only on the mesh $\mathbf{t}$ and the matrix D. Hence the problem (15) separates into the sum of n two-dimensional subproblems

$$\max_{\lambda \in \Lambda} \omega_{0,2}\lambda_2 + \left(\sum_{j=1}^{n-2} \max_{\lambda \in \Lambda} (\omega_{j,1}\lambda_1 + \omega_{j,2}\lambda_2) \right) + \max_{\lambda \in \Lambda} \omega_{n-1,1}\lambda_1 \ .$$

Since Λ is a closed, bounded, convex set the maxima in the subproblems occur at extreme points of Λ. Indeed the following lemma shows that the two-dimensional subproblems are trivial to solve.

LEMMA 3.1. *The set Λ defined above is the closed, bounded, convex set with boundary curves*

$$\gamma_1 = \left\{ \lambda = \begin{bmatrix} \alpha^2 - \frac{1}{2} \\ 2\alpha - \frac{1}{2} - \alpha^2 \end{bmatrix} : 0 < \alpha < 1 \right\}$$

and

$$\gamma_2 = \left\{ \lambda = \begin{bmatrix} 2\alpha - \frac{1}{2} - \alpha^2 \\ \alpha^2 - \frac{1}{2} \end{bmatrix} : 1 > \alpha > 0 \right\}$$

together with the points $A = \left(-\frac{1}{2}, -\frac{1}{2}\right)$ and $B = \left(\frac{1}{2}, \frac{1}{2}\right)$.

Proof. The proof is omitted.

Having defined Λ, we see that for each nonzero ω, $\omega^T\lambda$ is maximized at a unique point on the boundary of Λ. One can easily calculate that point. For example, when $\omega_1 > 0$, $\omega_2 < 0$, the maximum corresponds to the unique point in the interior of the lower boundary where ω is perpendicular to the tangent. Hence this maximum occurs at the point with parameter $\alpha = \omega_1/(\omega_1 - \omega_2)$. Other sign patterns for ω are dealt with similarly. This leads to a procedure for solving the two-dimensional subproblems and hence, with extra code, for computing $\|L''\|$.

REFERENCES

[1] R. K. Beatson, *On the convergence of some cubic spline interpolation schemes*, SIAM J. Numer. Anal., 23 (1986), pp. 903–912.

[2] C. de Boor, *A Practical Guide to Splines*, Springer-Verlag, New York, 1978.

[3] P. J. Davis, *Interpolation and Approximation*, Blaisdell, New York, 1963.

[4] C. A. Hall, *On error bounds for cubic spline interpolation*, J. Approx. Theory, 1 (1968), pp. 209–218.

[5] C. A . Hall and W. W. Meyer, *Optimal error bounds for cubic spline interpolation*, J. Approx. Theory, 16 (1976), pp. 105–122.

[6] D. Kershaw, *Inequalities on the elements of a certain tridiagonal matrix*, Math. Comp., 24 (1970), pp. 155–158.

[7] ———, *Two interpolatory cubic splines*, J. Inst. Math. Appl., 11 (1973), pp. 329–333.

[8] M. J. Marsden, *Cubic spline interpolation of continuous functions*, J. Approx. Theory, 10 (1974), pp. 103–111.

[9] M. J. D. Powell, *Approximation theory and methods*, Cambridge University Press, Cambridge, 1981.

[10] I. J. Schoenberg, *The elementary cases of Landau's problem of inequalities between derivatives*, Amer. Math. Monthly, 80 (1973), pp. 121–158.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 4, pp. 1025–1038, July 1992

011

INTEGRATING PRODUCTS OF B-SPLINES*

A. H. VERMEULEN†, R. H. BARTELS†, AND G. R. HEPPLER†

Abstract. This paper outlines several ways to evaluate the integral of the product of two B-spline functions, followed by a detailed description of an algorithm that is based on integration by parts. The algorithm reduces the integral to a sum of evaluations of a higher-order spline. This reduction involves differentiating one spline by differencing its coefficients, and integrating the other by summing its coefficients.

Key words. B-splines, integration, inner product, finite element method

AMS(MOS) subject classifications. 65D07, 65D30, 65N30

1. Introduction.

In this paper we consider the evaluation of integrals of the forms:

$$\int_a^b \left(\sum_i E_i B_{i,k,x}(t)\right)\left(\sum_j F_j B_{j,l,y}(t)\right) dt, \tag{1}$$

$$\int_a^b f(t)\left(\sum_i E_i B_{i,k,x}(t)\right) dt, \tag{2}$$

where $B_{i,k,x}$ is the ith B-spline of order k defined over the knots $x_i, x_{i+1}, \cdots, x_{i+k}$. We will consider B-splines normalized so that their integral is one. The splines may be of different orders and defined on different knot sequences x and y. Often the limits of integration will be the entire real line, $-\infty$ to $+\infty$. Note that (1) is a special case of (2) where $f(t)$ is a spline.

Integrals of these forms arise in applications such as the finite element method [1] and least squares function fitting [2], [3], [4] when B-splines are used as basis functions. In some problems, the function $f(t)$ in (2) may be nonpolynomial; for example, it may be a sinusoid [5]. It will be seen that the method we propose can be used to integrate such functions, provided that a sufficient number of antiderivatives of $f(t)$ are known.

There are five different methods for calculating (1) that will be considered:

1. Use Gauss quadrature on each interval.
2. Convert the integral to a linear combination of integrals of products of B-splines and provide a recurrence for integrating the product of a pair of B-splines.
3. Convert the sums of B-splines to piecewise Bézier format and integrate segment by segment using the properties of the Bernstein polynomials.
4. Express the product of a pair of B-splines as a linear combination of B-splines. Use this to reformulate the integrand as a linear combination of B-splines, and integrate term by term.
5. Integrate by parts.

Of these five, only methods 1 and 5 are suitable for calculating (2). The first four methods will be touched on and the last will be discussed at length.

* Received by the editors May 16, 1991; accepted for publication (in revised form) August 14, 1991.

† Department of Systems Design Engineering, University of Waterloo, Waterloo, Ontario N2L3G1, Canada (rhbartel@uwaterloo.ca, ahvermen@uwaterloo.ca, and heppler@dial.uwaterloo.ca).

2. Gauss quadrature. The integral (1) can be broken into a sum of integrals, where each integrand is a polynomial:

$$\int_a^b \left(\sum_i E_i B_{i,k,x}(t)\right)\left(\sum_j F_j B_{j,l,y}(t)\right) dt$$
$$= \sum_r \int_{t_r}^{t_{r+1}} \left(\sum_i E_i B_{i,k,x}(t)\right)\left(\sum_j F_j B_{j,l,y}(t)\right) dt,$$

where $t_1, \cdots, t_n$ is the union of the breakpoints of the two splines. Each integrand is polynomial, therefore the integrals can be computed exactly using Gauss quadrature:

$$\int_{t_r}^{t_{r+1}} e(t)f(t)dt = \sum_s H_s e(\xi_s)f(\xi_s). \tag{3}$$

The numbers ξ_s and H_s are the Gauss points and weights. Gauss quadrature can be used to approximate integrals of form (2), for general $f(t)$.

A different application of Gauss quadrature is to evaluate integrals of the form (2) by moving the summation out of the integral and integrating each term using Gauss quadrature, treating the B-spline as a weight function. If $f(t)$ is a polynomial,

$$\int_{-\infty}^{\infty} B_{i,k,x}(t)f(t)dt = \sum_s H_s f(\xi_s). \tag{4}$$

The numbers ξ_s and H_s are Gauss points and weights for the particular weighting function $B_{i,k,x}(t)$. Values for uniformly spaced B-splines are given in [6]. The Gauss points and weights must be recalculated for each different B-spline; if the basis is uniform this is not difficult, but if the basis is arbitrary then it may present a problem. This method is only capable of calculating integrals of form (1) approximately.

3. Integrating products of B-splines. The summations in (1) can be moved outside the integral to yield a linear combination of terms of the form:

$$\int_a^b B_{i,k,x}(t)B_{j,l,y}(t)dt. \tag{5}$$

In the case where integration is over the entire real line this quantity can be calculated using a recurrence [7]. If the limits are not $-\infty$ and $+\infty$ the procedure can still be used as follows. Insert enough knots [8], [9], [10] at a and b into the splines in the integrand so that no B-spline's support crosses the integration limits. Now consider only the B-splines in the integration region and integrate these over the entire real line.

The recurrence described in [7] to evaluate (5) proceeds as follows. Define the number $T_{i,j}^{k,l}$ as

$$T_{i,j}^{k,l} := (-1)^k[x_i, x_{i+1}, \cdots, x_{i+k} : t][y_j, y_{j+1}, \cdots, y_{j+l} : s](s-t)_+^{k+l+1}, \tag{6}$$

where $[x_i, x_{i+1}, \cdots, x_{i+k} : t]$ is the divided difference operator with respect to the sequence $x_i, \cdots, x_{i+k}$ acting on the variable t, and similarly for $[y_j, y_{j+1}, \cdots, y_{j+l} : s]$. It can be shown that

$$\int_{-\infty}^{\infty} B_{i,k,x}(t)B_{j,l,y}(t)dt = \frac{k!l!}{(k+l-1)!}T_{i,j}^{k,l}. \tag{7}$$

The quantities $T_{i,j}^{k,l}$ can be calculated using the definition of the divided difference and equation (6); however, this can lead to loss of significance if the knot spacing is uneven. A stable method of doing the calculation is to use the following recurrence. The recurrence begins by noting, from (6) and the divided difference definition of a B-spline, that for terms where one of $y_j = y_{j+l}$ or $x_i = x_{i+k}$ holds, $T_{i,j}^{k,l}$ is a scalar multiple of the value of a B-spline:

$$T_{i,j}^{k,l} = \begin{cases} \dfrac{(k+l-1)!}{k!l!} B_{i,k,x}(y_j), & y_j = y_{j+l}, \quad l \geq 0, \\ \dfrac{(k+l-1)!}{k!l!} B_{j,l,y}(x_i), & x_i = x_{i+k}, \quad k \geq 0. \end{cases} \tag{8}$$

Terms with higher numbers in the top indices can be calculated from terms of lower degree. The recurrence is

(9)

$$T_{i,j}^{k,l} = \begin{cases} \dfrac{(x_{i+k} - y_j)T_{i,j}^{k,l-1} + (y_{j+l} - x_{i+k})T_{i,j+1}^{k,l-1}}{y_{j+l} - y_j} + T_{i,j}^{k-1,l}, & x_{i+k} \leq y_{j+l}, \; y_j \leq y_{j+l} \\ \dfrac{(x_i - y_j)T_{i,j}^{k,l-1} + (y_{j+l} - x_i)T_{i,j+1}^{k,l-1}}{y_{j+l} - y_j} + T_{i+1,j}^{k-1,l}, & y_j \leq x_i, \qquad y_j \leq y_{j+l} \\ \dfrac{(y_j - x_i)T_{i,j}^{k-1,l} + (x_{i+k} - y_j)T_{i+1,j}^{k-1,l}}{x_{i+k} - x_i} + T_{i,j+1}^{k,l-1}, & x_i \leq y_j, \qquad x_i \leq x_{i+k} \\ \dfrac{(y_{j+l} - x_i)T_{i,j}^{k-1,l} + (x_{i+k} - y_{j+l})T_{i+1,j}^{k-1,l}}{x_{i+k} - x_i} + T_{i,j}^{k,l-1}, & y_{j+l} \leq x_{i+k}, \; x_i \leq x_{i+k}. \end{cases}$$

Thus the steps in the evaluation of (1) by this method are:

1. If necessary, insert enough knots into the two splines at the integration limits a and b, so that the support of no basis function extends past a or b.
2. Evaluate each of the splines $B_{i,k,x}$ at the points y_j and evaluate each of the splines $B_{j,l,y}$ at the points x_i.
3. Calculate the values $T_{i,j}^{k,l}$ recursively. The values obtained in step 2 provide an end to the recursion.
4. Evaluate the integral (1) by moving the summations and coefficients outside of the integral, replacing the integrals with scaled versions of $T_{i,j}^{k,l}$ according to (7), and summing over all pairs of B-splines.

4. Converting to Bézier form and integrating. The B-splines in the integrand of (1) can be converted to piecewise Bézier format and the result can be integrated segment by segment. On each segment, the B-spline combinations can be represented in Bézier form:

$$\sum_i E_i B_{i,k,x} = \sum_i G_i P_{i,k}, \tag{10}$$

$$\sum_j F_j B_{j,l,y} = \sum_j H_j P_{j,l} \,. \tag{11}$$

$P_{i,k}$ is the ith Bernstein polynomial of degree $k-1$ on the segment. The Bézier coefficients G_i and H_j can be calculated via knot insertion [8], [9], blossoming [11], or the tetrahedral algorithm of Sablonnière [12]. On each segment, the integral (1) can be expressed as a linear combination of inner products of Bernstein polynomials:

$$\int_a^b \left(\sum_i E_i B_{i,k,x}(t)\right)\left(\sum_j F_j B_{j,l,y}(t)\right) dt = \sum_i \sum_j G_i H_j \int_a^b P_{i,k}(t) P_{j,l}(t) dt. \tag{12}$$

The integrals can be evaluated using the formulae for products and integrals of Bernstein polynomials given by Farin [13]. For a segment of length L that lies in the interval (a, b), the integral evaluates to

$$\int_a^b P_{i+k}(t) P_{j,l}(t) dt = \left[\frac{(k+l-2)!i!j!(k-i-1)!(l-j-1)!}{(i+j)!(k+l-i-j-2)!(k-1)!(l-1)!(k+l)}\right] L. \tag{13}$$

5. Explicit multiplication of the integrand. The product of B-splines can be expressed as a linear combination of B-splines [14]:

$$\left(\sum_i E_i B_{i,k,x}(t)\right)\left(\sum_j F_j B_{j,l,y}(t)\right) = \sum_h G_h B_{h,p,z}(t). \tag{14}$$

The order of the product spline is $p = k + l - 1$; the knot vector z must contain sufficient knots to represent the product spline. Since the order of the product is higher than the order of the factors, yet the continuity is the same, the multiplicity of knots in the product spline is generally much higher than in the original splines. This means that there will, in general, be many more splines in the product than in either of the factors.

The procedure for constructing the knot vector z with the minimum possible number of knots will now be described. Begin by setting a single knot z_i at each point for which $z_i = x_\alpha$, a knot in x, or $z_i = y_\beta$, a knot in y. Assign multiplicity m_i to z_i as follows:

$$m_i = \begin{cases} \max(k + m_\beta - 1, l + m_\alpha - 1), & m_\alpha > 0 \quad \text{and} \quad m_\beta > 0, \\ k + m_\beta - 1, & m_\alpha = 0 \quad \text{and} \quad m_\beta > 0, \\ l + m_\alpha - 1, & m_\alpha > 0 \quad \text{and} \quad m_\beta = 0, \end{cases} \tag{15}$$

where m_α is the multiplicity of x_α and m_β is the multiplicity of y_β. Note that, if there is no knot in x at z_i or no knot in y at z_i, then either m_α or m_β will be zero.

The coefficients of the product spline are related to the coefficients of the factors via the following linear relationship [14]:

$$G_h = \sum_{i,j} \Gamma_{i,j,k,l}(h) E_i F_j. \tag{16}$$

The coefficients Γ can be calculated using the following recurrence:

$$\begin{aligned} \Gamma_{i,j,k,l}(h) = \frac{z_{h+p} - z_h}{p(z_{h+p-1} - z_h)} \Bigg(& \frac{k}{x_{i+k} - x_i} [(z_{h+p-1} - x_i)\Gamma_{i,j,k-1,l}(h) \\ & + (x_{i+k} - z_{h+p-1})\Gamma_{i+1,j,k-1,l}] \\ & + \frac{l}{y_{j+l} - y_j} [(z_{h+p-1} - y_j)\Gamma_{i,j,k,l-1}(h) \\ & + (y_{j+l} - z_{h+p-1})\Gamma_{i,j+1,k,l-1}] \Bigg). \end{aligned}$$

The recursion is initiated by specifying the values of Γ for $k = l = 1$:

(17)
$$\Gamma_{i,j,1,1}(h) = \begin{cases} \dfrac{(x_{i+1} - x_i)(y_{j+1} - y_j)}{z_{h+1} - z_h}, & x_i \leq z_h < x_{i+1} \quad \text{and} \quad y_j \leq z_h < y_{j+1}, \\ 0, & \text{otherwise.} \end{cases}$$

We can rewrite the integral (1) as:

$$\sum_{i,j,h} \Gamma_{i,j,k,l}(h) E_i F_j \int_a^b B_{h,p,z}(t)dt. \tag{18}$$

The integral of a B-spline over its entire support is one; we use this for cases where the entire B-spline support is within the limits of integration. This can be arranged by inserting knots at a and b as described previously. If the entire B-spline is not within the region of integration, the recurrence given to calculate definite integrals of B-splines described in [7] can be used, or the integrand can be converted to Bézier form and integrated as described previously.

6. Integration by parts. The final method of evaluating (1) is integration by parts.

$$\int_a^b e(t)f(t)dt = f^{(-1)}(b)e(b) - f^{(-1)}(a)e(a) - \int_a^b e^{(1)}(t)f^{(-1)}(t)dt, \tag{19}$$

where $f^{(-1)}(t)$ is the antiderivative of $f(t)$ and $e^{(1)}(t)$ is the derivative of $e(t)$. Integration by parts can be applied repeatedly to the integrand. After n applications:

$$\int_a^b e(t)f(t)dt = (-1)^n \int_a^b e^{(n)} f^{(-n)}(t)dt + \text{constant terms.} \tag{20}$$

If $e(t)$ and $f(t)$ are splines, then the degree of $e(t)$ will be lowered while the degree of $f(t)$ is raised. If we apply this enough times, $e(t)$ will be reduced to a simple enough form that its product with $f(t)$ can be integrated directly.

To apply this principle we need to recall some results relating the integration and differentiation of B-splines to Dirac delta functions.

6.1. Differentiating B-splines. The following two-term derivative formula for B-splines is well known:

$$\frac{d}{dt} B_{i,k}(t) = \frac{k}{u_{i+k} - u_i} \left[B_{i,k-1}(t) - B_{i+1,k-1}(t) \right]. \tag{21}$$

We will only concern ourselves with the case where $u_{i+k} > u_i$; thus the right side always has a nonzero denominator. When $u_{i+k-1} > u_i$ and $u_{i+k} > u_{i+1}$ the B-splines on the right side are defined in the normal way. Under the assumption that $u_{i+k} > u_i$, the only special cases that can arise are: (a) $u_{i+k-1} = u_i$ but $u_{i+k} > u_{i+1}$, for which $B_{i,k-1}(t)$ has zero support, and (b) $u_{i+k-1} > u_i$ but $u_{i+k} = u_{i+1}$, for which $B_{i+1,k-1}(t)$ has zero support.

Consider the case where

$$u_i = u_{i+1} = \cdots = u_{i+k-1} < u_{i+k}. \tag{22}$$

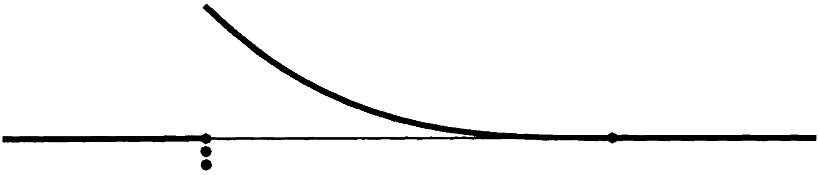


FIG. 1. *A cubic* B*-spline with a discontinuity.*

This situation is depicted for a cubic B-spline in Fig. 1. The spline $B_{i,k}$ is discontinuous at the point u_i. Specifically, with the convention that segment intervals are closed on the left,

$$B_{i,k}(u)\big|_{u<u_i} = 0, \qquad B_{i,k}(u)\big|_{u=u_i} = \frac{k}{u_{i+k} - u_i}. \tag{23}$$

Since the function is discontinuous at u_i, the derivative does not exist in the normal sense. One option is to consider only right-sided derivatives [15], [16]. With this approach splines with zero support are taken as zero. As pointed out in [16, p. 88], the problem with this option is that the Fundamental Theorem of Calculus does not hold. To see this, note that if the first Fundamental Theorem held, we would expect that

$$B_{i,k}(u) = \int_{-\infty}^{u} \frac{d}{dt} B_{i,k}(t)dt. \tag{24}$$

Substituting from (21) into the above integral, and using the definition that zero-support B-splines are to be interpreted as zero, yields

$$B_{i,k}(u) = \int_{-\infty}^{u} \frac{k}{u_{i+k} - u_i}\,[0 - B_{i+1,k-1}(t)]\,dt. \tag{25}$$

Yet at the point $u = u_{i+k}$ the left side of the equation evaluates to

$$\text{LS} = B_{i,k}(u_{i+k}) = 0, \tag{26}$$

while the right side yields (since the integral of a B-spline over its support is one)

$$\text{RS} = \int_{-\infty}^{u_{i+k}} \frac{k}{u_{i+k} - u_i}\,[0 - B_{i+1,k-1}(t)]\,dt = -\frac{k}{u_{i+k} - u_i} \neq 0. \tag{27}$$

This means that, if B-splines with zero support are taken as zero, we cannot use integration by parts to solve (1), because integration by parts is based on the Fundamental Theorem. Therefore, we will look at one of the alternative ways that B-splines with zero support have been defined.

We begin by recalling a result attributed to Curry and Schoenberg [7] that a kth order B-spline on the knots $u_i, \cdots, u_{i+k}$ can be defined as the function $B_{i,k}$, which satisfies

$$\int_{-\infty}^{\infty} B_{i,k}(t) g^{(k)}(t) dt = k! \left[u_i, u_{i+1}, \cdots, u_{i+k} : t \right] g(t) \tag{28}$$

for any function $g(t)$ with k continuous derivatives. Consider $B_{i,k}$ where $u_i = \cdots = u_{i+k} = \hat{u}$. In this case the right side of (28) becomes the kth derivative of $g(t)$ at $\hat{u}$. To be consistent, the left side must yield

$$\int_{-\infty}^{\infty} B_{i,k}(t) g^{(k)}(t) dt = g^{(k)}(\hat{u}). \tag{29}$$

Distribution theory [17], [18] provides an entity that behaves precisely as $B_{i,k}(t)$ must in this circumstance: the Dirac delta function $\delta(t-\hat{u})$. This function has the property that

$$\int_a^b \delta(t-\hat{u}) f(t) dt = \begin{cases} f(\hat{u}), & a < \hat{u} < b, \\ 0, & \hat{u} < a \text{ or } \hat{u} > b, \\ \text{undefined}, & \hat{u} = a \text{ or } \hat{u} = b \end{cases} \tag{30}$$

for any function $f(t)$ integrable on (a, b). Accordingly,

$$B_{i,k}(t) = \delta(t - \hat{u}), \quad u_i = u_{i+1} = \cdots = u_{i+k} = \hat{u}, \tag{31}$$

which is consistent with the Fundamental Theorem. Consider again the spline $B_{i,k}$ with knots as in (22). Using this definition, the two-term differentiation formula now yields

$$\frac{d}{dt} B_{i,k}(t) = \frac{k}{u_{i+k} - u_i} \left[\delta(t - \hat{u}) - B_{i+1,k-1}(t) \right]. \tag{32}$$

The revised version of (25) is

$$B_{i,k}(u) = \int_{-\infty}^{u} \frac{k}{u_{i+k} - u_i} \left[\delta(t - u_i) - B_{i+1,k-1}(t) \right]. \tag{33}$$

For $u < u_i$ or $u > u_{i+k}$ both left and right sides are zero. For $u_i < u < u_{i+k}$ we first note that the left side of (33) has polynomial form

$$\text{LS} = B_{i,k}(u) = \frac{k}{(u_{i+k} - u_i)} (u_{i+k} - u)^{k-1}. \tag{34}$$

The right side yields:

$$\text{RS} = \frac{k}{u_{i+k} - u_i} \left[\int_{\infty}^{u} \delta(t - u_i) dt - \int_{u_i}^{u} \frac{k-1}{(u_{i+k} - u_i)^{k-1}} (u_{i+k} - t)^{k-2} dt \right]. \tag{35}$$

Since $u > u_i$ the first integral has the value one. Expanding the second integral and simplifying yields

$$\text{RS} = \frac{k}{u_{i+k} - u_i} (u_{i+k} - u)^{k-1}. \tag{36}$$

Hence, the left side of (33) is equal to the right side.

The definition (31) is reasonable from another point of view as well. B-splines have been normalized to integrate to one:

$$\int_{-\infty}^{\infty} B_{i,k}(t)dt = 1. \tag{37}$$

This, too, is consistent with (30) for $f(t) = 1$.

6.1.1. Derivatives of B-spline combinations. The derivative of a linear combination of B-splines is a linear combination of B-splines of next lower order:

$$\frac{d}{dt}\sum_{i=0}^{m} V_i B_{i,k,u}(t) = \sum_{i=0}^{m+1} V_i^{(1)} B_{i,k-1,u}(t), \tag{38}$$

where the coefficients can be obtained by substituting the two-term differentiation formula into the left-hand side and shifting the summation:

$$V_i^{(1)} = \frac{k}{u_{i+k}-u_i}V_i - \frac{k}{u_{i+k-1}-u_{i-1}}V_{i-1}. \tag{39}$$

V_{-1} and V_{m+1} are defined to be zero.

6.2. Integrating B-splines. To obtain a formula for the indefinite integral of a B-spline combination, we integrate (38). This leads to this description of the indefinite integral of a B-spline combination

$$\int \sum_{i=0}^{m} V_i B_{i,k}(t) = \sum_{i=0}^{m} V_i^{(-1)} B_{i,k+1}(t), \qquad -\infty \le t < u_{m+1}, \tag{40}$$

where the coefficients are obtained by inverting (39):

$$V_i^{(-1)} = \frac{u_{i+k+1}-u_i}{k+1}\left[\frac{k+1}{u_{i+k}-u_{i-1}}V_{i-1}^{(-1)} + V_i\right]. \tag{41}$$

$V_{-1}^{(-1)}$ is defined to be zero.

The integral spline requires the existence of a new knot, u_{m+k+1}. The value of this new knot is arbitrary subject to $u_{m+k+1} \ge u_{m+k}$. Adopting the convention that knots with indices past the end of the given knot vector are equal to the last given knot can simplify implementation.

Also note that the integral spline in (40) is only valid on the interval $[-\infty, u_{m+1})$. This condition is necessary because the integral of a B-spline combination will, in general, have unbounded support. Such a function is not representable as a linear combination of a finite number of B-splines, but the portion to the left of u_{m+1} is representable in this way; hence the condition. An alternative method is to define B-spline–like basis functions that have unbounded support on one side. Suitable basis functions for this alternative are described in Barry and Goldman [19] and in de Boor, Lyche, and Schumaker [7].

6.3. The integration by parts algorithm. In this section the integration by parts algorithm will be described.

Begin by defining two splines

$$e(t) = \sum_{i=0}^{m} E_i B_{i,k,x}(t), \tag{42}$$

$$f(t) = \sum_{j=0}^{n} F_j B_{j,l,y}(t). \tag{43}$$

The integral we wish to compute is

$$\int_a^b e(t) f(t) dt. \tag{44}$$

Informally, the approach will be to use integration by parts to reduce the order of $e(t)$ while increasing the order of $f(t)$. This will reduce the support length of e's B-splines. When the support length of one of e's B-splines reaches zero, the B-spline becomes a Dirac delta function and thus the part of the integral on this basis function reduces to an evaluation of f. Eventually, all of e's B-splines will have zero support and the integral will be reduced to a sum of evaluations.

A detailed description of the algorithm is presented below. Step 0 serves to bring the integrand into a canonical form; steps 1–4 constitute the substance of the algorithm.

0. If the lower limit of integration, a, lies exactly on a knot, shift it an infinitesimal amount to the right. Similarly, if b lies exactly on a knot, shift it an infinitesimal amount to the left. This will not affect the value of the integral since the integral of a product of splines of order 1 or more varies continuously as the limits of integration are changed. The shifting of the limits is necessary to avoid the undefined condition in definition (30). In practical terms, however, such infinitesimal shifts correspond to consistently using one index ordering in making comparisons.

   Due to the formula chosen for spline integration, it is necessary that $b \leq y_{n+1}$. If this is not the case, let β be the index such that

$$y_\beta < b \leq y_{\beta+1}. \tag{45}$$

   Now add $\beta - n + 1$ knots into y, such that each of the new knots is larger than or equal to y_{n+l}. Correspondingly, increase the value of n by $\beta - n + 1$. This increases the number of basis splines used to represent $f(t)$. Since the new knots are outside the nonzero part of the spline, set the coefficients for each of the new basis splines to zero. Note that the function $f(t)$ is unchanged, but the new representation satisfies the condition that $b \leq y_{n+1}$.

   If $b = \infty$, then first replace b by $\min(y_{n+l}, x_{m+k})$. Since either $e(t)$ or $f(t)$ is zero past this point, this will not affect the value of the integral. Now adjust the representation of $f(t)$ as described above, if necessary.

   It is also required that y contain sufficient knots so that $B_{m,k+l,y}$ is defined. As previously mentioned, an easy way to implement this is to adopt the convention that added knots past the end of the given knot vector are equal to the last given knot.

1. Calculate the coefficients of $f^{(-1)}(t)$ using (41). We can now discard the coefficients of f itself and can calculate the two constant terms in the integration

by parts formula

$$\int_a^b e(t)f(t)dt = e(b)f^{(-1)}(b) - e(a)f^{(-1)}(a) - \int_a^b e^{(1)}(t)f^{(-1)}(t)dt, \tag{46}$$

leaving the integral term to be dealt with.

2. Calculate the coefficients of $e^{(1)}(t)$ using (39), and discard the coefficients of e.
3. Separate the basis splines of $e^{(1)}(t)$ into two categories: those with finite length support and those with zero length support. Let A and B be sets of integers such that:

$$i \in A \quad \text{if } 0 \le i \le m \quad \text{and} \quad x_i = x_{i+k-1}, \tag{47}$$

$$i \in B \quad \text{if } 0 \le i \le m \quad \text{and} \quad x_i < x_{i+k-1}. \tag{48}$$

The B-splines whose indices are in A are those whose support length is zero; they correspond to Dirac delta functions. We can write the spline $e^{(1)}(t)$ as

$$e^{(1)}(t) = \sum_{i\in A} E_i^{(1)}\delta(t - x_i) + \sum_{i\in B} E_i^{(1)} B_{i,k-1,x}(t). \tag{49}$$

4. Substitute (49) into the integral term in (46), noting that the integrals with delta terms reduce to function evaluations

$$\int_a^b e^{(1)}f^{(-1)}(t)dt = \sum_{i\in A} E_i^{(1)} f^{(-1)}(x_i) + \int_a^b \sum_{i\in B} E_i^{(1)} B_{i,k-1,x}(t)f^{(-1)}(t)dt. \tag{50}$$

The first set of terms require evaluations of the spline $f^{(-1)}(t)$. These terms can be calculated and added to the sum of terms calculated so far. If the set B is empty we are finished. If B is not empty then the integral term in (50) is the integral of a product of two splines. The first spline is the function $e^{(1)}(t)$ with the basis splines of zero support removed. The second is the spline $f^{(-1)}(t)$. Apply steps 1–4 recursively to this term, until all the splines in $e(t)$ are accounted for and B is reduced to the empty set.

Note that the same approach can be used to evaluate integrals of the form (2), provided that the first k antiderivatives of the function $f(t)$ can be calculated.

6.4. Computational cost. We will consider for simplicity problems of the form (1) where both splines are of order k. There are three computational components in the integration by parts algorithm:

1. Repeated differentiation of the spline $e(t)$. This step must be done k times. Each invocation takes one subtraction and one division per interval so the time spent on differentiation is on the order of k operations per segment.

2. Repeated integration of the spline $f(t)$. This step must be done k times. Each invocation takes one addition and one multiplication per interval so the time spent on antidifferentiation is also on the order of k operations per segment.

3. Evaluation of antiderivatives of the spline $f(t)$. One evaluation of an antiderivative of $f(t)$ must be done per segment; each evaluation takes on the order of k^2 operations.

The cost of the spline evaluations dominates the total computational cost. Thus the total cost is $O(k^2)$ operations per segment.

TABLE 1
Stability for order-4 B*-splines.*

r	Exact	Gauss Quadrature	Divided Differencing	Parts Method
0	4.19444444444444	4.19444444444444	4.19444444444444	4.19444444444444
1	4.06649773598049	4.06649773598049	$4.0664977359805\underline{0}$	$4.0664977359805\underline{0}$
2	4.04010964362323	4.04010964362322	4.04010964362323	4.04010964362322
3	4.03734554112486	4.03734554112486	$4.03734554112\underline{6}71$	4.03734554112486
4	4.03706789985594	4.03706789985594	$4.0370678998\underline{7}676$	4.03706789985593
5	4.03704012344300	4.03704012344300	$4.0370401232\underline{6}651$	$4.0370401234425\underline{5}1$
6	4.03703734567887	4.03703734567887	$4.037037345\underline{7}1633$	$4.0370373456\underline{8}062$
7	4.03703706790123	4.03703706790123	$4.03703707\underline{0}70117$	$4.0370370678\underline{9}959$
8	4.03703704012346	4.03703704012346	$4.03703703\underline{8}82036$	$4.0370370397\underline{7}985$
9	4.03703703734568	4.03703703734568	$4.0370361\underline{8}402446$	$4.0370370369\underline{3}127$
10	4.03703703706790	4.03703703706790	$4.03705\underline{0}93396563$	$4.0370370\underline{5}312282$
11	4.03703703704012	4.03703703704012	$4.03707\underline{1}98835796$	$4.0370367\underline{2}714926$
12	4.03703703703735	4.03703703703735	$4.038\underline{3}2729958782$	$4.037033\underline{3}3713402$
13	4.03703703703707	4.03703703703707	$4.0\underline{2}124815634075$	$4.03709\underline{1}74968579$
14	4.03703703703704	4.03703703703704	$4.\underline{1}7366372053858$	$4.037\underline{4}5891429760$
15	4.03703703703704	4.03703703703704	$3.9\underline{5}746527777776$	$4.03\underline{3}98753978588$

TABLE 2
Stability for order-6 B*-splines.*

r	Exact	Gauss Quadrature	Divided Differencing	Parts Method
0	30.3322685185185	30.3322685185185	30.3322685185185	30.3322685185185
1	28.8504734229846	28.8504734229846	28.8504734229846	28.8504734229845
2	28.6816125192285	28.6816125192285	$28.68161251922\underline{1}1$	28.6816125192286
3	28.6645841566786	28.6645841566786	$28.6645841567\underline{9}12$	28.6645841566787
4	28.6628799571565	28.6628799571565	$28.6628799578\underline{6}20$	28.6628799571566
5	28.6627095236305	28.6627095236305	$28.6627095\underline{0}75280$	28.6627095236304
6	28.6626924801422	28.6626924801422	$28.6626923\underline{7}29591$	28.6626924801421
7	28.6626907757920	28.6626907757920	$28.66268\underline{9}2195090$	28.6626907757921
8	28.6626906053570	28.6626906053570	$28.66269\underline{7}9434700$	$28.662690605357\underline{2}$
9	28.6626905883135	28.6626905883135	$28.6627\underline{6}89710028$	$28.662690588312\underline{5}$
10	28.6626905866091	28.6626905866091	$28.663\underline{5}029574743$	$28.6626905865\underline{7}84$
11	28.6626905864387	28.6626905864387	$28.65\underline{7}5636537495$	$28.662690585\underline{2}876$
12	28.6626905864216	28.6626905864216	$28.7\underline{4}88702056990$	$28.66269059\underline{5}0749$
13	28.6626905864199	28.6626905864199	$28.65\underline{5}8351597237$	$28.6626906\underline{8}02310$
14	28.6626905864198	28.6626905864198	$2\underline{0}.9915637713911$	$28.662691\underline{8}634761$
15	28.6626905864198	28.6626905864197	$\underline{9}4.3609932303722$	$28.66269\underline{5}5924918$

6.5. Stability. Of the three steps in the algorithm, only the third, evaluation, is unconditionally stable for all knot sequences. The other two steps, differentiation and antidifferentiation, may lead to numerical problems. It is also possible that forming the weighted sum of the evaluations may lead to loss of significance. To address these issues, a numerical comparison was made between integration by parts, Gauss quadrature, and direct evaluation of the divided difference formula (6). A similar comparison between Gauss quadrature and direct evaluation of (6) was made in [7].

The sample problem chosen was to evaluate the integral of the square of the order-k B-spline defined on the knot sequence $[5, 6, 6 + 10^{-r}, 8, \cdots, 5 + k]$. In Tables 1–3, values of $T_{0,0}^{k,k}$ are shown for r ranging from 0 to 15 for several different orders. The exact values were obtained symbolically using Maple [20]; the algorithmic results were calculated using double precision arithmetic on a DEC 5400 running Ultrix. In each result, the first significant digit in error is indicated using an underline. Note that the integration by parts algorithm is accurate to machine precision in the test case for all orders when the ratio of largest to smallest segment length is less than 10000:1; in

TABLE 3
*Stability for order-*10 B-*splines.*

r	Exact	Gauss Quadrature	Divided Differencing	Parts Method
0	2833.16953523513	2833.16953523514	2833.16953523513	$2833.1695352351\underline{7}$
1	2752.86392636369	2752.86392636369	$2752.863926362\underline{9}0$	$2752.8639263637\underline{2}$
2	2744.44592708222	2744.44592708222	$2744.44592709\underline{1}79$	$2744.4459270822\underline{6}$
3	2743.60112105862	2743.60112105862	$2743.60111\underline{9}69566$	$2743.6011210587\underline{2}$
4	2743.51661119805	2743.51661119805	$2743.51660\underline{9}88489$	$2743.516611198\underline{2}0$
5	2743.50815992021	2743.50815992021	$2743.5081\underline{3}576840$	$2743.5081599203\underline{1}$
6	2743.50731478951	2743.50731478950	$2743.50\underline{5}87880024$	$2743.507314789\underline{6}7$
7	2743.50723027641	2743.50723027641	$2743.4\underline{9}088954013$	$2743.5072302765\underline{4}$
8	2743.50722182510	$2743.5072218250\underline{9}$	$2743.\underline{3}4218803890$	$2743.5072218251\underline{2}$
9	2743.50722097996	2743.50722097996	$2743.\underline{6}6538821758$	$2743.5072209800\underline{1}$
10	2743.50722089545	2743.50722089545	$274\underline{5}.21802630369$	$2743.5072208953\underline{6}$
11	2743.50722088700	2743.50722088700	$2\underline{5}61.13229702871$	2743.50722088701
12	2743.50722088616	2743.50722088616	$2\underline{8}60.62453767450$	$2743.5072208861\underline{3}$
13	2743.50722088607	2743.50722088607	$\underline{1}7477.0284871979$	$2743.5072208860\underline{3}$
14	2743.50722088606	2743.50722088606	$\underline{-}159288.60966869$	$2743.5072208861\underline{3}$
15	2743.50722088606	2743.50722088606	$\underline{1}969830.68552676$	$2743.5072208860\underline{8}$

practical circumstances (e.g., finite elements) it is rare to see ratios this extreme. As r increases, the knots become less uniformly spaced and loss of significance becomes more pronounced in the integration by parts and divided difference algorithms. The accuracy of the divided difference scheme decreases as the order increases; it is interesting that the accuracy of the integration by parts algorithm *increases* as the order increases.

7. Comparison of the algorithms. In this section we present a brief comparison of the five methods considered for evaluation of the integral (1). We will consider for simplicity problems of the form (1) where both splines are of order k. Only rough estimates of the number of operations needed for each method are given.

The first method considered is the use of Gauss quadrature. To do the integration exactly requires evaluating each spline k times per interval. The cost of a B-spline evaluation is $O(k^2)$ operations, thus the total cost will be $O(k^3)$ operations per segment. The method exhibits no loss of significance for any order or knot sequence. Gauss quadrature extends to problems of the type given in (2).

The second method formulated the integrand as a linear combination of products of B-splines and used the recurrence given by de Boor, Lyche, and Schumaker [7] to calculate the integral of each B-spline pair. Each invocation of this recurrence requires $O(k^3)$ operations [7], and the recurrence must be carried out k times per segment, so the total cost is on the order of $O(k^4)$ operations per segment. The chief advantage of this approach is that it is very stable numerically. The disadvantages are that the recurrence is complicated, the approach does not extend to problems of the type given in (2), and this is the most expensive approach considered. It is worth noting that if a large number of integrations needs to be done using splines with the same bases but different coefficients, the inner products of the B-splines could be precalculated, thus reducing the computational expense.

The third method converted the splines to piecewise Bézier format and integrated segment by segment. To convert to Bézier format, we must compute the k Bézier coefficients on each segment. The calculation of these coefficients carries approximately the same price as the evaluation of a B-spline value, which is $k(k-1)/2$ linear combination operations. Thus the cost of converting both splines is about $k(k-1)$ linear combinations per segment. After the coefficients are obtained, we must calculate a weighted sum of all possible inner products of Bernstein polynomials on each interval. There are k^2 such inner products for two splines of order k. Thus the total cost of this method is of order $O(k^2)$. This algorithm cannot be extended to problems of the form (2).

The fourth method represented the integrand as a linear combination of B-splines. This involved the computation of numbers relating the coefficients of the factor splines to the coefficients of the product spline. It can be shown that these coefficients, Γ, are a generalization of the discrete B-splines. In fact, the computation of the Γ implicitly computes the discrete B-splines necessary to convert the splines to Bézier form. Thus we conclude that this method is at least as expensive as method 2. This algorithm also cannot be extended to problems of the form (2).

The final method is the integration by parts algorithm. Although it is not as stable as the other methods, good accuracy is obtained for reasonable knot vectors. The integration by parts method, in our implementation, is less expensive than the other methods; approximately three times as fast as Gauss quadrature for cubics, and about eight times as fast for degree 10 splines. In terms of order of operations, the method requires $O(k^2)$ operations, the same as the conversion to Bézier method and less than the other methods. In addition, this algorithm is extensible to problems of the form (2), provided that a sufficient number of antiderivatives of the function $f(t)$ can be calculated.

In conclusion, the integration by parts method is less expensive than the other methods considered, provides accurate results for reasonably uniform knot vectors, and generalizes to problems of the form (2).

Acknowledgment. The authors gratefully acknowledge the helpful comments of one of the referees.

REFERENCES

[1] O. C. Zienkiewicz, *The Finite Element Method in Engineering Science*, Fourth Edition, Prentice-Hall, Englewood Cliffs, NJ, 1987.

[2] S. D. Conte and C. de Boor, *Elementary Numerical Analysis*, Second Edition, McGraw Hill, New York, 1972.

[3] G. Dahlquist and A. Bjőrk, *Numerical Methods*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[4] P. Davis, *Interpolation and Approximation*, Blaidsell, New York, 1963.

[5] G. R. Heppler and J. S. Hansen, *A mindlin element for thick and deep shells*, Comput. Methods Appl. Mech. Engrg., 54 (1986), pp. 21–47.

[6] J. L. Phillips and R. J. Hanson, *Gauss quadrature rules with* B-*spline weight functions*, Math. Comp., 28 (1978), p. 666.

[7] C. de Boor, L. Lyche, and L. L. Schumaker, *On calculating with* B-*splines* ii: *Integration,* in Numerische Methoden der Approximationstheorie, L. Collatz, H. Werner, and G. Meinardus, eds., Birkhäuser, Basel, 1976, pp. 123–146.

[8] W. Boehm, *Inserting new knots into* B-*spline curves*, Computer Aided Design, 12 (1980), pp. 199–201.

[9] E. Cohen, T. Lyche, and R. Riesenfeld, *Discrete* B-*splines and subdivision techniques in computer-aided geometric design and computer graphics*, Computer Graphics and Image Processing, 14 (1980), pp. 87–111.

[10] R. Goldman, *Blossoming and knot insertion algorithms for* B-*spline curves*, Comput. Aided Geom. Design, 7 (1990), pp. 69–82.

[11] L. Ramshaw, *Blossoms are polar forms*, Comput. Aided Geom. Design, 6 (1989), pp. 323–358.

[12] P. Sablonnière, *Spline and Bézier polygons associated with a polynomial spline curve*, Computer Aided Design, 10 (1978), pp. 257–261.

[13] G. Farin, *Curves and Surfaces for Computer Aided Geometric Design*, Academic Press, New York, 1988.

[14] K. Morken, *Products of splines as linear combinations of* B-*splines*, manuscript.

[15] R. H. Bartels, J. C. Beatty, and B. A. Barsky, *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*, Morgan Kaufmann, San Mateo, CA, 1987.

[16] C. de Boor, *A Practical Guide to Splines*, Springer-Verlag, Berlin, New York, 1978.

[17] L. Schwartz, *Mathematics for the Physical Sciences*, Addison-Wesley, New York, 1966.

[18] A. H. Zemanian, *Distribution Theory and Transform Analysis*, McGraw Hill, New York, 1965.

[19] P. J. Barry and R. N. Goldman, *Algorithms for progressive curves: Extending* B-*spline and blossoming techniques to the monomial, power and Newton dual bases*, in Knot Insertion and Deletion Algorithms for B-Spline Modeling, R. Goldman and T. Lyche, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, to appear.

[20] B. W. Char, K. O. Geddes, G. H. Gonnet, M. B. Monogan, and S. M. Watt, *Maple Reference Manual*, Watcom Publications Limited, Waterloo, Ontario, Canada, 1988.

SIAM J. Sci. Stat. Comput.
Vol. 13, No. 5, pp. 1039-1061, September 1992

ONE-STAGE PARALLEL METHODS FOR THE NUMERICAL SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS*

H. W. TAM†

Abstract. This paper studies one-stage block methods for solving the numerical solution of ordinary differential equations (ODEs) in parallel, and a new family of one-stage methods is introduced. Each member in this family has a stability region equal to the interior of that of the Euler method, together with the origin. Thus the stability regions remain unchanged as the order is increased. These methods lead to the discovery of a specific approach in deriving parallel ODE methods with good stability regions; zero-stable block methods with perfect power stability polynomials are considered. It is also found that the coupling between the equations of individual time points for block methods must be taken into account. The coupling between time points restricts the use of the new one-stage parallel methods. By a modification of the method parameters, this coupling effect is eliminated. The resulting one-stage parallel methods outperform the Adams-Bashforth methods in both stability and accuracy.

Key words. one-stage, block methods, parallel processing, ordinary differential equations (ODEs)

AMS(MOS) subject classification. 65L05

1. Introduction. We study parallel methods for the numerical solution of the ordinary differential equation (ODE)

$$y(t_0)=\eta,$$
$$y'(t)=f(y(t)), \qquad t_0\leqq t\leqq t_{\text{out}},$$

where y and $f\in\mathcal{R}^n$. A nonautonomous ODE of the form $z'(t)=f(t, z(t))$ can always be written in the above autonomous form. Hence the above form involves no loss of generality.

Depending on the nature of the applications, existing ODE methods can be classified into stiff and nonstiff categories. A method suitable for stiff ODEs is usually implicit, which means that systems of linear equations of the form $[I-h\beta_0 J_n]x=y$ must be solved. Since the time spent on these linear systems usually dominates the overall computing time, a good speedup for stiff ODE problems can be achieved by performing the linear algebra in parallel. For nonstiff ODEs, we can exploit parallelism by evaluating function values simultaneously. The research for parallel methods in this category is still in its infancy. In this paper we focus on parallel nonstiff methods.

Stability and accuracy are the main considerations in deriving good ODE methods. It is fairly easy to derive sequential methods of good accuracy, e.g., in multistep methods we merely interpolate enough previous information to obtain the desired order. The same approach can be applied to parallel ODE methods. It is, however, the relatively small stability regions of many methods that constrain their usefulness. The main concern of this paper is to obtain parallel methods with good stability.

A detailed study on the effect of stability on the potential for parallelism for ODEs has been done in [17]. We give a brief review here on some of the key ideas in that paper, starting with the following definitions.

* Received by the editors July 28, 1989; accepted for publication November 14, 1991. This research was supported in part by U.S. Department of Energy grant DEF G02-87ER25026 and National Science Foundation grant NSF DMS 8703226.

† Department of Computer Science, University of Illinois, Urbana, Illinois 61801. Present address, Research and Development, Wolfram Research, Inc., 701 Budd Court, Campbell, California 95008.

DEFINITION 1.1. A *stage* of a method is a set of function evaluations performed in parallel (assuming there is an unlimited number of processors).

DEFINITION 1.2. The *stage number m* of a (parallel or serial) method is defined to be the number of stages (parallel or serial function evaluations) per step in the method.

DEFINITION 1.3. The *scaled stability region* of a method is defined to be

$$\frac{1}{m}S=\left\{\frac{1}{m}\mu\colon \mu\in S\right\},$$

where S is the stability region of the method and m is the stage number.

Since different methods have different numbers of stages, it is the scaled stability region that is appropriate for the comparison of stability properties of different methods.

DEFINITION 1.4. An *optimal* scaled stability region is one that cannot be properly contained in that of another method.

Optimality [12] is a desired condition for a method because there always exists a problem for which the method performs best due to its superior stability.

Roughly speaking, optimality theory [17] for parallel methods shows that if the scaled stability region of a serial method is optimal, it is still optimal among parallel and serial methods combined. Thus in the sense of optimal scaled stability regions, parallelism does not give any improvement. Due to the existence of optimal scaled stability regions among many ODE methods, numerous authors have proposed [12], [14], [5], [6] using instead the largest disk passing through the origin that can be inscribed into the scaled stability region of a method. Jeltsch and Nevanlinna [12] have demonstrated that such a measure is reasonable because the disks appear in an important way in the theory of error propagation for nonlinear problems. Again using optimality theory, we can prove that the largest disk that can be inscribed into a scaled stability region is the unit circle, for both parallel and serial methods. In this sense, the Euler method (the simplest serial method) has the best stability region. Unless otherwise noted, we will use the radius of the largest disk as a measure of how good the scaled stability region of a method is.

Although the above summary states that the largest scaled stability region a parallel ODE method can possibly have is the unit circle, there is no restriction on the order of such a method. The hope for success in finding good parallel methods is to find parallel methods with scaled stability regions that approach those of good (optimal) serial methods, such as the Euler method, but with higher order (or with significantly smaller error constants than serial methods of the same order). Alternatively, for a parallel and a serial ODE method of the same order, we would like to have a larger scaled stability region for the parallel method.

Existing work on finding parallel ODE methods with good stability properties has been unsatisfactory. In fact, many proposed parallel methods have smaller scaled stability regions than the Adams PECE method of the same order [17]. In this paper we study one-stage parallel ODE methods with good stability properties. In § 2 we introduce a two-processor, one-stage block method whose stability region is essentially that of the Euler method: the theoretical limit discussed above for a one-stage method, but now with an order equal to 2. This two-processor method can be generalized to higher orders. This new family of one-stage methods illustrates an important approach in deriving parallel methods with outstanding stability regions; we consider *zero-stable block methods with perfect power stability polynomials.* Methods with this property have an unchanged stability region as the order increases. This is an unconventional characteristic because the stability regions of existing methods all shrink as one raises the order. Although this family of one-stage methods is of restricted usage due to a phenomenon in block methods not seen before, the above specific approach nevertheless leads to the development of a successful family of two-stage methods that outperforms the Adams PECE methods by a sizeable margin. This two-stage family is described in an associated paper [18].

We show that conventional local error analysis is inadequate to predict the performance of the new one-stage parallel methods. For block methods we also need to take into account the coupling between time points, the effect of which appears in global errors. Although the local errors of these new parallel methods look reasonable in the sense of traditional multistep methods, global error analysis indicates that there is an accelerated error growth for eigenvalues along the imaginary axis of the $h\lambda$-plane. The details of the global error analysis will be studied in § 3. Despite the accelerated error growth, the new methods are the starting point from which useful one-stage parallel methods can be derived. In § 4 we will show that a slight variation of the above-mentioned methods yields one-stage parallel methods that outperform one-stage serial methods like the Adams–Bashforth methods. In § 5 we consider the effect of varying the spacing between time points within a block. It turns out that a nonuniform spacing has no benefit. Finally, numerical results are presented in § 6.

2. Derivation of a new one-stage method.

The discussion in § 1 states that the largest disk passing through the origin that can be inscribed into the scaled stability region of a (parallel or sequential) method is the unit circle centered at $(-1, 0)$. Any m-stage one-step method having the stability polynomial

$$\xi-\left(1+\frac{\mu}{m}\right)^m$$

possesses this scaled stability region. Such a stability polynomial may come from any variety of Runge–Kutta methods, e.g., a sequence of m identical Euler steps, each of length $\frac{h}{m}$ [10], or from a one-step method using higher derivatives [10]. The polynomial can also be realized by a method consisting of a predictor and $m-1$ correctors [12]. However, all these methods are of order one. Jeltsch and Nevanlinna [12] have shown that there exist explicit linear multistep methods of any order whose scaled stability regions can be made arbitrarily close to the unit circle, but the error coefficients also grow arbitrarily large, rendering these methods useless. Since the scaled stability regions of parallel methods are also limited by the unit circle, a natural question arises: "Can one obtain a parallel method of order-2 or higher with scaled stability region equal to (or close to) the unit circle?" Our goal is to find an order-2 or higher parallel method whose scaled stability region is close to that of Euler's method. In this paper we consider candidate methods with multiple saved values. Perhaps the simplest method of this nature is a one-stage block method that computes two new solution values from two previous solution values and their derivatives.

Let h be the stepsize (by which we mean the length of a block) for a block of two saved values y_n and $y_{n+1/2}$. In each computation step new values y_{n+1} and $y_{n+3/2}$ are calculated from y_n, $y_{n+1/2}$, $f_m=f(t_m, y_m)$, and $m=n$, $n+\frac{1}{2}$, as shown in Fig. 1. The stepsize h is the actual distance in t advanced per parallel function evaluation. (When we compare the performance of a block method to that of a serial method, the maximum allowable stepsize of the block method and the maximum allowable stepsize of the serial method per work unit for a fixed tolerance are the correct comparison quantities.) There are numerous ways to compute y_{n+1} and $y_{n+3/2}$ from y_n, $y_{n+1/2}$, f_n, and $f_{n+1/2}$ while maintaining second-order accuracy.

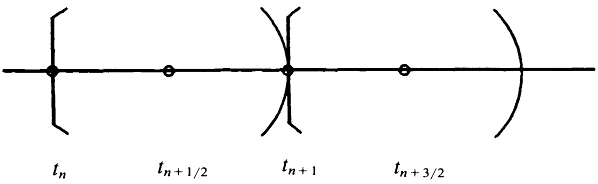


FIG. 1. *A simple block method.*

An example of an integration-based formula is given by

$$y_{n+1}=y_n+\int_{t_n}^{t_{n+1}} f_{n+1/2}+(t-t_{n+1/2})f_{n+1/2,n}\,dt=y_n+hf_{n+1/2}, \tag{2.1}$$

where $f_{n+1/2,n}$ is the backward divided difference involving $f_{n+1/2}$ and f_n. Alternatively, we can compute y_{n+1} from $y_{n+1/2}$, f_n, and $f_{n+1/2}$, and similarly in two ways for $y_{n+3/2}$. Taking a linear combination of the two equations for y_{n+1} and another linear combination of the two equations for $y_{n+3/2}$, we obtain a block method with two free parameters:

$$\begin{bmatrix} y_{n+3/2} \\ y_{n+1} \end{bmatrix}=\begin{bmatrix} 1-b & b \\ 1-a & a \end{bmatrix}\begin{bmatrix} y_{n+1/2} \\ y_n \end{bmatrix}+h\begin{bmatrix} 2+b/4 & -1+b/4 \\ \frac{3}{4}+a/4 & -\frac{1}{4}+a/4 \end{bmatrix}\begin{bmatrix} f_{n+1/2} \\ f_n \end{bmatrix}. \tag{2.2}$$

Equation (2.2) is the general second-order block method one obtained from the given previous values. Equation (2.2) can, of course, be derived from, e.g., the method of undetermined coefficients. However, an integration-based derivation paves the way for a variable stepsize extension.

We can manipulate free parameters a and b in (2.2) to obtain a good stability region. Of particular interest is the case $a=1$, $b=0$, giving

$$\begin{bmatrix} y_{n+3/2} \\ y_{n+1} \end{bmatrix}=\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} y_{n+1/2} \\ y_n \end{bmatrix}+h\begin{bmatrix} 2 & -1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} f_{n+1/2} \\ f_n \end{bmatrix}. \tag{2.3}$$

After applying (2.3) to the test problem $y'=\lambda y$, standard techniques give the stability polynomial

$$[\xi-(1+\mu)]^2,$$

where $\mu=h\lambda$. Conversely, the stability polynomial of (2.2) is such a perfect square only when $a=1$ and $b=0$. Conventionally, we do not consider methods having a perfect power stability polynomial because of the possible lack of zero stability. The lack of zero stability means that roundoff errors are amplified for the degenerate integration problem $y'(t)=f(t)$. However, (2.3) shows that when applied to the problem $y'=0$, the solution values remain unchanged.

In order to study the stability region of (2.4), we apply (2.3) to the test problem $y'=\lambda y$, $y(0)=1$ with exact initial values $y_0=1$, $y_{1/2}=e^{h\lambda/2}$, getting

$$\begin{aligned} \begin{bmatrix} y_{n+1/2} \\ y_n \end{bmatrix} &= \begin{bmatrix} 1+2\mu & -\mu \\ \mu & 1 \end{bmatrix}^n \begin{bmatrix} y_{1/2} \\ y_0 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 1+\mu \\ 1 & \mu \end{bmatrix}\begin{bmatrix} 1+\mu & \mu \\ 0 & 1+\mu \end{bmatrix}^n \begin{bmatrix} 1 & 1+\mu \\ 1 & \mu \end{bmatrix}^{-1}\begin{bmatrix} y_{1/2} \\ y_0 \end{bmatrix} \\ &= \begin{bmatrix} (1+\mu)^n+n\mu(1+\mu)^{n-1} & -n\mu(1+\mu)^{n-1} \\ n\mu(1+\mu)^{n-1} & (1+\mu)^n-n\mu(1+\mu)^{n-1} \end{bmatrix}\begin{bmatrix} y_{1/2} \\ y_0 \end{bmatrix}. \end{aligned} \tag{2.4}$$

Equation (2.4) shows that any point on the boundary of the unit disk other than the origin is unstable. Therefore the stability region of (2.3) contains the interior of the unit disk $\{\mu : |1+\mu| \leqq 1\}$ together with the origin.

With $t = nh$, (2.4) gives

$$\begin{aligned} y_n &= (1+\mu)^n \left[y_0 + \frac{n\mu}{(1+\mu)} (y_{1/2} - y_0) \right] \\ &= e^{n \log(1+\mu)} \left[1 + \frac{\lambda t}{(1+\mu)} (e^{h\lambda/2} - 1) \right] \\ &= e^{\lambda t} \cdot e^{(-h\lambda^2 t/2 + \mathcal{O}(h^2\lambda^3 t))} \left[1 + \frac{h\lambda^2 t}{2} + \mathcal{O}(h^2 \lambda^3 t) \right] \\ &= e^{\lambda t} \left[1 - \frac{h\lambda^2 t}{2} + \cdots \right] \left[1 + \frac{h\lambda^2 t}{2} + \cdots \right]. \end{aligned}$$

A similar expansion can be obtained for $y_{n+1/2}$. This demonstrates how the method (2.3) achieves second order. In essence, the term $n\mu(1+\mu)^{n-1}$ is being used to remove the dominant error term in $(1+\mu)^n$ as an approximation to $(e^{\mu})^n$.

The algebraic order p_a of an ODE method is defined by the largest real number p_a such that

$$p_1(\mu) - e^{\mu} = c\mu^{p_a+1} + \mathcal{O}(\mu^{p_a+2}), \qquad c \neq 0,$$

for some principal branch $p_1(\mu)$ of the algebraic function satisfying the stability polynomial of the method. Classical methods such as linear multistep, multiderivative methods, and Runge-Kutta methods have the property that $p \leqq p_a$, where p is the consistency order of the method. The algebraic order of (2.3) (here $p_1(\mu) = 1+\mu$) is only 1, but local error analysis indicates that (2.3) is of consistency order-2. This unusual phenomenon is possible because the stability matrix of (2.4) is defective. We believe that $p > p_a$ is possible only when the stability matrix of the underlying method is defective.

The parallel method (2.3) falls just short of our goal of finding a second-order method whose stability region is the unit disk $\{\mu : |1+\mu| \leqq 1\}$, although only the arc $\xi = -1 + e^{i\theta}$, $0 < \theta < 2\pi$ is unstable. We do not know whether one can improve the stability region of a second-order parallel method beyond that of (2.3).

We can generalize (2.3) to higher-order methods while retaining the same stability properties. By using polynomials interpolating f_n, $f_{n+1/3}$, $f_{n+2/3}$, and the appropriate y_n, $y_{n+1/3}$, $y_{n+2/3}$, we obtain the third-order method

$$\begin{aligned} \begin{bmatrix} y_{n+5/3} \\ y_{n+4/3} \\ y_{n+1} \end{bmatrix} &= \begin{bmatrix} 1-c_1-c_2 & c_2 & c_1 \\ 1-b_1-b_2 & b_2 & b_1 \\ 1-a_1-a_2 & a_2 & a_1 \end{bmatrix} \begin{bmatrix} y_{n+2/3} \\ y_{n+1/3} \\ y_n \end{bmatrix} \\ &+ h \begin{bmatrix} \frac{19}{4} + c_1/9 + 5c_2/36 & -6 + 4c_1/9 + 2c_2/9 & \frac{9}{4} + c_1/9 - c_2/36 \\ \frac{19}{9} + b_1/9 + 5b_2/36 & -\frac{20}{9} + 4b_1/9 + 2b_2/9 & \frac{7}{9} + b_1/9 - b_2/36 \\ \frac{23}{36} + a_1/9 + 5a_2/36 & -\frac{4}{9} + 4a_1/9 + 2a_2/9 & \frac{5}{36} + a_1/9 - a_2/36 \end{bmatrix} \begin{bmatrix} f_{n+2/3} \\ f_{n+1/3} \\ f_n \end{bmatrix}. \end{aligned} \tag{2.5}$$

For $a_1 = 1$, $a_2 = 0$, $b_1 = 0$, $b_2 = 1$, $c_1 = 0$, and $c_2 = 0$, (2.5) becomes

$$\begin{bmatrix} y_{n+5/3} \\ y_{n+4/3} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_{n+2/3} \\ y_{n+1/3} \\ y_n \end{bmatrix} + h \begin{bmatrix} \frac{19}{4} & -6 & \frac{9}{4} \\ \frac{9}{4} & -2 & \frac{3}{4} \\ \frac{3}{4} & 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} f_{n+2/3} \\ f_{n+1/3} \\ f_n \end{bmatrix}, \tag{2.6}$$

whose stability polynomial is

$$[\xi-(1+\mu)]^3.$$

The stability region of (2.6) is again the interior of the unit disk together with the origin. It is now obvious that the same procedure can lead to a qth-order method whose stability polynomial is $[\xi-(1+\mu)]^q$. Note that the qth-order method requires q processors in parallel. Henceforth we call the extension of (2.5) to q processors a q-processor block method and denote it by

$$Y_{n+1}=A_q Y_n + hB_q F_n. \tag{2.7}$$

Although (2.3) and (2.6) have the same stability region, the method (2.3) is nonetheless more stable in the following sense. Equation (2.4) shows that, when applied to the problem $y'=\lambda y$, the solution values y_n and $y_{n+1/2}$ of (2.3) consist of terms $(1+\mu)^n$ and $n\mu(1+\mu)^{n-1}$ multiplied by the initial values y_0 and $y_{1/2}$. The term $(1+\mu)^n$ behaves nicely for $|(1+\mu)|\leqq 1$. For $\mu=-2+\varepsilon$, $\varepsilon>0$, the term $n\mu(1+\mu)^{n-1}$ increases slowly before the effect of $n\mu$ is being annihilated by $(1+\mu)^{n-1}$ as n increases. The corresponding propagation matrix of (2.6) involves a term $n^2\mu^2(1+\mu)^{n-2}$ with even larger magnifying power than $n\mu(1+\mu)^{n-1}$ near the boundary of the unit circle. It is in this sense that the two-processor method is more stable than the three-processor method.

The above family of one-stage methods illustrates a simple but important approach in deriving parallel ODE methods with good stability properties; we consider zero-stable block methods with perfect power stability polynomials. Many proposed block methods contain arbitrary parameters with which we can manipulate the stability polynomials [15], [1], [4]. However, it is not usually possible to manipulate the method parameters so that the resulting stability polynomials are in the form of perfect powers of simple polynomials. As an example, let us consider a two-processor, two-block method (see Fig. 2), where the values $y_{n+3/2}$ and y_{n+1} of a new block are computed from the y and f values in two previous blocks. Using a derivation similar to that of (2.2), we can obtain the following fourth-order method with two free parameters.

Example 2.1. It holds that

$$\begin{aligned}\begin{bmatrix} y_{n+3/2} \\ y_{n+1}\end{bmatrix} &= \begin{bmatrix} 1-b & b \\ 1-a & a\end{bmatrix}\begin{bmatrix} y_{n+1/2} \\ y_n\end{bmatrix} + h\begin{bmatrix} \frac{9}{2}+3b/16 & -\frac{22}{3}+19b/48 \\ \frac{55}{48}+3a/16 & -\frac{59}{48}+19a/48\end{bmatrix}\begin{bmatrix} f_{n+1/2} \\ f_n\end{bmatrix} \\ &\quad + h\begin{bmatrix} \frac{31}{6}-5b/48 & -\frac{4}{3}+b/48 \\ \frac{37}{48}-5a/48 & -\frac{3}{16}+a/48\end{bmatrix}\begin{bmatrix} f_{n-1/2} \\ f_{n-1}\end{bmatrix}.\end{aligned} \tag{2.8}$$

There is no way to make the stability polynomial of (2.8) a perfect square for any choice of a and b. For the set $a=1$, $b=0$, the stability region is rather small (Fig. 3), with a stability interval equal to -0.18. Equation (2.8) is a simple method in the class of the multiblock methods of Chu and Hamilton [4]. Equation (2.8) is an example of a badly designed parallel method with a very small stability region (compared to the Adams–Bashforth method of the same order). Many existing parallel methods suffer from the weakness of having small stability regions [15], [7], [4], [1], so that their use

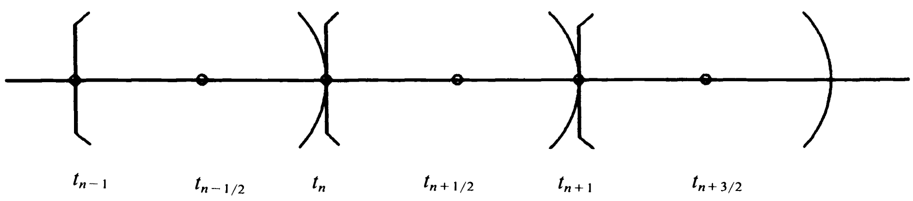


FIG. 2. *A two-block two-processor method.*

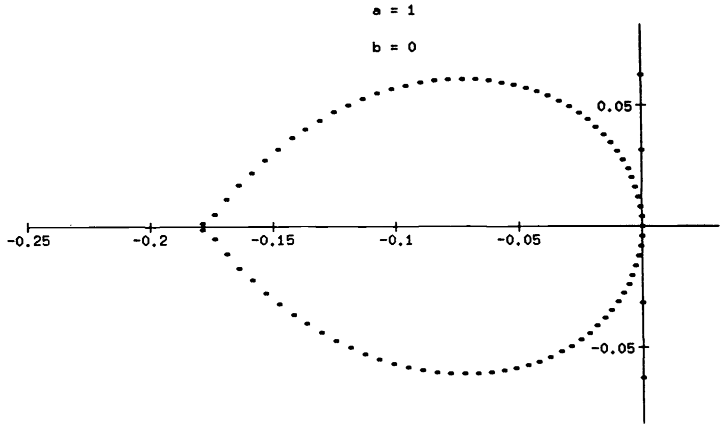


FIG. 3. *Stability region of Example* 2.1.

is limited to accuracy-bound applications. We believe, however, that the right combination of order, number of stored values, and method parameters will yield a parallel method with a good stability region. By choosing an appropriate order and number of stored values for the formula, we can derive useful zero-stable multiblock methods with stability polynomials equal to perfect powers of simple ones (see [17, Chap. 5]).

The one-stage block methods suggested in this section unfortunately have a drawback: due to coupling between time points, they are not accurate enough for eigenvalues along the imaginary axis of the $h\lambda$-plane. This drawback is significant for problems with natural frequencies on the imaginary axis that demand fairly high accuracy. An alternate approach is to have a different choice of the method parameters $a, b, \cdots$. The abnormal error behavior will be discussed in § 3 and the alternate approach in § 4.

3. Global error analysis of the block method. The local truncation error of the q-processor block method (2.7) is defined to be

$$\mathbf{d}_{n+1} = A_q Y(t_n) + hB_q f(Y(t_n)) - Y(t_{n+1}),$$

where $Y(t_n)$ is the exact solution $[y(t_{n+(q-1/q)}), \cdots, y(t_n)]^T$ of the given ODE. For the two-processor case,

$$\mathbf{d}_{n+1} = \begin{bmatrix} \dfrac{b-28}{96} h^3 y'''(\xi_1) \\ \dfrac{a-5}{96} h^3 y'''(\xi_2) \end{bmatrix}. \tag{3.1}$$

For comparison, the Euler method and the second-order Adams-Bashforth method have local errors $-\frac{1}{2}h^2 y''(\xi)$ and $-\frac{5}{12}h^3 y'''(\xi)$, respectively. Even though the individual components of (3.1) for $a=1$ and $b=0$ are smaller than that of the second-order Adams-Bashforth method, we should not be deceived by the behavior of (3.1). There is a classical theory in linear multistep methods (at least in the case of nonstiff ODEs) that relates the local error and global error. For the test problem $y' = \lambda y$, $y(0) = 1$, the second-order Adams-Bashforth method has a global error

$$y_n - y(t_n) = -\tfrac{5}{12}h^2 \int_0^{t_n} e^{(t_n - t)\lambda} y'''(t)\, dt + \mathcal{O}(h^3), \tag{3.2}$$

where $y(t)$ is the exact solution of the ODE. Note that the coefficient $-\frac{5}{12}$ of the local error is also that of the global error. For a block method, the individual equations for different time points may interact in a strange way so that the resulting global error has an unexpected growth. In this section we study the global error behavior of the method (2.3).

In linear multistep methods it is well known that an order-q method has a global error behaving like $\mathcal{O}(h^q)$. This relationship between the local error and global error extends to block methods. (See [9, Thm. 9.1].) Since the stability polynomial of (2.3) has no extraneous root at $\mu = 0$, if the given ODE has enough derivatives, (2.3) satisfies

$$\begin{bmatrix} y_{n+3/2} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} y(t_{n+3/2}) \\ y(t_{n+1}) \end{bmatrix} + \begin{bmatrix} h^2\, e_1(t_{n+3/2}) \\ h^2\, e_2(t_{n+1}) \end{bmatrix} + \begin{bmatrix} h^3 g_1(t_{n+3/2}) \\ h^3 g_2(t_{n+1}) \end{bmatrix} + \mathcal{O}(h^4), \tag{3.3}$$

where $e_1(t)$, $e_2(t)$, $g_1(t)$, and $g_2(t)$ are differentiable.

It is possible to explicitly determine the principal error functions $e_1(t)$ and $e_2(t)$. We first substitute (3.3) into the difference equation (2.3) and subtract both sides from the Taylor series

$$\begin{bmatrix} y(t_{n+3/2}) \\ y(t_{n+1}) \end{bmatrix} = \begin{bmatrix} y(t_{n+1/2}) \\ y(t_n) \end{bmatrix} + \begin{bmatrix} hy'(t_{n+1/2}) \\ hy'(t_n) \end{bmatrix} + \begin{bmatrix} \dfrac{h^2}{2} y''(t_{n+1/2}) \\ \dfrac{h^2}{2} y''(t_n) \end{bmatrix} + \begin{bmatrix} \dfrac{h^3}{6} y'''(t_{n+1/2}) \\ \dfrac{h^3}{6} y'''(t_n) \end{bmatrix} + \mathcal{O}(h^4).$$

Now expand every term in powers of h at the point t_n. The $\mathcal{O}(1)$, $\mathcal{O}(h)$, and $\mathcal{O}(h^2)$ terms on both sides of the resulting equation cancel each other. Dividing by h^3 yields

$$e_1'(t_n) = -\tfrac{7}{24} y'''(t_n) + f_y(y(t_n))[2e_1(t_n) - e_2(t_n)] + \mathcal{O}(h) \tag{3.4}$$

and

$$e_2'(t_n) = -\tfrac{1}{24} y'''(t_n) + f_y(y(t_n))\, e_1(t_n) + \mathcal{O}(h). \tag{3.5}$$

By assumption, (3.4) and (3.5) are true for any fixed value of t_n as $h \to 0$. Therefore one can drop the subscript n so that each of these equations holds for all values of t. Thus we have arrived at a set of ODEs from which $e_1(t)$ and $e_2(t)$ can be determined as follows:

$$\begin{bmatrix} e_1'(t) \\ e_2'(t) \end{bmatrix} = f_y(y(t)) \begin{bmatrix} 2 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} e_1(t) \\ e_2(t) \end{bmatrix} - \begin{bmatrix} \frac{7}{24} \\ \frac{1}{24} \end{bmatrix} y'''(t). \tag{3.6}$$

(Note that the coefficient matrix of the method (2.3), together with its local error (3.1), reappears in (3.6).)

If we use exact initial values for $y_{1/2}$ and y_0, set $n+1=0$ in (3.3), express every term in Taylor series at $t = t_0$, and equate coefficients of $\mathcal{O}(h^2)$, we obtain $e_1(t_0) = e_2(t_0) = 0$.

By making the transformation

$$\begin{bmatrix} \tilde{e}_1(t) \\ \tilde{e}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} e_1(t) \\ e_2(t) \end{bmatrix},$$

equation (3.6) becomes

$$\begin{bmatrix} \tilde{e}_1'(t) \\ \tilde{e}_2'(t) \end{bmatrix} = f_y(y(t)) \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \tilde{e}_1(t) \\ \tilde{e}_2(t) \end{bmatrix} + \begin{bmatrix} -\frac{1}{24} \\ -\frac{1}{4} \end{bmatrix} y'''(t),$$

which, together with the initial condition $\tilde{e}_1(t_0) = \tilde{e}_2(t_0) = 0$, determines the solution

$$\tilde{e}_2(t) = -\tfrac{1}{4}\int_{t_0}^{t} G(t,s)y'''(s)\,ds,$$

$$\tilde{e}_1(t) = -\tfrac{1}{24}\int_{t_0}^{t} G(t,s)y'''(s)\,ds + \int_{t_0}^{t} G(t,s)f_y(y(s))\tilde{e}_2(s)\,ds,$$

where $G(t,s)$ satisfies

$$\frac{\partial G(t,s)}{\partial t} = f_y(y(t))G(t,s), \qquad G(t,t) = I.$$

When applied to the test problem $y' = \lambda y$, $y(0) = 1$, $\lambda = \text{constant}$, we have $G(t,s) = e^{(t-s)\lambda}$, so that

$$\tilde{e}_2(t) = -\tfrac{1}{4}\int_{0}^{t} e^{(t-s)\lambda}\,y'''(s)\,ds,$$

which is similar to the global error function (3.2) of the second-order Adams-Bashforth method. However,

$$\tilde{e}_1(t) = \int_0^t e^{(t-s)\lambda}\left[-\frac{1}{24} - \frac{(t-s)\lambda}{4}\right]\lambda^3\,e^{\lambda s}\,ds = e^{t\lambda}\left[-\frac{t}{24} - \frac{t^2\lambda}{8}\right]\lambda^3.$$

If $\lambda = -1$, then $\tilde{e}_1(t) = -e^{-t}[-(t/24) + t^2/8]$ and the growth of t^2 is damped by e^{-t}. When $\lambda = i$,

$$\tilde{e}_1(t) = e^{it}\left[-\frac{t}{24} - \frac{t^2 i}{8}\right]i^3$$

and the growth of t^2 is undamped. The above global error analysis demonstrates that (2.3) has good accuracy for eigenvalues on the negative real axis. For eigenvalues along the imaginary axis, the effective error constant is of order $\mathcal{O}(t^2)$, as compared to a customary $\mathcal{O}(t)$ for a normal serial method. The corresponding global error expression for the q-processor block method with stability polynomial $[\xi - (1+\mu)]^q$ contains a leading term of order $\mathcal{O}(t^q)\cdot e^{\lambda t}$.

We can demonstrate that the global error of the method (2.3), when applied to the model problem $y' = \lambda y$, $y(0) = 1$, can be expressed in the form

$$y_n - y(t_n) = e^{\lambda t_n}\{h^2[-\tfrac{1}{8}\lambda^4 t_n^2 - \tfrac{1}{24}\lambda^3 t_n] + h^3[\tfrac{1}{24}\lambda^6 t_n^3 + \tfrac{3}{16}\lambda^5 t_n^2 + \tfrac{7}{48}\lambda^4 t_n] + \cdots\}.$$

For comparison, the Euler method has a global error of the form

$$y_n - y(t_n) = e^{\lambda t_n}\{-\tfrac{1}{2}h\lambda^2 t_n + h^2[\tfrac{1}{8}\lambda^4 t_n^2 + \tfrac{1}{3}\lambda^3 t_n] + h^3[-\tfrac{1}{48}\lambda^6 t_n^3 - \tfrac{1}{6}\lambda^5 t_n^2 - \tfrac{1}{4}\lambda^4 t_n] + \cdots\},$$

while that of the second-order Adams-Bashforth method is given by

$$y_n - y(t_n) = e^{\lambda t_n}\{-\tfrac{5}{12}h^2\lambda^3 t_n + \tfrac{1}{4}h^3\lambda^4 t_n + \cdots\}.$$

Thus (2.3) is indeed more accurate than the Euler method by considering the $\mathcal{O}(h)$ and $\mathcal{O}(h^2)$ terms in the global errors. The $\mathcal{O}(h^2)$ term in the global error of (2.3) is worse than that of the second-order Adams-Bashforth methods. However, when $|\lambda t_n| < 1$, the method (2.3) has a global error comparable to that of the second-order Adams-Bashforth method, but with much better stability properties. Therefore the consequence of the accelerated error growth of (2.3) is a restriction on the length of the integration interval.

The previous paragraph suggests that a remedy to the accelerated error growth is to restart the method (2.3) every n_s steps for some n_s. As $n_s \to \infty$, the scaled stability region of this restarted method approaches the unit circle, but the accuracy gets worse. We believe there is an appropriate value of n_s for a reasonable balance between stability and accuracy. The value of n_s may be problem dependent. An algorithm to find the appropriate n_s is left for further research.

Although (2.3) and its higher-order analogs are of restricted usage, they still enlighten us to consider zero-stable parallel methods with perfect power stability polynomials. Using this approach one can derive two-stage block methods that outperform the Adams PECE methods by a sizeable margin. These two-stage methods are described in a separate paper [18]. In the remainder of this paper we study useful one-stage parallel methods based on (2.3) and its higher-order analogs.

4. Methods without accelerated error growth. The growth of the error of the one-stage, two-processor block method is due to the coefficient matrix

$$\begin{bmatrix} 2+b/4 & -1+b/4 \\ \frac{3}{4}+a/4 & -\frac{1}{4}+a/4 \end{bmatrix}, \quad a=1, \quad b=0,$$

being defective. By a different choice of the parameters a and b we can make the above matrix diagonalizable. This choice of parameters at the same time reduces the stability region of the method (2.2). This reduction in stability is tolerable, so long as the resulting stability region is still comparable to those of existing widely used methods that are less accurate. (Recall that we compare two scaled stability regions by using the largest disks passing through the origin that can be inscribed into each of them.) Our goal is to manipulate a and b such that the stability region of the resulting method is comparable to that of the second-order Adams-Bashforth method (the reference method), while at the same time the accuracy of this new method exceeds that of the reference method. Note that the stability polynomial is not a perfect square any more after this order enhancement.

It is possible to choose a and b so that the method (2.2) is of order 3. This order enhancement can be achieved by an analysis of the global error of (2.2). However, for arbitrary a and b the stability polynomial of (2.2) has an extraneous root equal to $a-b$ at $\mu=0$. The consequence of this extraneous root is that the global error expression (3.3) is not valid anymore. We need to make use of a result in Skeel [16].

LEMMA 4.1. *Let*

$$\begin{aligned} &\mathbf{u}_0 = \sigma(\eta; h), \\ &\mathbf{u}_n = S\mathbf{u}_{n-1} + h\Psi(t_{n-1}, \mathbf{u}_{n-1}; h), \quad 1 \leqq n \leqq N, \end{aligned} \tag{4.1}$$

denote an ODE *method with starting procedure $\sigma(\eta; h)$, where the vector $\mathbf{u}_{n-1}$ contains the stored values that the method requires. Assume that S has a simple eigenvalue equal to* 1 *and that all other eigenvalues are less than* 1 *in modulus. Let*

$$\begin{aligned} &\hat{\mathbf{u}}_0 = \sigma(\eta; h) + \mathbf{r}_0, \\ &\hat{\mathbf{u}}_n = S\hat{\mathbf{u}}_{n-1} + h\Psi(t_{n-1}, \hat{\mathbf{u}}_{n-1}; h) + \mathbf{r}_n, \end{aligned} \tag{4.2}$$

be a perturbed solution based on (4.1) *with perturbations $\mathbf{r}_0, \mathbf{r}_1, \cdots, \mathbf{r}_N$. If $\mathbf{v}^T$ is a left eigenvector of S corresponding to the eigenvalue* 1, *then*

$$\max_{0 \leqq n \leqq N} \|\hat{\mathbf{u}}_n - \mathbf{u}_n\| \leqq C_1 \sum_{n=1}^{N} \|\mathbf{v}^T \mathbf{r}_n\| + C_2 \max_{0 \leqq n \leqq N} \|\mathbf{r}_n\| \tag{4.3}$$

for some constants C_1, C_2. In particular, if $\max_{1 \leqq n \leqq N} \|\mathbf{v}^T\mathbf{r}_n\| = \mathcal{O}(h^{q+1})$ *and* $\max_{0 \leqq n \leqq N} \|\mathbf{r}_n\| = \mathcal{O}(h^q)$, *then* $\max_{0 \leqq n \leqq N} \|\hat{\mathbf{u}}_n - \mathbf{u}_n\| = \mathcal{O}(h^q)$.

Proof. Lemma 4.1 is a direct consequence of Theorem 3.6 in Skeel [16]. □

We can treat the true solution $Y(t_n)$ of the ODE as a perturbed approximation to the numerical solution (2.2) with perturbations $\mathbf{d}_{n+1}$. Therefore, by Lemma 4.1, if

$$\mathbf{v}^T\mathbf{d}_{n+1}=\mathcal{O}(h^4) \tag{4.4}$$

and $\|y_{1/2}-y(t_{1/2})\|$ and $\|y_0-y(t_0)\|$ are of order $\mathcal{O}(h^3)$ where $\mathbf{v}^T=(1-a,b)$ is a left eigenvector of A_2 corresponding to the eigenvalue 1, then $y_n-y(t_n)$ and $y_{n+1/2}-y(t_{n+1/2})=\mathcal{O}(h^3)$. Condition (4.4) holds if

$$0=(1-a)\frac{b-28}{96}+b\frac{a-5}{96} \quad\Leftrightarrow\quad b=7(a-1). \tag{4.5}$$

Thus if we use $\mathcal{O}(h^3)$ accurate starting values, the method (2.2) is of third order when $b=7(a-1)$, $a\neq 1$.

We are still left with one free parameter to ensure a good stability region. With the relationship (4.5), the stability polynomial of the method (2.2) becomes

$$\xi^2+(-8+6a-2a\mu)\xi+[(2-a)\mu^2+(6-4a)\mu+(7-6a)]. \tag{4.6}$$

To ensure that the extraneous root of (4.6) at $\mu=0$ is bounded by 1, we also need to require that

$$|7-6a|\leqq 1 \quad\Leftrightarrow\quad 1\leqq a\leqq\tfrac{4}{3}. \tag{4.7}$$

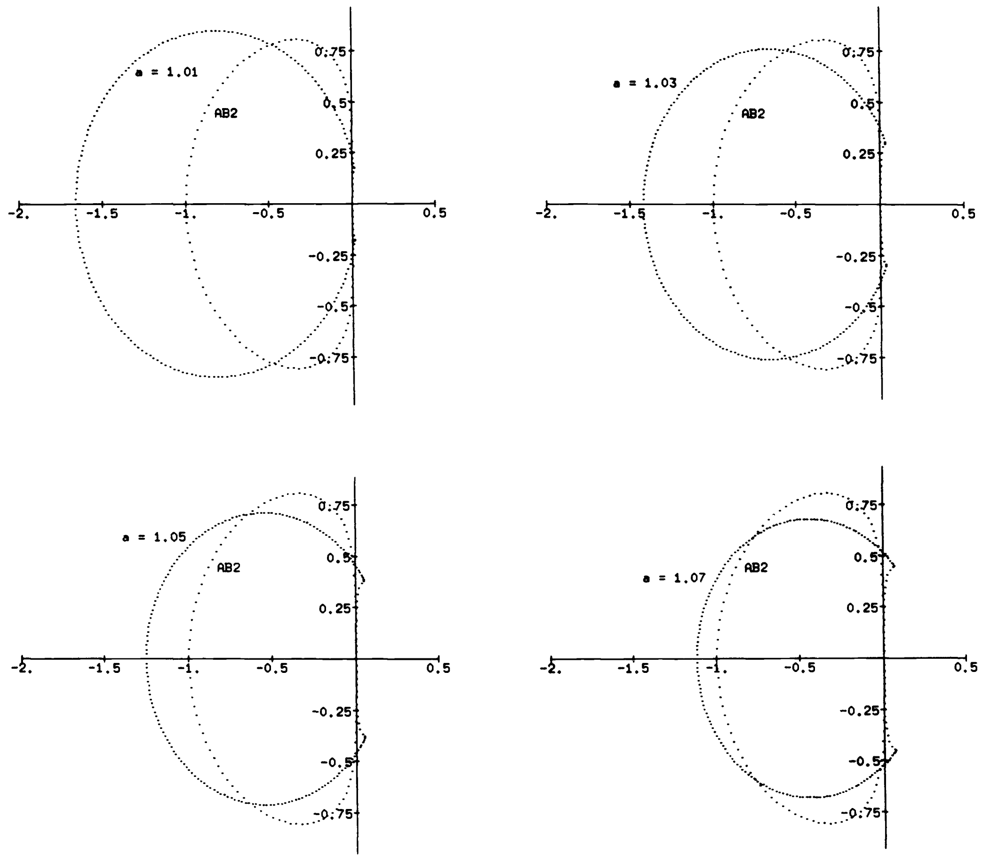


FIG. 4. *Stability region of* (2.2) *for different values of a.*

The stability regions of the method (2.2) for different choices of a's within the range (4.7) are shown in Fig. 4. The stability region of the reference method (second-order Adams-Bashforth method) is also sketched for comparison. The case $a=1$ gives the stability region with the largest inscribed circle. As a deviates from 1, the size of the inscribed circle shrinks. For $a \neq 1$, the stability region of the method (2.2) includes the boundary curves, like that of a conventional ODE method.

We must still ensure that the global error of (2.2) with $b=7(a-1)$, $a\neq 1$, behaves well. The following lemma shows that this is indeed the case.

LEMMA 4.2. *If* $b=7(a-1)$, $|a-b|\leqq 1$, $a\neq 1$ *in method* (2.2), *and* $Y_0 = Y(t_0)+\mathcal{O}(h^3)$, *then*

$$\begin{bmatrix} y_{n+1/2} \\ y_n \end{bmatrix} = \begin{bmatrix} y(t_{n+1/2}) \\ y(t_n) \end{bmatrix} + h^3 \begin{bmatrix} g_1(t_{n+1/2}) \\ g_2(t_n) \end{bmatrix} + h^3(7-6a)^n y'''(t_0) \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} + \mathcal{O}(h^4), \tag{4.8}$$

where $\delta_1 = 7(a-5)/576(1-a)$, $\delta_2=(a-5)/576(1-a)$, *and*

$$\begin{aligned} g_1(t) &= -\delta_1 G(t,t_0)y'''(t_0) \\ &\quad + \int_{t_0}^{t} G(t,s)\left[\frac{9a-37}{576(a-1)} y^{IV}(s) + \frac{5-a}{576(a-1)} f_y(y(s))y'''(s)\right] ds, \\ g_2(t) &= g_1(t) - \frac{a-5}{96(a-1)} y'''(t). \end{aligned} \tag{4.9}$$

Proof. We can consider

$$\mathbf{z}_n = Y(t_n) + h^3 \begin{bmatrix} g_1(t_{n+1/2}) \\ g_2(t_n) \end{bmatrix} + h^3(7-6a)^n y'''(t_0) \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix}$$

as a perturbed approximation to the solution of the method (2.2). It can be shown that

$$\mathbf{z}_0 - Y_0 = h^3 \begin{bmatrix} g_1(t_{1/2}) \\ g_2(t_0) \end{bmatrix} + h^3 y'''(t_0) \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} = \begin{bmatrix} \mathcal{O}(h^4) \\ 0 \end{bmatrix}.$$

If we define

$$\hat{\mathbf{d}}_{n+1} = \begin{bmatrix} \hat{d}_{n+3/2} \\ \tilde{d}_{n+1} \end{bmatrix} = A_2 \mathbf{z}_n + hB_2 f(\mathbf{z}_n) - \mathbf{z}_{n+1},$$

then

$$\begin{aligned} \hat{d}_{n+3/2} = {} & \frac{7(a-5)}{96} h^3 y'''(t_{n+1/2}) + \frac{-7a-1}{384} h^4 y^{IV}(t_{n+1/2}) + h^3 \delta_1 y'''(t_0)(7-6a)^n \\ & + 7(a-1)\left[h^3\left(g_2(t_{n+1/2}) - \frac{h}{2} g_2'(t_{n+1/2}) - g_1(t_{n+1/2}) \right) \right. \\ & \qquad \left. + h^3(\delta_2-\delta_1) y'''(t_0)(7-6a)^n \right] \\ & + \frac{7a+1}{4} hf_y(y(t_{n+1/2}))[h^3 g_1(t_{n+1/2}) + h^3\delta_1 y'''(t_0)(7-6a)^n] \\ & + \frac{7a-11}{4} hf_y(y(t_{n+1/2}))[h^3 g_2(t_{n+1/2}) + h^3\delta_2 y'''(t_0)(7-6a)^n] \\ & - h^4 g_1'(t_{n+1/2}) - h^3\delta_1 y'''(t_0)(7-6a)^{n+1} + \mathcal{O}(h^5), \end{aligned} \tag{4.10}$$

and

$$
\begin{aligned}
\hat{d}_{n+1} = {} & \frac{a-5}{96} h^3 y'''(t_n) + \frac{a-9}{384} h^4 y^{IV}(t_n) + (a-1)h^3\left[g_2(t_n) - g_1(t_n) - \frac{h}{2} g_1'(t_n)\right] \\
& + ah^3(\delta_2 - \delta_1) y'''(t_0)(7-6a)^n + h^3[\delta_1 - \delta_2(7-6a)] y'''(t_0)(7-6a)^n \\
& + \frac{3+a}{4} hf_y(y(t_n))[h^3 g_1(t_n) + h^3 \delta_1 y'''(t_0)(7-6a)^n] \\
& + \frac{-1+a}{4} hf_y(y(t_n))[h^3 g_2(t_n) + h^3 \delta_2 y'''(t_0)(7-6a)^n] - h^4 g_2'(t_n).
\end{aligned} \tag{4.11}
$$

As n becomes large, terms involving $(7-6a)^n$ become negligible. This argument and the second part of (4.9) imply that the $\mathcal{O}(h^3)$ terms in (4.10) and (4.11) cancel out. Hence $\hat{\mathbf{d}}_{n+1} = \mathcal{O}(h^4)$. We can also use the fact that $\sum_{n=1}^{N} (7-6a)^n = \mathcal{O}(1)$ and that

$$
\begin{aligned}
(1-a)g_1'(t) + 7(a-1)g_2'(t) = {} & \frac{7a^2 - 38a + 31}{192} y^{IV}(t) + \frac{7}{2}(1-a)^2(g_2'(t) - g_1'(t)) \\
& - 5(1-a) f_y(y(t)) g_1(t) - (1-a) f_y(y(t)) g_2(t)
\end{aligned}
$$

to show that $\sum_{n=1}^{N} \|\mathbf{v}^T \hat{\mathbf{d}}_n\| = \mathcal{O}(h^4)$. Hence by Lemma 4.1,

$$
Y_n - \mathbf{z}_n = Y_n - Y(t_n) - h^3 \begin{bmatrix} g_1(t_{n+1/2}) \\ g_2(t_n) \end{bmatrix} - h^3 (7-6a)^n y'''(t_0) \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} = \mathcal{O}(h^4),
$$

and the lemma follows. □

Lemma 4.2 shows that with $b = 7(a-1)$, $a \neq 1$, the leading term for the global error is given by $h^3\, e^{\lambda t}[C_1 \lambda^4 t + C_2 \lambda^3]$ for some constants C_1 and C_2, when applied to the model problem $y' = \lambda y$. This is comparable to the global error behavior of a regular ODE method.

With the order enhanced, we can vary a to get a stability region with an inscribed disk that is comparable to that of the reference method. The choice $a = 1.07$, $b = 0.49$ appears to give an appropriate method. We have run some numerical tests comparing this choice with the reference method. Test results for $a = 1$, $b = 0$ are also presented simultaneously to demonstrate the accelerated error growth mentioned at the end of § 3. The results show that the choice $a = 1$, $b = 0$ has good speedup over the reference method on the negative real axis (a speedup of almost 2 for the problem $y' = -y$ for an accuracy of two digits), while it performs poorly on the imaginary axis (consider the orbit problem). The new choice $a = 1.07$, $b = 0.49$ regains a speedup over the reference method on the imaginary axis.

We can enhance the order of the three-processor method (2.6) in a way analogous to the two-processor case. Let

$$
\bar{\mathbf{d}}_{n+1} = h^4 \begin{bmatrix} (c_2/1944 - \frac{25}{216}) y^{IV}(\xi_1) \\ (b_2/1944 - \frac{8}{243}) y^{IV}(\xi_2) \\ (a_2/1944 - \frac{1}{216}) y^{IV}(\xi_3) \end{bmatrix}
$$

denote the local truncation error of the three-processor method (2.5). By Lemma 4.1, if we use $\mathcal{O}(h^4)$ initial values and choose parameters to make

$$
\bar{\mathbf{v}}^T \bar{\mathbf{d}}_{n+1} = \mathcal{O}(h^5), \tag{4.12}
$$

where

$$
\bar{\mathbf{v}} = \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} 1 - a_1 - a_2 b_1 - b_2 + a_1 b_2 \\ a_2 c_1 + c_2 - a_1 c_2 \\ c_1 - b_2 c_1 + b_1 c_2 \end{bmatrix}
$$

is a left eigenvector of A_3 corresponding to the eigenvalue 1, then (2.5) becomes order 4. The condition (4.12) is satisfied when

$$25\alpha + 7\beta + \gamma = 0. \tag{4.13}$$

However, we have not been able to get a fourth-order method with a good stability region for (2.5). We do obtain a third-order method using parameters

$$a_1 = 0.9, \quad a_2 = 0.1, \quad b_1 = 0, \quad b_2 = 1, \quad c_1 = 0, \quad c_2 = 0.1, \tag{4.14}$$

with stability region shown in Fig. 5. This stability region is much larger than those of the second- or third-order Adams–Bashforth methods. Numerical experiments for this third-order variation are given in § 6. The numerical results show that this third-order variation is an improvement over the third-order Adams–Bashforth method in both accuracy and stability.

5. Nonuniformly distributed time points. Thus far, the time points within each individual block of the q-processor block method have been equally spaced. The decision to use equally spaced time points has been based on simplicity and convenience. We have given no analysis on what kind of spacing yields the best stability or accuracy. In light of the order enhancement of one-step collocation methods, we might suspect a similar phenomenon in block methods. In this section we investigate whether a nonuniform spacing between the time points of the two-processor block method has any benefit.

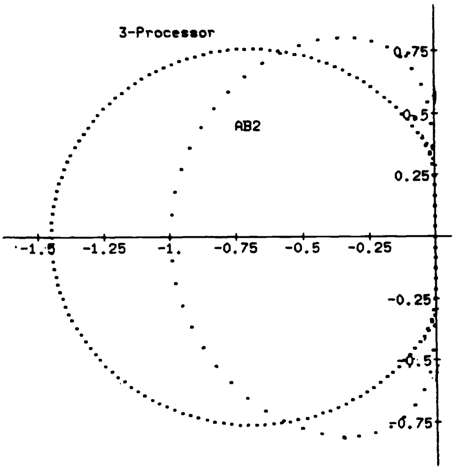


FIG. 5. *Stability region of* (2.5) *with parameters* (4.14).

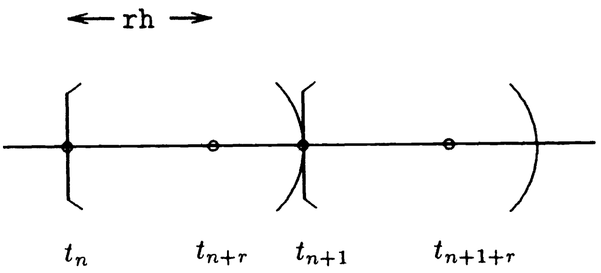


FIG. 6. *Variable spacing two-processor block method.*

Let the points of the variable spacing two-processor block method be represented by the diagram shown in Fig. 6. The two points t_n and t_{n+r} within a block are separated by the distance rh. The quantity r is assumed to be positive without loss of generality. We want to examine what the effect of the extra parameter r is on the stability or accuracy of the method. The quantity r can be either greater than or less than one. A value of $r=1$ makes the block method essentially a frontal method [13], [8]. The case $r>1$ represents a premature prediction of a block further ahead.

Equation (2.2) is a special case $r=\frac{1}{2}$ of the nonuniform time point two-processor block method

$$\begin{aligned}\begin{bmatrix} y_{n+1+r} \\ y_{n+1} \end{bmatrix} &= \begin{bmatrix} 1-b & b \\ 1-a & a \end{bmatrix}\begin{bmatrix} y_{n+r} \\ y_n \end{bmatrix} \\ &\quad + h\begin{bmatrix} 1+1/2r+br/2 & -1/2r+br/2 \\ 1/2r-r/2+ar/2 & 1-1/2r-r/2+ar/2 \end{bmatrix}\begin{bmatrix} f_{n+r} \\ f_n \end{bmatrix}.\end{aligned} \tag{5.1}$$

Equation (5.1) shows that the method coefficients grow large as the two time points approach one another. This indicates that allowing the time points to get too close together may have a negative effect on roundoff error.

The method (5.1) has a stability polynomial

$$\begin{aligned}&\xi^2+\xi\left[-1-a+b-2\mu+r\mu\left(\frac{1-a-b}{2}\right)\right]+\mu^2\left[\left(\frac{1+a-b}{2}\right)-r\left(\frac{1-a-b}{2}\right)\right] \\ &\quad +\mu\left[1+a-b-r\left(\frac{1-a-b}{2}\right)\right]+a-b,\end{aligned} \tag{5.2}$$

which, for $a=1$ and $b=0$, is equal to $[\xi-(1+\mu)]^2$. Thus the choice of r does not affect whether (5.2) can be made a perfect square. Conversely, the stability polynomial (5.2) is equal to $[\xi-(1+\mu)]^2$ only when $a=1$ and $b=0$.

The local truncation error of (5.1) is given by

$$\begin{bmatrix} (-\frac{1}{6}-r/4+br^3/12)h^3y'''(\xi_1) \\ (-\frac{1}{6}+r/4-r^3/12+ar^3/12)h^3y'''(\xi_2) \end{bmatrix}.$$

Again, since (5.1) has a second-order local truncation error, the global error for the case $a=1$, $b=0$ satisfies (by [9, Thm. 9.1]),

$$\begin{bmatrix} y_{n+1+r} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} y(t_{n+1+r}) \\ y(t_{n+1}) \end{bmatrix} + \begin{bmatrix} h^2e_1(t_{n+1+r}) \\ h^2e_2(t_{n+1}) \end{bmatrix} + \begin{bmatrix} h^3g_1(t_{n+1+r}) \\ h^3g_2(t_{n+1}) \end{bmatrix} + \mathcal{O}(h^4). \tag{5.3}$$

As in § 3, we can verify that $e_1(t)$ and $e_2(t)$ can be determined from

$$\begin{bmatrix} e_1'(t) \\ e_2'(t) \end{bmatrix} = f_y(y(t))\begin{bmatrix} 1+1/2r & -1/2r \\ 1/2r & 1-1/2r \end{bmatrix}\begin{bmatrix} e_1(t) \\ e_2(t) \end{bmatrix} + \begin{bmatrix} -\frac{1}{6}-r/4 \\ -\frac{1}{6}+r/4 \end{bmatrix} y'''(t). \tag{5.4}$$

If we make the transformation

$$\begin{bmatrix} \tilde{e}_1(t) \\ \tilde{e}_2(t) \end{bmatrix} = \begin{bmatrix} 1-1/2r & 1/2r \\ 1/2r & -1/2r \end{bmatrix}\begin{bmatrix} e_1(t) \\ e_2(t) \end{bmatrix},$$

then (5.4) becomes

$$\begin{bmatrix} \tilde{e}_1'(t) \\ \tilde{e}_2'(t) \end{bmatrix} = f_y(y(t)) \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \tilde{e}_1(t) \\ \tilde{e}_2(t) \end{bmatrix} + \begin{bmatrix} \frac{1}{12} - r/4 \\ -\frac{1}{4} \end{bmatrix} y'''(t).$$

Recall that in § 3, it is the defective error propagation matrix $[\begin{smallmatrix} 2 & -1 \\ 1 & 0 \end{smallmatrix}]$ in (3.6) that leads to an accelerated error growth on the imaginary axis. The error propagation matrix

$$\begin{bmatrix} 1+1/2r & -1/2r \\ 1/2r & 1-1/2r \end{bmatrix}$$

in (5.4) is always defective for any choice of $r \neq 0$. The defective error propagation matrix makes $\tilde{e}_1(t)$ explicitly dependent on $\tilde{e}_2(t)$. It is this explicit dependence of $\tilde{e}_1(t)$ on $\tilde{e}_2(t)$ that generates an $\mathcal{O}(t^2)$ term in $\tilde{e}_1(t)$ (when (5.1) is applied to the model problem $y' = \lambda y$). Because $\tilde{e}_2(t) \neq 0$ for any value of r, we cannot eliminate the $\mathcal{O}(t^2)$ term in the solution of $\tilde{e}_1(t)$. Hence it is impossible to avoid the accelerated error growth for eigenvalues along the imaginary axis for the case $a = 1$, $b = 0$ by varying the spacing between time points.

As in § 4, we can avoid the accelerated error growth on the imaginary axis by a different choice of the parameters a and b. When

$$0 = \left(\frac{-1+a-b}{6}\right) + r\left(\frac{-1+a+b}{r}\right)$$

and $a = 1$, $b = 0$ does not hold simultaneously, the method (5.1) becomes third order. By a proof analogous to that of Lemma 4.2, it can be shown that the global error of the third-order method has leading terms

$$g_1(t) - \frac{(2+3r)(-2+3r-r^3+ar^3)}{72r(a-1)} \left(\frac{-2-3r+6ra}{-2+3r}\right)^n y'''(t_0)$$

and

$$g_2(t) - \frac{(2-3r)(-2+3r-r^3+ar^3)}{72r(a-1)} \left(\frac{-2-3r+6ra}{-2+3r}\right)^n y'''(t_0),$$

where

$$\begin{aligned} g_1(t) = {} & \frac{(2+3r)(-2+3r-r^3+ar^3)}{72r(a-1)} G(t, t_0) y'''(t_0) \\ & + \int_{t_0}^{t} G(t, s) \left[\frac{-4+(a-1)r+9r^2+(1-a)r^3+(3a-3)r^4}{72r(a-1)} y^{IV}(s) \right. \\ & \qquad \left. + \frac{(1-3r)(-2+3r-r^3+ar^3)}{72r(a-1)} f_y(y(s)) y'''(s) \right] ds \end{aligned}$$

and

$$g_2(t) = g_1(t) - \frac{-\frac{1}{6} + r/4 - r^3/12 + ar^3/12}{(a-1)} y'''(t).$$

Thus the best we can do with varying r is to eliminate one of the component functions within the leading terms of the global error. Hence varying r does not improve the order of the variable time point two-processor block method. Based on this result, we believe that it is sufficient to consider uniformly spaced time points within a block.

6. Numerical results. We have tested the two-processor and three-processor block methods on selected problems of the standard test set DETEST [11]. These test problems are representative in demonstrating the stability and accuracy characteristics of the proposed methods on both the real and imaginary axes. The test problems, together with their analytic solutions, are listed below:

1.

$$y' = -y, \quad y(0) = 1, \quad t_{\text{out}} = 20, \quad y(t) = e^{-t},$$

2.

$$y' = -\frac{y^3}{2}, \quad y(0) = 1, \quad t_{\text{out}} = 20, \quad y(t) = \frac{1}{\sqrt{1+t}},$$

3.

$$y' = y \cos t, \quad y(0) = 1, \quad t_{\text{out}} = 20, \quad y(t) = e^{\sin t},$$

4.

$$y' = \frac{y}{4}\left(1 - \frac{y}{4}\right), \quad y(0) = 1, \quad t_{\text{out}} = 20, \quad y(t) = \frac{20}{1+19e^{-t/4}},$$

5.

$$\begin{bmatrix} y_1' \\ y_2' \\ y_3' \end{bmatrix} = \begin{bmatrix} -y_2 - y_2 y_3/r \\ y_1 - y_2 y_3/r \\ y_1/r \end{bmatrix}, \quad y(0) = \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix}, \quad y(t) = \begin{bmatrix} (2+\cos t) \cos t \\ (2+\cos t) \sin t \\ \sin t \end{bmatrix},$$

$$t_{\text{out}} = 20, \quad r = \sqrt{y_1^2 + y_2^2},$$

6.

$$\begin{bmatrix} y_1' \\ y_2' \\ y_3' \\ y_4' \end{bmatrix} = \begin{bmatrix} y_2 \\ -y_1/r^3 \\ y_4 \\ -y_3/r^3 \end{bmatrix}, \quad y(0) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad y(t) = \begin{bmatrix} \cos t \\ -\sin t \\ \sin t \\ \cos t \end{bmatrix},$$

$$t_{\text{out}} = 25, \quad r = \sqrt{y_1^2 + y_3^2},$$

7.

$$\begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} y_1/2(1+t) - 2ty_2 \\ y_2/2(1+t) + 2ty_1 \end{bmatrix}, \quad y(0) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad y(t) = \begin{bmatrix} \sqrt{1+t}\cos t^2 \\ \sqrt{1+t}\sin t^2 \end{bmatrix},$$

$$t_{\text{out}} = 6.$$

The test problems include both linear and nonlinear examples. Problems 1-4 have eigenvalues on the real axis, while problems 5-7 contain eigenvalues with imaginary parts. The integration interval in each test problem is long enough to indicate any deficiency in the long-term behavior of each method.

The two-processor and three-processor methods are compared to the second- and third-order Adams–Bashforth methods. For the two-processor case, we use parameters $a=1$ and $b=0$ (with unit circle stability region) and the alternate choice $a=1.07$, $b=0.49$. For the three-processor case, we use parameters $a_1=1$, $a_2=0$, $b_1=0$, $b_2=1$, and $c_1=c_2=0$ (with unit circle stability region) and the alternate choice $a_1=0.9$, $a_2=0.1$, $b_1=0$, $b_2=1$, $c_1=0$, and $c_2=0.1$. A constant stepsize test is enough to compare the accuracy and stability properties of these methods. We believe that if a parallel method is inferior to existing serial methods for constant stepsizes, this method has no hope in a variable stepsize implementation. Also, we do not want to obscure the performance of the new methods by a crude error estimator. Therefore all numerical tests are done using constant stepsizes.

Because the proposed and the reference methods all require more than one starting point, we need to specify a starting criterion for each method. We believe that a fair start for all methods is the provision of sufficient information for each of them to compute the solution value at t_1. Thus for a stepsize h and initial time t_0, the second-order and third-order Adams–Bashforth methods are given exact values at t_{-1} and t_{-1}, t_{-2}, respectively. For the two-processor block method, we give exact starting values at t_0 and $t_{1/2}$. For the three-processor case, exact values at t_0, $t_{1/3}$, and $t_{2/3}$ are provided.

As the block methods generate more intermediate points than the Adams method, a fair comparison is to sample only the errors at gridpoints $t_1, t_2, \cdots$ of the block methods. (Incidentally, we know from global error analysis that within a block, the error is smallest at the first point, e.g., the error at t_n is smaller than that at $t_{n+1/3}$ or $t_{n+2/3}$ for the three-processor case.) We measure the error of each method by $\text{err} = \max_n \|y_n - y(t_n)\|_2$.

Since the test examples are of small scale, when run on a parallel computer, the communication time between different processors in the parallel methods will be a significant portion of the run time. Moreover, the usual consensus in measuring the performance of nonstiff ODE methods on a particular problem is to count the number of function evaluations required by each of them. In practice, when the problem size is large, the interprocessor communication time becomes relatively insignificant. Therefore, in the subsequent performance plots, we show the number of function evaluations per processor for each method. Because the new methods are conventional in nature (they belong to Butcher's [2], [3] class of general linear methods) and do not exploit special properties of the ODE system, it is reasonable to expect that their performance on small systems is indicative of their performance on large systems.

We run each method on a particular test problem using different constant values of h to complete the given integration interval. Dividing the integration interval by the stepsize gives the number of function evaluations per processor. The decimal places of accuracy in each plot are computed by $-\log_{10} \text{err}$. The tests are run on an Alliant FX/8 using double precision arithmetic, although it is obvious that the tests do not require the actual use of a parallel computer. The plots of accuracy versus number of function evaluations per processor for the test problems are shown in Figs. 7–13.

The results of problem 1 clearly indicates the improved stability of the two- and three-processor block methods with unit circle stability regions over the second- and third-order Adams–Bashforth methods. The stability properties of the two-processor case with a different parameter choice still match those of the second-order Adams–Bashforth method, while the stability characteristics of the three-processor block method with modified parameters are even better. In all test examples except problem 5, the modified parameter three-processor method has better accuracy than the

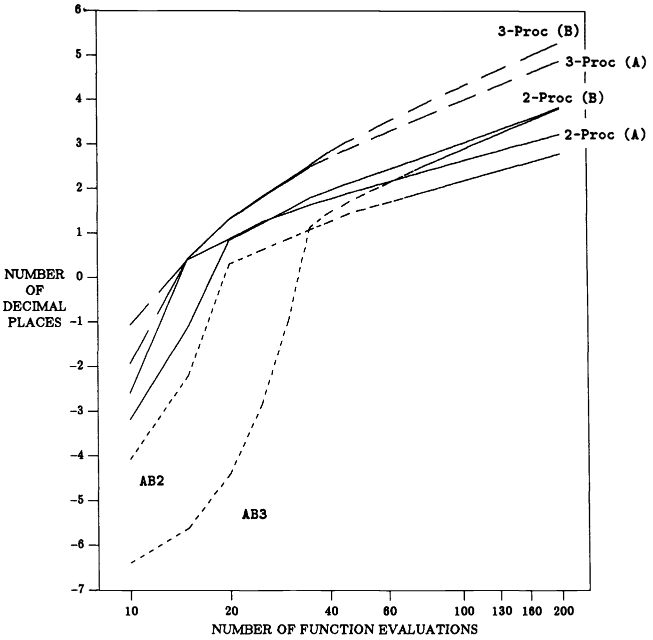


FIG. 7. *Problem* 1. AB2, AB3: *Adams-Bashforth method* (*second- and third-order*); 2- *or* 3-Proc (A): *two- or three-processor method with unit circle stability region*; 2- *or* 3-Proc (B): *two- or three-processor method with modified parameters.*

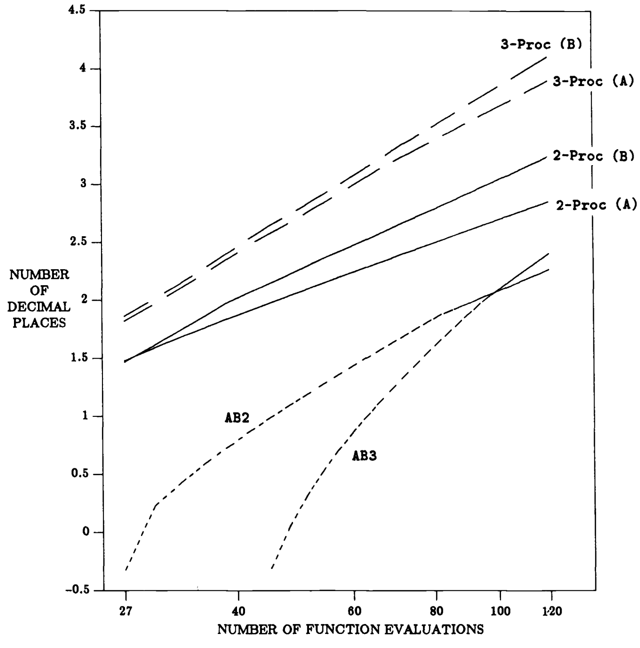


FIG. 8. *Problem* 2.

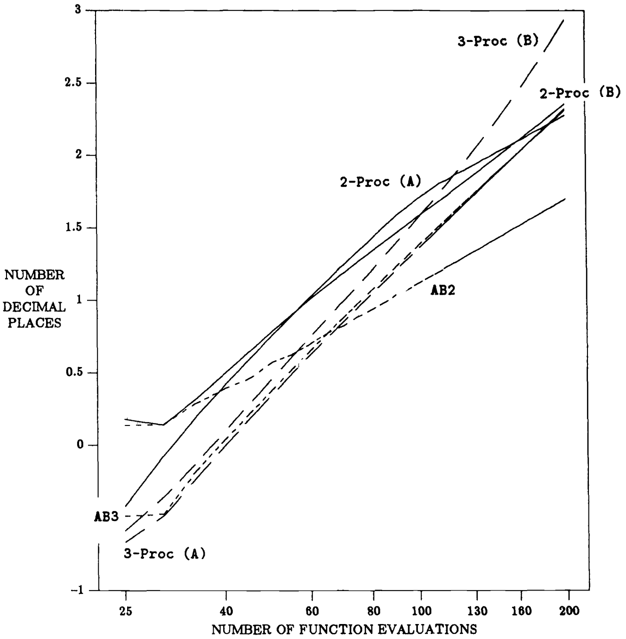


FIG. 9. *Problem* 3.

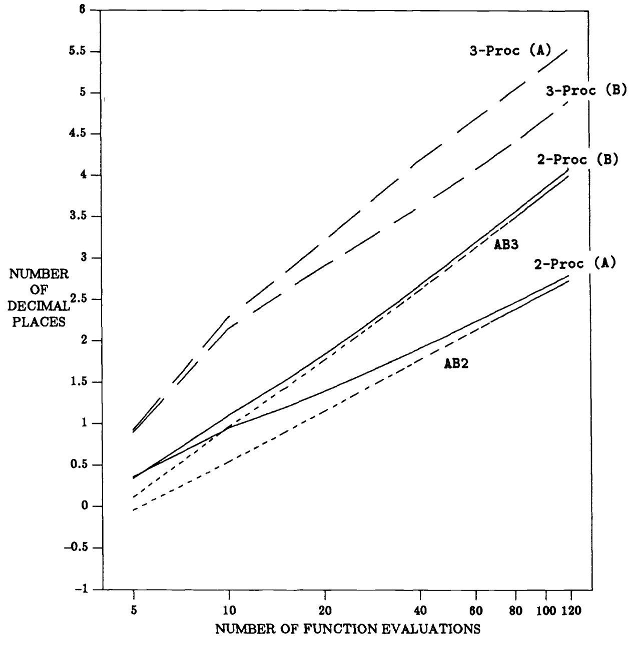


FIG. 10. *Problem* 4.

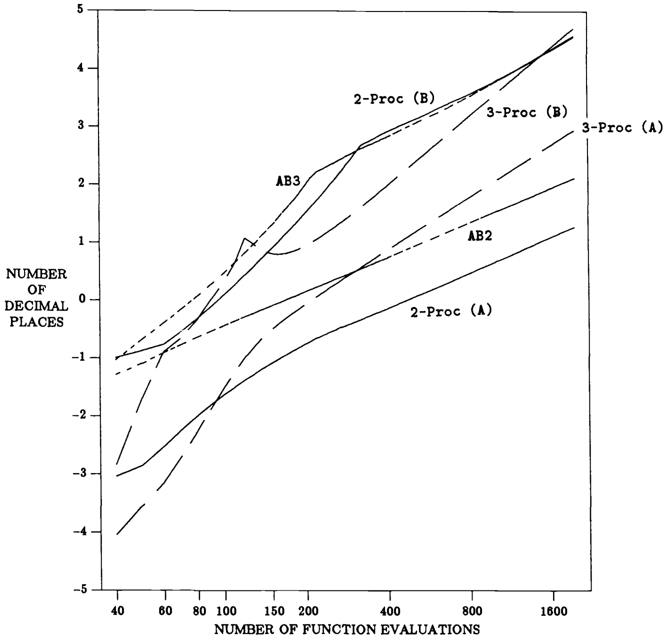


FIG. 11. *Problem* 5.

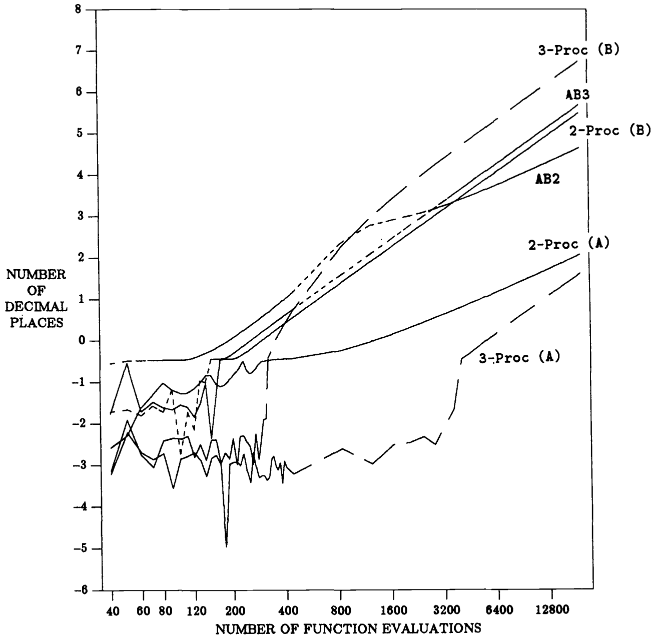


FIG. 12. *Problem* 6.

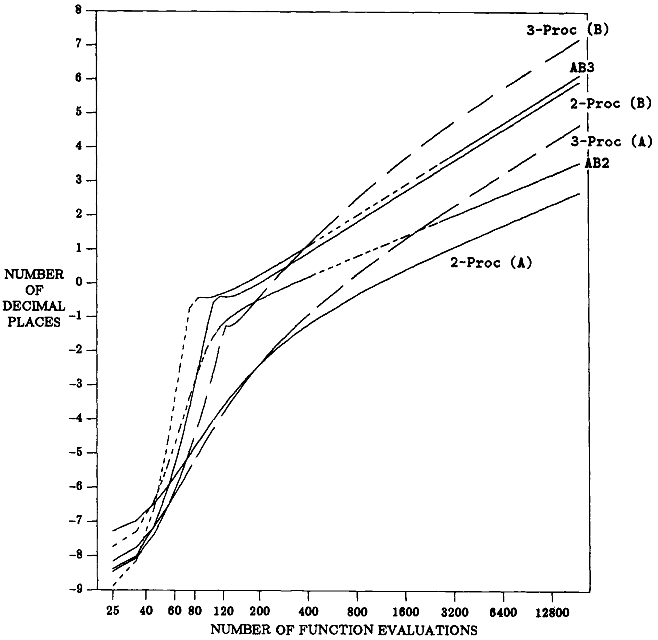


FIG. 13. *Problem* 7.

third-order Adams–Bashforth method, with a speedup of up to 3 in some cases (e.g., in problem 2 for an accuracy of two digits). Problem 6 shows the effect of imaginary eigenvalues on the two- and three-processor methods with unit circle stability regions. Both methods perform poorly on the imaginary axis. However, with a different choice of the parameters, both methods regain good accuracy.

7. Summary. In this paper we have derived a new family of one-stage q-processor parallel methods. The q-processor element has order q. This family of methods contains free parameters, which are at our disposal. For any number of processors q, there exists a unique set of parameters so that the stability region of the resulting method is essentially that of the Euler method. We discover that besides stability and local error analysis, we must also consider the coupling between time points for a block method. The effect of this coupling on the above methods with unit circle stability regions leads to an accelerated error growth on the imaginary axis of the $h\lambda$-plane. This accelerated error growth restricts the length of the integration interval for a given problem. Although the methods with unit circle stability regions are of restricted practical usage, the unique set of parameters is the starting point from which we obtain good one-stage parallel methods. By making modifications of the parameters around the unique set, we are able to obtain methods with relatively large stability regions, compared to the Adams–Bashforth methods, but now of better accuracy. The modified parameter methods of the two- and three-processor cases outperform the second- and third-order Adams–Bashforth methods, respectively, in both accuracy and stability. However, we have not been able to generalize this parameter modification to higher-order cases.

This paper also illustrates an important approach to obtaining parallel methods with good stability properties; we study zero-stable parallel methods with perfect power stability polynomials. Although the methods with unit circle stability regions studied in this paper are of restricted usage, they allow us to consider two-stage block methods with very little accelerated error growth. These two-stage methods also have unchanged stability regions as the order increases. They outperform the Adams PECE methods in both stability and accuracy by a sizeable margin. These methods are presented in [18].

REFERENCES

[1] L. G. BIRTA AND O. ABOU-RABIA, *Parallel block predictor-corrector methods for ODEs*, IEEE Trans. Comput., C36 (1987), pp. 299-311.

[2] J. C. BUTCHER, *On the convergence of numerical solutions to ordinary differential equations*, Math. Comp., 20 (1966), pp. 1-10.

[3] ———, *The numerical analysis of ordinary differential equations—Runge-Kutta and general linear methods*, John Wiley, New York, 1987.

[4] M. CHU AND H. HAMILTON, *Parallel solution of ODEs by multi-block methods*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 342-353.

[5] G. DAHLQUIST, *G-stability is equivalent to A-stability*, BIT, 18 (1978), pp. 384-401.

[6] G. DAHLQUIST AND R. JELTSCH, *Generalized disks of contractivity for explicit and implicit Runge-Kutta methods*, Computer Science Report TRITA-NA-7906, Royal Institute of Technology, Stockholm, Sweden, 1979.

[7] M. A. FRANKLIN, *Parallel solution of ordinary differential equations*, IEEE Trans. Comput., C-27 (1978), pp. 413-420.

[8] C. W. GEAR, *Parallel methods for ordinary differential equations*, Report No. UIUCDCS-R-87-1369, University of Illinois, Urbana, IL, 1987.

[9] E. HAIRER, S. P. NORSETT, AND G. WANNER, *Solving Ordinary Differential Equations I—Nonstiff Problems*, Springer-Verlag, Berlin, New York, 1987.

[10] P. J. VAN DER HOUWEN, *Construction of Integration Formulas for Initial Value Problems*, North-Holland, Amsterdam, 1977.

[11] T. E. HULL, W. H. ENRIGHT, B. M. FELLEN, AND A. E. SEDGWICK, *Comparing numerical methods for ordinary differential equations*, SIAM J. Numer. Anal., 9 (1972), pp. 603-637.

[12] R. JELTSCH AND O. NEVANLINNA, *Stability of explicit time discretizations for solving initial value problems*, Numer. Math., 37 (1981), pp. 61-91.

[13] W. L. MIRANKER AND W. LINIGER, *Parallel methods for the numerical integration of ordinary differential equations*, Math. Comp., 21 (1967), pp. 303-320.

[14] O. NEVANLINNA, *On the numerical integration of nonlinear initial value problems by linear multistep methods*, BIT, 17 (1977), pp. 58-71.

[15] L. F. SHAMPINE AND H. W. WATTS, *Block implicit one-step methods*, Math. Comp., 23 (1969), pp. 731-740.

[16] R. D. SKEEL, *Analysis of fixed-stepsize methods*, SIAM J. Numer. Anal., 13 (1976), pp. 664-685.

[17] H. W. TAM, *Parallel methods for the numerical solution of ordinary differential equations*, Report No. UIUCDCS-R-89-1516, University of Illinois, Urbana, IL, 1989.

[18] ———, *Two-stage parallel methods for ordinary differential equations*, SIAM J. Sci. Statist. Comput., this issue, pp. 1062-1084.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 5, pp. 1062-1084, September 1992

TWO-STAGE PARALLEL METHODS FOR THE NUMERICAL SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS*

H. W. TAM†

Abstract. This is a continuation of a previous work on finding parallel ODE methods with good stability regions. Based on the approach of zero-stable methods with perfect power stability polynomials, a family of two-stage block methods with stability regions essentially equal to that of the second-order Taylor series method is derived. The stability regions of these methods remain unchanged as the order increases. These methods also have very low interprocessor communication cost compared to existing PECE block methods.

Key words. two-stage, block methods, parallel processing, ordinary differential equations

AMS(MOS) subject classification. 65L05

1. Introduction. This is a continuation of the work on parallel ordinary differential equation (ODE) methods in [10]. We would like to study parallel methods for the numerical solution of the ODE

$$y(t_0)=\eta,$$
$$y'(t)=f(y(t)), \qquad t_0 \leqq t \leqq t_{\text{out}},$$

where y and $f \in \mathscr{R}^n$. A nonautonomous ODE of the form $z'(t)=f(t, z(t))$ can always be written in the above autonomous form. Hence the above form involves no loss of generality.

A family of one-stage block methods, the two-processor element of which is given by

$$\begin{bmatrix} y_{n+(3/2)} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} y_{n+(1/2)} \\ y_n \end{bmatrix} + h \begin{bmatrix} 2 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} f_{n+(1/2)} \\ f_n \end{bmatrix}, \tag{1.1}$$

has been studied in [10]. Equation (1.1) has the special property that its stability polynomial is equal to $[\xi-(1+\mu)]^2$ while the method remains zero-stable. (Usually a method whose stability polynomial is the power of another lacks zero-stability.) The stability region of (1.1) is equal to the interior of the unit circle centered at $(-1, 0)$ of the $h\lambda$-plane, together with the origin. This family of one-stage block methods has the peculiarity that the stability regions do not change with order. It has been shown in [9] that the best scaled stability region[1] one can achieve for explicit ODE methods is the unit circle. Therefore this upper bound is almost achieved by the above family of methods.

However, the coupling between each component of (1.1) renders the use of the method rather restrictive. If the global error of (1.1) satisfies

$$\begin{bmatrix} y_{n+(3/2)} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} y(t_{n+(3/2)}) \\ y(t_{n+1}) \end{bmatrix} + \begin{bmatrix} h^2 e_1(t_{n+(3/2)}) \\ h^2 e_2(t_{n+1}) \end{bmatrix} + \mathscr{O}(h^3), \tag{1.2}$$

* Received by the editors July 28, 1989; accepted for publication (in revised form) November 11, 1991. This research was supported in part by U.S. Department of Energy grant DEFG02-87ER25026 and National Science Foundation grant DMS 8703226.

† Department of Computer Science, University of Illinois, Urbana, Illinois 61801. Present address, Research and Development, Wolfram Research, Inc., 701 Budd Court, Campbell, California 95008.

[1] The scaled stability region of a method is defined to be its stability region divided by m, where m is the number of stages per step of the method. Scaled stability regions are the fair entities for the comparison of stability properties of different methods.

where $e_1(t)$, $e_2(t)$ are differentiable, it is possible to show that, after a linear transformation

$$\begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = T \begin{bmatrix} \tilde{e}_1 \\ \tilde{e}_2 \end{bmatrix},$$

$$\begin{bmatrix} \tilde{e}_1'(t) \\ \tilde{e}_2'(t) \end{bmatrix} = f_y(y(t)) \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \tilde{e}_1(t) \\ \tilde{e}_2(t) \end{bmatrix} + \begin{bmatrix} -\frac{1}{24} \\ -\frac{1}{4} \end{bmatrix} y'''(t).$$

The solutions of $e_1(t)$ and $e_2(t)$, when applied to $y' = \lambda y$, $y(0) = 1$, contain the component

$$e^{t\lambda}\left[-\frac{t}{24} - \frac{t^2\lambda}{8}\right]\lambda^3.$$

(For comparison, the leading term in the global error of a conventional ODE method is of the form $\mathcal{O}(t)\, e^{\lambda t}$). If $\lambda = -1$, the growth of t^2 is damped by e^{-t}. When $\lambda = i$, $|e^{it}| = 1$ and the growth of t^2 is undamped. We call this phenomenon an accelerated error growth. The above global error analysis demonstrates that (1.1) has good accuracy for eigenvalues on the negative real axis of the $h\lambda$-plane, but not for eigenvalues along the imaginary axis.

Method (1.1), nevertheless, is an example of a member of a broad class of formulas having stability polynomials that are perfect powers of simple ones and possessing good stability properties. Since two-stage methods have been proved to be useful in the sequential case (as in the Adams PECE methods), we would like to consider two-stage parallel methods. In this paper we show that there exists a family of zero-stable two-stage block methods that satisfies the aforementioned perfect power stability polynomial property. In § 2 we show that by using a one-stage method of the type discussed in [10] as the predictor, and by choosing an appropriate corrector, we are able to derive a family of zero-stable methods with stability polynomials equal to perfect powers of

$$\left[\xi - \left(\frac{\mu^2}{2} + \mu + 1\right)\right].$$

These methods have stability regions essentially identical to that of the second-order Taylor series method. However, the stability regions remain unchanged for all orders. (Stability regions of conventional ODE methods usually shrink as the order increases.) These methods also have the advantage of requiring much less interprocessor communication than existing predictor-corrector block methods. In § 3 we use global error analysis to show that the error functions of this new family of methods do not have as much accelerated growth as the family of (1.1). The global error analysis also suggests a unique way to calculate the predictor parameters leading to the above stability polynomials. In § 4 we discuss variants of the new family of two-stage methods. In § 5 we demonstrate how to enhance the accuracy of the methods by doing post-processing at the end of the integration. Numerical simulations in § 6 show that the new parallel methods outperform the Adams PECE family in both accuracy and stability by a sizable margin.

2. The corrector formula. In this section we would like to derive a family of two-stage formulas whose stability polynomials are perfect powers of simple ones. We believe that the derivation of the lowest-order formula serves as a pattern for the generalization to higher orders. Let us use the two-processor, one-stage block method of [10] as the predictor of our (as yet to be described) two-stage method:

$$\begin{bmatrix} y^p_{n+(3/2)} \\ y^p_{n+1} \end{bmatrix} = \begin{bmatrix} 1-b & b \\ 1-a & a \end{bmatrix} \begin{bmatrix} y_{n+(1/2)} \\ y_n \end{bmatrix} + h \begin{bmatrix} 2+\frac{b}{4} & -1+\frac{b}{4} \\ \frac{3}{4}+\frac{a}{4} & -\frac{1}{4}+\frac{a}{4} \end{bmatrix} \begin{bmatrix} f_{n+(1/2)} \\ f_n \end{bmatrix}. \tag{2.1}$$

As in the Adams–Moulton methods, we can use the now available values $f^p_{n+(3/2)} = f(t_{n+(3/2)}, y^p_{n+(3/2)})$ and $f^p_{n+1} = f(t_{n+1}, y^p_{n+1})$, as well as the stored quantities y_n, $y_{n+(1/2)}$, f_n, $f_{n+(1/2)}$ to compute the corrected values y_{n+1} and $y_{n+(3/2)}$. We find that making the corrector one order higher than the predictor is a good decision. Since the predictor (2.1) is second order, we would like to make the corrector formula third order. We will use f^p_{n+1} in the computation of y_{n+1}, and $f^p_{n+(3/2)}$ in the computation of $y_{n+(3/2)}$. The third-order formula to calculate y_{n+1} from $y_n, f^p_{n+1}, f_{n+(1/2)}, f_n$ is given by

$$\begin{aligned} y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} & f^p_{n+1} + (t-t_{n+1}) f^p_{n+1,n+(1/2)} \\ & + (t-t_{n+1})(t-t_{n+(1/2)}) f^p_{n+1,n+(1/2),n}\, dt \\ = y_n + h[\tfrac{1}{6} & f^p_{n+1} + \tfrac{2}{3} f_{n+(1/2)} + \tfrac{1}{6} f_n], \end{aligned} \tag{2.2}$$

where the divided differences $f^p_{n+1,n+(1/2),\dots}$ are evaluated from f^p_{n+1}, $f_{n+(1/2)}, \cdots$. Similarly, we obtain third-order formulas

$$y_{n+1} = y_{n+(1/2)} + h[\tfrac{5}{24} f^p_{n+1} + \tfrac{1}{3} f_{n+(1/2)} - \tfrac{1}{24} f_n], \tag{2.3}$$

$$y_{n+(3/2)} = y_n + h[\tfrac{3}{8} f^p_{n+(3/2)} + \tfrac{9}{8} f_{n+(1/2)}], \tag{2.4}$$

$$y_{n+(3/2)} = y_{n+(1/2)} + h[\tfrac{7}{18} f^p_{n+(3/2)} + \tfrac{5}{6} f_{n+(1/2)} - \tfrac{2}{9} f_n]. \tag{2.5}$$

Linear combinations of (2.2) and (2.3), and of (2.4) and (2.5) yield the corrector formula

$$\begin{aligned} \begin{bmatrix} y_{n+(3/2)} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} 1-b^* & b^* \\ 1-a^* & a^* \end{bmatrix} \begin{bmatrix} y_{n+(1/2)} \\ y_n \end{bmatrix} & + h \begin{bmatrix} \frac{7}{18}-\frac{b^*}{72} & 0 \\ 0 & \frac{5}{24}-\frac{a^*}{24} \end{bmatrix} \begin{bmatrix} f^p_{n+(3/2)} \\ f^p_{n+1} \end{bmatrix} \\ & + h \begin{bmatrix} \frac{5}{6}+\frac{7}{24}b^* & -\frac{2}{9}+\frac{2}{9}b^* \\ \frac{1}{3}+\frac{1}{3}a^* & -\frac{1}{24}+\frac{5}{24}a^* \end{bmatrix} \begin{bmatrix} f_{n+(1/2)} \\ f_n \end{bmatrix}, \end{aligned} \tag{2.6}$$

with a^*, b^* as free parameters. The coefficient matrix for the f^{p}'s is specifically chosen to be diagonal to reduce the communication costs associated with a parallel implementation of this formula.

We need to choose the parameters a, b, a^*, b^* so that the stability polynomial of (2.6) is a perfect power. Since the largest power of ξ in the stability polynomial of (2.6) is ξ^2, we would like to make this stability polynomial of the form $[\xi - (\frac{1}{2}\mu^2 + \mu + 1)]^2$. The choice $[\xi - (\frac{1}{2}\mu^2 + \mu + 1)]^2$ gives a method of high formal order [8]. It turns out that for

$$a^* = 1, \quad b^* = 0, \quad a = 5, \quad b = \tfrac{4}{7}$$

or

$$a^* = 17, \quad b^* = 16, \quad a = 5, \quad b = 28,$$

the stability polynomial of (2.6) is indeed $[\xi - (\frac{1}{2}\mu^2 + \mu + 1)]^2$. There are also some other uninteresting parameter choices such that the stability polynomial of (2.6) is also a perfect square. These cases are not discussed in this paper.

If we apply our two-processor, two-stage block method to the simple test problem $y'(t)=0$, we get

$$\begin{bmatrix} y_{n+(3/2)} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} 1-b^* & b^* \\ 1-a^* & a^* \end{bmatrix} \begin{bmatrix} y_{n+(1/2)} \\ y_n \end{bmatrix}.$$

If $a^*=17$ and $b^*=16$,

$$\begin{aligned} \begin{bmatrix} 1-b^* & b^* \\ 1-a^* & a^* \end{bmatrix} &= \begin{bmatrix} 1-b^* & b^* \\ -b^* & 1+b^* \end{bmatrix} \\ &= \begin{bmatrix} 1 & 1 \\ 1 & \frac{1+b^*}{b^*} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & \frac{1+b^*}{b^*} \end{bmatrix}^{-1}, \end{aligned} \tag{2.7}$$

so that

$$\begin{bmatrix} y_{n+(3/2)} \\ y_{n+1} \end{bmatrix}$$

becomes unbounded as n increases. Therefore the only possibly useful set of parameters is $a=5$, $b=\frac{4}{7}$, $a^*=1$, $b^*=0$.

The corresponding three-processor, two-stage block method, where the second stage is derived in such a way that $y_{n+(i/3)}$ depends on $f^p_{n+(i/3)}$, $i=3, 4, 5$, is given by

$$\begin{aligned} \begin{bmatrix} y^p_{n+(5/3)} \\ y^p_{n+(4/3)} \\ y^p_{n+1} \end{bmatrix} &= \begin{bmatrix} 1-c_1-c_2 & c_2 & c_1 \\ 1-b_1-b_2 & b_2 & b_1 \\ 1-a_1-a_2 & a_2 & a_1 \end{bmatrix} \begin{bmatrix} y_{n+(2/3)} \\ y_{n+(1/3)} \\ y_n \end{bmatrix} \\ &\quad + h \begin{bmatrix} \frac{19}{4}+\frac{c_1}{9}+\frac{5c_2}{36} & -6+\frac{4c_1}{9}+\frac{2c_2}{9} & \frac{9}{4}+\frac{c_1}{9}-\frac{c_2}{36} \\ \frac{19}{9}+\frac{b_1}{9}+\frac{5b_2}{36} & -\frac{20}{9}+\frac{4b_1}{9}+\frac{2b_2}{9} & \frac{7}{9}+\frac{b_1}{9}-\frac{b_2}{36} \\ \frac{23}{36}+\frac{a_1}{9}+\frac{5a_2}{36} & -\frac{4}{9}+\frac{4a_1}{9}+\frac{2a_2}{9} & \frac{5}{36}+\frac{a_1}{9}-\frac{a_2}{36} \end{bmatrix} \begin{bmatrix} f_{n+(2/3)} \\ f_{n+(1/3)} \\ f_n \end{bmatrix}, \end{aligned} \tag{2.8}$$

$$\begin{aligned} \begin{bmatrix} y_{n+(5/3)} \\ y_{n+(4/3)} \\ y_{n+1} \end{bmatrix} &= \begin{bmatrix} 1-c_1^*-c_2^* & c_2^* & c_1^* \\ 1-b_1^*-b_2^* & b_2^* & b_1^* \\ 1-a_1^*-a_2^* & a_2^* & a_1^* \end{bmatrix} \begin{bmatrix} y_{n+(2/3)} \\ y_{n+(1/3)} \\ y_n \end{bmatrix} \\ &\quad + h \begin{bmatrix} \frac{5}{16}-\frac{c_2^*}{720} & 0 & 0 \\ 0 & \frac{2}{9}-\frac{b_2^*}{288} & 0 \\ 0 & 0 & \frac{1}{8}-\frac{a_2^*}{72} \end{bmatrix} \begin{bmatrix} f^p_{n+(5/3)} \\ f^p_{n+(4/3)} \\ f^p_{n+1} \end{bmatrix} \\ &\quad + h \begin{bmatrix} \frac{13}{8}+\frac{c_1^*}{9}+\frac{11c_2^*}{72} & -\frac{21}{16}+\frac{4c_1^*}{9}+\frac{29c_2^*}{144} & \frac{3}{8}+\frac{c_1^*}{9}-\frac{7c_2^*}{360} \\ \frac{7}{9}+\frac{b_1^*}{9}+\frac{23b_2^*}{144} & -\frac{4}{9}+\frac{4b_1^*}{9}+\frac{7b_2^*}{36} & \frac{1}{9}+\frac{b_1^*}{9}-\frac{5b_2^*}{288} \\ \frac{19}{72}+\frac{a_1^*}{9}+\frac{13a_2^*}{72} & -\frac{5}{72}+\frac{4a_1^*}{9}+\frac{13a_2^*}{72} & \frac{1}{72}+\frac{a_1^*}{9}-\frac{a_2^*}{72} \end{bmatrix} \begin{bmatrix} f_{n+(2/3)} \\ f_{n+(1/3)} \\ f_n \end{bmatrix}. \end{aligned} \tag{2.9}$$

As in the two-processor case, we now want to choose the parameters to make the stability polynomial of the form $[\xi-(\frac{1}{2}\mu^2+\mu+1)]^3$. Due to the large number of parameters and the complexity of the stability polynomial, we believe that the choice of the parameters requires some intuition. The two-processor, two-stage method suggests that an appropriate partial choice is $a_1^*=1$, $a_2^*=0$, $b_1^*=0$, $b_2^*=1$, $c_1^*=c_2^*=0$, which it is. However, this time more than a unique set of predictor parameters satisfy our requirement. For example, both

$$a_1=19, \quad a_2=-27, \quad b_1=\frac{45}{7}, \quad b_2=-8, \quad c_1=\frac{18}{5}, \quad c_2=-\frac{27}{5}$$

and

$$a_1=7, \quad a_2=-3, \quad b_1=-\frac{3}{7}, \quad b_2=\frac{40}{7}, \quad c_1=-\frac{6}{5}, \quad c_2=\frac{21}{5}$$

make the stability polynomial equal to

$$\left[\xi-\left(\frac{\mu^2}{2}+\mu+1\right)\right]^3.$$

We do not know yet how many more possible choices of the a's, b's, and c's there are.

It turns out that the stability regions of the proposed family of two-stage block methods are equal to the interior of that of the second-order Taylor series method (also the lowest-order Adams PECE method), together with the origin. What is interesting is that the stability regions do not shrink as we increase the order. Thus the new two-stage block methods have stability properties essentially identical to those of the lowest-order Adams PECE method. This is a significant improvement over the family of Adams PECE methods because the stability regions of the Adams PECE methods shrink by a large amount as one increases the order [6].

We will study the error growth of this two-stage family of methods in the next section. This will assist us in finding the right predictor parameters, the number of which is enormous as one increases the number of processors.

3. Global error analysis of the two-stage block method. In § 1 we have demonstrated that the one-stage block methods with stability polynomials equal to $[\xi-(1+\mu)]^q$ are not practical because of accelerated error growth. In this section we use global error analysis to show that accelerated error growth no longer plays a significant role in the two-stage case. This analysis also gives us a clue to compute the method parameters.

As in the previous section, we want to investigate the behavior of the two-processor case (with the hope that the conclusions generalize). The derivation of (2.6) indicates that the corrector formula for the two-processor, two-stage method is of the third order. Based on global error theory for ODE methods [3], it is reasonable to assume that, since the stability polynomial has no extraneous root at $\mu=0$,

$$\begin{bmatrix} y_{n+(3/2)} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} y(t_{n+(3/2)}) \\ y(t_{n+1}) \end{bmatrix} + \begin{bmatrix} h^3 e_1(t_{n+(3/2)}) \\ h^3 e_2(t_{n+1}) \end{bmatrix} + \begin{bmatrix} h^4 g_1(t_{n+(3/2)}) \\ h^4 g_2(t_{n+1}) \end{bmatrix} + \mathcal{O}(h^5), \tag{3.1}$$

where the error functions $e_1(t)$, $e_2(t)$ as well as $g_1(t)$, $g_2(t)$ are differentiable, and $e_1(t)$, $e_2(t)$ are different from those of the one-stage methods in § 1. With the parameters $a=5$, $b=\frac{4}{7}$, $a^*=1$, $b^*=0$, if we substitute (3.1) into the corrector (2.6), subtract the Taylor series

$$\begin{bmatrix} y(t_{n+(3/2)}) \\ y(t_{n+1}) \end{bmatrix} = \begin{bmatrix} y(t_{n+(1/2)}) \\ y(t_n) \end{bmatrix} + \begin{bmatrix} hy'(t_{n+(1/2)}) \\ hy'(t_n) \end{bmatrix} + \begin{bmatrix} \frac{h^2}{2} y''(t_{n+(1/2)}) \\ \frac{h^2}{2} y''(t_n) \end{bmatrix}$$
$$+ \begin{bmatrix} \frac{h^3}{6} y'''(t_{n+(1/2)}) \\ \frac{h^3}{6} y'''(t_n) \end{bmatrix} + \begin{bmatrix} \frac{h^4}{24} y^{IV}(t_{n+(1/2)}) \\ \frac{h^4}{24} y^{IV}(t_n) \end{bmatrix} + \mathcal{O}(h^5)$$

from both sides of the resulting equation, and express all solution and function values at the point t_n, then the error functions $e_1(t)$ and $e_2(t)$ satisfy

$$\begin{aligned} h^4 e_1'(t_n) &= \tfrac{1}{36} h^4 y^{IV}(t_n) + h^4 \tfrac{7}{18}(-\tfrac{2}{7}) f_y(y(t_n)) y'''(t_n) \\ &\quad + h^4 \tfrac{7}{18} f_y(y(t_n))[\tfrac{3}{7} e_1(t_n) + \tfrac{4}{7} e_2(t_n)] \\ &\quad + h^4 \tfrac{5}{6} f_y(y(t_n)) e_1(t_n) - h^4 \tfrac{2}{9} f_y(y(t_n)) e_2(t_n) + \mathcal{O}(h^5), \end{aligned} \tag{3.2}$$

and

$$h^4 e_2'(t_n) = h^4 \tfrac{1}{6} f_y(y(t_n))[5e_2(t_n) - 4e_1(t_n)] + h^4 \tfrac{2}{3} f_y(y(t_n)) e_1(t_n) \\ + h^4 \tfrac{1}{6} f_y(y(t_n)) e_2(t_n) + \mathcal{O}(h^5). \tag{3.3}$$

Not surprisingly, the local errors of (2.1) and (2.6),

$$\begin{bmatrix} -\frac{2}{7} h^3 y'''(\tau_1) \\ 0 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \frac{1}{36} h^4 y^{IV}(\tau_2) \\ 0 \end{bmatrix},$$

are embedded in (3.2) and (3.3).

If we fix t_n in (3.2) and (3.3) and let $h \to 0$, we can drop the subscript n and obtain

$$\begin{bmatrix} e_1'(t) \\ e_2'(t) \end{bmatrix} = f_y(y(t)) \left(\begin{bmatrix} \frac{7}{18} & 0 \\ 0 & \frac{1}{6} \end{bmatrix} \begin{bmatrix} \frac{3}{4} & \frac{4}{7} \\ -4 & 5 \end{bmatrix} + \begin{bmatrix} \frac{5}{6} & -\frac{2}{9} \\ \frac{2}{3} & \frac{1}{6} \end{bmatrix} \right) \begin{bmatrix} e_1(t) \\ e_2(t) \end{bmatrix} \\ + f_y(y(t)) \begin{bmatrix} \frac{7}{18} & 0 \\ 0 & \frac{1}{6} \end{bmatrix} \begin{bmatrix} -\frac{2}{7} \\ 0 \end{bmatrix} y'''(t) + \begin{bmatrix} \frac{1}{36} \\ 0 \end{bmatrix} y^{IV}(t). \tag{3.4}$$

For comparison, the corresponding corrector formula (2.6) is given by

$$\begin{bmatrix} y_{n+(3/2)} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} y_{n+(1/2)} \\ y_n \end{bmatrix} + h \begin{bmatrix} \frac{7}{18} & 0 \\ 0 & \frac{1}{6} \end{bmatrix} \begin{bmatrix} f^p_{n+(3/2)} \\ f^p_{n+1} \end{bmatrix} \\ + h \begin{bmatrix} \frac{5}{6} & -\frac{2}{9} \\ \frac{2}{3} & \frac{1}{6} \end{bmatrix} \begin{bmatrix} f_{n+(1/2)} \\ f_n \end{bmatrix}. \tag{3.5}$$

The analogous error function equation for the second-order Adams–Bashforth–third-order Adams–Moulton PECE method has the form

$$e'(t) = f_y(y(t)) e(t) - [\gamma_3^* y^{IV}(t) + \beta_{3,0}^* \gamma_2 f_y(y(t)) y'''(t)], \tag{3.6}$$

where $\gamma_3^* = -\frac{1}{24}$, $\beta_{3,0}^* = \frac{5}{12}$, $\gamma_2 = \frac{5}{12}$. Notice the similarity between the arrangements of the coefficients of (3.4) and (3.6). It is obvious that (3.4) can be easily generalized to error functions of the higher-order two-stage block methods.

We are concerned that the propagation matrix

$$\begin{bmatrix} \frac{7}{18} & 0 \\ 0 & \frac{1}{6} \end{bmatrix} \begin{bmatrix} \frac{3}{7} & \frac{4}{7} \\ -4 & 5 \end{bmatrix} + \begin{bmatrix} \frac{5}{6} & -\frac{2}{9} \\ \frac{2}{3} & \frac{1}{6} \end{bmatrix} \tag{3.7}$$

of the error functions $e_1(t)$, $e_2(t)$ in (3.4) may be defective and lead to an accelerated error growth on the imaginary axis similar to that of the one-stage method (1.1). Surprisingly, (3.7) is actually the identity matrix. Recall that the defective error propagation matrix of the one-stage method (1.1) leads to the failure of that method. Such a defectiveness is absent in the current two-stage methods.

At this point there appears to be a natural choice of parameters for our q-processor, two-stage block method. This natural set of parameters simultaneously makes the stability polynomial equal to

$$\left[\xi - \left(\frac{\mu^2}{2} + \mu + 1 \right) \right]^q,$$

and the propagation matrix of the error function equation equal to an identity matrix. Although the number of these parameters increases tremendously as q increases, we will see shortly that the task of computing them is not very difficult.

For the natural set of parameters, the error function equation (3.4) becomes

$$\begin{bmatrix} e_1'(t) \\ e_2'(t) \end{bmatrix} = f_y(y(t)) \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} e_1(t) \\ e_2(t) \end{bmatrix} + \begin{bmatrix} -\frac{1}{9} f_y(y(t)) y'''(t) + \frac{1}{36} y^{IV}(t) \\ 0 \end{bmatrix}. \tag{3.8}$$

The initial conditions are given by $e_1(t_0)=0$ and $e_2(t_0)=0$. Hence the component $e_2(t)$ is identically zero. Thus there is superconvergence for our two-processor, two-stage method at the gridpoints $t_1, t_2, \cdots, t_n, \cdots$. At the gridpoints, the two-processor method becomes fourth-order.

The error function equation provides us with a clue to calculate the natural choice of parameters for the family of two-stage block methods. Let us denote the predictor and corrector of the q-processor, two-stage block method by, respectively,

$$Y^p_{n+1}=A\cdot Y_n+B\cdot hF_n \tag{3.9}$$

and

$$Y_{n+1}=A^*\cdot Y_n+B^*\cdot hF_n+C^*\cdot hF^{\mathrm{p}}_{n+1}, \tag{3.10}$$

where A, B, A^*, B^*, and C^* are $q\times q$ matrices. Previous experience suggests that we should choose $A^*=I_{q\times q}$, the $q\times q$ identity matrix, and C^* a diagonal matrix. By making the corrector order $q+1$ we automatically fix B^*. The error analysis suggests that we compute the parameters $a_1, a_2, \cdots, b_1, b_2, \cdots, c_1, c_2, \cdots$ by setting the propagation matrix

$$C^*\cdot A+B^*=I_{q\times q}. \tag{3.11}$$

Thus we can solve for the a's, b's, $\cdots$ whenever C^* is nonsingular. (See Theorem 4.1.)

As an example, let us look at the three-processor, two-stage block method (2.8)-(2.9). Making $A^*=I_{3\times 3}$ transforms the second stage, (2.9), into

$$\begin{bmatrix} y_{n+(5/3)} \\ y_{n+(4/3)} \\ y_{n+1} \end{bmatrix}=\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} y_{n+(2/3)} \\ y_{n+(1/3)} \\ y_n \end{bmatrix}+h\begin{bmatrix} \frac{5}{16} & 0 & 0 \\ 0 & \frac{7}{32} & 0 \\ 0 & 0 & \frac{1}{8} \end{bmatrix}\begin{bmatrix} f^p_{n+(5/3)} \\ f^p_{n+(4/3)} \\ f^p_{n+1} \end{bmatrix}$$
$$+h\begin{bmatrix} \frac{13}{8} & -\frac{21}{16} & \frac{3}{8} \\ \frac{15}{16} & -\frac{1}{4} & \frac{3}{32} \\ \frac{3}{8} & \frac{3}{8} & \frac{1}{8} \end{bmatrix}\begin{bmatrix} f_{n+(2/3)} \\ f_{n+(1/3)} \\ f_n \end{bmatrix}.$$

Equating

$$\begin{bmatrix} \frac{5}{16} & 0 & 0 \\ 0 & \frac{7}{32} & 0 \\ 0 & 0 & \frac{1}{8} \end{bmatrix}\begin{bmatrix} 1-c_1-c_2 & c_2 & c_1 \\ 1-b_1-b_2 & b_2 & b_1 \\ 1-a_1-a_2 & a_2 & a_1 \end{bmatrix}+\begin{bmatrix} \frac{13}{8} & -\frac{21}{16} & \frac{3}{8} \\ \frac{15}{16} & -\frac{1}{4} & \frac{3}{32} \\ \frac{3}{8} & \frac{3}{8} & \frac{1}{8} \end{bmatrix}=\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

gives $a_1=7$, $a_2=-3$, $b_1=-\frac{3}{7}$, $b_2=\frac{40}{7}$, $c_1=-\frac{6}{5}$, $c_2=\frac{21}{5}$. The corresponding stability polynomial is indeed

$$\left[\xi-\left(\frac{\mu^2}{2}+\mu+1\right)\right]^3.$$

We also mentioned in § 2 that $a_1=19$, $a_2=27$, $b_1=\frac{45}{7}$, $b_2=-8$, $c_1=\frac{18}{5}$, $c_2=-\frac{27}{5}$ is an alternative set of parameters leading to the same stability polynomial. This alternative set makes the error propagation matrix

$$C^*\cdot A+B^*=\begin{bmatrix} \frac{5}{2} & -3 & \frac{3}{2} \\ \frac{3}{2} & -2 & \frac{3}{2} \\ \frac{3}{2} & -3 & \frac{5}{2} \end{bmatrix},$$

with eigenvalues also equal to $\{1, 1, 1\}$. However, this alternative error propagation matrix is defective, with a consequence of the error functions having an accelerated growth. In a straightforward way, one can now easily compute the predictor parameters for the four-processor, two-stage block method to be

$$a_1=\tfrac{83}{7},\quad a_2=-\tfrac{32}{7},\quad a_3=-\tfrac{12}{7},\quad b_1=\tfrac{32}{67},\quad b_2=\tfrac{255}{67},\quad b_3=\tfrac{160}{67},\quad c_1=\tfrac{380}{277},$$
$$c_2=-\tfrac{1728}{277},\quad c_3=\tfrac{4185}{277},\quad d_1=\tfrac{2560}{817},\quad d_2=-\tfrac{11060}{817},\quad d_3=\tfrac{17472}{817}.$$

With the natural choice of parameters for the two-stage block methods determined, one can show that the global error functions (for q-processors) satisfy

$$\begin{bmatrix} e_1'(t) \\ \vdots \\ e_q'(t) \end{bmatrix} = f_y(y(t)) \begin{bmatrix} e_1(t) \\ \vdots \\ e_q(t) \end{bmatrix} + C^*\mathbf{v} f_y(y(t)) y^{(q+1)}(t) + \mathbf{v}^* y^{(q+2)}(t), \tag{3.12}$$

where $\mathbf{v}$ and $\mathbf{v}^*$ are the coefficient vectors of the local errors of the predictor and corrector, respectively. For $q=3$ and 4, the driving terms in (3.12) are given by

$$\begin{bmatrix} -\frac{23}{648} \\ -\frac{17}{2592} \\ -\frac{1}{1296} \end{bmatrix} f_y(y(t)) y^{IV}(t) + \begin{bmatrix} \frac{23}{3240} \\ \frac{17}{12960} \\ \frac{1}{6480} \end{bmatrix} y^V(t)$$

and

$$\begin{bmatrix} -\frac{29}{3200} \\ -\frac{23}{9600} \\ -\frac{1}{2400} \\ 0 \end{bmatrix} f_y(y(t)) y^V(t) + \begin{bmatrix} \frac{29}{19200} \\ \frac{23}{57600} \\ \frac{1}{14400} \\ 0 \end{bmatrix} y^{VI}(t),$$

respectively. In general, it is possible to show that the order of the q-processor, two-stage block method at gridpoints $t_n, t_{n+1}, \cdots$ is given by

$$\begin{cases} q+1, & q \text{ odd}, \\ q+2, & q \text{ even}. \end{cases}$$

When q is even, the last component $e_q(t)$ in (3.12) turns out to be zero. We will investigate the form of the dominating term in the global error in this situation. As usual, the two-processor case may give us an idea of the general behavior. Therefore, we will compare the global error of the two-processor method, when expanded in a power series of h, to that of the Adams PECE method of comparable order. In this way we can be certain that there is no hidden error growth.

One can extend (3.1) to (for $a=5$ and $b=\frac{4}{7}$)

$$\begin{aligned} \begin{bmatrix} y_{n+(3/2)} \\ y_{n+1} \end{bmatrix} &= \begin{bmatrix} y(t_{n+(3/2)}) \\ y(t_{n+1}) \end{bmatrix} + \begin{bmatrix} h^3 e_1(t_{n+(3/2)}) \\ 0 \end{bmatrix} + \begin{bmatrix} h^4 g_1(t_{n+(3/2)}) \\ h^4 g_2(t_{n+1}) \end{bmatrix} \\ &\quad + \begin{bmatrix} h^5 w_1(t_{n+(3/2)}) \\ h^5 w_2(t_{n+1}) \end{bmatrix} + \mathcal{O}(h^6), \end{aligned} \tag{3.13}$$

with $w_1(t)$ and $w_2(t)$ being differentiable. By a tedious generalization of the derivations of (3.2) and (3.3), and making use of the now available function $e_1(t)$, one can prove that $g_1(t)$ and $g_2(t)$ in (3.13) satisfy

$$\begin{aligned} \begin{bmatrix} g_1'(t) \\ g_2'(t) \end{bmatrix} &= f_y(y(t)) \begin{bmatrix} g_1(t) \\ g_2(t) \end{bmatrix} + \begin{bmatrix} \frac{7}{960} \\ \frac{1}{2880} \end{bmatrix} y^V(t) + \begin{bmatrix} -\frac{5}{576} \\ -\frac{1}{576} \end{bmatrix} f_y(y(t)) y^{IV}(t) \\ &\quad + \begin{bmatrix} \frac{5}{6} \\ \frac{1}{3} \end{bmatrix} f_y(y(t)) f_y(y(t)) e_1(t) \\ &\quad + \begin{bmatrix} f_{yy}(y(t)) y'(t)(-\frac{1}{9} y'''(t) + \frac{1}{6} e_1(t)) - \frac{1}{2} e_1''(t) \\ -\frac{1}{3} f_{yy}(y(t)) y'(t) e_1(t) \end{bmatrix}. \end{aligned}$$

We would like to compare the global error expansion of the two-processor, two-stage method in power series form to those of the Adams PECE methods when applied to a linear test problem $y'(t)=\lambda y$. From the knowledge of the $e_i(t)$'s and $g_i(t)$'s, the two-processor, two-stage method has an error expansion of the form

$$\begin{bmatrix} y_{n+(3/2)} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} y(t_{n+(3/2)}) \\ y(t_{n+1}) \end{bmatrix} + \begin{bmatrix} h^3\, e^{\lambda t_{n+(3/2)}} C_1 \lambda^4 t_{n+(3/2)} \\ 0 \end{bmatrix} \tag{3.14}$$
$$+ \begin{bmatrix} h^4\, e^{\lambda t_{n+(3/2)}} [C_2\lambda^6 t_{n+(3/2)}^2 + C_3\lambda^5 t_{n+(3/2)} + C_4\lambda^4] \\ h^4\, e^{\lambda t_{n+1}} [\hat{C}_2\lambda^6 t_{n+1}^2 + \hat{C}_3\lambda^5 t_{n+1}] \end{bmatrix} + \cdots .$$

The fourth-order Adams PECE method has a global error expansion of the form

$$y_{n+1} = y(t_{n+1}) + h^4\, e^{\lambda t_{n+1}} \tilde{C}_1 \lambda^5 t_{n+1} + \cdots .$$

The second component of (3.14) shows that even though the new two-processor, two-stage method is of order 4 at gridpoints, its global error is worse than that of the Adams PECE method with a fourth-order corrector, at least when $|\lambda t| > 1$. (One could think of the order of the two-processor, two-stage method as being 4^-.) This new method, however, outperforms the third-order Adams PECE method in accuracy because the $h^3\, e^{\lambda t}\mathcal{O}(\lambda^4 t)$ term in the global error of the block method is eliminated. Thus in general, the q-processor, two-stage method is more accurate than the $(q+1)$th-order Adams PECE method. (For the case when q is odd, the block method has a smaller error constant for the leading term of the global error.) Moreover, as q increases, the stability regions of the new methods do not shrink, while the stability regions of the Adams PECE methods shrink by a large amount. The stability regions for the first few elements of the Adams PECE family are plotted against that of the new two-stage block methods in Fig. 1. Based on the above discussion, we believe that the proposed family of two-stage parallel methods outperforms the Adams PECE

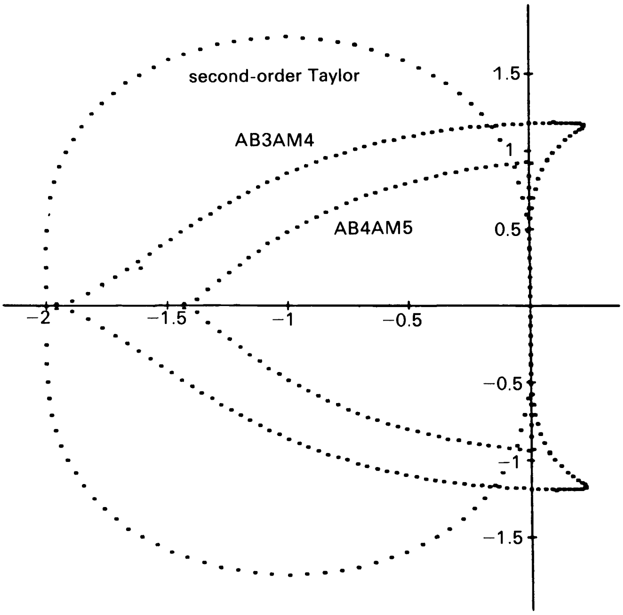


FIG. 1. *Stability regions of Adams* PECE *methods.*

family (a widely used class of serial methods) with a good speedup when executed on a parallel computer. Numerical simulations in § 6 appear to agree with this claim.

We would also like to discuss the implementation issues of the new family of two-stage block methods for a parallel computer. Existing two-stage parallel ODE methods, such as the predictor-corrector block methods in Shampine and Watts [7] or the predictor-corrector multiblock methods in Chu and Hamilton [2], all require two broadcasts per step. In these methods one has to share the function values of the predicted values computed in each processor. In the two-stage block methods in this paper, each processor uses only the function value of the predicted value computed by itself. Thus the number of broadcasts is cut by one-half. One should be aware that the amount of data to be communicated after the prediction, should it be necessary, and the amount of data communicated after the correction are not the same. In other existing parallel methods where one needs to communicate after the prediction, one broadcasts the function values of the intermediate solution values only. After the correction, however, one broadcasts both the solution values and their derivatives. Thus the amount of data broadcasted after the correction is twice that after the prediction. However, it takes startup overhead to initiate a broadcast in a parallel computer. Therefore the interprocessor communication time of the two-stage block methods in this paper is less than two-thirds of that of existing PECE parallel methods with the same number of saved values.

The way the new two-stage block methods are derived makes it easily transformable to a variable-step fashion. In a variable-step implementation of an ODE method, one needs to compute new method coefficients as the stepsize is changed. Shampine and Gordon [6] have derived an efficient way to evaluate the coefficients of the Adams PECE methods. One can derive a similar algorithm for the new two-stage block methods. To share this computational overhead, each processor can evaluate its own set of coefficients. Thus for the same number of previous time points, the amount of work each processor incurs is approximately equal to that of the Adams PECE methods. Since this load can be evenly distributed, the computational overhead of the new block methods is not much more than that of the Adams PECE methods. If we compare the two methods by order, the new block methods even have a smaller computational overhead per processor than the Adams PECE methods of the same order. Moreover, there are fewer step ratios in the block methods.

4. Variants of the two-stage block methods. We have abbreviated the q-processor, two-stage block method studied in § 3 by

$$Y_{n+1}^{p} = A \cdot Y_n + B \cdot hF_n, \tag{4.1}$$

$$Y_{n+1} = A^* \cdot Y_n + B^* \cdot hF_n + C^* \cdot hF_{n+1}^{p}. \tag{4.2}$$

Equations (4.1) and (4.2) represent, in fact, a whole class of methods of which the two-stage family of methods in the previous section are only some. It is interesting to study whether there exists any other element in the class of (4.1) and (4.2) that possesses a similar stability property, namely, that its stability polynomial is the perfect power of that of a simple method.

As in § 2, the complexity of the stability polynomial of (4.1) and (4.2) indicates that some insight is necessary. Previous experience in both the one-stage and two-stage methods suggests that $A^* = I_{q \times q}$ may be required. The condition (3.11) also turns out to be crucial. With these criteria it is possible to prove the following.

THEOREM 4.1. *Given an order $q+1$, $q \geq 0$ corrector for the two-stage block method* (4.1)-(4.2) *such that $A^* = I_{q\times q}$ and C^* is nonsingular, if A solves $C^* \cdot A + B^* = I_{q\times q}$, then*

(1) *the predictor* (4.1) *satisfies the zeroth order-condition, i.e., $Aw_1 = w_1$, where $w_1 = [1, \cdots, 1]^T$;*

(2) *if* (4.1) *is of order q, then B is uniquely determined, and the resulting stability polynomial is of the form*

$$\left[\xi - \left(\frac{\mu^2}{2} + \mu + 1\right)\right]^q.$$

Proof. The first order-condition for the corrector requires that $(B^* + C^*)w_1 = w_1$, which can be rewritten as $C^{*-1}(I - B^*)w_1 = Aw_1 = w_1$.

Since A satisfies the zeroth order-condition, if the predictor is of order q, B is uniquely determined by classical results on linear multistep methods.

The two-stage method, when applied to $y' = \lambda y$, is equivalent to

$$Y_{n+1} = [I + \mu B^* + \mu C^* \cdot (A + \mu B)]Y_n.$$

A necessary and sufficient condition for the stability polynomial of the two-stage method to equal

$$\left[\xi - \left(\frac{\mu^2}{2} + \mu + 1\right)\right]^q$$

is that A and B make

$$\left[\left(1 + \mu + \frac{\mu^2}{2}\right)I - (I + \mu(B^* + C^* \cdot A) + \mu^2 C^* \cdot B)\right]^q \tag{4.3}$$

equal to the zero matrix. The necessary condition is readily seen by the Cayley-Hamilton theorem. Conversely, if A and B make (4.3) equal to zero, then

$$\frac{\mu^2}{2} + \mu + 1$$

is the only eigenvalue of $I + \mu B^* + \mu C^* \cdot (A + \mu B)$. Hence the stability polynomial of the two-stage method must be

$$\left[\xi - \left(\frac{\mu^2}{2} + \mu + 1\right)\right]^q$$

by counting the number of eigenvalues.

Let

$$W(\mu) = \begin{bmatrix} e^{((q-1)/q)\mu} \\ \vdots \\ e^{(1/q)\mu} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 + (\frac{q-1}{q})\mu + \cdots + (\frac{q-1}{q})^{q-1}\frac{\mu^{q-1}}{(q-1)!} \\ \vdots \\ 1 + (\frac{1}{q})\mu + \cdots + (\frac{1}{q})^{q-1}\frac{\mu^{q-1}}{(q-1)!} \\ 1 \end{bmatrix} + \mathcal{O}(\mu^q).$$

Since $C^* \cdot A + B^* = I_{q\times q}$, (4.3) becomes

$$[(\tfrac{1}{2}I - C^* \cdot B)\mu^2]^q.$$

If the predictor is of order q, the $(q+1)$th order-condition of the two-stage pair implies that

$$e^{\mu}W(\mu)=[(1+\mu)I+\mu^2C^*\cdot B]W(\mu)+\mathcal{O}(\mu^{q+2})$$

$$\Rightarrow(\tfrac{1}{2}I-C^*\cdot B)W(\mu)=\mathcal{O}(\mu)W(\mu)+\mathcal{O}(\mu^q)$$

$$\Rightarrow(\tfrac{1}{2}I-C^*\cdot B)^qW(\mu)=\mathcal{O}(\mu^q),$$

which, after equating coefficients of powers of μ, gives

$$(\tfrac{1}{2}I-C^*\cdot B)^q w_i(\mu)=0,$$

where

$$w_1=\begin{bmatrix}1\\ \vdots\\ 1\\ 1\end{bmatrix},\quad w_i=\begin{bmatrix}(\frac{q-1}{q})^{i-1}\\ \vdots\\ (\frac{1}{q})^{i-1}\\ 0\end{bmatrix},\quad i=2,\cdots,q.$$

Since the w_i's are linearly independent, $(\frac{1}{2}I-C^*\cdot B)^q=0$. Thus the stability polynomial of the two-stage method is equal to

$$\left[\xi-\left(\frac{\mu^2}{2}+\mu+1\right)\right]^q. \qquad \square$$

Example 4.1. With predictor parameters

$$a_1=7,\quad a_2=-3,\quad b_1=-21,\quad b_2=16,\quad c_1=42,\quad c_2=-39 \tag{4.4}$$

for (2.8), the coefficient matrices of the following corrector formula

$$\begin{bmatrix}y_{n+(5/3)}\\ y_{n+(4/3)}\\ y_{n+1}\end{bmatrix}=\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}\begin{bmatrix}y_{n+(2/3)}\\ y_{n+(1/3)}\\ y_n\end{bmatrix}+h\begin{bmatrix}\frac{1}{8}&\frac{3}{8}&\frac{3}{8}\\0&\frac{1}{8}&\frac{3}{8}\\0&0&\frac{1}{8}\end{bmatrix}\begin{bmatrix}f^{\mathrm{P}}_{n+(5/3)}\\ f^{\mathrm{P}}_{n+(4/3)}\\ f^{\mathrm{P}}_{n+1}\end{bmatrix}$$
$$+h\begin{bmatrix}\frac{1}{8}&0&0\\ \frac{3}{8}&\frac{1}{8}&0\\ \frac{3}{8}&\frac{3}{8}&\frac{1}{8}\end{bmatrix}\begin{bmatrix}f_{n+(2/3)}\\ f_{n+(1/3)}\\ f_n\end{bmatrix} \tag{4.5}$$

satisfy

$$\begin{bmatrix}\frac{1}{8}&\frac{3}{8}&\frac{3}{8}\\0&\frac{1}{8}&\frac{3}{8}\\0&0&\frac{1}{8}\end{bmatrix}\begin{bmatrix}1-c_1-c_2&c_2&c_1\\1-b_1-b_2&b_2&b_1\\1-a_1-a_2&a_2&a_1\end{bmatrix}+\begin{bmatrix}\frac{1}{8}&0&0\\ \frac{3}{8}&\frac{1}{8}&0\\ \frac{3}{8}&\frac{3}{8}&\frac{1}{8}\end{bmatrix}=\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}.$$

Equation (4.5) is simply the three-eighths rule in a repeated mode. The stability polynomial of this two-stage method is indeed

$$\left[\xi-\left(\frac{\mu^2}{2}+\mu+1\right)\right]^3.$$

By a global error analysis similar to the one given in § 3, one can verify that the propagation matrix of the error function of this two-stage method is the identity matrix $I_{3\times3}$.

Theorem 4.1 gives a constructive way to determine the predictor of a two-stage method with stability polynomial

$$\left[\xi-\left(\frac{\mu^2}{2}+\mu+1\right)\right]^q$$

once the corrector is fixed. The criterion that C^* be nonsingular seems to be essential in obtaining a perfect power stability polynomial. A fourth-order corrector example is given in [9] where no third-order predictor can be found to make the stability polynomial a perfect power. In that example C^* is singular.

Since there are numerous extensions to the two-stage block methods in § 3, we are led to question of which is the best and in what sense. It can be verified that when applied to the test problem $y'=\lambda y$, the error functions of both Example 4.1 and the pair (2.8), (2.9) with natural choice of parameters are equal. This is, incidentally, not surprising because both of them have the same stability polynomial. The following shows three two-processor, two-stage methods and their corresponding error function equations.

Example 4.2. The corrector (3.5) with the natural choice of parameters $a=5$, $b=\frac{4}{7}$ in the predictor (2.1), has an error function satisfying (3.8).

Example 4.3. The corrector

$$\begin{bmatrix} y_{n+(3/2)} \\ y_{n+1} \end{bmatrix}=\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} y_{n+(1/2)} \\ y_n \end{bmatrix}+h\begin{bmatrix} \frac{1}{6} & \frac{2}{3} \\ -\frac{2}{9} & \frac{5}{6} \end{bmatrix}\begin{bmatrix} f^p_{n+(3/2)} \\ f^p_{n+1} \end{bmatrix}$$
$$+h\begin{bmatrix} \frac{1}{6} & 0 \\ 0 & \frac{7}{18} \end{bmatrix}\begin{bmatrix} f_{n+(1/2)} \\ f_n \end{bmatrix},$$

with $a=\frac{11}{31}$, $b=-\frac{44}{31}$ in (2.1), has an error function equation

$$\begin{bmatrix} e_1'(t) \\ e_2'(t) \end{bmatrix}=f_y(y(t))\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} e_1(t) \\ e_2(t) \end{bmatrix}+\begin{bmatrix} -\frac{1}{12}f_y(y(t))y'''(t) \\ \frac{1}{36}f_y(y(t))y'''(t)-\frac{1}{36}y^{IV}(t) \end{bmatrix}.$$

Example 4.4. The corrector

$$\begin{bmatrix} y_{n+(3/2)} \\ y_{n+1} \end{bmatrix}=\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} y_{n+(1/2)} \\ y_n \end{bmatrix}+h\begin{bmatrix} \frac{1}{6} & \frac{2}{3} \\ 0 & \frac{1}{6} \end{bmatrix}\begin{bmatrix} f^p_{n+(3/2)} \\ f^p_{n+1} \end{bmatrix}$$
$$+h\begin{bmatrix} \frac{1}{6} & 0 \\ \frac{2}{3} & \frac{1}{6} \end{bmatrix}\begin{bmatrix} f_{n+(1/2)} \\ f_n \end{bmatrix},$$

with $a=5$, $b=-20$ in (2.1), has an error function equation

$$\begin{bmatrix} e_1'(t) \\ e_2'(t) \end{bmatrix}=f_y(y(t))\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} e_1(t) \\ e_2(t) \end{bmatrix}+\begin{bmatrix} -\frac{1}{12}f_y(y(t))y'''(t) \\ 0 \end{bmatrix}.$$

For the test problem $y'(t)=\lambda y$, the driving term in each of the above three error functions is equal to $-\frac{1}{12}f_y(y(t))y'''(t)$. Hence, at least for the test problem, all these methods perform equally accurately.

Formula (2.9) and its higher-order extensions do give the lowest communication cost as compared to the other cases. Therefore we believe that (2.9) and its higher-order analogs are the most interesting ones.

There is only one minor drawback in the two-stage block methods. The magnitude of the coefficients increases as the order gets higher. The four-processor analog of

(2.8), (2.9) is given by

$$\begin{bmatrix} y^p_{n+(7/4)} \\ y^p_{n+(6/4)} \\ y^p_{n+(5/4)} \\ y^p_{n+1} \end{bmatrix} = \begin{bmatrix} \frac{8155}{817} & \frac{17482}{817} & -\frac{11060}{817} & \frac{2560}{817} \\ -\frac{2560}{277} & \frac{4185}{817} & -\frac{1728}{277} & \frac{380}{277} \\ -\frac{380}{67} & \frac{160}{67} & \frac{255}{67} & \frac{32}{67} \\ -\frac{32}{7} & -\frac{12}{7} & -\frac{32}{7} & \frac{83}{7} \end{bmatrix} \begin{bmatrix} y_{n+(3/4)} \\ y_{n+(2/4)} \\ y_{n+(1/4)} \\ y_n \end{bmatrix}$$

$$+h \begin{bmatrix} \frac{595}{43} & -\frac{21840}{817} & \frac{16590}{817} & -\frac{4480}{817} \\ \frac{1920}{277} & -\frac{2835}{277} & \frac{2160}{277} & -\frac{570}{277} \\ \frac{190}{67} & -\frac{120}{67} & \frac{195}{67} & -\frac{40}{67} \\ \frac{8}{7} & \frac{6}{7} & \frac{24}{7} & 1 \end{bmatrix} \begin{bmatrix} f_{n+(3/4)} \\ f_{n+(2/4)} \\ f_{n+(1/4)} \\ f_n \end{bmatrix},$$

$$\begin{bmatrix} y_{n+(7/4)} \\ y_{n+(6/4)} \\ y_{n+(5/4)} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_{n+(3/4)} \\ y_{n+(2/4)} \\ y_{n+(1/4)} \\ y_n \end{bmatrix}$$

$$+h \begin{bmatrix} \frac{817}{3150} & 0 & 0 & 0 \\ 0 & \frac{277}{1350} & 0 & 0 \\ 0 & 0 & \frac{67}{450} & 0 \\ 0 & 0 & 0 & \frac{7}{90} \end{bmatrix} \begin{bmatrix} f^p_{n+(7/4)} \\ f^p_{n+(6/4)} \\ f^p_{n+(5/4)} \\ f^p_{n+1} \end{bmatrix}$$

$$+h \begin{bmatrix} \frac{323}{90} & -\frac{416}{75} & \frac{158}{45} & -\frac{256}{315} \\ \frac{256}{135} & -\frac{21}{10} & \frac{32}{25} & -\frac{38}{135} \\ \frac{38}{45} & -\frac{16}{45} & \frac{13}{30} & -\frac{16}{225} \\ \frac{16}{45} & \frac{2}{15} & \frac{16}{45} & \frac{7}{90} \end{bmatrix} \begin{bmatrix} f_{n+(3/4)} \\ f_{n+(2/4)} \\ f_{n+(1/4)} \\ f_n \end{bmatrix}$$

with largest corrector coefficient $\frac{416}{75} \approx 5.5$ and largest predictor coefficient $\frac{21840}{817} \approx 26.7$. For the five-processor analog, the largest corrector coefficient has magnitude 21.5. The four-processor, two-stage method is still acceptable on a single precision machine, but the five-processor method may amplify noise in the derivative values. The corrector formula (4.5) and its higher-order analogs do have small enough coefficients, but the corresponding predictor coefficients are even larger than those of (2.9). For example, the corresponding predictor for (4.5) is given by

$$\begin{bmatrix} y^p_{n+(5/3)} \\ y^p_{n+(4/3)} \\ y^p_{n+1} \end{bmatrix} = \begin{bmatrix} -2 & -39 & 42 \\ 6 & 16 & -21 \\ -3 & -3 & 7 \end{bmatrix} \begin{bmatrix} y_{n+(2/3)} \\ y_{n+(1/3)} \\ y_n \end{bmatrix} + h \begin{bmatrix} 4 & 4 & 8 \\ 2 & -8 & -2 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} f_{n+(2/3)} \\ f_{n+(1/3)} \\ f_n \end{bmatrix},$$

the largest coefficient of which is 42. Thus there is always a tradeoff between the magnitudes of the predictor and corrector coefficients. At this point it is not known how high one can push the order of case (4.5) without introducing intolerable noise amplification.

The problem of large formula coefficients is the nature of one-block parallel methods. It is the relatively large distance from t_{n+1} to $t_{n+1+((q-1)/q)}$, compared to h/q, that causes this drawback. To clarify this statement, let us look at the higher-order analogs of (2.9). The first component, with equation

$$y_{n+1+((q-1)/q)} = y_{n+((q-1)/q)} + \int_{t_{n+((q-1)/q)}}^{t_{n+1+((q-1)/q)}} \text{polynomial interpolating}$$

$$\{f^{\mathrm{p}}_{n+1+((q-1)/q)}, f_n, f_{n+(1/q)}, \cdots, f_{n+((q-1)/q)}\}\, dt,$$

is essentially a variable-stepsize Adams–Moulton formula having the stepsize changed abruptly from h/q to h. Hence it is not surprising that this equation will cause trouble (i.e., large coefficients) as q gets larger. A similar argument is valid for the predictor coefficients. One can, of course, manipulate the method parameters to keep the coefficients small (as is done in Shampine and Watts [7]), but the stability region will simultaneously diminish [1]. A better cure is simply to generalize the current one-block, q-processor, two-stage methods to multiblock methods.

5. Order enhancement by postprocessing. As the numerical integration of the ODE reaches the given output point t_{out}, where $t_N \leqq t_{\text{out}} \leqq t_{N+1}$, we need to generate an output value y_{out} at t_{out}. Usually t_{out} does not coincide with the gridpoints $t_N, \cdots, t_{N+((q-1)/q)}$. There are q values $y_N, y_{N+(1/q)}, \cdots, y_{N+((q-1)/q)}$ at our disposal. These values have global errors of the form

$$(5.1)\quad \begin{bmatrix} y_{N+((q-1)/q)} \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} y(t_{N+((q-1)/q)}) \\ \vdots \\ y(t_N) \end{bmatrix} + h^{q+1} \begin{bmatrix} e_1(t_{N+((q-1)/q)}) \\ \vdots \\ e_q(t_N) \end{bmatrix} + h^{q+2} \begin{bmatrix} g_1(t_{N+((q-1)/q)}) \\ \vdots \\ g_q(t_N) \end{bmatrix} + \cdots.$$

We would like to obtain y_{out} as accurately as possible by interpolating these q values of y. Recall that y_N is of order $q+2$ when q is even and order $q+1$ when q is odd, and the other y's are only of order $q+1$. It turns out that if we are willing to use the $f(y_n), \cdots, f(y_{N+((q-1)/q)})$ needed for the next step, we can make y_{out} of order $q+2$ at any point t_{out} for any q. The interpolation procedure can be illustrated by the case $q=3$. Let $t_{\text{out}} = t_{N+r} = t_N + rh$. When $q=3$, we have from (5.1) that

$$(5.2)\quad \begin{bmatrix} y_{N+(2/3)} \\ y_{N+(1/3)} \\ y_N \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} y(t_{N+r}) + h \begin{bmatrix} \frac{2}{3}-r \\ \frac{1}{3}-r \\ -r \end{bmatrix} y'(t_{N+r}) + \cdots + \frac{h^4}{4!} \begin{bmatrix} (\frac{2}{3}-r)^4 \\ (\frac{1}{3}-r)^4 \\ (-r)^4 \end{bmatrix} y^{IV}(t_{N+r}) + h^4 \begin{bmatrix} e_1(t_{N+r}) \\ e_2(t_{N+r}) \\ e_3(t_{N+r}) \end{bmatrix} + \mathcal{O}(h^5)$$

and

$$(5.3)\quad \begin{aligned} h \begin{bmatrix} f(y_{N+(2/3)}) \\ f(y_{N+(1/3)}) \\ f(y_N) \end{bmatrix} &= h \begin{bmatrix} f(y(t_{N+(2/3)}) + h^4 e_1(t_{N+(2/3)}) + \mathcal{O}(h^5)) \\ f(y(t_{N+(1/3)}) + h^4 e_1(t_{N+(1/3)}) + \mathcal{O}(h^5)) \\ f(y(t_N) + h^4 e_1(t_N) + \mathcal{O}(h^5)) \end{bmatrix} \\ &= h \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} y'(t_{N+r}) + h^2 \begin{bmatrix} \frac{2}{3}-r \\ \frac{1}{3}-r \\ -r \end{bmatrix} y''(t_{N+r}) + \cdots \\ &\quad + \frac{h^4}{3!} \begin{bmatrix} (\frac{2}{3}-r)^3 \\ (\frac{1}{3}-r)^3 \\ (-r)^3 \end{bmatrix} y^{IV}(t_{N+r}) + \mathcal{O}(h^5). \end{aligned}$$

Let

$$M_1=\begin{bmatrix} 1 & \frac{2}{3}-r & \frac{1}{2!}(\frac{2}{3}-r)^2 \\ 1 & \frac{1}{3}-r & \frac{1}{2!}(\frac{1}{3}-r)^2 \\ 1 & -r & \frac{1}{2!}(-r)^2 \end{bmatrix};$$

then (5.3) implies that

$$\begin{bmatrix} hy'(t_{N+r}) \\ h^2y''(t_{N+r}) \\ h^3y'''(t_{N+r}) \end{bmatrix} = M_1^{-1}\left\{ h\begin{bmatrix} f(y_{N+(2/3)}) \\ f(y_{N+(1/3)}) \\ f(y_N) \end{bmatrix} - \frac{h^4}{3!}\begin{bmatrix} (\frac{2}{3}-r)^3 \\ (\frac{1}{3}-r)^3 \\ (-r)^3 \end{bmatrix} y^{IV}(t_{N+r})\right\} + \mathcal{O}(h^5),$$

so that from (5.2),

$$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} y(t_{N+r}) = \begin{bmatrix} y_{N+(2/3)} \\ y_{N+(1/3)} \\ y_N \end{bmatrix} - M_2M_1^{-1}h\begin{bmatrix} f(y_{N+(2/3)}) \\ f(y_{N+(1/3)}) \\ f(y_N) \end{bmatrix} - h^4\begin{bmatrix} e(t_{N+r}) \\ e_2(t_{N+r}) \\ e_3(t_{N+r}) \end{bmatrix}$$

$$+\left\{ M_2M_1^{-1}\begin{bmatrix} \frac{1}{3!}(\frac{2}{3}-r)^3 \\ \frac{1}{3!}(\frac{1}{3}-r)^3 \\ \frac{1}{3!}(-r)^3 \end{bmatrix} - \begin{bmatrix} \frac{1}{4!}(\frac{2}{3}-r)^4 \\ \frac{1}{4!}(\frac{1}{3}-r)^4 \\ \frac{1}{4!}(-r)^4 \end{bmatrix}\right\} h^4y^{IV}(t_{N+r}) - \mathcal{O}(h^5),$$

where

$$M_2=\begin{bmatrix} \frac{2}{3}-r & \frac{1}{2!}(\frac{2}{3}-r)^2 & \frac{1}{3!}(\frac{2}{3}-r)^3 \\ \frac{1}{3}-r & \frac{1}{2!}(\frac{1}{3}-r)^2 & \frac{1}{3!}(\frac{1}{3}-r)^3 \\ -r & \frac{1}{2!}(-r)^2 & \frac{1}{3!}(-r)^3 \end{bmatrix}.$$

Using (3.12), if $\mathbf{u}^T$ satisfies

$$\mathbf{u}^T\left\{ M_2M_1^{-1}\begin{bmatrix} \frac{1}{3!}(\frac{2}{3}-r)^3 \\ \frac{1}{3!}(\frac{1}{3}-r)^3 \\ \frac{1}{3!}(-r)^3 \end{bmatrix} - \begin{bmatrix} \frac{1}{4!}(\frac{2}{3}-r)^4 \\ \frac{1}{4!}(\frac{1}{3}-r)^4 \\ \frac{1}{4!}(-r)^4 \end{bmatrix}, \begin{bmatrix} \frac{23}{648} \\ \frac{17}{2592} \\ \frac{1}{1296} \end{bmatrix}\right\} = 0,$$

then

$$\mathbf{u}^T\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} y(t_{N+r}) = \mathbf{u}^T\begin{bmatrix} y_{N+(2/3)} \\ y_{N+(1/3)} \\ y_N \end{bmatrix} - \mathbf{u}^TM_2M_1^{-1}h\begin{bmatrix} f(y_{N+(2/3)}) \\ f(y_{N+(1/3)}) \\ f(y_N) \end{bmatrix} + \mathcal{O}(h^5).$$

The choice for $\mathbf{u}^T$ is not uniquely determined.

It is, however, impossible to further increase the order because it turns out that the g_i's in (5.1) cannot be annihilated.

The above procedure can be applied to any number of processors. The requirement to solve for M_1^{-1}, a $q \times q$ matrix, is insignificant since q is relatively small.

As a result of postprocessing, our two-stage block methods have order $q+2$ for q-processors.

6. Numerical results. The two-processor and four-processor two-stage methods are compared to the Adams PECE methods with third- and fifth-order correctors. We have done a number of experiments using selected test problems from [5] and [4]. The test problems and their analytic solutions are listed below:

1.

$$y'=-y, \quad y(0)=1, \quad t_{\text{out}}=20, \quad y(t)=e^{-t},$$

2.

$$y' = -\frac{y^3}{2}, \quad y(0) = 1, \quad t_{\text{out}} = 20, \quad y(t) = \frac{1}{\sqrt{1+t}},$$

3.

$$\begin{bmatrix} y_1' \\ y_2' \\ y_3' \end{bmatrix} = \begin{bmatrix} -y_2 - y_2 y_3/r \\ y_1 - y_2 y_3/r \\ y_1/r \end{bmatrix}, \quad y(0) = \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix}, \quad y(t) = \begin{bmatrix} (2+\cos t)\cos t \\ (2+\cos t)\sin t \\ \sin t \end{bmatrix},$$
$$t_{\text{out}} = 20, \quad r = \sqrt{y_1^2 + y_2^2},$$

4.

$$\begin{bmatrix} y_1' \\ y_2' \\ y_3' \\ y_r' \end{bmatrix} = \begin{bmatrix} y_2 \\ -y_1/r^3 \\ y_4 \\ -y_3/r^3 \end{bmatrix}, \quad y(0) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad y(t) = \begin{bmatrix} \cos t \\ -\sin t \\ \sin t \\ \cos t \end{bmatrix},$$
$$t_{\text{out}} = 25, \quad r = \sqrt{y_1^2 + y_3^2},$$

5.

$$\begin{bmatrix} y_1' \\ y_2' \\ \vdots \\ y_{10}' \end{bmatrix} = \begin{bmatrix} -1 & & & \\ 1 & \ddots & & \\ & \ddots & -1 & \\ & & 1 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{10} \end{bmatrix}, \quad y(0) = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad t_{\text{out}} = 20,$$
$$y(t) = \begin{bmatrix} e^{-t} \\ te^{-t} \\ \vdots \\ \frac{t^8}{8!} e^{-t} \\ 1 - (1 + t + \cdots + \frac{t^8}{8!}) e^{-t} \end{bmatrix},$$

6.

$$\begin{bmatrix} y_1' \\ y_2' \\ \vdots \\ y_{10}' \end{bmatrix} = \begin{bmatrix} -2 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{10} \end{bmatrix} = T\Lambda T^{-1} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{10} \end{bmatrix},$$
$$t_{\text{init}} = 0, \quad t_{\text{out}} = 20, \quad y(t) = Te^{\Lambda t},$$

7.

$$y' = y, \quad y(0) = 1, \quad t_{\text{out}} = 10, \quad y(t) = e^t,$$

8.

$$\begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} -\lambda & -\omega \\ \omega & -\lambda \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \quad y(0) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad y(t) = \begin{bmatrix} e^{-\lambda t} \cos \omega t \\ e^{-\lambda t} \sin \omega t \end{bmatrix},$$
$$t_{\text{out}} = 20, \quad \lambda = 1, \quad \omega = \sqrt{3},$$

9.

$$y' = t(1-y) + (1-t)e^{-t}, \quad y(0) = 1, \quad t_{\text{out}} = 10,$$
$$y(t) = e^{-(t^2/2)} - e^{-t} + 1.$$

Problem 7 is an accuracy-bound example, while the rest are stability bound. These problems include tests on both the real and imaginary axes of the $h\lambda$-plane. Problems 5 and 6 show the behavior of the new methods on moderately large systems. The integration interval in each test problem is long enough to indicate any deficiency in the long-term behavior of each method. The problems are run using constant stepsizes on an Alliant FX/8. Exact starting information for each method is provided to compute the solution value at t_1. The number of decimal places of accuracy is computed by

$$-\log_{10}[\max_n \|y_n - y(t_n)\|_2].$$

Due to the small scale of the examples, we do not calculate the total run time, since the interprocessor communication time would have been a dominating portion of the overall timing.

The number of decimal places versus the number of functional evaluations per processor for various values of h are plotted in Figs. 2-10. These figures are sufficient to demonstrate the improvement of the block methods over the Adams PECE methods. In almost all cases, the four-processor block method outperforms the Adams PECE method with fifth-order corrector in both stability and accuracy. The average speedup is about 2.5 at an accuracy of four decimal places. This speedup does not include the effect of improved stability of the block method. A more conclusive speedup can be obtained by comparing the block methods against Adams PECE methods of all orders. As mentioned in the previous paragraph, we have not taken into account the delay in interprocessor communication. One has also to consider the overhead of the intrinsically sequential stepsize and order control in an ODE solver. These factors will degrade the performance of the block methods in practice. The current examples only indicate the

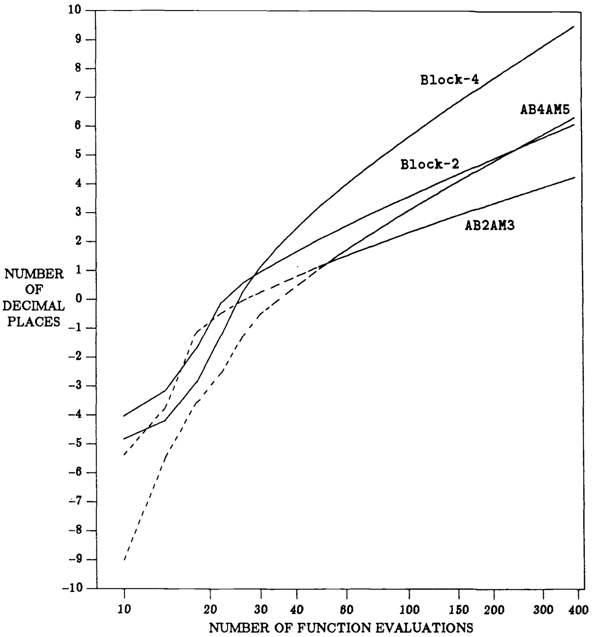


FIG. 2. *Problem* 1. AB2AM3: *Second-order Adams-Bashforth-third-order Adams-Moulton* PECE; AB4AM5: *Fourth-order Adams-Bashforth-fifth-order Adams-Moulton* PECE; Block-2 *or* Block-4: *two- or four-processor two-stage block.*

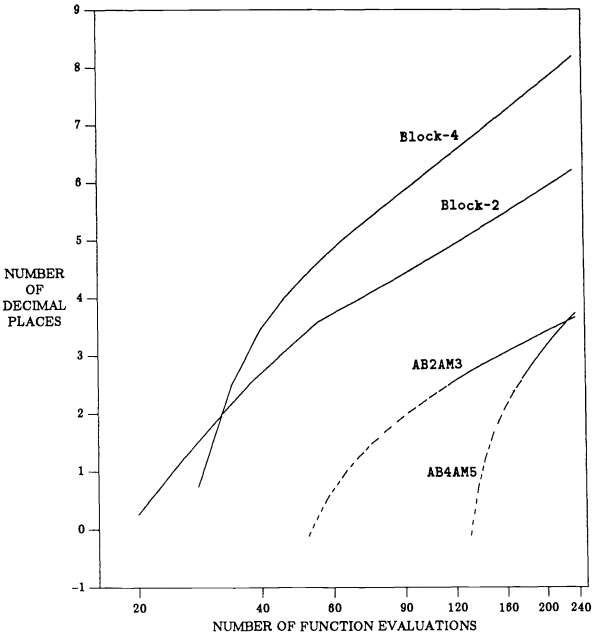


FIG. 3. *Problem* 2.

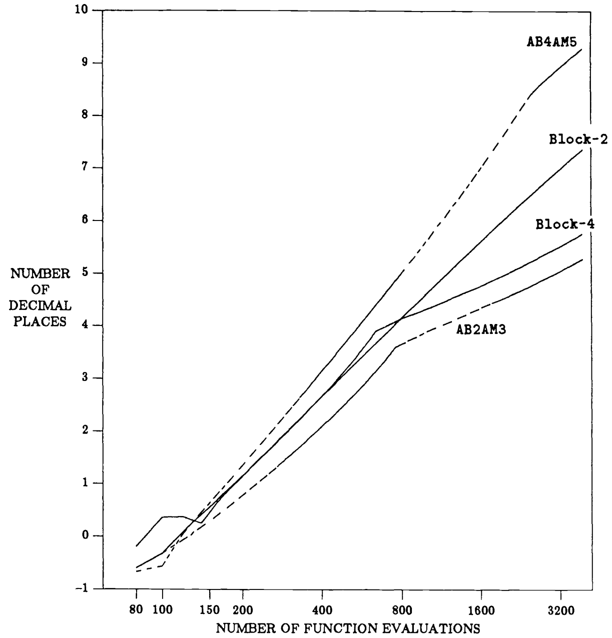


FIG. 4. *Problem* 3.

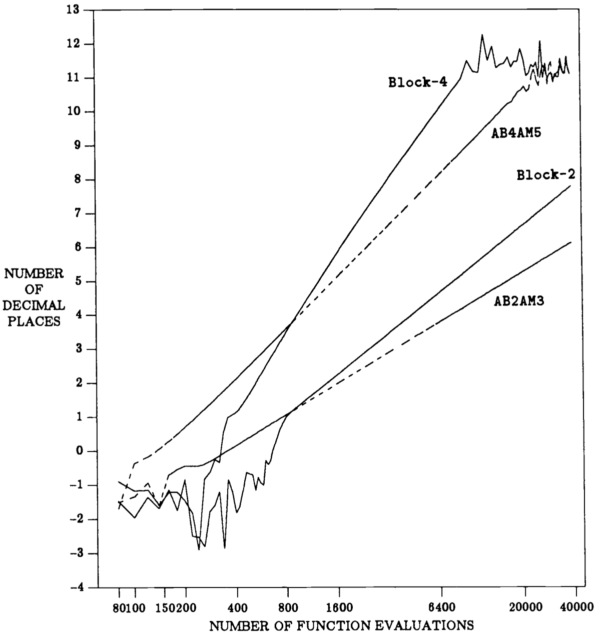


FIG. 5. *Problem* 4.

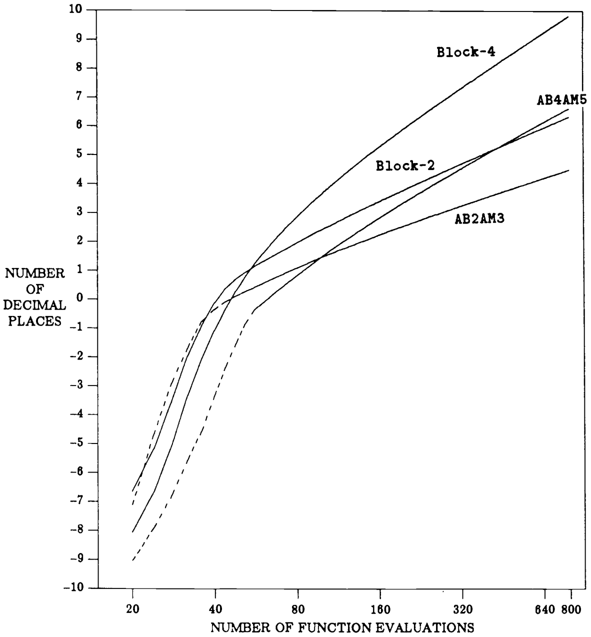


FIG. 6. *Problem* 5.

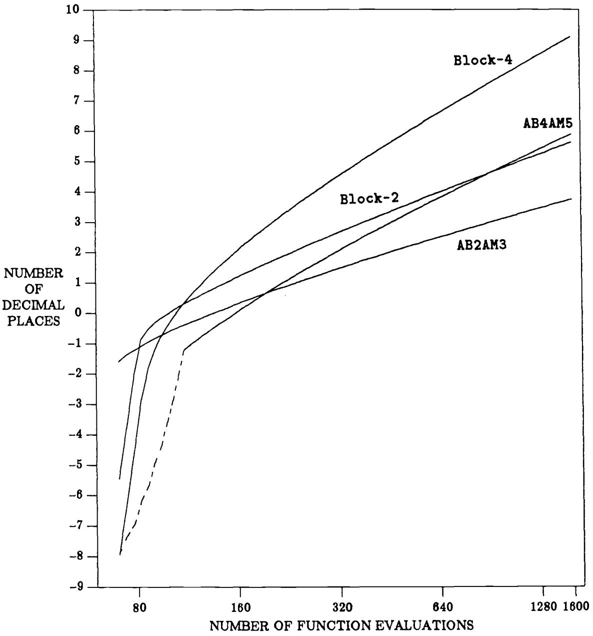


FIG. 7. *Problem 6.*

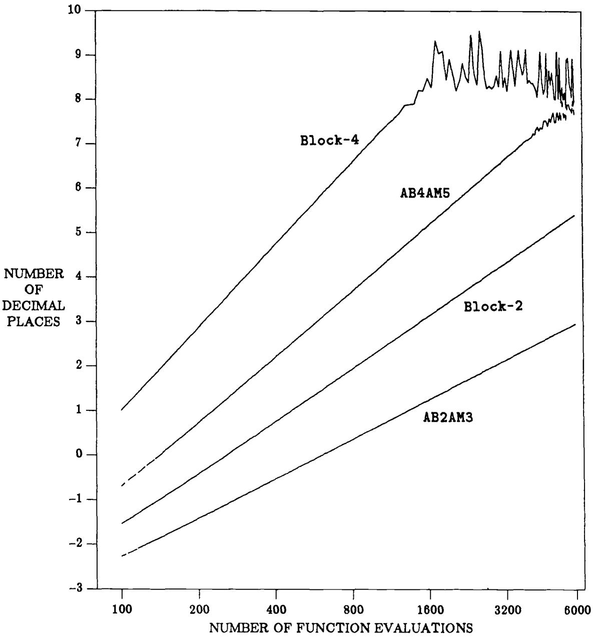


FIG. 8. *Problem 7.*

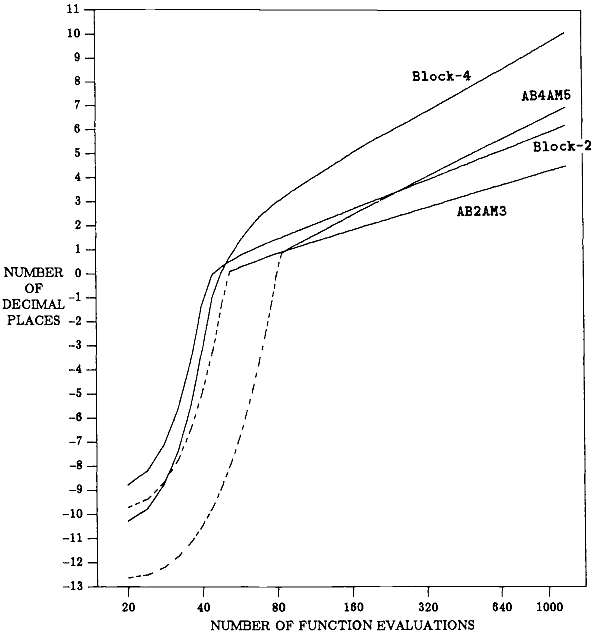


FIG. 9. *Problem* 8.

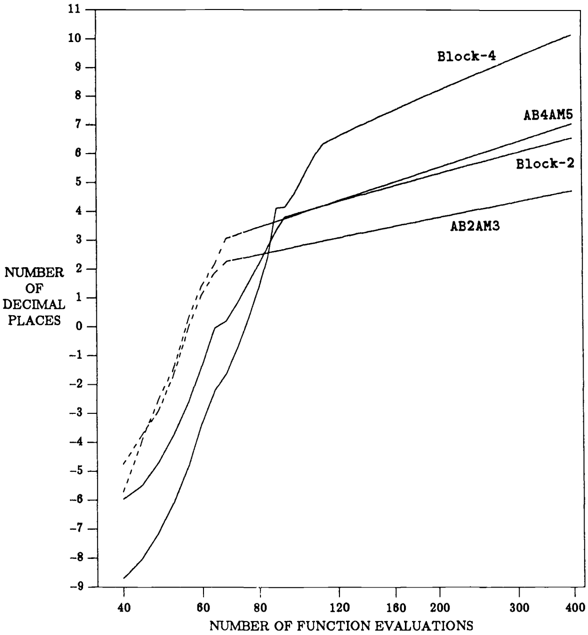


FIG. 10. *Problem* 9.

potential of the new methods. A more detailed comparison would take considerable effort and is not performed here.

7. Summary. In this paper we have discussed a useful family of parallel methods based on the approach of zero-stable methods with perfect power stability polynomials. These parallel methods have stability polynomials equal to

$$\left[\xi-\left(\frac{\mu^2}{2}+\mu+1\right)\right]^q$$

and stability regions essentially that of the second-order Taylor series method. Unlike conventional ODE methods, the stability regions of this new family remain unchanged as the order increases. While the one-stage methods in [10] with stability polynomials $[\xi-(1+\mu)]^q$ suffer from accelerated error growth due to coupling between time points, this family of two-stage methods has tolerable error growth. Numerical experiments show that the q-processor, two-stage block method outperforms the $(q+1)$th-order Adams PECE method in both stability and accuracy. The new family of methods also has a significantly lower interprocessor communication cost than other proposed parallel PECE methods. Extensions of this family of two-stage block methods to multiblock methods will be studied in a forthcoming paper.

REFERENCES

[1] L. G. Birta and O. Abou-Rabia, *Parallel block predictor-corrector methods for* ODE'*s*, IEEE Trans. Comput., C36 (1987), pp. 299-311.

[2] M. Chu and H. Hamilton, *Parallel solution of* ODE'*s by multi-block methods*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 342-353.

[3] E. Hairer, S. P. Norsett, and G. Wanner, *Solving Ordinary Differential Equations* I—*Nonstiff Problems*, Springer-Verlag, Berlin, New York, 1987.

[4] T. E. Hull, W. H. Enright, B. M. Fellen, and A. E. Sedgwick, *Comparing numerical methods for ordinary differential equations*, SIAM J. Numer. Anal., 9, No. 4, (1972), pp. 603-637.

[5] F. T. Krogh, *On testing a subroutine for the numerical integration of ordinary differential equations*, J. Assoc. Comput. Mach., 20 (1973), pp. 545-562.

[6] L. F. Shampine and M. K. Gordon, *Computer Solution of Ordinary Differential Equations*, W. H. Freeman and Co., San Francisco, CA, 1975.

[7] L. F. Shampine and H. W. Watts, *Block implicit one-step methods*, Math. Comp., 23 (1969), pp. 731-740.

[8] R. D. Skeel, *Equivalent forms of multistep formulas*, Math. Comp., 33 (1979), pp. 1229-1250.

[9] H. W. Tam, *Parallel methods for the numerical solution of ordinary differential equations*, Report UIUCDCS-R-89-1516, Department of Computer Science, University of Illinois, Urbana, IL, 1989.

[10] H. W. Tam, *One-stage parallel methods for the numerical solution of ordinary differential equations*, SIAM J. Sci. Statist. Comput., this issue, pp. 1039-1061.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 5, pp. 1085-1096, September 1992

003

SIMULATION AND APPROXIMATION OF STOCHASTIC PROCESSES BY SPLINE FUNCTIONS*

MICHAEL WEBA†

Abstract. Error bounds for simultaneous approximation of stochastic processes by means of spline functions are derived. As opposed to conventional methods, conditions such as regularity of covariances, stationarity, continuity of sample paths, etc. can be dropped, and the error bounds are valid with respect to arbitrary norms. Several applications are indicated: simulating solutions of some stochastic differential equations, computing distributions of continuous functionals by simulation as well as interpolation, numerical differentiation, and numerical integration of stochastic processes by splines.

Key words. stochastic processes, spline functions

AMS(MOS) subject classifications. 60G, 60H, 62E, 62M, 65C, 65D, 65U

1. Introduction. Let X be a stochastic process in continuous time that has been simulated or observed at distinct timepoints $t_1, \cdots, t_n \in [a, b]$. Many problems require "good" approximations of X on the whole interval $[a, b]$. The following may serve as examples:

—Simulate the output process

$$Y(s) = F(s, X(\cdot)), \qquad s \in \mathbb{R}$$

of a possibly nonlinear filter F depending on s and a complete trajectory of X on [a, b]. (In particular, solutions of certain stochastic differential or integral equations admit the above representation.) To this end the input process X has to be simulated at design points $t_i \in [a, b]$. The simulated values must be used to define an approximate version $\hat{X}$ of X; finally,

$$\hat{Y}(s) = F(s, \hat{X}(\cdot)), \qquad s \in \mathbb{R}$$

will be regarded as the simulated output process.

—Simulate the distribution of a continuous functional $\varphi \circ X$ by using $\varphi \circ \hat{X}$ where the approximation $\hat{X}$ of X is based on a finite number of simulated observations from X. For Gaussian processes this method has been studied by Eplett [9] and it can be applied to obtain the asymptotic distribution of rank test statistics.

—Produce a "smooth" representation of a time series X that has been recorded at distinct timepoints. If necessary, reconstruct missing data.

—Compute Stieltjes integrals and derivatives of stochastic processes and specify error bounds for the numerical calculations.

Spline functions are known to yield high-order approximations for differentiable nonrandom functions, and it is the purpose of this paper to examine the rate of convergence if random functions are considered instead. In certain cases splines can be interpreted as minimum variance unbiased linear (MVUL) predictors [14].

Hitherto, splines have frequently been used in nonparametric regression analysis where a nonrandom function $f(t)$ must be estimated if observations

$$Y(t_i) = f(t_i) + \varepsilon_i$$

* Received by the editors March 1, 1989; accepted for publication (in revised form) November 12, 1991.

† Institut für Mathematische Stochastik, Universität Hamburg, Bundesstraße 55, 2000 Hamburg 13, Germany.

are available. The quantities ε_i are random errors and spline estimators $h(t)$ of $f(t)$ are minimizing functionals such as

$$\lambda \cdot \int (h''(t))^2\,dt + \frac{1}{n} \cdot \sum_{i=1}^{n} (Y(t_i) - h(t_i))^2$$

with smoothing parameter $\lambda > 0$. (See Rice and Rosenblatt [19] and Wahba [28], [29].) The topics of this paper are completely different from this curve-fitting problem: the process itself is to be approximated rather than deterministic residuals, and the approximation is based on correct simulations or observations. For various applications of spline functions in stochastics, see also Karlin, Micchelli, and Rinott [12]; Miller and Wegman [17]; Wegman [31]; and Wegman and Wright [32]. Convergence of order-preserving splines is treated in Weba [30].

2. Basic definitions. $X = X(t)$, $t \in [a, b]$, always denotes a stochastic process on the probability space $(\Omega, \mathscr{A}, P)$ with real state space and the closed interval $[a, b] \subset \mathbb{R}$, $a < b$, as index set. Consider the linear space L_o of Borel measurable mappings from Ω into $\mathbb{R}$. It will be assumed that each random variable $X(t)$ lies in the normed vector space $(L, \|\cdot\|)$ where L is a linear subspace of L_o containing the constant mapping $Z_o(\omega) = 1$, $\omega \in \Omega$, and $\|\cdot\|$ stands for an arbitrary norm. (As usual, equivalent elements of L with respect to P will be identified. For simplicity, Z_o is required to satisfy $\|Z_o\| = 1$.) For example, L may be the space $L = L^p(\Omega, \mathscr{A}, P)$, $1 \leqq p \leqq \infty$, of random variables with finite pth moments being equipped with the L^p-norm, or L may be some Orlicz space.

Spline approximations are known to be efficient for differentiable functions $f: [a, b] \to \mathbb{R}$; therefore, some smoothness conditions must be imposed on X. Theorems on stochastic splines will be based on the space $C^m_\Omega[a, b]$ of processes having continuous derivatives[1] up to order $m \geqq 0$. Formally, $C^m_\Omega[a, b]$ is defined as

$$C^m_\Omega[a, b] = \{X: X^{(\mu)} \in C^0([a, b], L) \text{ for } 0 \leqq \mu \leqq m\},$$

where $[a, b]$ is endowed with the natural metric and $C^0([a, b], L)$ denotes the space of continuous mappings from $[a, b]$ into L. Under pointwise addition and scalar multiplication, $C^m_\Omega[a, b]$ becomes a normed linear space by setting

$$\|X\|_\infty = \max_{0 \leqq \mu \leqq m} \sup_{t \in [a, b]} \|X^{(\mu)}(t)\|.$$

By an obvious embedding, the space $C^m[a, b]$ of ordinary nonrandom functions with continuous derivatives up to order m can be viewed as a linear subspace of $C^m_\Omega[a, b]$.

In what follows, Δ always stands for a partition $a = t_0 < t_1 < \cdots < t_n = b$, $n \geqq 1$, of $[a, b]$ with fineness

$$|\Delta| = \max_{0 \leqq j \leqq n-1} h_{j+1}$$

where

$$h_{j+1} = t_{j+1} - t_j, \qquad 0 \leqq j \leqq n - 1.$$

Spline approximations $S_\Delta f$, which depend on Δ and $f \in C^m[a, b]$, are usually expressible as finite linear combinations

$$(S_\Delta f)(t) = d_0(t) + \sum_\mu \sum_i d_{\mu i}(t) \cdot f^{(\mu)}(t_{\mu i}), \qquad t \in [a, b]$$

[1] Derivatives are taken with respect to the norm on L.

with prescribed real-valued functions d_0 and $d_{\mu i}$ and fixed knots $t_{\mu i} \in \Delta$. Then the corresponding spline approximation $S_\Delta X$ of a process $X \in C^m_\Omega[a, b]$ is defined by

$$(S_\Delta X)(t) = d_0(t) + \sum_\mu \sum_i d_{\mu i}(t) \cdot X^{(\mu)}(t_{\mu i}), \qquad t \in [a, b].$$

Finally, the stochastic modulus of continuity $\eta(X; \varepsilon)$ of a process $X \in C^0_\Omega[a, b]$ is the function

$$\eta(X; \varepsilon) = \sup_{\substack{t_1, t_2 \in [a,b] \\ |t_1 - t_2| \leqq \varepsilon}} \|X(t_1) - X(t_2)\|, \qquad \varepsilon \geqq 0.$$

Note that, for $f \in C^0[a, b]$, $\eta(f; \varepsilon)$ coincides with the ordinary modulus of continuity because of $\|Z_o\| = 1$.

3. Convergence of stochastic splines. Error bounds for spline approximations of nonrandom functions $f \in C^m[a, b]$ can be established without additional conditions such as "regularity" or "stationarity"; therefore, it seems quite natural to pin one's hopes on the well-known procedures being applied to the paths of a stochastic process $X \in C^m_\Omega[a, b]$. Actually, an extension argument shows that the convergence properties can be carried over.

THEOREM. *Let $t_{\mu i} \in [a, b]$ be given knots and consider prescribed natural numbers $I(\mu)$, real numbers a_0* and *$a_{\mu i}$ and nonnegative real numbers b_μ, c_μ,and d_μ. Suppose the inequality*

$$\left| a_o + \sum_{\mu=0}^{m} \left(\sum_{i=1}^{I(\mu)} a_{\mu i} \cdot f^{(\mu)}(t_{\mu i}) \right) \right| \leqq \sum_{\mu=0}^{m} \left(b_\mu \cdot \sup_{t \in [a,b]} |f^{(\mu)}(t)| + c_\mu \cdot \eta(f^{(\mu)}; d_\mu) \right)$$

is valid for each $f \in C^m[a, b]$. Then each process $X \in C^m_\Omega[a, b]$ satisfies

$$\left\| a_0 \cdot Z_o + \sum_{\mu=0}^{m} \left(\sum_{i=1}^{I(\mu)} a_{\mu i} \cdot X^{(\mu)}(t_{\mu i}) \right) \right\| \leqq \sum_{\mu=0}^{m} \left(b_\mu \cdot \sup_{t \in [a,b]} \|X^{(\mu)}(t)\| + c_\mu \cdot \eta(X^{(\mu)}; d_\mu) \right).$$

Proof. Let $X \in C^m_\Omega[a, b]$ be a fixed process and consider

$$\xi = a_0 \cdot Z_o + \sum_{\mu=0}^{m} \left(\sum_{i=1}^{I(\mu)} a_{\mu i} \cdot X^{(\mu)}(t_{\mu i}) \right).$$

ξ lies in L; without loss of generality, one can assume $\xi \neq 0$. By the Hahn-Banach therorem there exists a bounded linear functional $\varphi : L \to \mathbb{R}$ having the properties $\|\varphi\| = 1$ and $\|\xi\| = \varphi(\xi)$. (See Corollary (14.13) in Hewitt and Stromberg [11].) Suppose $a_o \geqq 0$. Since the function $\varphi \circ X$ lies in $C^m[a, b]$ with

$$(\varphi \circ X)^{(\mu)} = \varphi \circ (X^{(\mu)}), \qquad 0 \leqq \mu \leqq m,$$

we obtain

$$\begin{aligned} \|\xi\| &= a_o \cdot \varphi(Z_o) + \sum_{\mu=0}^{m} \left(\sum_{i=1}^{I(\mu)} a_{\mu i} \cdot (\varphi \circ X)^{(\mu)}(t_{\mu i}) \right) \\ &\leqq \sum_{\mu=0}^{m} \left(b_\mu \cdot \sup_{t \in [a,b]} |(\varphi \circ X)^{(\mu)}(t)| + c_\mu \cdot \eta((\varphi \circ X)^{(\mu)}; d_\mu) \right) \\ &\leqq \sum_{\mu=0}^{m} \left(b_\mu \cdot \sup_{t \in [a,b]} \|X^{(\mu)}(t)\| + c_\mu \cdot \eta(X^{(\mu)}; d_\mu) \right). \end{aligned}$$

The case $a_0 < 0$ is treated analogously if we make use of the relations $\|(-\varphi)\| = 1$ and $\|\xi\| = (-\varphi)(-\xi)$. □

The above theorem will now be used to establish error bounds for simultaneous approximation of stochastic processes by piecewise cubic splines.

Let $f:[a, b]\to\mathbb{R}$ be a given function. A cubic interpolating spline function $S_\Delta f$ with respect to a partition Δ of $[a, b]$ is a real function with the properties: $S_\Delta f$ is twice continuously differentiable on $[a, b]$, $S_\Delta f$ coincides on every subinterval $[t_j, t_{j+1}]$ with a polynomial of maximal degree three, and $(S_\Delta f)(t_j)=f(t_j)$ holds for each j. In order to ensure uniqueness, the additional requirement

$$(S_\Delta f)'(a)=f'(a), \qquad (S_\Delta f)'(b)=f'(b)$$

is considered. (It is, of course, assumed that $f'(a)$ and $f'(b)$ exist.)

In the following corollary the symbols $(S_\Delta f)^{(3)}(t_j)$ and $(S_\Delta X)^{(3)}(t_j)$ stand for left-hand or right-hand limits. For any partition Δ of $[a, b]$ the quantity $\beta(\Delta)$ is defined by

$$\beta(\Delta)=\max_{0\leqq j\leqq n-1}\frac{|\Delta|}{h_{j+1}}.$$

In part (ii) of the corollary the space $L^p(\Omega, \mathcal{A}, P)$, $1\leqq p<\infty$, of random variables with finite pth moments is endowed with the usual L^p-norm

$$\|Z\|_p=(E|Z|^p)^{1/p}=\left(\int_\Omega |Z(\omega)|^p P(d\omega)\right)^{1/p}.$$

COROLLARY. *Suppose $X\in C^4_\Omega[a, b]$.*

(i) *For each partition Δ of $[a, b]$ and each $k\in\{0, 1, 2, 3\}$ the spline approximation $S_\Delta X$ fulfills*

$$\sup_{t\in[a, b]}\|X^{(k)}(t)-(S_\Delta X)^{(k)}(t)\|\leqq\beta_k\cdot\beta(\Delta)\cdot|\Delta|^{4-k}\cdot\sup_{t\in[a,b]}\|X^{(4)}(t)\|$$

with

$$\beta_0=\tfrac{7}{8}, \quad \beta_1=\beta_2=\tfrac{7}{4}, \quad \beta_3=2.$$

(ii) *Let L be the space $L=L^p(\Omega, \mathcal{A}, P)$, $1\leqq p<\infty$. If Δ_j, $j\geqq 1$, is a sequence of partitions satisfying the condition*

$$\sum_{j=1}^{\infty}(\beta(\Delta_j)\cdot|\Delta_j|^{4-k})^p<\infty$$

for some fixed $k\in\{0, 1, 2, 3\}$, then the relation

$$\lim_{j\to\infty}(S_{\Delta_j}X)^{(k)}(t)=X^{(k)}(t) \qquad (P\ \textit{almost surely})$$

holds for all $t\in[a, b]$.

Proof. Hold $t^*\in[a, b]$ and $k\in\{0, 1, 2, 3\}$ fixed. For each $f\in C^4[a, b]$, the approximation error $f^{(k)}(t^*)-(S_\Delta f)^{(k)}(t^*)$ is expressible as linear combination of $f(t_0)$, $f(t_1), \cdots, f(t_n)$, and $f'(t_0), f'(t_n), f^{(k)}(t^*)$, where coefficients depend on t^* but not on f. Moreover, we have

$$|f^{(k)}(t^*)-(S_\Delta f)^{(k)}(t^*)|\leqq\beta_k\cdot\beta(\Delta)\cdot|\Delta|^{4-k}\cdot\sup_{t\in[a,b]}|f^{(4)}(t)|$$

(see, e.g., [26, pp. 97-106]).

Therefore, part (i) of the corollary is an immediate consequence of the theorem. Part (ii) follows from (i) and the inequality

$$\sum_{j=1}^{\infty} E|X^{(k)}(t)-(S_{\Delta_j}X)^{(k)}(t)|^p<\infty. \qquad \square$$

Remarks. Let us note the following.

(1) By definition, the paths of the process $S_\Delta X$ are obtained by formal application of the spline operator S_Δ on the paths of X. For numerical calculations, we may therefore use standard algorithms for spline approximations. Note that the above error bounds hold with respect to the norm $\|\cdot\|$ of L; no explicit assumptions are made about the sample paths of X and nothing is said about the sample path behavior of $S_\Delta X$. However, if L is the space $L=L^p(\Omega, \mathcal{A}, P)$, then the pathwise approximation will be good "on the average," and under the mild summability condition of part (ii) convergence also occurs with probability 1.

(2) The corollary treats piecewise cubic splines with the boundary condition $(S_\Delta X)'(a)=X'(a)$, $(S_\Delta X)'(b)=X'(b)$. Analogously, error bounds can be carried over if the boundary conditions are changed or if splines are considered being defined by means of noncubic functions. Error bounds in the nonrandom case may be found in [4], [5], [24], or [27].

4. Applications. In the following, $S_\Delta X$ always denotes a fixed spline approximation of the process X, and $r_\Delta^{(k)}(X)$, $k\geqq 0$, stands for the error

$$r_\Delta^{(k)}(X)=\sup_{t\in[a,b]}\|X^{(k)}(t)-(S_\Delta X)^{(k)}(t)\|.$$

For piecewise cubic splines, upper bounds for $r_\Delta^{(k)}(X)$ have been derived in the preceding section.

4.1. Interpolation of stochastic processes. A classic approach to the problem of interpolating random functions, particularly in the field of time series analysis, is the following: given observations $X(t_0), \cdots, X(t_n)$, specification of a good approximation $\hat{X}(t)$ of $X(t)$, $t\in[t_0, t_n]$, is viewed as a special prediction problem. Typically, $\hat{X}(t)$ is chosen to be the linear combination of the observations, which minimizes the mean square error $E|X(t)-\hat{X}(t)|^2$.

Then $\hat{X}(t)$ may be calculated by using filtering techniques or by solving Yule–Walker equations (see [18] and [20]). To this end, however, assumptions about stationarity or the behavior of the spectrum have to be made. Also, the quantities involved such as autocovariances, spectral densities, etc. are usually unknown and must be estimated from the data. Unlike these methods, spline interpolation works for stationary and nonstationary processes. No information on spectral properties is needed and it is unnecessary to cope with estimation problems. Moreover, the derivatives of the process are approximated as well.

4.2. Numerical integration of stochastic processes. Similar to procedures in "ordinary" numerical analysis, splines can be utilized to compute integrals such as

$$\int_a^b g(t)\cdot X(t)\,dt.$$

Here $g:[a,b]\rightarrow\mathbb{R}$ is a given weight function, and the integral is an abstract Riemann integral; since this notion requires completeness, L is now assumed to be a Banach space.

If $S_\Delta(g\cdot X)$ is integrated instead of $g\cdot X$ one at least obtains the error bound

$$\left\| \int_a^b (S_\Delta(g\cdot X))(t)\,dt - \int_a^b g(t)\cdot X(t)\,dt \right\| \leqq (b-a)\cdot r_\Delta^{(0)}(g\cdot X),$$

provided $g\cdot X$ is sufficiently smooth.

In the literature, the problem of estimating the above integral is usually treated as a problem of statistical design or as a special regression problem. Conventional methods offer the following advantages: the sampling designs may be random designs, and within small classes of processes specific sampling designs such as median sampling (quantile sampling, respectively) are shown to yield asymptotically optimal estimates, and exact error rates including constants, rather than bounds are obtained. However, there are serious drawbacks. In addition to differentiability, regularity conditions have to be imposed on the covariance structure of the underlying process. It is also postulated that the process has exactly K derivatives; if the process has derivatives of arbitrary order, the theory gives no information. In particular, stationary processes with band-limited spectra cannot be treated. Furthermore, the whole theory deals exclusively with L^2-processes. For details, see [2], [3], [6], [8], [10], [21], [22], and [23].

Trapezoidal Monte Carlo integration of random processes is discussed in [16]. No regularity conditions are assumed, and for processes that are once mean-square continuously differentiable, it is shown that the rate of convergence of the mean-square approximation error is precisely n^{-4} (n is the sample size). The rate n^{-5} can be achieved for processes being twice mean-square continuously differentiable, provided stratified Monte Carlo integration is performed [7].

We now consider numerical integration by means of splines. No regularity conditions are required and the error bounds are valid with respect to an arbitrary Banach space $L\subset L_o$. The above rates of convergence may be substantially improved for smooth processes. For example, if piecewise cubic splines are used with equidistant knots $t_j = a+j\cdot h$, $0\leqq j\leqq n$, $h=(b-a)/n$, then $g\cdot X\in C_\Omega^4[a,b]$ already guarantees

$$E\left| \int_a^b (S_\Delta(g\cdot X))(t)\,dt - \int_a^b g(t)\cdot X(t)\,dt \right|^2 = O(n^{-8})$$

for $L=L^2(\Omega, \mathscr{A}, P)$.

4.3. Simulating solutions of ordinary stochastic differential equations. Assume $L=L^2(\Omega, \mathscr{A}, P)$. Consider the stochastic differential equation

$$X'(t) = -\alpha\cdot X(t) + Y'(t), \qquad t\in[0,T]$$

with initial condition

$$X(0)=Z,$$

where $\alpha>0$ is a given constant, Z a random variable with finite variance, and $Y\in C_\Omega^1[0,T]$ a stochastic process. If the random fluctuations of X are to be studied by simulation we have to simulate Z and the forcing function Y at timepoints t_i of a prescribed partition Δ of $[a,b]=[0,T]$. However, calculation of the unique solution

$$X(t) = Y(t) + e^{-\alpha t}\cdot(Z-Y(0)) - \alpha\cdot\int_0^t e^{-\alpha(t-s)}\cdot Y(s)\,ds, \qquad t\in[0,T]$$

requires knowledge of Y on the whole interval. Using $S_\Delta Y$, which is based on the simulated values $Y(0),\cdots, Y(T)$, and computing the approximate solution

$$\hat{X}_\Delta(t) = (S_\Delta Y)(t) + e^{-\alpha t}\cdot(Z-Y(0)) - \alpha\cdot\int_0^t e^{-\alpha(t-s)}\cdot(S_\Delta Y)(s)\,ds,$$

the discretization error satisfies

$$\|X(t)-\hat{X}_\Delta(t)\|_2 \leqq (2-e^{-\alpha t}) \cdot r_\Delta^{(0)}(Y)$$

for each $t \in [0, T]$. (Recall that $\|\cdot\|_2$ stands for the L^2-norm.) For simplicity, the discussion has been restricted to the above equation. Analogously, any functional equation in terms of the forcing function yields an approximation $\hat{X}_\Delta$. Representations for more complicated equations can be found, e.g., in [1] and [13].

4.4. Simulating distributions of continuous functionals. Let L be the space $L = L^2(\Omega, \mathscr{A}, P)$. Some topics in statistics require the distribution of $\varphi \circ X$ where φ is a continuous functional on a set of stochastic processes. Since it is often difficult to derive an explicit formula, this distribution must be computed by simulations. If X is a Gaussian process this problem has been studied by Eplett [9]. For certain functionals related to L^p-norms, a result is given by Weba [30] that holds for non-Gaussian processes as well. The idea is to simulate $\varphi \circ X$ by computing $\varphi \circ \hat{X}$; $\hat{X}$ is based on a finite number of simulated observations from X. For a smooth process X, spline approximations may be used, and there is the additional advantage that φ is also allowed to depend on derivatives of X. For example, assume $X \in C^1_\Omega[a, b]$ and consider

$$\varphi \circ X = \int_a^b X(t) \cdot X'(t)\, dt.$$

(Since both X and X' are L^2-continuous, the integrand is L^1-continuous. Hence $\varphi \circ X$ is a well-defined random variable with finite expectation being the L^1-limit of Riemann sums.) Set

$$\varphi \circ S_\Delta X = \int_a^b (S_\Delta X)(t) \cdot (S_\Delta X)'(t)\, dt$$

on condition that $(S_\Delta X)'$ is piecewise L^2-continuous. Then

$$\begin{aligned} &E|\varphi \circ X - \varphi \circ S_\Delta X| \\ &\quad \leqq (b-a) \cdot \sup_{t\in[a,b]} E|X(t) \cdot X'(t) - (S_\Delta X)(t) \cdot (S_\Delta X)'(t)| \\ &\quad \leqq (b-a) \cdot \sup_{t\in[a,b]} E|X(t) \cdot (X'(t) - (S_\Delta X)'(t))| \\ &\qquad + (b-a) \cdot \sup_{t\in[a,b]} E|(S_\Delta X)'(t) \cdot (X(t) - (S_\Delta X)(t))|, \end{aligned}$$

and the inequality of Cauchy-Schwarz gives

$$\begin{aligned} &E|\varphi \circ X - \varphi \circ S_\Delta X| \leqq (b-a) \\ &\qquad \cdot \left(r_\Delta^{(1)}(X) \cdot \sup_{t\in[a,b]} \|X(t)\|_2 + r_\Delta^{(0)}(X) \cdot \sup_{t\in[a,b]} \|(S_\Delta X)'(t)\|_2 \right). \end{aligned}$$

Furthermore, arbitrary $\alpha \in \mathbb{R}$ and $\varepsilon > 0$ satisfy

$$\begin{aligned} &P(\varphi \circ S_\Delta X \leqq \alpha - \varepsilon) - P(|\varphi \circ X - \varphi \circ S_\Delta X| \geqq \varepsilon) \\ &\quad \leqq P(\varphi \circ X \leqq \alpha) \leqq P(\varphi \circ S_\Delta X \leqq \alpha + \varepsilon) + P(|\varphi \circ X - \varphi \circ S_\Delta X| \geqq \varepsilon). \end{aligned}$$

By Markov's inequality,

$$P(|\varphi \circ X - \varphi \circ S_\Delta X| \geqq \varepsilon) \leqq \varepsilon^{-1} \cdot E|\varphi \circ X - \varphi \circ S_\Delta X|.$$

The distance between two distribution functions F and G is often expressed in terms of the Lévy distance

$$D(F, G)=\inf\{\delta>0\colon F(x-\delta)-\delta \leqq G(x) \leqq F(x+\delta)+\delta \text{ for each } x \in \mathbb{R}\}.$$

Consequently, the Lévy distance $D(\varphi \circ X, \varphi \circ S_\Delta X)$ between the distribution functions of $\varphi \circ X$ and $\varphi \circ S_\Delta X$ fulfills

$$D^2(\varphi \circ X, \varphi \circ S_\Delta X) \leqq (b-a) \cdot \Big(r_\Delta^{(1)}(X) \cdot \sup_{t \in [a,b]} \|X(t)\|_2 + r_\Delta^{(0)}(X) \cdot \sup_{t \in [a,b]} \|X'(t)\|_2 + r_\Delta^{(0)}(X) \cdot r_\Delta^{(1)}(X) \Big).$$

Under the assumptions of the above corollary one obtains

$$D(\varphi \circ X, \varphi \circ S_\Delta X) = O(|\Delta|^{3/2})$$

for piecewise cubic splines, provided that $\beta_\Delta = O(1)$ holds.

5. A numerical example. Stochastic differential equations of the type

$$X'(t) = F(t, X(t), Y(t))$$

are frequently used to describe phenomena in electrodynamics, population genetics, economics, etc., where random perturbations are involved. The stochastic properties of the forcing function Y may lead to substantial stochastic fluctuations of the solution X.

The probabilistic behavior of $X(t)$ can be studied by means of simulations. Consider, e.g., the assumptions of § 4.3, where the equation

$$X'(t) = -\alpha \cdot X(t) + Y'(t), \qquad t \in [0, T],$$

with $\alpha > 0$ and initial condition $X(0) = Z$, has been treated. This equation can be regarded as a model for a population that is dying out where the constant rate of "growth" is given by $-\alpha$.

As indicated above, we may simulate Y at timepoints t_i and compute the approximate solution $\hat{X}_\Delta$ by means of some spline function $S_\Delta Y$; the discretization error depends on Y and satisfies

$$\|X(t) - \hat{X}_\Delta(t)\|_2 \leqq (2 - e^{-\alpha t}) \cdot r_\Delta^{(0)}(Y)$$

for each $t \in [0, T]$.

The numerical calculations are based on the following assumptions:

—$T = 5$, $\alpha = (\ln 2)/5$, $\mathrm{X}(0) = \text{const.} = 100$.

—$\Delta = \{t_0, t_1, \cdots, t_n\}$, $t_i = ih$, $h = 0.2$, $n = 25$.

—The random variables $Y(t_0), \cdots, Y(t_n)$ are given by

$$Y(t_0) = 0, \quad Y(t_i) = Y(t_{i-1}) + \delta_i, \quad 1 \leqq i \leqq n,$$

where $\delta_1, \cdots, \delta_n$ are independent and normally distributed with mean value zero and standard deviation $\sigma > 0$.

—$S_\Delta Y$ is the piecewise linear spline function with

$$(S_\Delta Y)(t_i) = Y(t_i), \qquad 0 \leqq i \leqq n.$$

(In other words, $S_\Delta Y$ is a linear polynomial on each subinterval $[t_i, t_{i+1}]$ and interpolates Y. The theorem of the third section may be applied to derive upper bounds for $r_\Delta^{(0)}(Y)$; for instance, $Y \in C_\Omega^2[a, b]$ implies $r_\Delta^{(0)}(Y) = O(|\Delta|^2)$.)

TABLE 1
Values of $\hat{X}_\Delta$ $(\sigma = 1)$.

i	$f_0(t_i)$	$\hat{X}_\Delta(t_i)$ (First sample)	$\hat{X}_\Delta(t_i)$ (Second sample)	$\hat{X}_\Delta(t_i)$ (Third sample)
0	100.00	100.00	100.00	100.00
1	97.27	97.26	95.61	95.13
2	94.61	93.99	90.63	93.09
3	92.02	90.90	89.28	91.87
4	89.50	89.38	87.95	88.04
5	87.06	86.24	85.76	84.31
6	84.67	83.42	83.95	81.21
7	82.36	81.35	83.03	79.46
8	80.11	79.00	80.60	75.42
9	77.92	78.03	78.55	74.80
10	75.79	76.25	76.10	74.06
11	73.71	75.45	72.67	72.90
12	71.70	72.82	69.36	69.62
13	69.74	69.65	67.91	67.38
14	67.83	67.95	65.78	66.15
15	65.98	66.93	63.02	64.70
16	64.17	66.31	63.13	61.33
17	62.42	64.93	62.15	60.19
18	60.71	63.19	60.12	58.01
19	59.05	61.73	56.60	57.03
20	57.43	60.63	55.68	55.21
21	55.86	59.55	55.70	53.70
22	54.34	56.50	54.48	52.56
23	52.85	55.52	52.76	50.98
24	51.41	54.88	51.52	49.96
25	50.00	54.35	50.67	49.04

TABLE 2
Values of $\hat{X}_\Delta$ $(\sigma = 2)$.

i	$f_0(t_i)$	$\hat{X}_\Delta(t_i)$ (First sample)	$\hat{X}_\Delta(t_i)$ (Second sample)	$\hat{X}_\Delta(t_i)$ (Third sample)
0	100.00	100.00	100.00	100.00
1	97.27	95.11	95.50	98.22
2	94.61	95.19	91.65	92.39
3	92.02	94.55	91.40	90.30
4	89.50	92.12	88.90	90.97
5	87.06	90.82	88.92	88.49
6	84.67	89.90	84.40	83.52
7	82.36	86.99	84.83	81.61
8	80.11	82.30	79.34	76.69
9	77.92	81.16	78.39	74.96
10	75.79	78.72	78.28	73.74
11	73.71	73.38	74.41	69.01
12	71.70	71.30	72.10	71.50
13	69.74	72.15	72.06	68.43
14	67.83	69.20	71.67	65.16
15	65.98	70.38	69.73	64.66
16	64.17	65.43	71.45	61.66
17	62.42	63.98	70.26	64.59
18	60.71	58.61	70.59	59.77
19	59.05	59.14	68.28	56.09
20	57.43	59.15	66.33	53.38
21	55.86	56.74	66.46	51.77
22	54.34	54.45	63.11	48.20
23	52.95	52.55	64.31	46.22
24	51.41	50.89	65.54	45.47
25	50.00	52.34	62.15	43.71

TABLE 3
Values of $\hat{X}_\Delta$ $(\sigma = 3)$.

i	$f_0(t_i)$	$\hat{X}_\Delta(t_i)$ (First sample)	$\hat{X}_\Delta(t_i)$ (Second sample)	$\hat{X}_\Delta(t_i)$ (Third sample)
0	100.00	100.00	100.00	100.00
1	97.27	97.40	97.39	101.80
2	94.61	93.79	90.63	97.30
3	92.02	93.65	83.53	95.79
4	89.50	85.73	84.29	98.42
5	87.06	91.44	84.23	96.69
6	84.67	84.56	80.20	93.17
7	82.36	76.65	79.46	90.08
8	80.11	70.66	72.99	91.59
9	77.92	62.55	71.58	92.73
10	75.79	59.33	72.77	93.49
11	73.71	52.84	71.27	87.66
12	71.70	52.47	64.92	82.62
13	69.74	52.37	58.38	80.59
14	67.83	51.86	56.79	78.69
15	65.98	46.80	51.50	79.08
16	64.17	48.40	52.59	79.30
17	62.42	47.80	51.75	75.20
18	60.71	41.73	46.83	68.94
19	59.05	45.88	44.96	72.46
20	57.43	44.00	47.42	70.32
21	55.86	40.44	40.77	70.39
22	54.34	40.50	33.51	68.91
23	52.85	36.96	32.25	69.71
24	51.41	36.47	30.50	70.76
25	50.00	35.62	29.79	65.09

TABLE 4
Values of $\hat{X}_\Delta$ $(\sigma = 5)$.

i	$f_0(t_i)$	$\hat{X}_\Delta(t_i)$ (First sample)	$\hat{X}_\Delta(t_i)$ (Second sample)	$\hat{X}_\Delta(t_i)$ (Third sample)
0	100.00	100.00	100.00	100.00
1	97.27	93.63	97.36	96.46
2	94.61	91.58	103.99	95.91
3	92.03	93.68	109.43	95.54
4	89.50	78.30	101.51	86.89
5	87.06	71.21	102.54	82.23
6	84.67	76.56	109.73	89.03
7	82.36	74.67	105.41	87.09
8	80.11	72.17	103.48	85.05
9	77.92	74.36	99.52	76.97
10	75.79	74.96	99.97	60.42
11	73.71	75.61	99.90	56.48
12	71.70	79.98	93.75	50.67
13	69.74	83.29	91.90	43.51
14	67.83	81.31	81.69	45.92
15	65.98	76.23	70.80	43.50
16	64.17	76.02	67.77	44.90
17	62.42	77.91	65.84	53.30
18	60.71	75.17	59.84	50.60
19	59.05	78.98	56.42	59.61
20	57.43	83.74	53.55	51.58
21	55.86	88.48	49.21	48.73
22	54.34	82.81	46.56	40.18
23	52.85	79.68	39.66	34.52
24	51.41	76.14	36.97	30.58
25	50.00	80.84	35.26	31.40

First, three different (independent) samples from $\delta_1, \delta_2, \cdots, \delta_n$ have been generated; in each case $(S_\Delta Y)(t)$ has been used to compute $\hat{X}_\Delta(t)$ at times $t_0, t_1, \cdots, t_n$. The results for $\sigma=1$ can be found in Table 1. In order to show that the influence of the perturbation process Y is substantial the procedure has been repeated with $\sigma=2$, $\sigma=3$, and $\sigma=5$; see Tables 2-4. In addition, the values of the function

$$f_0(t)=100\cdot\exp\left(-\frac{\ln 2}{5}\cdot t\right)$$

are also given where f_0 is the solution of the differential equation in the absence of random perturbation.

Acknowledgment. The author wishes to express his gratitude to an unknown referee for numerous suggestions and improvements.

REFERENCES

[1] R. B. ASH AND M. F. GARDNER, *Topics in Stochastic Processes*, Academic Press, New York, San Francisco, London, 1975.

[2] K. BENHENNI AND S. CAMBANIS, *Sampling designs for estimating integrals of stochastic processes*, Tech. Report No. 265, Department of Statistics, University of North Carolina, Chapel Hill, NC, 1989.

[3] K. BENHENNI AND S. CAMBANIS, *Sampling designs for estimating integrals of stochastic processes using quadratic mean derivatives*, Tech. Report No. 293, Department of Statistics, University of North Carolina, Chapel Hill, NC, 1990.

[4] K. BÖHMER, *Spline-Funktionen*, Teubner, Stuttgart, 1974.

[5] C. DE BOOR, *A practical guide to splines*, Springer-Verlag, New York, 1978.

[6] S. CAMBANIS, *Sampling designs for time series*, Handbook of Statistics, Vol. 5, Elsevier, Amsterdam, 1985, pp. 337-362.

[7] S. CAMBANIS AND E. MASRY, *Trapezoidal stratified Monte Carlo integration*, Tech. Report No. 286, Department of Statistics, University of North Carolina, Chapel Hill, NC, 1990.

[8] N. CRESSIE, *Estimation of the integral of a stochastic process*, Bull. Austral. Math. Soc., 18 (1978), pp. 83-93.

[9] W. J. R. EPLETT, *Approximation theory for the simulation of continuous Gaussian processes*, Probab. Theory Related Fields, 73 (1986), pp. 159-181.

[10] R. L. EUBANK, P. L. SMITH, AND P. W. SMITH, *A note on optimal and asymptotically optimal designs for certain time series models*, Ann. Statist., 10 (1982), pp. 1295-1301.

[11] E. HEWITT AND K. STROMBERG, *Real and abstract analysis*, Springer-Verlag, Berlin, 1975.

[12] S. KARLIN, C. A. MICCHELLI, AND Y. RINOTT, *Multivariate splines: A probabilistic perspective*, J. Multivariate Anal., 20 (1986), pp. 69-90.

[13] S. KARLIN AND H. M. TAYLOR, *A second course in stochastic processes*, Academic Press, Orlando, FL, 1981.

[14] G. S. KIMELDORF AND G. WAHBA, *Spline functions and stochastic processes*, Sankhyā A, 32 (1970), pp. 173-180.

[15] M. LOÈVE, *Probability Theory* II, Springer-Verlag, New York, 1978.

[16] E. MASRY AND S. CAMBANIS, *Trapezoidal Monte Carlo integration*, SIAM J. Numer. Anal., 27 (1990), pp. 225-246.

[17] J. J. MILLER AND E. J. WEGMAN, *Vector function estimation using splines*, J. Statist. Planning Inf., 17 (1987), pp. 173-180.

[18] M. B. PRIESTLEY, *Spectral analysis and time series*, Academic Press, London, 1981.

[19] J. RICE AND M. ROSENBLATT, *Smoothing splines: Regression, derivatives and deconvolution*, Ann. Statist., 11 (1983), pp. 141-156.

[20] Y. A. ROZANOV, *Stationary random processes*, Holden-Day, San Francisco, 1967.

[21] J. SACKS AND D. YLVISAKER, *Statistical designs and integral approximation*, in Proc. 12th Biennial Seminar of the Canad. Math. Congress, 1970, pp. 115-136.

[22] F. J. SAMANIEGO, *The optimal sampling design for estimating the integral of a process with stationary independent increments*, IEEE Trans. Inform. Theory, IT-22 (1976), pp. 375-376.

[23] C. Schoenfelder and S. Cambanis, *Random designs for estimating integrals of stochastic processes*, Ann. Statist., 10 (1982), pp. 526-538.

[24] L. L. Schumaker, *Spline functions. Basic theory*, John Wiley, New York, 1981.

[25] B. W. Silverman, *Spline smoothing: The equivalent variable kernel method*, Ann. Statist., 12, (1984), pp. 898-916.

[26] J. Stoer and R. Bulirsch, *Introduction to numerical analysis*, Springer-Verlag, New York, 1983.

[27] B. K. Swartz and R. S. Varga, *Error bounds for spline and L-spline interpolation*, J. Approx. Theory, 6 (1972), pp. 6-49.

[28] G. Wahba, *Smoothing noisy data with spline functions*, Numer. Math, 24 (1975), pp. 383-393.

[29] ——— *A comparison of* GCV *and* GML *for choosing the smoothing parameter in the generalized spline smoothing problem*, Ann. Statist., 13 (1985), pp. 1378-1402.

[30] M. Weba, *Quantitative results on monotone approximation of stochastic processes*, Probab. Math. Statist., 11 (1990), pp. 109-120.

[31] E. J. Wegman, *Vector splines and the estimation of filter functions*, Technometrics 14 (1981), pp. 533-546.

[32] E. J. Wegman and I. W. Wright, *Splines in statistics*, J. Amer. Statist. Assoc., 78 (1983), pp. 351-365.

[33] E. Wong and B. Hajek, *Stochastic Processes in Engineering Systems*, Springer-Verlag, New York, 1985.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 5, pp. 1097–1122, September 1992

004

A COMPARISON OF THREE MIXED METHODS FOR THE TIME-DEPENDENT MAXWELL'S EQUATIONS*

PETER MONK†

Abstract. Three mixed finite-element methods for approximating Maxwell's equations are compared. A dispersion analysis provides a Courant–Friedrichs–Lewy (CFL) bound that is necessary for convergence when a uniform mesh is used. The dispersion analysis also allows a comparison of the stability properties of the methods. Superconvergence at the interpolation points is proved for uniform grids, and demonstrated by three numerical examples. All three methods are shown to be able to handle discontinuous media without modification of the finite-element spaces. Since all three methods have three-dimensional counterparts, this study suggests that all three methods could be the basis of a successful three-dimensional code.

Key words. Maxwell's equations, finite elements, mixed methods

AMS(MOS) subject classifications. 65N30, 78-08

1. Introduction.

In a recent paper, Lee and Madsen [9] present numerical results for a novel mixed finite-element method for approximating the time-dependent Maxwell's equations. Applying the method to two-dimensional problems, they show, by numerical examples, that their method has good dispersion properties and can handle discontinuous dielectric constants without special modification of the finite-element spaces. The purpose of this paper is to investigate the dispersion properties of the Lee–Madsen method analytically, and to compare the algorithm to two other mixed finite-element methods due (in three dimensions) to Nédélec [13] and Monk [12]. The Nédélec scheme is of particular interest since it uses a special space of piecewise linear polynomials to approximate the electric field, yet does not need modification where the dielectric constant is discontinuous. This is in contrast to methods based on standard continuous piecewise linear elements (cf. [9]). Our analysis will also show the connection between the mixed finite-element methods analyzed in this paper and the standard Yee finite-difference method for Maxwell's equations [18].

Our study is limited to Maxwell's equations in two space dimensions. Of course this is not the setting of real physical interest; however, the two-dimensional case makes a convenient test problem for rapid comparison of methods. Unfortunately, some features of Maxwell's equations, such as the divergence-free nature of the magnetic flux density, are lost in the two-dimensional case.

Besides the paper of Lee and Madsen [9], Adam, Serveniere, Nédélec, and Raviart [1] have also examined the use of mixed methods to discretize Maxwell's equations. The method of Adam et al. [1] is closely related to the two-dimensional analogue of the method of Monk [12], but does not allow for discontinuous dielectric constant or conductivity. Adam et al. [1] present a dispersion and error analysis for their method in the case when the mass matrix is lumped. In this paper we deal with the full mass matrix and allow discontinuous coefficients.

The method of Lee and Madsen is also related to a prior method suggested by Cangellaris, Lin, and Mei [3]. They proposed and discussed a point-matching (or collocation) method based on the use of continuous bilinear finite-element spaces for

* Received by the editors September 24, 1990; accepted for publication (in revised form) June 28, 1991. This research was supported in part by grants from the Air Force Office of Scientific Research and the National Science Foundation.

† Department of Mathematical Sciences, University of Delaware, Newark, Delaware 19716 (monk@math.udel.edu).

both the electric and magnetic fields, but on staggered grids. For other references concerning the use of finite-element methods in discretizing Maxwell's equations, see [9], and for a discussion of developments in finite-difference methods, see [16].

The plan of the paper is as follows. In §2 we discuss two general classes of variational principles for the time-dependent Maxwell system. Then in §3 we give details of the three finite-element methods examined in this paper. We discuss error estimates and derive the discrete equations in the special case of a uniform grid. These equations show the connections between the methods in this paper and standard finite-difference methods (cf. [18]). In §4, we use the discrete equations on a uniform grid to compare the three methods via a dispersion analysis. Section 5 is devoted to numerical experiments with the three methods. For ease of comparison, we have used examples from [9] as test problems.

2. Variational principles for Maxwell's equations. We begin by deriving two general classes of methods for Maxwell's equations [9], [13], [12], [11]. Since our numerical examples are all in two space dimensions, we consider Maxwell's equations for a linear isotropic material in which the magnetic field $\boldsymbol{H}$ is z-polarized. Thus if $\boldsymbol{E} = (E^{(1)}(\boldsymbol{x},t), E^{(2)}(\boldsymbol{x},t))$ and $\boldsymbol{H} = H(\boldsymbol{x},t)$ where $\boldsymbol{x} = (x,y)$, we have that

$$\epsilon \frac{\partial \boldsymbol{E}}{\partial t} + \sigma \boldsymbol{E} = \vec{\nabla} \times H - \boldsymbol{J}, \tag{1}$$

$$\mu \frac{\partial H}{\partial t} = -\nabla \times \boldsymbol{E}, \tag{2}$$

where ϵ, σ, and μ are known functions of $\boldsymbol{x}$ giving the dielectric constant (ϵ), permeability (μ), and conductivity (σ), respectively. $\boldsymbol{J} = (J^{(1)}, J^{(2)})$ is a function of space and time giving the current density. The vector curl ($\vec{\nabla}\times$) is defined by

$$\vec{\nabla} \times f = \left(\frac{\partial f}{\partial y}, -\frac{\partial f}{\partial x}\right), \tag{3}$$

and the scalar curl ($\nabla\times$) by

$$\nabla \times \boldsymbol{v} = \frac{\partial}{\partial x} v^{(2)} - \frac{\partial}{\partial y} v^{(1)} \tag{4}$$

(we use $v^{(j)}$ to refer to the jth component of a vector $\boldsymbol{v}$). Equations (1) and (2) are solved in a plane polygonal region Ω with boundary Γ and we assume the boundary condition

$$\boldsymbol{n} \times \boldsymbol{E} = \gamma, \tag{5}$$

where $\boldsymbol{n}$ is the unit outward normal to Ω, $\boldsymbol{n} \times \boldsymbol{E} = E^{(1)}n^{(2)} - E^{(2)}n^{(1)}$ and γ is a specified function. If $\gamma \equiv 0$, this boundary condition models a perfect conducting boundary. In addition, initial data must be specified so that we assume that

$$\boldsymbol{E}(0) = \boldsymbol{E}_0 \quad \text{and} \quad H(0) = H_0, \tag{6}$$

where $\boldsymbol{E}_0$ and H_0 are given functions. Equations (1), (2), (5), and (6) are a well-posed system of equations (cf. [5], [1], [10]) provided $\epsilon, \mu, \gamma, \sigma$, and $\boldsymbol{J}$ are sufficiently smooth and satisfy standard positivity and boundedness assumptions.

To derive a weak or variational formulation of (1)–(6), we proceed formally (for details of the function space setting, see [5], [12]). Multiplying (1) by a test function

$\boldsymbol{\phi}(\boldsymbol{x})$ and (2) by $\psi(\boldsymbol{x})$ and integrating over Ω, we obtain the following equations where $(\boldsymbol{u},\boldsymbol{v}) = \int_\Omega u^{(1)}v^{(1)} + u^{(2)}v^{(2)}\,dA$ and $(u,v) = \int_\Omega uv\,dA$,

$$(\epsilon \boldsymbol{E}_t + \sigma \boldsymbol{E}, \boldsymbol{\phi}) = (\vec{\nabla} \times H, \boldsymbol{\phi}) - (\boldsymbol{J}, \boldsymbol{\phi}), \tag{7}$$

$$(\mu, H_t, \psi) = -(\nabla \times \boldsymbol{E}, \psi). \tag{8}$$

This pair of equations is not suitable for discretization since, in general, it would not result in an energy-conserving discrete problem. Instead we choose to integrate one of the curl terms in (7) or (8) by parts. One choice will lead to the method of Lee and Madsen, the other to the method of Nédélec.

Let

$$H(\text{curl};\Omega) = \{\boldsymbol{u} \in (L^2(\Omega))^2 \mid \nabla \times \boldsymbol{u} \in L^2(\Omega)\},$$
$$H_0(\text{curl};\Omega) = \{\boldsymbol{u} \in H(\text{curl};\Omega) \mid \boldsymbol{n} \times \boldsymbol{u} = 0 \text{ on } \Gamma\},$$

and let $\boldsymbol{E}(t) = \boldsymbol{E}(\cdot,t)$ and $H(t) = H(\cdot,t)$ (see [6] for a complete discussion of the curl spaces). If we integrate (7) by parts, then provided $\boldsymbol{\phi} \in H_0(\text{curl};\Omega)$ we are led to the conclusion that $\boldsymbol{E}(t) \in H(\text{curl};\Omega), H(t) \in L^2(\Omega)$ satisfies

$$(\epsilon \boldsymbol{E}_t + \sigma \boldsymbol{E}, \boldsymbol{\phi}) = (H, \nabla \times \boldsymbol{\phi}) - (\boldsymbol{J}, \boldsymbol{\phi}) \quad \forall \boldsymbol{\phi} \in H_0(\text{curl};\Omega), \tag{9}$$

$$(\mu H_t, \psi) = -(\nabla \times \boldsymbol{E}, \psi) \quad \forall \psi \in L^2(\Omega), \tag{10}$$

and

$$\boldsymbol{n} \times \boldsymbol{E} = \gamma \quad \text{on}\, \Gamma, \tag{11}$$

together with the initial condition (6). Note that in this case the test function $\boldsymbol{\phi}$ must have a well-defined scalar curl and must satisfy the boundary condition $\boldsymbol{n} \times \boldsymbol{\phi} = 0$. In this formulation the boundary conditions are essential. Nédélec's method is based on (9)–(11).

Alternatively, we may integrate (8) by parts to obtain that $\boldsymbol{E}(t) \in (L^2(\Omega))^2$ and $H(t) \in H(\vec{\text{curl}};\Omega) \equiv \{v \in L^2(\Omega) | \vec{\nabla} \times v \in (L^2(\Omega))^2\}$ satisfy

$$(\epsilon \boldsymbol{E}_t + \sigma \boldsymbol{E}, \boldsymbol{\phi}) = (\vec{\nabla} \times H, \boldsymbol{\phi}) - (\boldsymbol{J}, \boldsymbol{\phi}) \quad \forall \boldsymbol{\phi} \in (L^2(\Omega))^2, \tag{12}$$

$$(\mu H_t, \psi) = -(\boldsymbol{E}, \vec{\nabla} \times \psi) + \langle \gamma, \psi \rangle \quad \forall \psi \in H(\vec{\text{curl}};\Omega), \tag{13}$$

together with the initial condition (6). Note that in this case neither trial nor test functions need to satisfy any boundary conditions, and the boundary condition $\boldsymbol{n} \times \boldsymbol{E} = \gamma$ is imposed naturally. The system (12)–(13) is the basis for the method of Lee–Madsen [9] and the method of Monk [12].

Next we discretize in space to produce an approximate method of lines for Maxwell's equations. Later we shall detail discretization in time. To discretize (9)–(11) we let $U_h^N \subset H(\text{curl};\Omega), U_{ho}^N = U_h^N \cap H_0(\text{curl};\Omega)$, and $V_h^N \subset L^2(\Omega)$ be finite-dimensional spaces indexed by h. In this paper each space will be a finite-element space, but at this stage we could allow other possibilities, such as spectral method spaces. The semidiscrete problem corresponding to (9)–(11) is to find $(\boldsymbol{E}^h(t), H^h(t)) \in U_h^N \times V_h^N$ such that

$$(\epsilon \boldsymbol{E}_t^h + \sigma \boldsymbol{E}^h, \boldsymbol{\phi}^h) = (H^h, \nabla \times \boldsymbol{\phi}^h) - (\boldsymbol{J}, \boldsymbol{\phi}^h) \quad \forall \boldsymbol{\phi}^h \in U_{ho}^N, \tag{14}$$

$$(\mu H_t^h, \psi^h) = -(\nabla \times \boldsymbol{E}^h, \psi^h) \quad \forall \psi^h \in V_h^N, \tag{15}$$

$$\boldsymbol{n} \times \boldsymbol{E}^h = \gamma^h \quad \text{on } \Gamma, \tag{16}$$

and

$$E^h(0) = E_0^h \quad \text{and} \quad H^h(0) = H_0^h, \tag{17}$$

where $\gamma^h \in \boldsymbol{n} \times U_h^N$ approximates $\gamma, \boldsymbol{E}_0^h \subset U_h^N$ approximates $\boldsymbol{E}_0$, and $H_0^h \subset V_h^N$ approximates H_0. Note that U_h^N need only be a subspace of $H(\text{curl};\Omega)$ and need not be in $(H^1(\Omega))^2$. Standard continuous linear finite elements are in $(H^1(\Omega))^2$ and thus are too smooth. This manifests itself in the complex way that standard elements must be adapted to handle interfaces where ϵ is discontinuous (cf. [9] and §5.3 of this paper). Nédélec's construction gives elements in $H(\text{curl};\Omega)$ but not in $(H^1(\Omega))^2$.

To discretize (12)–(13) we take finite-dimensional spaces $U_h^L \subset (L^2(\Omega))^2$ and $V_h^L \subset H(\vec{\text{curl}};\Omega)$. Then seek $(\boldsymbol{E}^h(t), H^h(t)) \in U_h^L \times V_h^L$ such that

$$(\epsilon \boldsymbol{E}_t^h + \sigma \boldsymbol{E}^h, \boldsymbol{\phi}^h) = (\vec{\nabla} \times H^h, \boldsymbol{\phi}^h) - (J, \boldsymbol{\phi}^h) \quad \forall \boldsymbol{\phi} \in U_h^L, \tag{18}$$

$$(\mu H_t^h, \psi^h) = -(\boldsymbol{E}^h, \vec{\nabla} \times \psi^h) + \langle \gamma, \psi^h \rangle \quad \forall \psi^h \in V_h^L, \tag{19}$$

and

$$\boldsymbol{E}^h(0) = E_0^h \quad \text{and} \quad H^h(0) = H_0^h, \tag{20}$$

where $\boldsymbol{E}_0^h \subset U_h^L$ and $H_0^h \subset V_h^L$, respectively, approximate $\boldsymbol{E}_0$ and H_0.

In this case, the boundary condition need not be imposed on the finite-element functions. Note that $V_h^L \subset H(\vec{\text{curl}};\Omega)$ and so, in this two-dimensional problem, we need $V_h^L \subset H^1(\Omega)$. Thus in this case, standard finite elements can be used to construct V_h^L. In fact, Lee and Madsen [9] use standard isoparametric piecewise bilinear functions on quadrilateral elements. The space $U_h^L \subset (L^2(\Omega))^2$, and thus a candidate space, is the space of piecewise constant vectors, which is the choice used by Lee and Madsen [9]. Monk [12] uses a slightly augmented space.

One final remark is that the general methods (14)–(17) and (18)–(20) both conserve energy (before time discretization). To see this, consider the case when $\sigma \equiv 0$, $\boldsymbol{J} \equiv 0$, $\gamma \equiv 0$. Then taking $\psi^h = H^h$ and $\boldsymbol{\phi}^h = \boldsymbol{E}^h$ in (14)–(17) or (18)–(19) and adding (14)–(15) or (18)–(19) we obtain

$$(\epsilon \boldsymbol{E}_t^h, \boldsymbol{E}^h) + (\mu H_t^h, H^h) = 0. \tag{21}$$

Thus

$$\frac{1}{2}\frac{d}{dt}\{(\epsilon \boldsymbol{E}^h, \boldsymbol{E}^h) + (\mu H^h, H^h)\} = 0,$$

and so,

$$(\epsilon \boldsymbol{E}^h(t), \boldsymbol{E}^h(t)) + (\mu H^h(t), H^h(t)) = (\epsilon \boldsymbol{E}^h(0), \boldsymbol{E}^h(0)) + (\mu H^h(0), H^h(0)), \tag{22}$$

which states that the energy in the discrete system is independent of time. This energy conservation is the reason for integrating by parts in (7) and (8) to ensure that the curl terms cancel. The above energy equality can be used to derive error estimates for the finite-element methods (see [11] and §3 of this paper).

3. Discretization in space and time. In this section we give details of each of the methods to be analyzed. We start by subdividing Ω using a collection of quadrilateral elements $\tau_h = \{K_i\}_{i=1}^{N_h}$ with maximum diameter h, which obey the usual finite-element mesh restrictions [4]. The basis functions for each finite-element space are obtained using the isoparametric method by mapping basis functions on a reference element to the target element. Let the domain of the reference element be

$$\hat{K} = \{(x, y) \mid 0 \le \hat{x} \le 1,\ 0 \le \hat{y} \le 1\}, \tag{23}$$

then let $\boldsymbol{F}_{K_i}$ be the bilinear map from $\hat{K}$ to K_i. If K_i has vertices with coordinates $\{\boldsymbol{a}_j^{(i)}\}_{j=1}^4$ and $\hat{\boldsymbol{x}} = (\hat{x}, \hat{y})$, then

$$\boldsymbol{F}_{K_i}(\hat{\boldsymbol{x}}) = \boldsymbol{a}_1^{(i)}(1-\hat{x})(1-\hat{y}) + \boldsymbol{a}_2^{(i)}\hat{x}(1-\hat{y}) + \boldsymbol{a}_3^{(i)}\hat{x}\hat{y} + \boldsymbol{a}_4^{(i)}(1-\hat{x})\hat{y}. \tag{24}$$

We denote the Jacobian of $\boldsymbol{F}_{K_i}$ by $\boldsymbol{J}_{K_i}(\hat{x})$.

In general,

$$\boldsymbol{E}^h(t) = \sum_{j=1}^{N_E} E_j(t)\boldsymbol{\phi}_j(\boldsymbol{x}) \quad \text{and} \quad H^h(t) = \sum_{j=1}^{N_H} H_j \psi_j(\boldsymbol{x}), \tag{25}$$

where N_E and N_H are the number of degrees of freedom for $\boldsymbol{E}^h$ and H^h, respectively, and $\{\boldsymbol{\phi}_j(\boldsymbol{x})\}_{j=1}^{N_E}$ and $\{\psi_j(x)\}_{j=1}^{N_H}$ are suitable basis functions.

In the remainder of the paper we use the notation that Q_{ij} is the set of all polynomials of degree at most i in x and at most j in y. Thus Q_{00} is the space of constant functions and $Q_{10} = \{p \mid p = a + bx \quad a, b \in \mathbb{R}\}$.

3.1. Lee and Madsen's method. This method has the simplest spaces. The electric field space is just the space of piecewise constant vectors. Thus

$$U_h^L = \{\boldsymbol{u}^h \mid \boldsymbol{u}^h|_K \in Q_{00} \times Q_{00} \quad \forall K \in \tau_h\}. \tag{26}$$

A suitable basis for this space is the set of piecewise constant vector functions in which one component is nonzero on exactly one element in the grid. The space V_h^L is the standard continuous piecewise bilinear isoparametric space. Thus to construct V_h^L, we define the reference element $(\hat{K}, P_{\hat{K}}, \Sigma_{\hat{K}})$ [4] by taking $\hat{K}$ to be the reference domain in (23) with vertices $\{\hat{\boldsymbol{a}}_i\}_{i=1}^4$, the local space $P_{\hat{K}}$ to be the standard space of bilinear polynomials so that $P_{\hat{K}} = Q_{11}$, and the local degrees of freedom to be

$$\Sigma_{\hat{K}} = \{p(\hat{\boldsymbol{a}}_i) \mid 1 \le i \le 4\}. \tag{27}$$

Then on each element $K \in \tau_h$ we have that the jth basis function is given by $\psi_j|_K(\boldsymbol{x}) = \hat{\psi}(F_K^{-1}(\boldsymbol{x}))$ for some $\hat{\psi} \in P_{\hat{K}}$. Thus

$$V_h^L = \{v_h \in C(\overline{\Omega}) \mid v_h|_K = \hat{v}_K \circ F_K^{-1} \quad \text{for some} \quad \hat{v}_K \in Q_{11} \quad \forall K \in \tau_h\}. \tag{28}$$

Following the isoparametric philosophy, all integrations are performed on $\hat{K}$ by mapping from a given element K to $\hat{K}$ using the mapping F_K^{-1}. We use four-point Gaussian quadrature on $\hat{K}$ to compute integrals. This is exact if F_K is affine and of precision three in general.

In summary, the degrees of freedom for the discontinuous electric field space U_h^L are the values of the electric field at the centroid of each element. The degrees of freedom for the magnetic field space V_h^L are the values of the field at the vertices of the mesh. In keeping with Lee and Madson [9], we refer to this method as ECHL (E constant, H linear).

We now apply the ECHL algorithm on a uniform mesh and consider the equations satisfied by the degrees of freedom. Figure 1 shows a portion of the mesh for the

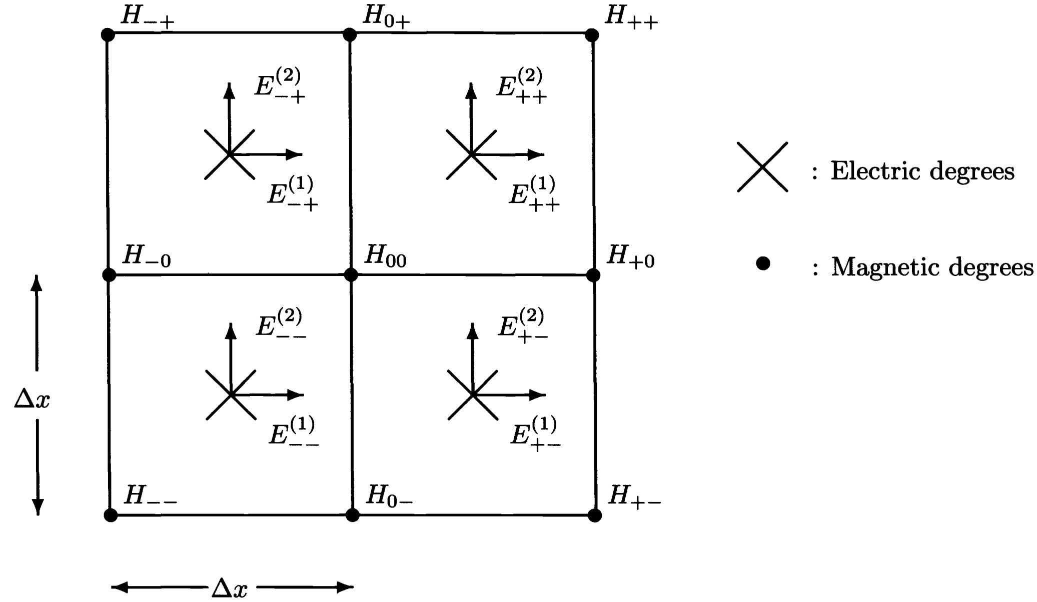


FIG. 1. *A portion of a uniform quadrilateral grid showing the positions of the degrees of freedom for Lee's method. The equations for the indicated degrees of freedom are given in* (29)–(31).

method with the degrees of freedom labeled. If we consider the very special case of (18)–(19) with $\boldsymbol{J} \equiv 0, \sigma \equiv 0, \epsilon = \mu = 1$, and (U_h^L, V_h^L) given by (26) and (28), we obtain the following equations for the degrees of freedom:

$$\frac{d}{dt}E^{(1)}_{++} - \left(\frac{(H_{0+}+H_{++})-(H_{00}+H_{+0})}{2\Delta x}\right) = 0, \tag{29}$$

$$\frac{d}{dt}E^{(2)}_{++} + \left(\frac{(H_{+0}+H_{++})-(H_{0+}+H_{00})}{2\Delta x}\right) = 0, \tag{30}$$

$$\begin{aligned}&\frac{d}{dt}\left[\frac{1}{36}[H_{+-}+H_{++}+H_{--}+H_{-+}]\right.\\ &\qquad\left.+\frac{1}{9}[H_{0-}+H_{-0}+H_{+0}+H_{0+}]+\frac{4}{9}H_{00}\right]\\ &+\frac{1}{2\Delta x}\left\{[(E^{(2)}_{++}+E^{(2)}_{+-})-(E^{(2)}_{-+}+E^{(2)}_{--})]\right.\\ &\qquad\left.-[(E^{(1)}_{++}+E^{(1)}_{-+})-(E^{(1)}_{+-}+E^{(1)}_{--})]\right\} = 0.\end{aligned} \tag{31}$$

Clearly (29)–(31) are a centered difference approximation to (1)–(2) and thus have local truncation error $O((\Delta x)^2)$ provided $\boldsymbol{E}$ and H are smooth enough. Given the stability result (21), this implies that on a uniform mesh, ECHL converges with error $O((\Delta x)^2)$ at the points indicated in Fig. 1 (convergence is in the discrete L^2 norm at each timestep [11]). More precisely, let us define the norms

$$\|u\|_{0,\epsilon} = \sqrt{(\epsilon u, u)} \quad \text{and} \quad \|u\|_{0,\mu} = \sqrt{(\mu u, u)}, \tag{32}$$

and let $r_h\boldsymbol{E} \in U_h^L$ interpolate $\boldsymbol{E}$ at the interpolation points shown in Fig. 1. Let $r_h H \in V_h^L$ be the interpolant for the magnetic field space. Then the stability estimate

(22) and the second-order local truncation error of the method imply that

$$\|(r_h\boldsymbol{E} - \boldsymbol{E}^h)(t)\|_{\epsilon,0} + \|(r_hH - H^h)(t)\|_{\mu,0} = O((\Delta x)^2), \tag{33}$$

provided the initial data is chosen to satisfy (33) at $t = 0$.

In trying to prove convergence on more general domains, we see that ECHL suffers from a big theoretical drawback in that the spaces (U_h^L, V_h^L) do not satisfy an appropriate inf-sup condition and hence ECHL is not included in the general theory of Brezzi [2]. In this context, the inf-sup condition would state that for every nonconstant $H^h \in V_h^L$ (i.e., a function such that $\int_\Omega |\nabla H^h|^2\, d\boldsymbol{x} \neq 0$) there should exist a function $\phi^h \in U_h^L$ such that $(\vec{\nabla} \times H^h, \phi^h) \neq 0$. However, at least for some meshes, it is easy to construct a function $H^{*h} \in V_h^L$ such that

$$(\vec{\nabla} \times H^{*h}, \phi^h) = 0 \quad \forall \phi^h \in U_h^L, \tag{34}$$

so that the inf-sup condition does not hold. To see this, consider first the reference element $\hat{K} = [0,1] \times [0,1]$. Let $\hat{H}^* \in Q_{11}$ be the bilinear function such that $\hat{H}^*(0,0) = \hat{H}^*(1,1) = 1$ and $\hat{H}^*(1,0) = \hat{H}^*(0,1) = -1$. Direct computation shows that $\vec{\nabla} \times \hat{H}^* = (2-4\hat{x}, -2+4\hat{y})^T$ and hence $\int_{\hat{K}} \vec{\nabla} \times \hat{H}^* \cdot \phi = 0$ for any constant vector ϕ. For a general mesh, copies of this function can often be used to construct a function $H^{*h} \in V_h^L$ that satisfies (34). For example, on a square domain, meshed with a uniform grid of squares, H^{*h} is the piecewise bilinear function that interpolates ± 1 at the vertices with the values of $+1$ and -1 arranged in a checkerboard configuration. We can use H^{*h} to show nonphysical behavior in ECHL. Suppose we solve (21)–(25) with $\boldsymbol{J} \equiv 0$ and $\gamma \equiv 0$ and with initial data

$$\boldsymbol{E}_0^h = 0, \qquad H_0^h = H^{*h}.$$

Then we see that the solution at later times is given by

$$\boldsymbol{E}^h(t) = 0, \qquad H^h(t) = H^{*h}.$$

Given the perfect conducting boundary condition, this solution is nonphysical. Of course, H^{*h} oscillates on a wavelength of order h and thus it is not surprising that the numerical method propagates this high frequency solution poorly. In §4, we use a dispersion analysis to investigate the accuracy of wave propagation as a function of frequency and wavelength for each of the methods discussed in this paper. The lack of an inf-sup condition makes general error estimation difficult and it is not proven, to our knowledge, that ECHL is convergent on general grids. Nevertheless, for smooth data it is possible for the method to converge and work well in practice, as we shall see.

The next method is based on the same variational formulation as ECHL, but the space for the electric field has been modified so that the inf-sup condition holds.

3.2. The MECHL method. This method is based on (18)–(20) and is obtained by restricting the three-dimensional method of [12] to two dimensions by assuming H is z-polarized and independent of z. Let us denote the spaces for the method by U_h^M and V_h^M. The method has the same space for the magnetic field as is used by Lee and Madsen, thus $V_h^M = V_h^L$. The electric field space U_h^M is the space U_h^L of ECHL augmented by some linear terms. Therefore, we refer to this method as the MECHL (modified ECHL).

To define the electric field space, denoted U_h^M, we first give the reference element $(\hat{K}, P_{\hat{K}}, \Sigma_{\hat{K}})$. Again $\hat{K}$ is given by (23), and

$$P_{\hat{K}} = \{\boldsymbol{p} \mid \boldsymbol{p} \in Q_{10} \times Q_{01}\},$$
$$\Sigma_{\hat{K}} = \left\{\boldsymbol{p} \cdot \hat{\boldsymbol{n}}(\boldsymbol{b}_i)\, 1 \leq i \leq 4 \mid, \text{ where } \boldsymbol{b}_i = \frac{\boldsymbol{a}_i + \boldsymbol{a}_{i+1}}{2}(\boldsymbol{a}_5 = \boldsymbol{a}_1)\right\}.$$

In these definitions $\hat{\boldsymbol{n}}$ is the unit outward normal to $\hat{K}$. This choice of $P_{\hat{K}}$ is made since $\vec{\nabla} \times Q_{11} \subset Q_{10} \times Q_{01}$, and we want to construct U_h^M such that $\nabla \times V_h^M \subset U_h^M$. Since V_h^M is an isoparametric space, this implies some care with the way that U_h^M is obtained from the reference element. By computing $\vec{\nabla} \times \psi$ for $\psi \in V_h^M$, we find that the correct choice is

$$U_h^M = \{\boldsymbol{u}^h \mid \boldsymbol{u}^h|_K = ((\det \boldsymbol{J}_K)^{-1}\boldsymbol{J}_K\hat{\boldsymbol{u}}_K) \circ F_K^{-1} \text{ for some } \hat{\boldsymbol{u}}_K \in Q_{10} \times Q_{01} \quad \forall K \in \tau_h\}. \tag{35}$$

This space has as degrees of freedom $\boldsymbol{u}^h \cdot \boldsymbol{n}$ at the midpoint of each edge of each element. However, since we wish to allow ϵ, the permittivity, to be discontinuous, we do not require continuity of $\boldsymbol{u}^h \cdot \boldsymbol{n}$ across element boundaries. So the actual degrees of freedom on element K are

$$\lim_{\substack{\boldsymbol{x} \to \boldsymbol{b} \\ \boldsymbol{x} \in K}} \boldsymbol{u}^h(\boldsymbol{x}) \cdot \boldsymbol{n}_K(\boldsymbol{b}),$$

where $\boldsymbol{n}_K$ is the outward normal to K and $\boldsymbol{b}$ is the midpoint of an edge. If we know that ϵ is continuous, we can decrease the dimension of U_h^M by enforcing continuity of normal components across element boundaries. In that case, U_h^M becomes the lowest-order Raviart–Thomas divergence-conforming space [14] and MECHL becomes essentially the method of Adam et al. [1] (but without mass lumping and with different boundary conditions).

The transformation used in (35) to obtain $\boldsymbol{u}^h|_K$ from $\hat{\boldsymbol{u}}$ implies that

$$\nabla \times V_h^M \subset U_h^M \tag{36}$$

so that the analysis of Monk [12] holds and we can be sure that, on a general mesh, the method converges with error

$$\|(\boldsymbol{E} - \boldsymbol{E}^h)(t)\|_{0,\epsilon} + \|(H - H^h)(t)\|_{0,\mu} = O(h), \tag{37}$$

provided the standard assumptions on isoparametric meshes hold [4], the exact solution is sufficiently smooth, and the initial data for the discrete problem satisfies (37). This is an optimal global estimate since U_h^M does not contain all vector linear polynomials on the reference element.

Note also that in the special case when $\epsilon = \mu = 1$ (or constant) and $\sigma \equiv 0, \boldsymbol{J} \equiv 0$, the fact that $\nabla \times V_h^M \subset U_h^M$ implies that (1) is satisfied pointwise exactly. In particular, we have that

$$\boldsymbol{E}_t^h = \vec{\nabla} \times H^h,$$

and hence $\nabla \cdot \boldsymbol{E}_t^h = 0$. Thus if the initial data $\boldsymbol{E}_0^h$ is chosen so that $\nabla \cdot \boldsymbol{E}_0^h$ is defined and $\nabla \cdot \boldsymbol{E}_0^h = 0$ in Ω, then we can be sure that

$$\nabla \cdot \boldsymbol{E}^h = 0$$

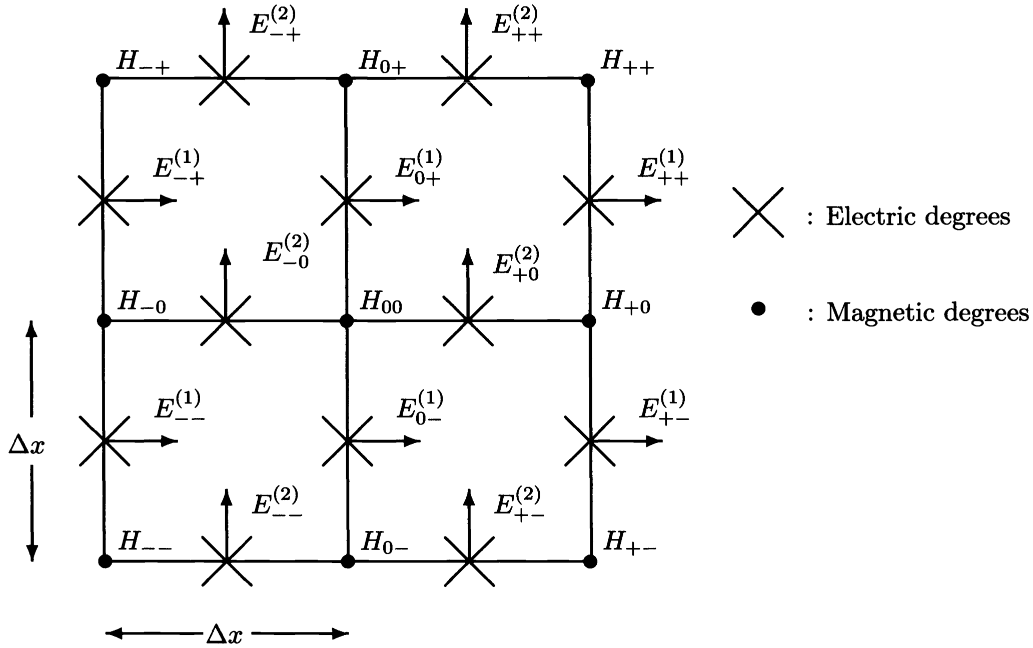


FIG. 2. *The degrees of freedom for* MECHL *on a uniform square grid. In this case we assume that the electric field has continuous normal component across the edge of each element. Thus the electric field degrees of freedom are the normal component of the field at the midpoint of each edge marked* ×. *The magnetic degrees of freedom (marked* •) *are the same as for* ECHL (*see Fig.* 1).

for all time. Thus in some cases, MECHL allows the exact satisfaction of the equations for the electric field. We remark that $\nabla \cdot \boldsymbol{E}_0^h$ is defined if $\boldsymbol{E}_0^h$ is chosen so that the normal components of $\boldsymbol{E}_0^h$ are continuous across each edge of the mesh [13].

We can also analyze MECHL on a uniform grid. We assume, as before, that $\epsilon = \mu = 1, \sigma \equiv 0, \boldsymbol{J} \equiv 0$, and since ϵ is continuous we assume that $\boldsymbol{E}^h$ has continuous normal component across each edge of the mesh. The arrangement of degrees of freedom is shown in Fig. 2. On the uniform mesh, we can derive the following equations for the degrees of freedom.

$$\frac{d}{dt}E^{(1)}_{0+} - \frac{H_{0+} - H_{00}}{\Delta x} = 0, \tag{38}$$

$$\frac{d}{dt}E^{(2)}_{+0} + \frac{H_{+0} - H_{00}}{\Delta x} = 0, \tag{39}$$

$$\begin{aligned} &\frac{d}{dt}\left[\frac{1}{36}[H_{++} + H_{--} + H_{+-} + H_{-+}]\right. \\ &\qquad \left. + \frac{1}{9}[H_{-0} + H_{0-} + H_{+0} + H_{0+}] + \frac{4}{9}H_{00}\right] \\ &\quad + \frac{1}{6\Delta x}\left[[(E^{(2)}_{++} + 4E^{(2)}_{+0} + E^{(2)}_{+-}) - (E^{(2)}_{-+} + 4E^{(2)}_{-0} + E^{(2)}_{--})]\right. \\ &\qquad \left. -[(E^{(1)}_{-+} + 4E^{(1)}_{0+} + E^{(1)}_{++}) - (E^{(1)}_{--} + 4E^{(1)}_{0-} + E^{(1)}_{+-})]\right] = 0. \end{aligned} \tag{40}$$

We see that this is another centered finite-difference analogue of (1)–(2) and hence has local truncation error $O((\Delta x)^2)$. Again the stability of the method implies that in the weighted discrete L^2 norm (cf. (32) and (33)), the order of convergence for the method is $O((\Delta x)^2)$ at the points shown in Fig. 2. The precise form of the estimate is exactly as given in (33), provided $\boldsymbol{r}_h$ and r_h are taken to be the interpolation operators for U_h^M and V_h^M, respectively. This is a superconvergence result since the order at selected points is one greater than the global error expected using U_h^M.

By comparison, a Yee-type [18] finite-difference scheme for this problem, with the grid and degrees of freedom shown in Fig. 2, would be

$$\frac{d}{dt}E_{0+}^{(1)} - \frac{H_{0+} - H_{00}}{\Delta x} = 0, \tag{41}$$

$$\frac{d}{dt}E_{+0}^{(2)} + \frac{H_{+0} - H_{00}}{\Delta x} = 0, \tag{42}$$

$$\frac{d}{dt}H_{00} + \frac{1}{\Delta x}\left[E_{+0}^{(2)} - E_{-0}^{(2)} - [E_{0+}^{(1)} - E_{0-}^{(1)}]\right] = 0. \tag{43}$$

Thus we see that the MECHL finite-element method on a uniform grid is just an averaged version of the Yee method.

3.3. Nédélec's method. This method is based on (14)–(17) and was first proposed in [13] (in three dimensions). The version presented here is modified to allow a discontinuity in μ. The magnetic field space is simple, but the electric field space is a subspace of $H(\text{curl};\Omega)$. Nédélec [13] shows that for a finite-element space U_h^N to be a subspace of $H(\text{curl};\Omega)$ it suffices that functions in U_h^N have continuous tangential component across edges in the mesh (i.e., $\boldsymbol{n} \times \boldsymbol{u}^h$ continuous across each internal edge in the mesh for each $\boldsymbol{u}^h \in U_h^N$). Following Nédélec [13] we define the reference element as follows:

$$\begin{aligned}
&\hat{K} \equiv \text{unit square} \quad \text{(see (23))},\\
&P_{\hat{K}} \equiv Q_{01} \times Q_{10},\\
&\Sigma_{\hat{K}} \equiv \left\{ (\hat{\boldsymbol{u}} \cdot \hat{\boldsymbol{t}})(\hat{\boldsymbol{b}}_i) \mid \boldsymbol{b}_i = \frac{\hat{\boldsymbol{a}}_i + \hat{\boldsymbol{a}}_{i+1}}{2},\ 1 \le i \le 4, \right.\\
&\qquad\qquad \left. \text{where } \boldsymbol{a}_5 = \boldsymbol{a}_1 \text{ and } \hat{\boldsymbol{t}} \text{ is the unit tangent to } \partial\hat{K} \right\}.
\end{aligned}$$

Nédélec [13] shows that a finite-element space constructed from this reference element is $H(\text{curl};\Omega)$-conforming (strictly speaking, Nédélec's results are for $\mathbb{R}^3$ but can be easily reinterpreted for $\mathbb{R}^2$). Care must be taken in the isoparametric method to ensure continuity of tangential components. Thus Nédélec shows that

$$\begin{aligned}
U_h^N = \{\boldsymbol{u}^h \in H(\text{curl};\Omega) | \boldsymbol{u}^h|_K = (\boldsymbol{J}_K^{-T}\hat{\boldsymbol{u}}_K) \circ F_K^{-1},\\
\hat{\boldsymbol{u}}_K \in Q_{01} \times Q_{10} \quad \forall K \in \tau_h\}.
\end{aligned} \tag{44}$$

Using the above space we define

$$U_{h0}^N = \{\boldsymbol{u}^h \in U_h^N \mid \boldsymbol{n} \times \boldsymbol{u}^h = 0 \text{ on } \Gamma\},$$

which can be constructed by enforcing a zero tangential component (zero degree of freedom) on each edge on Γ. The function γ^h required in (16) is the piecewise constant function interpolating γ at the midpoint of each boundary edge. Having defined U_h^N, we can essentially take V_h^N to be the space of piecewise constants. However, we would like $\nabla \times U_h^N \subset V_h^N$ and thus use an isoparametric mapping of the constant space:

$$V_h^N = \{v^h \mid v^h|_K = ((\det \boldsymbol{J}_K)^{-1}\hat{v}_K) \circ F_K^{-1}, \quad \hat{v}_K \in Q_{00} \quad \forall K \in \tau_h\}. \tag{45}$$

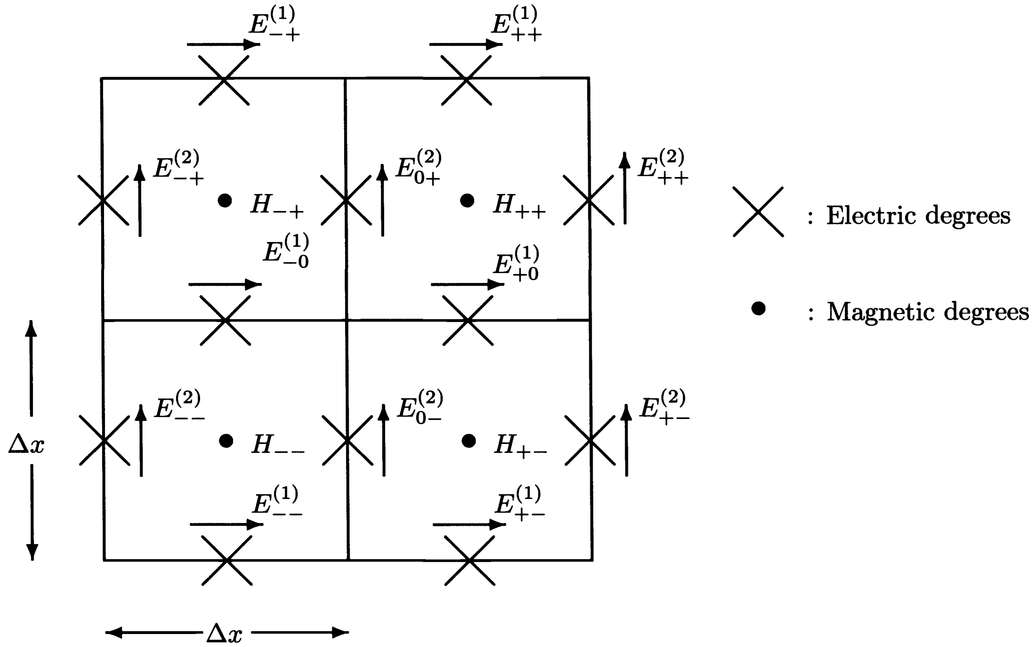


FIG. 3. *The degrees of freedom for* EL$_N$HC. *Electric field degrees of freedom are the tangential components of the electric field at the midpoint of each edge of the mesh, marked* × (*the nearby arrows show the direction of the degree of freedom at the interpolation point*). *The magnetic field has degrees of freedom marked* •.

Since $\nabla \times U_h^N \subset V_h^N$, we know that, provided μ is constant (as it is in all the examples here),

$$\mu H_t^h + \nabla \times \boldsymbol{E}^h = 0$$

pointwise in Ω, and the magnetic field equation is satisfied exactly. Since the electric field is approximated by Nédélec's linear elements, we refer to this method as EL$_N$HC. We emphasize that this is a different method than the method ELHC in [9], which is based on standard continuous linear elements.

This method is analyzed in [11] for the full Maxwell system in three space dimensions. In general, the method has global L^2 error at least $O(h)$ (see (37) for a more precise statement). Also in [11], we prove higher-order convergence rates at special points when the mesh is uniform. This is done as follows. If we use the notation in Fig. 3, we can derive the equations satisfied by the degrees of freedom as follows:

$$\frac{d}{dt}\frac{1}{6}\left\{E^{(1)}_{++} + 4E^{(1)}_{+0} + E^{(1)}_{+-}\right\} - \frac{H_{++} - H_{+-}}{\Delta x} = 0, \tag{46}$$

$$\frac{d}{dt}\frac{1}{6}\left\{E^{(2)}_{++} + 4E^{(2)}_{0+} + E^{(2)}_{-+}\right\} + \frac{H_{++} - H_{-+}}{\Delta x} = 0, \tag{47}$$

$$\frac{d}{dt}H_{++} + \frac{1}{\Delta x}\left\{(E^{(2)}_{++} - E^{(2)}_{0+}) - (E^{(1)}_{++} - E^{(1)}_{+0})\right\} = 0. \tag{48}$$

Again, we see that the method has local truncation error $O((\Delta x)^2)$ and the stability result (22) implies convergence in a suitable weighted discrete L^2 norm at the mesh points in the same sense as for the two previous methods discussed in this section [11].

For this grid the Yee finite-difference scheme [18] is

$$\begin{aligned}
&\frac{d}{dt}\, E^{(1)}_{+0} - \frac{H_{++} - H_{+-}}{\Delta x} = 0,\\
&\frac{d}{dt}\, E^{(2)}_{0+} + \frac{H_{++} - H_{-+}}{\Delta x} = 0,\\
&\frac{d}{dt}\, H_{++} + \frac{1}{\Delta x}\left\{(E^{(2)}_{++} - E^{(2)}_{0+}) - (E^{(1)}_{++} - E^{(1)}_{+0})\right\} = 0,
\end{aligned}$$

which we see is just (41)–(43) with a renaming of nodes. Thus, on a uniform grid, both MECHL and EL$_N$HC are essentially averaged Yee schemes.

3.4. Timestepping. In contrast to the rather unusual finite-element spaces used in the spatial discretization, we use the standard leapfrog method to discretize in time [18]. Each of the methods outlined above gives rise to a matrix problem of the following type. Let $(\vec{E}(t), \vec{H}(t))$ be the vectors of free degrees of freedom for the electric and magnetic field, respectively. Then using either (14)–(17) or (18)–(20), $(\vec{E}(t), \vec{H}(t))$ satisfies a matrix problem of the form

$$M^{\epsilon}_{uu}\frac{d\vec{E}}{dt} + M^{\sigma}_{uu}\vec{E} - C_{uv}\vec{H} = \vec{G}, \tag{49}$$

$$M^{\mu}_{vv}\frac{d\vec{H}}{dt} + [C_{uv}]^T\vec{E} = \vec{F}, \tag{50}$$

where $\vec{G}$ and $\vec{F}$ are vectors taking into account boundary data and the applied currents. For example, in the ECHL (respectively, MECHL) methods (see (25))

$$\begin{aligned}
[M^{\epsilon}_{uu}]_{ij} &= \textstyle\int_\Omega \epsilon\phi_i\phi_j dA, && 1 \le i,j \le N_E,\\
[M^{\sigma}_{uu}]_{ij} &= \textstyle\int_\Omega \sigma\phi_i\phi_j dA, && 1 \le i,j \le N_E,\\
[C_{uv}]_{ij} &= \textstyle\int_\Omega (\vec{\nabla} \times \psi_j)\phi_i dA, && 1 \le j \le N_H, \quad 1 \le i \le N_E,\\
[\vec{G}]_i &= \textstyle\int_\Omega \boldsymbol{J}\phi_i dA, && 1 \le i \le N_E,\\
[M^{\mu}_{vv}]_{ij} &= \textstyle\int_\Omega \mu\psi_i\psi_j dA, && 1 \le i,j \le N_H,\\
[\vec{F}]_j &= \textstyle\int_\Gamma \gamma\psi_j dA, && 1 \le j \le N_H,
\end{aligned}$$

where $\{\phi_i\}_{i=1}^{N_E}$ is a basis for U_h^L (respectively, U_h^M) and $\{\psi_i\}_{i=1}^{N_E}$ is a basis for V_h^L (respectively, V_h^M). Similar definitions hold for the Nédélec method after allowing for the fact that boundary degrees of freedom for U_h^N are specified via the boundary data (16). The fully discrete scheme is to compute a sequence $\{\vec{E}^n, \vec{H}^{n+1/2}\}_{n=0}^{\infty}$ that approximates $\{\vec{E}(t_n), \vec{H}(t_{n+1/2})\}_{n=1}^{\infty}$, where $t_n = n\Delta t$ and $t_{n+1/2} = (n + 1/2)\Delta t$. Given $(\vec{E}^n, \vec{H}^{n+1/2})$, we compute $(\vec{E}^{n+1}, \vec{H}^{n+3/2})$ by solving the system

$$M^{\epsilon}_{uu}\left(\frac{\vec{E}^{n+1} - \vec{E}^n}{\Delta t}\right) + M^{\sigma}_{uu}\left(\frac{\vec{E}^{n+1} + \vec{E}^n}{2}\right) - C_{uv}\vec{H}^{n+1/2} = \vec{G}^{n+1/2} \tag{51}$$

where $\vec{G}^{n+1/2} = \vec{G}(t_{n+1/2})$, then solving

$$M^{\mu}_{vv}\left(\frac{\vec{H}^{n+3/2} - \vec{H}^{n+1/2}}{\Delta t}\right) + [C_{uv}]^T\vec{E}^{n+1} = \vec{F}^{n+1}. \tag{52}$$

For ECHL, M^{ϵ}_{uu} and M^{σ}_{uu} are diagonal, while for MECHL they are block diagonal (with 4 × 4 diagonal blocks) and thus (51) may be solved rapidly. M^{μ}_{vv} is sparse (and identical for ECHL and MECHL) and we solve (52) using the preconditioned conjugate gradient method [7], using the mass-lumped matrix as a preconditioner. Precisely, we define the diagonal matrix $\overline{M}^{\mu}_{vv}$ with diagonal entry $[\overline{M}^{\mu}_{vv}]_i$ by

$$[\overline{M}^{\mu}_{vv}]_i = \sum_{j=1}^{N_H} [M^{\mu}_{vv}]_{ij},$$

and then solve

$$C\vec{Y} = (\overline{M}^{\mu}_{vv})^{-1/2} \left(\vec{F}^{n+1} - [C_{uv}]^T \vec{E}^{n+1} \right)$$

where $C = (\overline{M}^{\mu}_{vv})^{-1/2}(M^{\mu}_{vv})(\overline{M}^{\mu}_{vv})^{-1/2}$, and hence compute

$$\vec{H}^{n+3/2} = \vec{H}^{n+1/2} + \Delta t (\overline{M}^{\mu}_{vv})^{-1/2} \vec{Y}.$$

This method converges rapidly and we always iterate to completion (i.e., so that successive conjugate gradient iterates differ by less than 10^{-6} in the L^2 norm). In practice, a coarser error tolerance would produce good results.

In the case of the EL$_N$HC or Nédélec method, M^{μ}_{vv} is diagonal and hence easily inverted. Then (51) must be solved by conjugate gradients, and we also precondition with the mass-lumped matrix, although on a regular grid this is not necessary. The number of conjugate gradient iterations per step for the preconditioned EL$_N$HC matrix is usually less than for the preconditioned ECHL or MECHL methods. However, for EL$_N$HC, the dimension of M^{ϵ}_{uu} is approximately the number of edges in the mesh, while for ECHL/MECHL the dimension of M^{μ}_{vv} is the number of nodes in the mesh. Thus we must solve a larger system when timestepping EL$_N$HC than ECHL/MECHL.

To obtain $\vec{H}^{1/2}$, we use a Runge–Kutta-type method. First, we predict a value for $\vec{E}^{1/4}$ by solving

$$M^{\epsilon}_{uu} \left(\frac{\vec{E}^{1/4} - \vec{E}^0}{(\Delta t)/4} \right) + M^{\sigma}_{uu} \vec{E}^0 - C_{uv} \vec{H}^0 = \vec{J}^0,$$

where $\vec{E}^0$ and $\vec{H}^0$ are obtained by interpolating the initial data. Then we compute $\vec{H}^{1/2}$ by solving

$$M^{\mu}_{vv} \left(\frac{\vec{H}^{1/2} - \vec{H}^0}{(\Delta t)/2} \right) + [C_{uv}]^T \vec{E}^{1/4} = \vec{F}^{1/4}.$$

This procedure produces an $O((\Delta t)^2)$ approximation to $\vec{H}(t_{1/2})$. The timestepping scheme (51)–(52) is an $O((\Delta t)^2)$ scheme locally, and we expect that if $\Delta t/h$ is sufficiently small (a standard CFL condition [15]) the overall method will be stable and second order. On uniform grids, this condition is discussed further in §4. For more general grids and problems, convergence has been confirmed numerically, but has not yet been proven for the methods used in this paper.

4. Dispersion analysis. In this section, we assume that ϵ and μ are constants (which for simplicity we take to be unity), $\sigma \equiv 0$, $\boldsymbol{J} \equiv 0$. In this case, if Ω is an infinite domain (or if Ω is finite and γ chosen appropriately), (1) and (2) have solutions of the form

$$E(x,t) = E_0 \exp\left(i(wt - k \cdot x)\right), \tag{53}$$

$$H(x,t) = H_0 \exp\left(i(wt - k \cdot x)\right), \tag{54}$$

where k is a constant vector. Substituting these solutions into (1) and (2) (recalling $\epsilon = \mu = 1$) shows that w and k are related by the dispersion relation

$$w = |k|, \tag{55}$$

where $|k|$ is the Euclidean norm of k (other solutions are $w = 0$ or $w = -|k|$). The group velocity C is given [17] by

$$C = \nabla_{k} w = \frac{k}{|k|}, \tag{56}$$

and hence regardless of the wave number $|k|$ all plane waves move with the same group speed $|C|$. We can also analyze the dispersion relation for each of the numerical methods under consideration. Such an analysis describes how waves propagate in the numerical method far from boundaries, and gives information on the expected accuracy of the methods [17].

For a dispersion analysis, we assume a uniform grid of square elements of dimension $\Delta x \times \Delta x$ (see Figs. 1–3). We assume an infinite grid, and seek solutions of the discrete equations of the form (53)–(54). First we consider the case of exact integration in time, and start with the ECHL method. Substituting (53) and (54) into (29)–(31) we find that, if $E_0 = (E_0^{(1)}, E_0^{(2)}), \zeta_1 = k^{(1)}\Delta x, \zeta_2 = k^{(2)}\Delta x$, $\eta = w\Delta x$, and

$$\gamma(\zeta_1,\zeta_2) = \frac{4}{9} + \frac{2}{9}(\cos(\zeta_1) + \cos(\zeta_2)) + \frac{1}{18}(\cos(\zeta_1 - \zeta_2) + \cos(\zeta_1 + \zeta_2)), \tag{57}$$

then

$$\begin{pmatrix} \sin\left(\frac{\zeta_2-\zeta_1}{2}\right) + \sin\left(\frac{\zeta_1+\zeta_2}{2}\right) & \sin\left(\frac{\zeta_2-\zeta_1}{2}\right) - \sin\left(\frac{\zeta_2+\zeta_1}{2}\right) & \eta\,\gamma(\zeta_1,\zeta_2) \\ \eta & 0 & \sin\left(\frac{\zeta_2-\zeta_1}{2}\right) + \sin\left(\frac{\zeta_2+\zeta_1}{2}\right) \\ 0 & \eta & -\sin\left(\frac{\zeta_1-\zeta_2}{2}\right) - \sin\left(\frac{\zeta_1+\zeta_2}{2}\right) \end{pmatrix}$$

$$\cdot \begin{pmatrix} E_0^{(1)} \\ E_0^{(2)} \\ H_0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}. \tag{58}$$

For (58) to have a nontrivial solution, the determinant of the matrix must be zero. This defines η as a function of (ζ_1, ζ_2). If η°_{ECHL} denotes the positive solution, we find that

$$\eta^{\circ}_{ECHL}(\zeta_1,\zeta_2) = \left\{ \frac{6\sqrt{\sin^2\left(\frac{\zeta_1-\zeta_2}{2}\right) + \sin^2\left(\frac{\zeta_1+\zeta_2}{2}\right)}}{\sqrt{8 + 4\cos(\zeta_2) + \cos(\zeta_1 - \zeta_2) + 4\cos(\zeta_1) + \cos(\zeta_1 + \zeta_2)}} \right\} \tag{59}$$

(other solutions are $\eta = 0$ or η is the negative of the above expression). Hence, depending on the magnitude and direction of k, the numerically computed wave possesses an erroneous phase. This implies that a plane wave of the form (53) and (54) generally moves in the wrong direction at the wrong speed. Of course if $|\boldsymbol{\zeta}|$ is small, the frequency w_{ECHL} for ECHL with exact time integration is

$$w^{\circ}_{ECHL}(\zeta_1,\zeta_2,\Delta x) \equiv \frac{\eta^{\circ}_{ECHL}(\zeta_1,\zeta_2)}{\Delta x} \sim \sqrt{\frac{\zeta_1^2+\zeta_2^2}{(\Delta x)^2}} = |\boldsymbol{k}|,$$

so that waves with a sufficiently long wavelength compared to the grid move with essentially the correct phase. To derive the frequency w for the fully discrete scheme, we see that discretization in time corresponds to replacing η in (58) by $2\frac{\Delta x}{\Delta t}\sin\left(\frac{w\Delta t}{2}\right)$, and thus if we denote the frequency for the fully discrete scheme by $w^{\Delta t}_{ECHL}(\zeta_1,\zeta_2,\lambda,\Delta t)$ and let $\lambda = \Delta t/\Delta x$, we find that

$$w^{\Delta t}_{ECHL}(\zeta_1,\zeta_2,\lambda,\Delta t) = \frac{2}{\Delta t}\sin^{-1}\left(\frac{\lambda\eta^{\circ}_{ECHL}(\zeta_1,\zeta_2)}{2}\right) \tag{60}$$

and the group velocity

$$\boldsymbol{C}^{\Delta t}_{ECHL}(\zeta_1,\zeta_2,\lambda) = \frac{2}{\lambda}\nabla_{\zeta}\left[\sin^{-1}\left(\frac{\lambda\eta^{\circ}_{ECHL}(\zeta_1,\zeta_2)}{2}\right)\right]. \tag{61}$$

In Fig. 4 we show a plot of $|\boldsymbol{C}^{\Delta t}_{ECHL}-\boldsymbol{C}|$ as a function of $\boldsymbol{\zeta} \in [0,\pi]\times[0,\pi]$ (higher values of $\boldsymbol{\zeta}$ are aliased onto the grid [17]). $\boldsymbol{C}^{\Delta t}_{ECHL}$ depends only on $\lambda = \Delta t/\Delta x$, not Δt or Δx individually, and we choose $\lambda = 0.25$ in accordance with later numerical results (graphs for smaller λ are similar). The shaded area is where $|\boldsymbol{C}^{\Delta t}_{ECHL} - \boldsymbol{C}| < 0.1$, so that for a given $\boldsymbol{k}$, if we choose Δx such that $\boldsymbol{k}\Delta x$ lies in the shaded region, we can be sure that the wave will move with a wave speed and direction in error by less than 10 percent. A diagram like Fig. 4 can be invaluable for choosing step sizes.

Note also that λ must be chosen sufficiently small that

$$\lambda\left(\max_{(\zeta_1,\zeta_2)\in[0,\pi]\times[0,\pi]}|\eta^{\circ}_{ECHL}(\zeta_1,\zeta_2)|\right) \le 2, \tag{62}$$

so that $w^{\Delta t}_{ECHL}$ is real. This implies the condition that

$$\lambda \le \frac{2}{\sqrt{12}} \simeq 0.577\cdots. \tag{63}$$

With mass lumping, an examination of the stencil for ECHL suggests a CFL condition $\lambda \le 1$, but this CFL condition is not correct when the full finite-element mass matrix is used. Thus for the split-step, mixed, finite-element methods considered here some extra care is needed to ensure stability (the condition $\lambda \le 0.577\cdots$ has been checked numerically, and gives an accurate picture of stability for ECHL applied to boundary value problems on a uniform grid (see §5.1)).

We can perform the same dispersion analysis on EL$_N$HC and MECHL. Not surprisingly, these methods have identical dispersion relations (when $\epsilon = \mu = 1$) since both methods are restrictions of three-dimensional methods based on the same spaces (cf. [13] and [11]). Hence we need only analyze EL$_N$HC. In this case the matrix corresponding to (58) is

$$\begin{pmatrix} 2\sin\left(\frac{\zeta_2}{2}\right) & -2\sin\left(\frac{\zeta_1}{2}\right) & \eta \\ \eta\left(\frac{4+2\cos(\zeta_2)}{6}\right) & 0 & 2\sin\left(\frac{\zeta_2}{2}\right) \\ 0 & \eta\left(\frac{4+2\cos(\zeta_1)}{6}\right) & -2\sin\left(\frac{\zeta_1}{2}\right) \end{pmatrix},$$

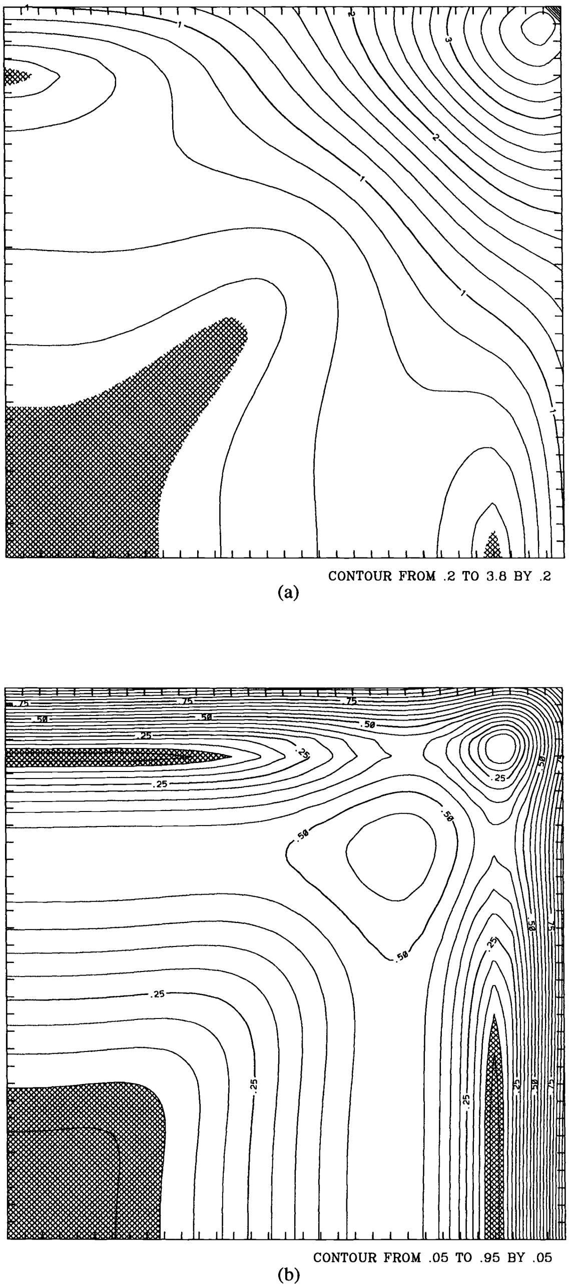


FIG. 4. *Graphs of the error in the group velocity for* $k\Delta x \in [0,\pi] \times [0,\pi]$. *We show contours of* $|C^{\Delta t} - C|$ *where* $C^{\Delta t}$ *is the numerical group velocity and* C *is the exact group velocity. The shaded region is the region in which* $|C^{\Delta t} - C| < 0.1$. *All three methods have the same dispersion behavior for waves oriented along grid lines, and this implies that the worst-case error for a particular* $|k|\Delta x$ *is the same for each method.* (a) *The group velocity error for* ECHL. *For values of* $|k|\Delta x \sim \pi$ *the group velocity can be in error by up to about* 400 *percent.* (b) *The group velocity error for* MECHL *and* EL$_N$HC. *In this case the maximum error is less than for* ECHL. *The error contours for* MECHL/EL$_N$HC *are more nearly circular than for* ECHL, *implying less grid anisotropy.*

and thus

$$\overset{\circ}{\eta}_{EL_NHC}(\zeta_1,\zeta_2)$$
$$= 2\sqrt{3}\sqrt{\frac{2\sin^2\left(\frac{\zeta_1}{2}\right)+\cos(\zeta_2)\sin^2\left(\frac{\zeta_1}{2}\right)+2\sin^2\left(\frac{\zeta_2}{2}\right)+\cos(\zeta_1)\sin^2\left(\frac{\zeta_2}{2}\right)}{4+2\cos(\zeta_1)+2\cos(\zeta_2)+\cos(\zeta_1)\cos(\zeta_2)}}.$$

The condition corresponding to (63) is

$$\lambda \leq 2/\sqrt{24}.$$

For the fully discrete method, $w^{\Delta t}_{EL_NHC}$ and $\boldsymbol{C}^{\Delta t}_{EL_NHC}$ are given by the analogues of (60) and (61). In Fig. 4(b) we show $|\boldsymbol{C}^{\Delta t}_{EL_NHC} - \boldsymbol{C}|$ against (ζ_1, ζ_2) and again shade the region with less than 10 percent error. Notice that the EL$_N$HC, ECHL, and MECHL methods have the same dispersion behavior for waves with $\boldsymbol{k} = (k_1, 0)$ or $\boldsymbol{k} = (0, k_2)$ (i.e., parallel to grid lines). This is because MECHL and ECHL are identical schemes in these cases (see (29)–(31), (38)–(40)), and MECHL and EL$_N$HC are based on the same spaces (but used in different ways) in three dimensions.

Clearly, from Fig. 4, the area in the (ζ_1, ζ_2) plane for which the numerical group velocity is in error by less than 10 percent is larger for ECHL than MECHL or EL$_N$HC. On the other hand, the error contours for MECHL/EL$_N$HC are more nearly circular than for ECHL, implying less grid anisotropy. Furthermore, the maximum error for MECHL and EL$_N$HC is approximately 100 percent, whereas that for ECHL is 400 percent. Thus ECHL will cause much worse dispersion for very high frequency waves than either MECHL or EL$_N$HC. In general, we must choose a grid on the basis of the worst-case behavior of the method. This occurs in all three methods for waves along the coordinate axis, and is identical for the three methods.

Since the worst case for plane wave propagation is for waves along a grid line, we can gain useful information on error by considering a one-dimensional plot of group velocity error, as shown in Fig. 5. Here we consider waves with wave number $\boldsymbol{k} = (k_1, 0)$ moving along the x-axis. We plot group velocity error against number of grid cells per half wave length defined by $N_p = \pi/(k_1 \Delta x)$. This graph implies that for an error of 10 percent in the group velocity, we need about eight grid cells per wavelength. For more complex waves (i.e., wave consisting of a superposition of plane waves), we can still use Fig. 5 by considering the plane wave components separately.

On the basis of dispersion, it is difficult to choose between the three methods. In addition, in real problems the mesh is not uniform, the coefficients not constant, and boundary conditions are important. We investigate these problems in §5 using essentially the same numerical examples as [9].

All algebra in the preceding section was manipulated using Mathematica.

5. Numerical results.

In this section, we further compare the three finite-element methods described in §3 by applying them to selected numerical examples. The examples are all taken from [9].

5.1. Plane wave propagation.

We shall compare the numerical solution of a simple wave propagation problem with the exact solution. We choose the functions in Maxwell's equations as follows: $\epsilon = \mu = 1$, $\sigma = 0$, and $\boldsymbol{J} = 0$. Then an exact solution of Maxwell's equations (on the entire plane) is

$$\boldsymbol{E}_e(\boldsymbol{x}, t) = \begin{pmatrix} -k_2 \\ k_1 \end{pmatrix} g(t - \boldsymbol{k}\cdot\boldsymbol{x}),$$
$$H_e(\boldsymbol{x}, t) = g(t - \boldsymbol{k}\cdot\boldsymbol{x}),$$

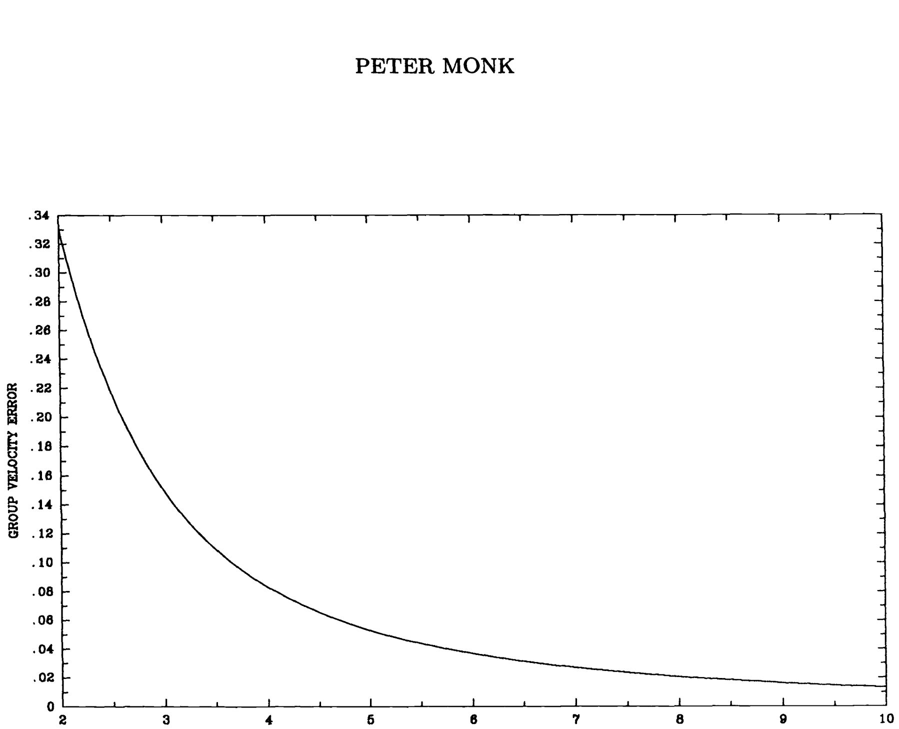


FIG. 5. *A graph of the error in the group velocity for a wave traveling along the x-axis with wave number $k = (k_1, 0)$. The error is plotted against number of grid blocks per half wave length defined as $N_p = \pi/(k_1 \Delta x)$. The* CFL *number $\lambda = 0.25$. A wave parallel to a grid direction is the worst case for all three methods. This graph can be used to select the grid size.*

where we take $\boldsymbol{k} = (\cos(1), \sin(1))$ and

$$g(s) = \begin{cases} \frac{\exp(-10(s-1)^2) - \exp(-10)}{1 - \exp(-10)}, & 0 \le s \le 2, \\ 0, & s > 2 \text{ or } s < 0. \end{cases} \tag{64}$$

We choose a finite domain $\Omega = [0,2] \times [0,2]$ and take $\boldsymbol{E}_0 = 0$, $H_0 = 0$, and $\boldsymbol{J}_0 = 0$. The boundary data is chosen consistent with the above exact solution so that $\gamma = \boldsymbol{n} \times \boldsymbol{E}_e$. We discretize the problem using a uniform grid on $\Omega = [0,2] \times [0,2]$ with N subintervals on each edge (thus N^2 quadrilaterals). In this case $\Delta x = 2/N$, $h = \sqrt{2}\Delta x$, and we choose the CFL parameter $\lambda = \Delta t / \Delta x = 0.25$. For each method, and a variety of N, we integrate until $t = 2$, and then compute the relative discrete L^2 norm error in the magnetic field H (for simplicity). Thus we compute

$$\|(H_e - H^h)(t)\|_{L^2,h} = \sqrt{\sum_{i=1}^{N_H} |H_e(\boldsymbol{x}_i, t) - H^h(\boldsymbol{x}_i, t)|^2}$$

at $t = 2$ where $\boldsymbol{x}_i$ is the ith node if using ECHL or MECHL, or the centroid of the ith quadrilateral if using EL$_N$HC. Then we define

$$\|H_e - H^h\|_{\text{rel}} = \frac{\|H_e - H^h\|_{L^2,h}}{\|H_e\|_{L^2,h}}. \tag{65}$$

A plot of error against N and error against CPU time (on a SUN SparcStation 1) is shown in Fig. 6. The plot of error against N confirms that on a uniform grid all three

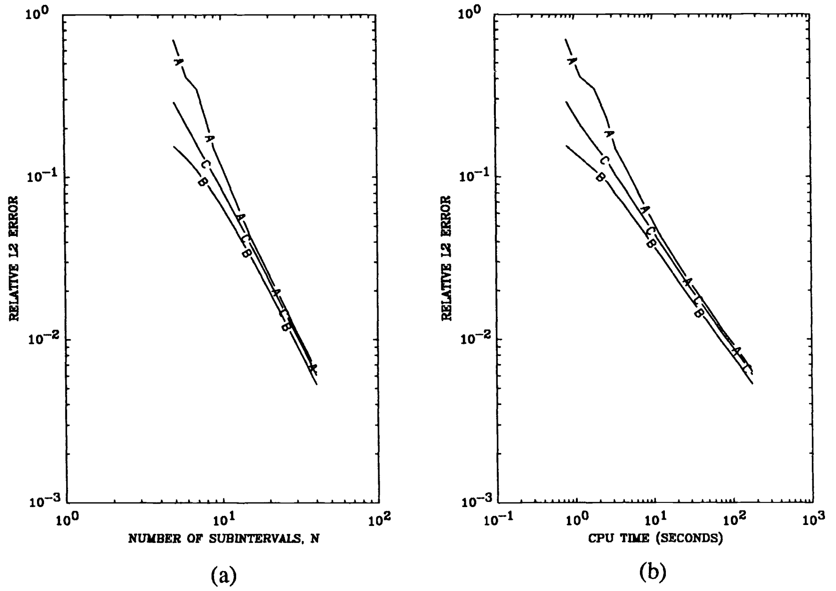


FIG. 6. *Plots of error against grid parameter N and* CPU *time for the example in* §5.1. *A plane wave moves across a uniform grid at an angle to the grid lines. The error reported is the relative discrete L^2 error given by* (65). *In each case* —A— *denotes* ECHL, —B— *denotes* EL$_N$HC, *and* —C— *denotes* MECHL. (a) *A plot of error against N. Here $h = 2\sqrt{2}/N$. This graph suggests second-order convergence in the norm of* (65) *for all three methods.* EL$_N$HC *is notably more accurate for coarse meshes.* (b) *A graph of error against* CPU *time* (*on a* SUN *SparcStation* 1). *Except possibly for very fine grids,* EL$_N$HC *produces a given error in least time* (*but see* §5.3). *For this example the conjugate gradient method in* EL$_N$HC *was not preconditioned, since satisfactory convergence was seen without preconditioning.*

methods are ultimately second-order convergent at the degrees of freedom (the slope of the graphs is 2). For a given N, EL$_N$HC is always the most accurate and ECHL the least accurate. The disparity is particularly obvious for small N. In view of the dispersion analysis, the difference between MECHL and EL$_N$HC must be due to the way that the boundary data is imposed. A slightly different picture emerges when we view error against CPU time. For a given error, EL$_N$HC is the most rapid method, but for a very low error (a fine mesh), it appears that ECHL may become the fastest method. Most time is spent in the conjugate gradient solver. ECHL and MECHL solve the same matrix problem, which has about half as many unknowns as EL$_N$HC. However, despite preconditioning, the ECHL/MECHL takes approximately twice as many iterations to solve the matrix problem to comparable accuracy.

Using this example, we have also checked the validity of the stability estimates (62) and (63) in the case of a boundary value problem. For the EL$_N$HC, scheme (63) implies that $\lambda \leq 2/\sqrt{24}$ is necessary for stability (for the pure initial value problem). When $N = 30$ this implies that $\Delta t \leq 0.0272\cdots$. We find divergence when $\Delta t = 0.029$, but find satisfactory results with $\Delta t = 0.027$ (at least up to time $t = 4$). For ECHL, $\lambda \leq 2/\sqrt{12}$, and hence if $N = 30, \Delta t \leq 0.0385\cdots$. When $\Delta t = 0.04$ the method is unstable, but converges satisfactorily until $t = 4$ when $\Delta t = 0.038$. These results indicate that (62) and (63) are pertinent to the stability of the boundary value problem.

5.2. Simple scattering. In the previous example we used a uniform grid and a plane wave. Our next example uses a nonuniform grid (which is uniform in polar coordinates) and more complex wave motion. Again following [9], we consider an infinite domain problem of scattering of a plane wave off a circular perfect conductor. We consider a plane wave

$$\boldsymbol{E}_i(\boldsymbol{x},t) = \begin{pmatrix} 0 \\ \sqrt{\mu_0/\epsilon_0}\, g((t-(x-0.1)\sqrt{\epsilon_0\mu_0})10^9) \end{pmatrix}, \tag{66}$$

$$H_i(\boldsymbol{x},t) = g((t-(x-0.1)\sqrt{\epsilon_0\mu_0})10^9) \tag{67}$$

(where g is given by (64)) incident on a perfectly conducting circular cylinder of radius 0.1m centered at the origin. We choose

$$\epsilon = \epsilon_0 = 8.85 \times 10^{-12} F/m^2, \qquad \mu = \mu_0 = 1.2566 \times 10^{-6} N/A^2.$$

Using special function theory, an exact solution is available for this problem [8].

For the numerical problem we take an annular domain with inner radius 0.1m and an outer radius of 1.1m. Since the problem is symmetric about the x-axis we use the half-domain

$$\Omega = \left\{(x,y) \mid 0.1 < \sqrt{x^2+y^2} < 1.1, y > 0\right\}.$$

On $y = 0$ we impose the symmetry condition $\boldsymbol{n} \times \boldsymbol{E} = 0$, and on $r = 1.1$ we impose the perfect reflecting boundary condition $\boldsymbol{n} \times \boldsymbol{E} = \gamma \equiv 0$. On $r = 0.1$ we impose $\boldsymbol{n} \times \boldsymbol{E} = \gamma \equiv \boldsymbol{n} \times \boldsymbol{E}_i$, which describes how the incident field scatters off a perfect conductor. The boundary condition $\boldsymbol{n} \times \boldsymbol{E} = 0$ on $r = 1.1$ generates spurious reflections for times greater than about time $t = 4$ nanoseconds (ns), so the exact solution is not useful beyond that time. The grid used has mesh points distributed uniformly in the (r, θ) plane, i.e., the mesh points are (r_i, θ_j), $0 \le i \le N_r$, $0 \le j \le N_\theta$, where

$$r_i = 0.1 + \frac{i}{N_r} \quad \text{and} \quad \theta_j = \pi\frac{j}{N_\theta}.$$

In Fig. 7, we show the computed and exact solutions at two interpolation points when $N_r = 30$, $N_\theta = 15$, and $\Delta t = 1 \times 10^{-3}$. With the small timestep used here, most of the error is due to spatial discretization. These results can be compared to [9, Fig. 3]. Clearly, all three methods compute reasonable solutions.

Table 1 shows the relative L_2 error in space at various times (error defined by (65)). The error ratios are consistent with second-order convergence (the ratios are shown in parentheses in the table). The small timestep $\Delta t = 1 \times 10^{-3}$ is chosen since we wish to focus on spatial error, and numerical experiments show that, with this timestep, the timestepping error is negligible compared to the spatial error. Thus the second-order convergence shown in Table 1 is evidence of second-order convergence in the spatial error. As yet, we have no theory to predict this rate of convergence. However, the fact of second-order convergence at the mesh points is encouraging since the mesh is slightly nonuniform.

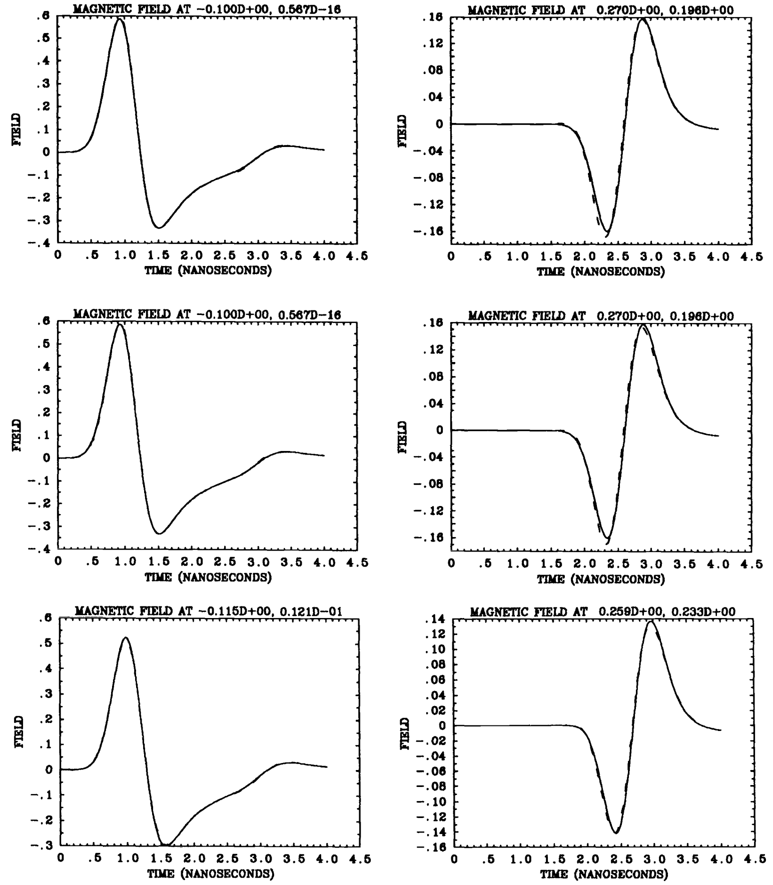


FIG. 7. *A comparison of exact and numerical solution for the simple scattering problem in* §5.2 *at selected spatial points. The exact solution is the solid line and the dashed line shows the computed solution. Here* $N_r = 30$, $N_\theta = 15$, *and* $\Delta t = 1 \times 10^{-3}$. *The coordinates of the spatial points are shown on each plot, and are chosen to be the interpolation point closest to the points A and B in* [9, *Fig.* 2]. *Top row: Results for* ECHL. *Middle row: Results for* MECHL. *Bottom row: Results for* EL$_N$HC.

5.3. Scattering by a dielectric cylinder. Our final example investigates a very nonuniform mesh and a case in which ϵ is discontinuous. Again the example is taken from [9]. At a line of discontinuity L separating two regions Ω_1 and Ω_2 with $\epsilon = \epsilon_1$ in Ω_1 and $\epsilon = \epsilon_2$ in Ω_2, we have the continuity condition

$$\epsilon_1(\boldsymbol{n} \cdot \boldsymbol{E})_1 = \epsilon_2(\boldsymbol{n} \cdot \boldsymbol{E})_2, \tag{68}$$

so there is a jump in the normal component of the electric field across L (cf. [9]). If standard continuous finite elements are used to discretize $\boldsymbol{E}$, a complex modification must be made along L to ensure (68) [9]. This modification is not required with any of the methods in this paper, although L must coincide with mesh lines.

To construct an exact solution, we consider the infinite domain-scattering problem

TABLE 1

Relative L_2 error in the magnetic field (as defined by (65)) at various times for two different discretizations of the example in §5.2 (see text for details of the domain and mesh). The numbers in parentheses give the ratio of errors in the table with a ratio of 4 being exactly second-order convergence. A ratio of 3.7 indicates $O(h^{1.9})$ convergence. In this case $\Delta t = 1 \times 10^{-3}$, thus the second-order convergence suggested by this table is the result of second-order convergence of spatial error and not due to the time discretization. By $t = 4$, the solution is polluted by reflections from the artificial boundary.

	Time (ns)	ECHL	MECHL	EL$_N$HC
$N_r = 30$	1	0.0222	0.0125	0.0180
$N_\theta = 15$	2	0.0638	0.0623	0.0464
	3	0.143	0.157	0.107
	4	0.337	0.358	0.236
$N_r = 60$	1	0.00547 (4.1)	0.00338 (3.7)	0.00485 (3.7)
$N_\theta = 30$	2	0.0161 (4.0)	0.0164 (3.8)	0.0124 (3.7)
	3	0.0366 (3.9)	0.0402 (3.9)	0.0289 (3.7)
	4	0.124 (2.7)	0.128 (2.8)	0.0982 (2.4)

in which a plane wave is incident on an inhomogeneous cylinder of radius 0.25m centered at the origin. In this cylinder we assume that the dielectric constant $\epsilon = \epsilon_0/16$ and $\mu = \mu_0$ (see also Fig. 8). This problem can be solved by using special function theory on the infinite domain [8].

To approximate the infinite domain problem, we use the domain shown in Fig. 8. We use the half-domain $y \geq 0$ since the problem is symmetric about the x-axis, and use a symmetry boundary condition on the line $y = 0$. Following [9], we take $\boldsymbol{E}_0 = 0$, $H_0 = 0$, $\sigma \equiv 0$, $J \equiv 0$, and $\gamma = 0$ on $\Gamma\backslash\Sigma$. On Σ we take

$$\gamma = \begin{cases} 1/2\sqrt{\mu_0/\epsilon_0}(1 - \cos 2\pi t), & 0 \leq t \leq 1, \\ 0, & t > 1. \end{cases}$$

A wave with the cross-section of γ above has more slowly decaying harmonics than the incident wave in §5.2. Thus dispersion is more severe for this example than for the example in §5.2.

Using the finite domain shown in Fig. 8(a), we can compute an accurate approximate solution to the infinite domain-scattering problem until reflections from the artificial boundary are significant. We have not attempted to use absorbing boundary conditions on this boundary and instead, to minimize spurious reflections in the output shown in Fig. 9, we compute only up to $t = 6ns$.

We use a mesh similar to the one in [9]. An example is shown in Fig. 8(b) when $N = 16$. In this case N refers to the number of subintervals along the line $x = -1$. Table 2 shows the relative L^2 error on the subdomain $\{(x, y) \mid |x| \leq 0.5,\ 0 \leq y \leq 0.5\}$ at various times for $N = 16$ and $N = 32$ ($N = 32$ corresponds to subdividing every quadrilateral in the $N = 16$ mesh into four subquadrilaterals). We compute the error on a subdomain for two reasons: first, we want to investigate error behavior on the nonuniform portion of the mesh; and second, we want to avoid errors due to spurious reflections from the boundary. The timestep used gives an approximate CFL number of 0.25 in the inner circle where $\epsilon = \epsilon_0/16$. The results in Table 2 are consistent with an order convergence of at least $O(h^{1.6})$ for all three methods. It thus appears that three-halfs-order or possibly even second-order superconvergence is seen even on nonuniform meshes and in the presence of material discontinuities. This is unexpected and needs to be justified by further numerical and theoretical investigations. It is possible that for finer meshes the order of convergence would approach first order.

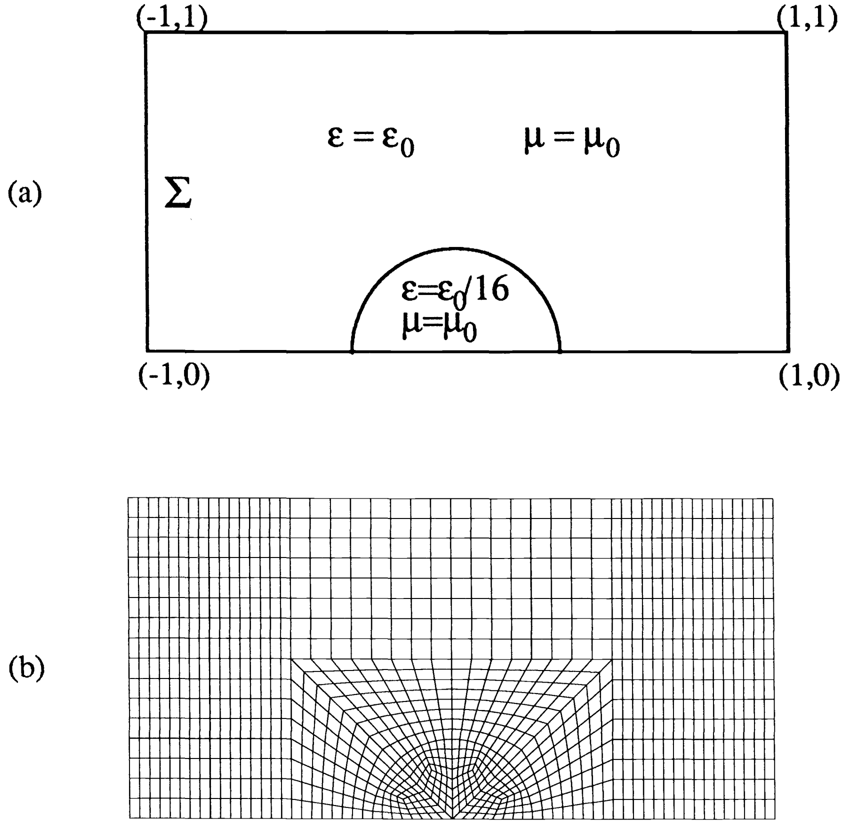


FIG. 8. *The domain Ω for the example in* §5.3. (a) *We show the domain Ω indicating the values of the material parameters in each subdomain. Σ marks the boundary edge on which inhomogeneous boundary data is specified.* (b) *An example of the mesh. Here $N = 16$ (N is number of subintervals along $x = -1$).*

TABLE 2

Relative L_2 error in the magnetic field (as defined by (65)) *on the subdomain $\{(x, y) \mid |x| \leq 0.5\ 0 \leq y \leq 1\}$ at various times for two different discretizations of the example in* §5.3 (*see text and Fig.* 8 *for details of the domain and mesh). In this case the* CFL *number λ is approximately* 0.25 *in the inner domain. The numbers in parentheses give the ratio of errors in the table. Errors at all times show faster than $O(h)$ convergence (a decrease of error by a factor of* 3 *is $O(h^{1.6})$) which is unexpected.*

	Time (ns)	ECHL	MECHL	EL$_N$HC
$N = 16$ $\Delta t = 0.001$	2	0.234	0.235	0.263
	3	0.226	0.268	0.215
	4	0.243	0.291	0.245
	5	0.406	0.447	0.407
	6	0.551	0.545	0.383
$N = 32$ $\Delta t = 0.0005$	2	0.0730 (3.2)	0.0748 (3.1)	0.0762 (3.5)
	3	0.0635 (3.6)	0.0757 (3.5)	0.0635 (3.4)
	4	0.0719 (3.4)	0.0891 (3.3)	0.0756 (3.2)
	5	0.132 (3.1)	0.163 (2.7)	0.140 (2.9)
	6	0.181 (3.0)	0.208 (2.6)	0.135 (2.8)

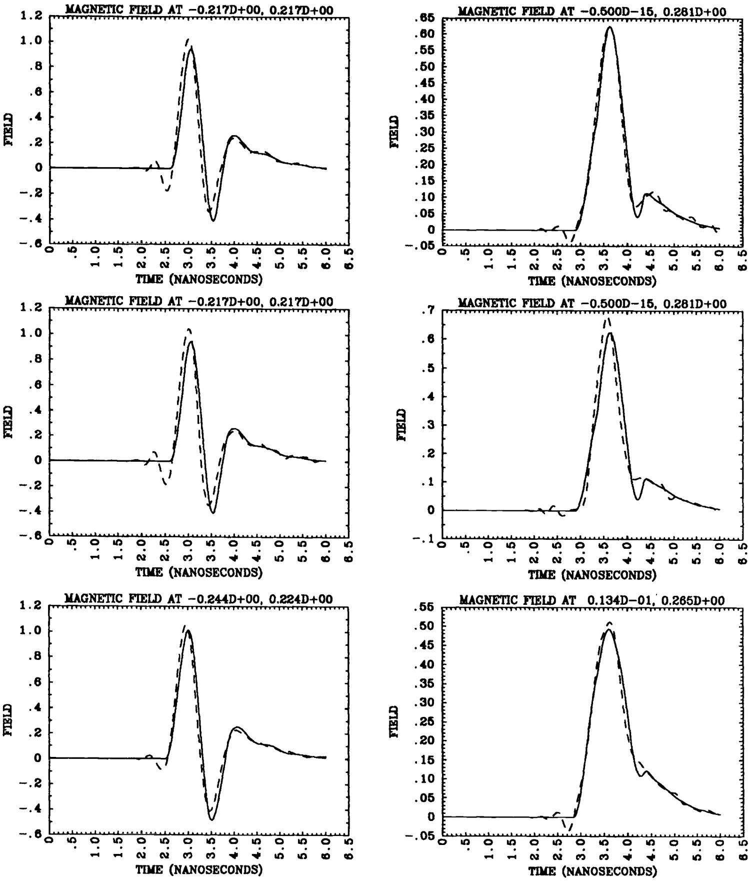


FIG. 9. *A comparison of exact and numerical solution for the simple scattering problem in* §5.3 *at selected spatial points. The exact solution is the solid line and the dashed line shows the computed solution. Here $N = 16$ and the* CFL *number for the inhomogeneous cylinder is approximately* 0.25. *The coordinates of the spatial points are shown on each plot, and are chosen to be the interpolation point closest to the points A and B in* [9, Fig. 4]. *Dispersion is visible in all plots, particularly along the leading edge of the wave. Top row: Results for* ECHL. *Middle row: Results for* MECHL. *Bottom row: Results for* $\mathrm{EL}_N\mathrm{HC}$.

Figure 9 shows the results of all three methods when $N = 16$ at two points in the domain. These are the interpolation points closest to the points A and B in [9, Fig. 4]. All three methods give qualitatively similar results. Significant dispersion occurs in the coarsely meshed region above the cylinder shown in Fig. 8(b), but this tends not to effect the solution at the points given in Fig. 9. Clearly, although dispersion is evident, all three methods produce qualitatively similar and reasonable results. In particular, all three methods can handle discontinuous media with no problem.

We have used a significantly smaller timestep in this example compared to that used in [9], and we find that all three methods are unstable with the mesh and time step parameters used in [9]. Note that our implementation of ECHL differs from that in [9] in that we solve (49) and (50) by conjugate gradients with a stringent stopping tolerance. If the iterative method is only allowed to perform a small number of iterations per timestep, it is possible that this incomplete iteration will smooth the solution and allow longer timesteps.

As a final remark on this example, we note that even with preconditioning, the conjugate gradient scheme for $\mathrm{EL}_N\mathrm{HC}$ was very slowly convergent for this example (particularly when $N = 32$). In this example, $\mathrm{EL}_N\mathrm{HC}$ was much slower than ECHL or MECHL.

6. Conclusions. The methods under consideration in this paper have quite similar features. All are based on mixed finite-element spaces approximating the full Maxwell system. Moreover, in the examples computed, none of the methods has a decisive edge over the rest. The ECHL method has a larger stability boundary or CFL condition, but is sometimes less accurate than the other methods and suffers from greater grid anisotropy. Generally, the $\mathrm{EL}_N\mathrm{HC}$ method is most accurate, and MECHL is either more accurate or approximately the same accuracy as ECHL. The state of theory for ECHL is not as well developed as for MECHL or $\mathrm{EL}_N\mathrm{HC}$, and in all three cases an improved understanding of superconvergence is desirable.

Clearly, a second important area of research is to examine other timestepping methods. The stability condition for these schemes should be better understood, and less restrictive timestepping methods should be investigated (cf. [1]).

All three schemes have three-dimensional counterparts, and the results in this paper and in [9] suggest they will be successful schemes. Given the ease of implementation of methods with natural boundary conditions, ECHL or MECHL may be preferable to $\mathrm{EL}_N\mathrm{HC}$ when the perfectly conducting boundary condition is appropriate. For more complex conditions, $\mathrm{EL}_N\mathrm{HC}$ may be preferable.

Acknowledgments. I would like to thank Professor F. Santosa for several useful discussions regarding dispersion relations.

REFERENCES

[1] J. Adam, A. Serveniere, J. Nédélec, and P. Raviart, *Study of an implicit scheme for integrating Maxwell's equations*, Comput. Methods Appl. Mech. Engrg., 22 (1980), pp. 327–346.

[2] F. Brezzi, *On the existence and uniqueness of saddle-point problems arising from Lagrange multipliers*, RAIRO Anal. Numér., 8-R2 (1974), pp. 129–151.

[3] A. Cangellaris, C.-C. Lin, and K. Mei, *Point-matched time domain finite element methods for electromagnetic radiation and scattering*, IEEE Trans. Antennas and Propagation, AP-35 (1987), pp. 1160–1173.

[4] P. G. Ciarlet, *The Finite Element Method for Elliptic Problems*, Vol. 4, Studies In Mathematics and Its Applications, Elsevier, North–Holland, New York, 1978.

[5] G. Duvaut and J.-L. Lions, *Inequalities in Mechanics and Physics*, Springer–Verlag, New York, 1976.

[6] V. Girault and P. Raviart, *Finite Element Methods for Navier–Stokes Equations*, Springer-Verlag, New York, 1986.

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

[8] D. S. Jones, *The theory of electromagnetism*, MacMillan, New York, 1964.

[9] R. L. Lee and N. K. Madsen, *A mixed finite element formulation for Maxwell's equations in the time domain*, J. Comput. Phys., 88 (1990), pp. 284–304.

[10] R. Leis, *Initial Boundary Value Problems in Mathematical Physics*, John Wiley, New York, 1988.

[11] P. Monk, *An analysis of Nédélec's method for the spatial discretization of Maxwell's equations*, SIAM J. Numer. Anal., 28 (1991), pp. 1610–1634.

[12] P. Monk, *A mixed method for approximating Maxwell's equations*, J. Comput. Appl. Math., to appear.

[13] J. Nédélec, *Mixed finite elements in $\mathbb{R}^3$*, Numer. Math., 35 (1980), pp. 315–341.

[14] P. A. Raviart and J. M. Thomas, *A mixed finite element method for 2nd order elliptic problems*, in Mathematical Aspects of the Finite Element Method, A. Dold and B. Eckmann, eds., Lecture Notes in Math. 606, Springer-Verlag, Berlin, 1977.

[15] R. Richtmeyer and K. Morton, *Difference methods for initial value problems*, Wiley-Interscience, New York, 1976.

[16] A. Taflove, K. R. Umashankar, B. Beker, F. Harfoush, and K. S. Yee, *Detailed FD-TD analysis of electromagnetic fields penetrating narrow slots and lapped joints in thick conducting screens*, IEEE Trans. Antennas and Propagation, 36 (1988), pp. 247–257.

[17] L. N. Trefethen, *Group velocity in finite difference schemes*, SIAM Rev., 24 (1982), pp. 113–136.

[18] K. Yee, *Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media*, IEEE Trans. Antennas and Propagation, AP–16 (1966), pp. 302–307.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 5, pp. 1123–1141, September 1992

ADAPTIVE APPROXIMATION BY PIECEWISE LINEAR POLYNOMIALS ON TRIANGULATIONS OF SUBSETS OF SCATTERED DATA*

SHMUEL RIPPA†

Abstract. Given a set V of data points in R^2 with corresponding data values, the problem of adaptive piecewise polynomial approximation is to choose a subset of points of V, to create a triangulation of this subset, and to define a piecewise linear surface over the triangulation such that the deviation of this surface from the data set is no more than a prescribed error tolerance. A typical numerical scheme starts with some initial triangulation and adds more points (and triangles) as necessary until the resulting piecewise linear surface satisfies the error bound. In this paper two ingredients of such schemes are discussed. The first problem is that of constructing a suitable triangulation of a subset of points. The use of data-dependent triangulations that depend on the given function values at the data points is discussed, and some data-dependent criteria for optimizing a triangulation are presented and compared to the Delaunay criterion leading to the well-known Delaunay triangulation traditionally used for this purpose. The second problem addressed in this paper is how to select a piecewise linear surface approximating the given data. A common approach is to use an interpolating surface, i.e., to require that the surface interpolates the data at the nodes of the triangulation. In this paper the least-square approximation to the data from the space of piecewise linear polynomials defined over a triangulation of a subset of V is used. It is proved that the matrix of the normal equations is always nonsingular and a bound for its condition number is derived. This bound is relatively low, hence the least-square surface can be computed by solving the normal equations, e.g., by the conjugate gradient scheme with no preconditioning. Various numerical experiments demonstrate the improvement in the quality of the approximation when certain data-dependent triangulations are used. Improvement is also reported when least-square surfaces are compared to interpolating surfaces.

Key words. triangulation, data-dependent triangulation, piecewise linear interpolation, least-squares fitting

AMS(MOS) subject classifications. 65D05, 65D10

1. Introduction.

Scattered data interpolation in R^2 consists of constructing a function $F_I = F_I(x,y)$ such that

$$F_I(x_i, y_i) = f_i, \qquad i = 1, \cdots, N,$$

where $V = \{v_i = (x_i, y_i) \in R^2, i = 1, \cdots, N\}$ is a set of distinct and noncollinear data points, and $f = (f_1, \cdots, f_N)$ is a (real) data vector.

The data vector f is regarded, in most cases, as being sampled from a (usually unknown) surface $F = F(x,y)$, which we wish to approximate by the interpolating surface F_I. In many applications interpolation is not desired, e.g., when the data is subject to errors or when there is much more data than is required to get a good approximation to the surface. In many cases, we would like to replace the interpolating surface F_I by an approximating function F_A, which depends on far fewer parameters than the number of data points. Interpolation and approximation methods are reviewed by Barnhill [1], Franke [8], Powell [16], and Schumaker [19].

In this paper we discuss some methods of approximation using spaces of piecewise polynomials defined on triangulations of subsets of V. Let Ω be a polygonal domain

* Received by the editors January 8, 1990; accepted for publication (in revised form) July 9, 1991.

† School of Mathematical Sciences, Sackler Faculty of Exact Sciences, Tel Aviv University, Ramat Aviv 69978, Tel Aviv, Israel. Present address, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Silver Street, Cambridge CB3 9EW, England.

such that $V \subset \Omega$, and its boundary $\partial\Omega$ is a polygon with vertices in V. We denote by $V^{(M_0)}$ the set of all vertices of $\partial\Omega$, M_0 being the number of vertices.

DEFINITION 1.1. We call the set $V^{(M)} \subset V$ a proper subset of V if $V^{(M_0)} \subset V^{(M)}$.

When considering a proper subset $V^{(M)}$ of V that contains M points, we assume for simplicity that the data points are numbered such that

$$V^{(M)} = \{v_i = (x_i, y_i)\}_{i=1}^{M}.$$

DEFINITION 1.2. Let $V^{(M)}$ be a proper subset of V. A set $T = \{T_i\}_1^t$ of nondegenerate, open triangles is a $V^{(M)}$-triangulation of Ω if:

- $V^{(M)}$ is the set of all vertices of triangles in T;
- Every edge of a triangle in T contains only two points from $V^{(M)}$, namely, its endpoints;
- $\overline{\Omega} = \bigcup_{i=1}^{t} \overline{T_i}$;
- $T_i \bigcap T_j = \emptyset\,,\ i \neq j$.

When there is no risk of ambiguity, we use the term "triangulation" instead of "$V^{(M)}$-triangulation."

DEFINITION 1.3. The space $S_d^r(T)$ is the space of piecewise polynomials of degree d and smoothness r defined over T, i.e.,

$$S_d^r(T) = \{g \in C^r(\Omega) \mid \ g|_{T_i} \in \Pi_d\},$$

where Π_d is the space of all bivariate polynomials of total degree d. (See [20], [22], and references therein for a comprehensive discussion of these spaces.)

In the present paper, we restrict ourselves to the approximation from the space $S_1^0(T)$ of piecewise linear functions defined on a $V^{(M)}$-triangulation T of Ω. Many of the ideas are, however, applicable to more general spaces. As a basis to $S_1^0(T)$, we use the pyramidal functions

$$(\phi_1(x,y,T), \phi_2(x,y,T), \cdots, \phi_M(x,y,T)),$$

satisfying

$$\phi_i(x_j, y_j, T) = \delta_{ij}, \qquad 1 \leq j \leq M.$$

The support of ϕ_i is the cell Ω_i, which is the union of all (closed) triangles in T having v_i as a vertex.

The problem of adaptive approximation with piecewise linear polynomials is that of selecting a proper subset $V^{(M)}$ of V and a $V^{(M)}$-triangulation of Ω such that the deviation of the piecewise linear approximating surface, defined on that triangulation, from the data set will be within a prescribed error tolerance ϵ. A basic scheme for adaptive approximation was proposed by various authors (see, e.g., Floriani, Falcidieno, and Pienovi [5] and Lee and Schachter [11]). The algorithm begins by constructing an initial proper subset $V^{(M_0)}$ consisting of the M_0 points in $\partial\Omega$. In the ith iteration a subset $V^{(M)}$ of M vertices, chosen during the previous iteration, is given. The Delaunay $V^{(M)}$-triangulation is constructed and the deviation of the piecewise linear interpolator $F_{T,f}$ from the data points is computed. If the maximal deviation is greater than ϵ, then the point attaining the maximal deviation is added to the subset $V^{(M)}$ resulting in a new proper subset $V^{(M+1)}$ of V. The iterations continue until the maximal deviation is less than ϵ. The output of the scheme is a

$V^{(M)}$-triangulation T and an interpolating surface $F_{T,f}$, which deviates from the data set by no more than ϵ.

In this paper we discuss and test two modifications of the basic scheme; namely, the use of data-dependent triangulations instead of Delaunay triangulations, and the use of least-square surfaces instead of interpolating surfaces.

One important question is how to select a proper $V^{(M)}$-triangulation T of Ω. The standard approach is to use the constrained Delaunay triangulation [2], [5]. A different approach, recently suggested in [3], is that of *data-dependent triangulation.* The basic idea is to use triangulations that depend on the data vector f rather than triangulations that are constructed by geometrical criteria that consider only the locations of the data points. The numerical experiments presented in [3] demonstrate that interpolating surfaces defined over data-dependent triangulations provide a considerably better approximation to the underlying surface F than interpolating surfaces defined over the Delaunay triangulation. Further studies on this matter are presented in [14] and [18]. In §2, two data-dependent criteria and the Delaunay criterion for optimizing a $V^{(M)}$-triangulation are presented together with an algorithm for the construction of a locally optimal triangulation. We note that the LS criterion that is described in §2 was independently suggested and tested by Quak and Schumaker [15].

The interpolating surface $F_{T,f}$ is defined by

$$F_{T,f}(x,y) = \sum_{i=1}^{M} f_i \phi_i(x,y,T),$$

and it interpolates the data vector f at the nodes of $V^{(M)}$, i.e.,

$$F_{T,f}(x_j,y_j) = F(x_j,y_j) = f_j, \qquad j = 1,\cdots,M.$$

The interpolating surface is easily constructed but it is obviously not the best approximation to the data set, since the values of F on the set $V \backslash V^{(M)}$, are not used in the definition of $F_{T,f}$. A better alternative is to consider the least-square surface $U_{T,f}(x,y)$, which minimizes the quantity

$$E(g) = \sum_{i=1}^{N} (f_i - g(x_i,y_i))^2$$

among all functions $g \in S_1^0(T)$. In §3, we show that the above least-square problem has a unique solution and that the matrix of the normal equations has a relatively low condition number and a sparse structure, which can be exploited to develop efficient schemes for the solution of the normal equations.

In §4, the basic adaptive scheme is reformulated and several variants are presented and compared numerically to the basic scheme. The numerical comparisons, involving various test functions, demonstrate that significant improvement in the performance of the scheme is achieved when using data-dependent triangulations and/or least-square approximation. This means that for a prescribed error tolerance ϵ fewer points are needed to represent the surface or, alternatively, for the same number of degrees of freedom, many more accurate approximations of the underlying surface are obtained.

2. Data-dependent triangulation of a subset. Given a proper subset $V^{(M)}$ of V, we construct a $V^{(M)}$-triangulation of Ω. There are many ways to triangulate a set of data points, and we look for a triangulation that is optimal in some sense. The

Delaunay triangulation has the property that it maximizes, over all possible triangulations, the minimal angle in a triangulation [2], [5], [10], [11], [21], [24]. In [3] the concept of data-dependent triangulation was introduced and it was shown, by various numerical examples, that data-dependent triangulations are superior to the Delaunay triangulation for piecewise linear interpolation schemes defined on triangulations.

In this section we discuss data-dependent ($V^{(M)}$-)triangulation of Ω. Data-dependent triangulations are chosen to be optimal with respect to a given data-dependent criterion. A given criterion selects a preferred triangulation from among several alternatives, thus defining an ordering on the set of all triangulations, and we use the notation $T' < T$ to denote that the criterion prefers T' to T.

In the current investigation, we have considered two data-dependent criteria for optimizing a $V^{(M)}$-triangulation. The first criterion tries to minimize the sum of square errors between the approximating function and the data vector f. The second criterion is taken from [3] and is one of the more successful criteria discussed there.

To introduce these criteria we need some notations: Let $T^{(0)}$ be an initial $V^{(M)}$-triangulation, and let $W_{T^{(0)},f} \in S_1^0(T^{(0)})$ be an approximating function, e.g., $W_{T^{(0)},f}$ might be the interpolating surface $F_{T^{(0)},f}$ or the least-square surface $U_{T^{(0)},f}$ (see the Introduction). We define the coefficient vector as

$$w = (w_1, \cdots, w_M),$$

where

$$w_i = W_{T^{(0)},f}(x_i, y_i), \qquad i = 1, \cdots, M,$$

and on any other $V^{(M)}$-triangulation T, we consider the function $F_{T,w}$ interpolating the vector w, i.e.,

$$F_{T,w}(x_i, y_i) = w_i \ , \qquad i = 1, \cdots, M.$$

Of course, $F_{T^{(0)},w} = W_{T^{(0)},f}$.

The least-square (LS) criterion. By this criterion, triangulation T' is preferred to T if $E(F_{T',w}) < E(F_{T,w})$, where

$$E(F_{T,w}) = \sum_{i=1}^{N} (f_i - F_{T,w}(x_i, y_i))^2.$$

This criterion was also suggested by Quak and Schumaker [15].

The angle between normals (ABN) criterion. This is one of the more successful criteria described in [3]. To each interior edge e of T, the ABN cost function $s = s(F_{T,w}, e)$ is assigned, measuring the angle between the two normal vectors to the planes defined in the two adjacent triangles having e as a common edge. The cost of an entire triangulation T is then defined by

$$R(F_{T,w}) = \sum_{\text{all interior edges } e} |s(F_{T,w}, e)|,$$

and the ABN criterion prefers triangulation T' to T if $R(F_{T',w}) < R(F_{T,w})$.

The third criterion considered here is a purely geometrical criterion, which yields the Delaunay triangulation.

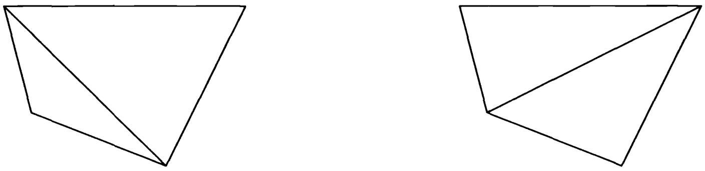


FIG. 1. *Two triangulations of a convex quadrilateral.*

The Delaunay criterion. For each triangle $T_i \in T$ a value σ_i, which is the minimum of the three interior angles of T_i, is assigned. The vector N_T is a vector of length t (the number of triangles in a triangulation T) containing the values σ_i. Furthermore, suppose that N_T is ordered in a nondecreasing manner. The Delaunay criterion (or MaxMin angle criterion) imposes the following ordering on triangulations: $T' < T$ means that $N_{T'}$ is lexicographically larger than N_T.

The Delaunay triangulation, which is globally optimal according to the Delaunay criterion, is a well-understood triangulation of a set of points and many efficient algorithms for its construction exist (see, e.g., [10], [11], and [20] and references therein). We do not know of any efficient schemes for the construction of triangulations that are globally optimal according to the LS or ABN and, in this case, we are content with locally optimal triangulations, a notion we now define. Let T be a triangulation of Ω, e an internal edge of T, and Q_e a quadrilateral formed from the two triangles having e as a common edge. If Q_e is strictly convex, then there are two possible ways of triangulating it (see Fig. 1).

DEFINITION 2.1. An edge e is called locally optimal if one of the following conditions holds:

1. The quadrilateral Q_e is not strictly convex;
2. The quadrilateral Q_e is strictly convex and $T \leq T'$ where T' is obtained from T by replacing e by the other diagonal of Q_e.

DEFINITION 2.2. A locally optimal triangulation of Ω is a triangulation T' in which all edges are locally optimal.

The term "data-dependent triangulation" frequently replaces the term "locally optimal triangulation of Ω" in cases where the triangulation criterion is data dependent.

Locally optimal triangulations are constructed by the local optimization procedure (LOP) of Lawson [10].

```
ALGORITHM 2.1 (LOP).
  1. Construct an initial $V^{(M)}$-triangulation $T$ of $\Omega$.
  2. As long as $T$ is not locally optimal:
         choose an interior edge $e$ that is not locally optimal and swap the
         edge: Replace it by the other diagonal of $Q_e$.
```

Each time an edge swap occurs, the resulting triangulation is strictly better with respect to the triangulation criterion. Since the number of triangulations of Ω is finite, the LOP converges, after a finite number of edge swaps, to a locally optimal triangulation. The locally optimal triangulation obtained by the LOP may depend on the specific order in which edges are swapped. Different ordering strategies are presented and compared in [4]. We note that in the case of the Delaunay criterion, the LOP always converges to a globally optimal triangulation independently of the order of edge swaps (see, e.g., [10]).

3. The least-square problem. In this section we discuss the least-square problem in the space $S_1^0(T)$ of piecewise linear functions defined over a $V^{(M)}$-triangulation T of Ω. The problem is to find a function

$$U_{T,f}(x,y) = \sum_{i=1}^{M} u_i \phi_i(x,y,T) \in S_1^0(T)$$

such that the sum of squares of the deviation from all data values

$$\sum_{j=1}^{N} (f_j - U_{T,f}(x_j,y_j))^2$$

is minimized.

In matrix notations we look for a vector $u = (u_1, u_2, \cdots, u_M)$ such that

$$||f - Au||_2 = \min_{v \in R^M} ||f - Av||_2,$$

where A is the $N \times M$ rectangular matrix

$$A = \{a_{ij}\} = \{\phi_j(x_i,y_i)\}, \qquad i = 1, \cdots, N, \quad j = 1, \cdots, M.$$

An equivalent formulation is to find a solution vector u to the normal equations

$$Bu = b, \tag{1}$$

where $B = A^T A$ and $b = A^T f$. System (1) is known to have at least one solution. We show that, in fact, the matrix B is of full rank.

THEOREM 3.1. *The system of equations* (1) *is nonsingular for any configuration of data points V and any selection of a proper subset $V^{(M)}$ of V.*

Proof. We prove the theorem by using a lower bound for the minimal singular value $\sigma_{\min}(A)$ of A (i.e., the minimal eigenvalue of $(A^T A)^{1/2}$). A convenient lower bound is given by Johnson [9]:

$$\sigma_{\min}(A) \geq \min_{1 \leq j \leq M} \left\{ |a_{jj}| - \frac{1}{2} \left(\sum_{\substack{i=1 \\ i \neq j}}^{M} (|a_{ij}| + |a_{ji}|) \right) \right\}. \tag{2}$$

Since the points of V were numbered such that

$$V^{(M)} = \{v_i \, , \, i = 1, \cdots, M\},$$

we have, by the choice of the basis functions $\phi_1, \cdots, \phi_M$, that

$$a_{ij} = \delta_{ij}, \qquad 1 \leq i,j \leq M,$$

and thus from (2) we conclude that

$$\sigma_{\min}(A) \geq 1, \tag{3}$$

and the theorem is proved. □

The solution of the normal equations is usually considered a bad alternative to solving the least-square problem, as these equations very often tend to become ill conditioned. It is interesting, therefore, to derive an upper bound for the condition number of the matrix B of the normal equations.

The spectral condition number of B is given by

$$\mathrm{cond}(B) = ||B||_2 \cdot ||B^{-1}||_2, \quad \text{where } ||B||_2 = \sigma_{\max}(B)$$

and $\sigma_{\max}(B)$ is the largest singular value of B.

THEOREM 3.2. *Let Ω_i be the union of all (closed) triangles having a vertex at the point v_i , $i = 1, \cdots, M$, and let N_i be the number of data points from V in the interior of Ω_i; then*

$$\mathrm{cond}(B) \leq N_{\max} = \max_{1 \leq i \leq M} N_i.$$

Proof. From the definition of B and (3) we conclude that

$$||B^{-1}||_2 = \frac{1}{\sigma_{\min}(B)} = \left(\frac{1}{\sigma_{\min}(A)}\right)^{1/2} \leq 1,$$

and thus

$$\mathrm{cond}(B) \leq ||B||_2.$$

The estimation of $||B||_2$ is done by using the well-known bound

$$||B||_2 \leq (||B||_1 \cdot ||B||_\infty)^{1/2},$$

where

$$||B||_\infty = \max_{1 \leq i \leq M} \sum_{j=1}^{M} |B_{ij}|$$

and

$$||B||_1 = \max_{1 \leq j \leq M} \sum_{i=1}^{M} |B_{ij}|.$$

Since B is symmetric, $||B||_1 = ||B||_\infty$ and $||B||_2 \leq ||B||_\infty$.

In the following, we prove that $||B||_\infty \leq N_{\max}$. We consider the ith row of B, i.e.,

$$B_{iq} = \sum_{s=1}^{N} \phi_i(x_s, y_s, T)\phi_q(x_s, y_s, T), \qquad q = 1, \cdots, M.$$

Since a data point $v_s \in V$ contributes to the ith row only if it is inside the cell Ω_i, we have that

$$B_{iq} = \sum_{(x_s, y_s) \in \Omega_i} \phi_i(x_s, y_s, T)\phi_q(x_s, y_s, T), \qquad q = 1, \cdots, M.$$

Let $v_s \in \Omega_i$, $s \neq i$ be some point from V. We denote by $v_i, v_{j(s)}$ and $v_{k(s)}$ the vertices of the triangle from T containing v_s (if v_s lies on an edge of two triangles in Ω_i, we assign it arbitrarily to one of the triangles). The point v_s contributes only to the elements B_{ii}, $B_{ij(s)}$, and $B_{ik(s)}$ of B. We note also that $\phi_i(x_s, y_s, T)$, $\phi_{j(s)}(x_s, y_s, T)$, and $\phi_{k(s)}(x_s, y_s, T)$ are just the barycentric coordinates of v_s with respect to the triangle $\triangle v_i v_{j(s)} v_{k(s)}$; hence

$$\phi_i(x_s, y_s, T) + \phi_{j(s)}(x_s, y_s, T) + \phi_{k(s)}(x_s, y_s, T) = 1,$$

and the contribution of v_s to the sum

$$\sum_{q=1}^{M} |B_{iq}| = \sum_{q=1}^{M} B_{iq}$$

is

$$\begin{aligned}\phi_i^2(x_s, y_s, T) &+ \phi_i(x_s, y_s, T)\phi_{j(s)}(x_s, y_s, T) \\ &+ \phi_i(x_s, y_s, T)\phi_{k(s)}(x_s, y_s, T) = \phi_i(x_s, y_s, T).\end{aligned}$$

Summing over all contributions of points inside the cell Ω_i, we get that

$$\sum_{q=1}^{M} B_{iq} = \sum_{(x_s, y_s) \in \Omega_i} \phi_i(x_s, y_s, T) \leq N_i,$$

and from the definition of $||B||_\infty$ we obtain the required upper bound:

$$||B||_\infty \leq N\text{max} = \max_{1 \leq i \leq M} N_i.$$

Using the above upper bound, we conclude that

$$\text{cond}(B) \leq ||B||_2 \leq ||B||_\infty \leq N\text{max},$$

and the proof of the theorem is completed. □

The fact that, in the worst possible case, the condition number of B depends linearly on the number of data points N, and is in fact much smaller in most cases, suggests that the solution of the normal equation is a practical way of solving the least-square problem. Of special interest are methods that take account of the sparsity and the special structure of B.

The storage requirements for the matrix B are very modest, as it is a sparse matrix that has a nice structure: any elements of $B_{ij} \neq 0$, $i \neq j$ correspond to an edge of the triangulation connecting the vertices v_i and v_j. Thus the nonzero B_{ij}'s can be stored in an array of M_e places where M_e denotes the number of edges in the $V^{(M)}$-triangulation T. The diagonal elements B_{ii} can be stored in an additional array of M places. Since $M_e \leq 3M$, the total memory requirement for the storage of the matrix B is $4M$ places at most. The structure of B can be used to derive efficient numerical schemes for the solution of the normal equations. In our tests we solved those equations with the conjugate gradient (CG) scheme [23, p. 572]. For its computation, the CG algorithm uses only operations of multiplications of vectors in R^M by the matrix B and scalar products of vectors in R^M. The sparsity of B is exploited by writing an efficient code for multiplication of a vector in R^M by B.

4. Numerical experiments with the COMPRESS scheme.

4.1. The COMPRESS scheme. The modified adaptive approximation scheme COMPRESS, which is a slight generalization of the basic scheme presented in [5] and [11], is formulated in the following algorithm.

ALGORITHM 4.1 (COMPRESS).

```
1. Let $\Omega$ be a region with a polygonal boundary $\partial\Omega$ consisting of $M_0$ vertices from $V$.
   Set $M \leftarrow M_0$ and let $V^{(M)} \subset V$ be the set of the $M_0$ vertices of $\partial\Omega$.
   Construct an initial $V^{(M)}$-triangulation $T$ of $\Omega$.
2. Construct a locally optimal $V^{(M)}$-triangulation $T'$ by the LOP scheme (Algorithm 2.1) with respect to a chosen criterion starting from the given triangulation $T$.
   Set $T \leftarrow T'$.
3. Construct $W_{T,f}$, an approximating surface to the data vector $f$ from $S_1^0(T)$.
   Compute the errors $E_i(W_{T,f}) = |f_i - W_{T,f}(x_i, y_i)|$, $i = 1, \cdots, N$ and let
       $E = \max_{1 \le i \le N} E_i(W_{T,f}).$
4. If $E \le \epsilon$ end the procedure; else go to step 5.
5. Select a point $v_k = (x_k, y_k) \in V \backslash V^{(M)}$ for which $E_k(W_{T,f})$ is maximal and add it to $V^{(M)}$:
       $V^{(M+1)} \leftarrow V^{(M)} \bigcup \{v_k\}.$
6. Update the triangulation $T$ to include the point $v_k$ in it, and attain a $V^{(M+1)}$-triangulation.
   Set $M \leftarrow M + 1$ and go to step 2.
```

The above algorithm in fact defines a family of schemes depending on the selection of the approximating function $W_{T,f}$ and on the triangulation criterion used for optimizing a triangulation during the LOP step. In the basic scheme [5], [11], the approximating surface was taken to be the interpolating surface $F_{T,f}$, and the LOP step was performed with the Delaunay criterion. In the following we describe the testing procedure and review some variants of the scheme that we have tested.

4.2. The testing procedure. In the numerical experiments a set V of data points, uniformly placed over the unit square, was constructed, i.e.,

$$V = \left\{ \left(\frac{i}{N}, \frac{j}{N} \right), 0 \le i, j \le N, N = 30 \right\}.$$

The data vectors $f = (f_1, \cdots, f_N)$ were obtained by evaluating five test functions $FI = FI(x, y)$ at the data points. Our experiments were performed with several data sets, including scattered data sets. These scattered test sets were generated from regular uniform grids by randomly moving the points of the regular grid by a distance not exceeding $h/4$, where h is the length of the grid cell. The results of those experiments, obtained using the data set V, were similar to the results presented in this section.

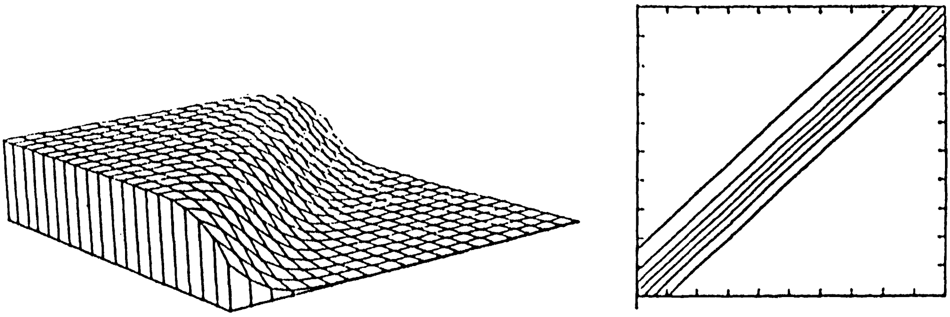


FIG. 2. *Perspective view and level curves of $F2$.*

All of the test functions are defined on the unit square. The first two test functions were taken from Franke [7]:

$$\begin{aligned} F1 = {} & .75\exp\left(-\frac{(9x-2)^2+(9y-2)^2}{4}\right) \\ & +.75\exp\left(-\frac{(9x+1)^2}{49}-\frac{9y+1}{10}\right) \\ & +.5\exp\left(-\frac{(9x-7)^2+(9y-3)^2}{4}\right) \\ & -.2\exp\left(-(9x-4)^2-(9y-7)^2\right), \\ F2 = {} & \frac{\tanh(9y-9x)+1}{9}. \end{aligned}$$

The function $F1$ is composed of two Gaussian peaks and a sharp Gaussian dip. The function $F2$ simulates a sharp rise running diagonally across the unit square (Fig. 2).

The third test function is taken from Ritchie [17]:

$$F3 = \begin{cases} 1 & \text{if } y-\xi \geq \frac{1}{2}, \\ 2(y-\xi) & \text{if } 0 \leq y-\xi \leq \frac{1}{2}, \\ (\cos(4\pi r(\xi,y))+1)/2 & \text{if } r(\xi,y) \leq \frac{1}{4}, \\ 0 & \text{otherwise,} \end{cases}$$

where

$$\begin{aligned} r(\xi,y) &= \sqrt{(\xi-\tfrac{3}{2})^2+(y-\tfrac{1}{2})^2}, \\ \xi &= 2.1x-0.1. \end{aligned}$$

This function represents a "hill" on a plane and a ramp leading to another plane (Fig. 3). It is a function with discontinuous first derivatives.

The last two test functions were taken from Lyche and Morken [12]:

$$F4 = \tanh(-3g(x,y))+1, \quad g(x,y) = 0.595576(y+3.79762)^2 - x - 10,$$

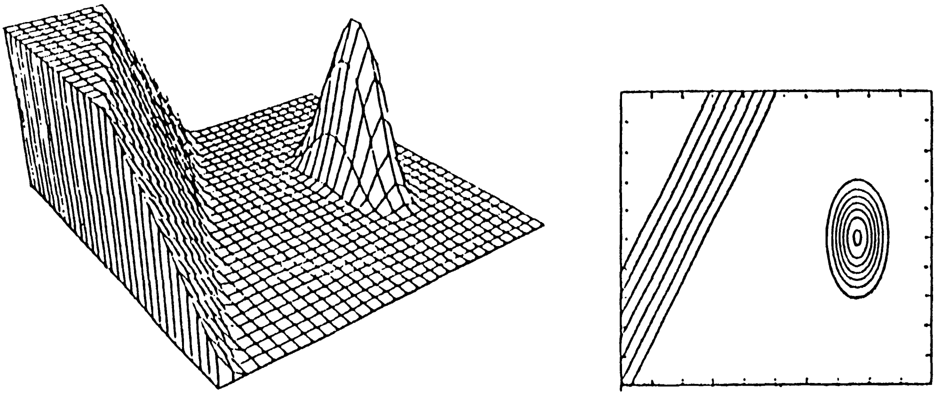


FIG. 3. *Perspective view and level curves of* $F3$.

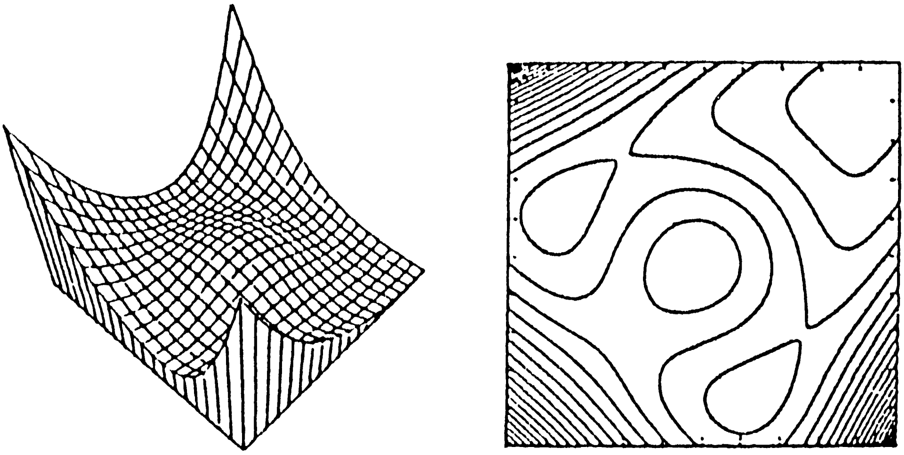


FIG. 4. *Perspective view and level curves of* $F5$.

$$F5 = \left(1-\frac{x}{2}\right)^6\left(1-\frac{y}{2}\right)^6 + 1000(1-x)^3x^3(1-y)^3y^3$$
$$+y^6\left(1-\frac{x}{2}\right)^6 + x^6\left(1-\frac{y}{2}\right)^6.$$

Function $F4$ resembles $F2$. Its contour lines are the parabolas $3g(x,y) = \text{const}$, while those of $F2$ are the straight lines $y - x = \text{const}$. Function $F5$ is a polynomial surface of degree 12 (Fig. 4).

It is convenient, during the numerical experiments, to use the number of points in the subset as a stopping criterion for the COMPRESS scheme, i.e., the iterations stop when the subset contains a predetermined number of points (and not when the deviation of the approximating surface at the data points is less than ϵ). For each of the five data sets we applied each variant of the COMPRESS algorithm twice for generating the final subsets $V^{(M)}$, with $M = 25$ and $M = 50$.

On each of the resulting subsets, with its corresponding triangulation T, the

TABLE 1
Mean errors using a subset of 25 *points.*

Trng. criterion	Approximation	$F1$	$F2$	$F3$	$F4$	$F5$
Delaunay	Interpolation	0.02237	0.00544	0.03662	0.04311	0.02685
Delaunay	Least squares	0.01890	0.00357	0.02737	0.01878	0.01331
LS	Interpolation	0.03016	0.00106	0.05584	0.00939	0.02114
LS	Least squares	0.02509	0.00055	0.05022	0.00491	0.01321
ABN	Interpolation	0.04829	0.00139	0.05013	0.01072	0.02036
ABN	Least squares	0.03750	0.00073	0.06127	0.00558	0.01509
Hybrid	Interpolation	0.02237	0.00107	0.03662	0.01326	0.02006
Hybrid	Least squares	0.01890	0.00059	0.02737	0.00654	0.01203

TABLE 2
Max errors using a subset of 25 *points.*

Trng. criterion	Approximation	$F1$	$F2$	$F3$	$F4$	$F5$
Delaunay	Interpolation	0.12966	0.03870	0.28225	0.30443	0.13794
Delaunay	Least squares	0.10373	0.02920	0.26520	0.24924	0.08270
LS	Interpolation	0.19905	0.00757	0.71336	0.09526	0.11527
LS	Least squares	0.21662	0.00630	0.71867	0.06533	0.10124
ABN	Interpolation	0.23160	0.00938	0.64746	0.20181	0.08931
ABN	Least squares	0.20189	0.00668	0.54788	0.16919	0.07955
Hybrid	Interpolation	0.12966	0.00757	0.28225	0.20181	0.08359
Hybrid	Least squares	0.10373	0.00659	0.26520	0.17597	0.06952

TABLE 3
Mean errors using a subset of 50 *points.*

Trng. criterion	Approximation	$F1$	$F2$	$F3$	$F4$	$F5$
Delaunay	Interpolation	0.01343	0.00238	0.01889	0.01567	0.01401
Delaunay	Least squares	0.01007	0.00114	0.01560	0.00728	0.00675
LS	Interpolation	0.01784	0.00028	0.04202	0.00225	0.00826
LS	Least squares	0.01390	0.00021	0.04106	0.00114	0.00547
ABN	Interpolation	0.01798	0.00035	0.02007	0.00239	0.00837
ABN	Least squares	0.01353	0.00023	0.01737	0.00109	0.00589
Hybrid	Interpolation	0.01343	0.00044	0.01889	0.00273	0.00727
Hybrid	Least squares	0.01007	0.00026	0.01560	0.00121	0.00500

piecewise linear approximating surface $W_{T,f}$ is constructed, and its deviation from the test function FI (generating f) is computed on a grid of 33 × 33 nodes placed uniformly over the unit square. The mean and the maximum values of these errors are tabulated and serve as a primary criterion for evaluating the various variants of the scheme. We note that this grid size was taken from the experiments in [7], and we found it satisfactory compared to a finer set of points.

4.3. Review of the numerical results. There are basically two alternatives for the use of least-square surfaces within the COMPRESS scheme. First, it is possible to perform all steps of the scheme using the interpolating surface $F_{T,f}$ and to compute the least-square surface $U_{T,f}$ only once upon leaving the routine. This strategy has the advantage of being economical in computer time and was found in our testing to be the most reasonable strategy. Only variants of the COMPRESS scheme using this strategy are presented in Tables 1–4. It is very clear from the tables that the final least-square surfaces usually provide better approximation to the selected test functions than the interpolating surfaces. The reduction of the errors is sometimes very significant, e.g., function $F4$ in Tables 1 and 3. The solution to the normal equations in our tests is computed by using conjugate gradient (CG) iterations with

TABLE 4
Max errors using a subset of 50 *points.*

Trng. criterion	Approximation	$F1$	$F2$	$F3$	$F4$	$F5$
Delaunay	Interpolation	0.05936	0.01729	0.09281	0.14981	0.05700
Delaunay	Least squares	0.04997	0.01303	0.08887	0.10151	0.03231
LS	Interpolation	0.19905	0.00138	0.78815	0.03800	0.08893
LS	Least squares	0.21286	0.00116	0.77550	0.03892	0.09088
ABN	Interpolation	0.10004	0.00268	0.12464	0.02407	0.03299
ABN	Least squares	0.11819	0.00262	0.12563	0.02609	0.02607
Hybrid	Interpolation	0.05936	0.00268	0.09281	0.02407	0.03299
Hybrid	Least squares	0.04997	0.00267	0.08887	0.02435	0.02566

the values of the data vector $f|_{V(M)}$ taken as an initial guess. Since the condition number of the normal equations is relatively low, no preconditioning is required for the CG scheme, and since the matrix of the normal equations is sparse and structured, the CG code is very efficient. The condition number of the normal equations, for the model problems displayed in the tables, was between 3 and 65, and the CG scheme for the solution of these equations took between 3 and 13 iterations to converge.

A second alternative is to use the least-square surface as the approximating surface $W_{T,f}$ in various stages of the COMPRESS algorithm. The use of the least-square surface during the triangulation optimization (LOP) step resulted in slightly better locally optimal triangulations when the triangulation criterion was data dependent. The least-square surfaces may also be used to compute the deviation from the data set. Since the least-square error $E(U_{T,f})$ is usually smaller than the interpolation error $E(F_{T,f})$, a faster termination of the COMPRESS scheme results when using the former error. These advantages are not, however, enough to justify the extra computation needed for the generation of the least-square surfaces during intermediate stages of the COMPRESS scheme. We note that our numerical experiments indicate that it is *not advisable* to select the new point to be inserted (step 5 of the COMPRESS algorithm) using the errors $E_i(U_{T,f})$. Therefore, the selection step should always be carried out using the interpolation errors $E_i(F_{T,f})$, even if other steps of the COMPRESS scheme make use of the least-square surface. To conclude, we do not recommend the use of the least-square surface in intermediate steps of the COMPRESS algorithm when the data is exact.

If the data is subject to errors, it is recommended that all steps of the COMPRESS algorithm using the least-square surface instead of the interpolating surface be carried out. We note that the normal equations change only slightly when an edge swap occurs during triangulation optimization (LOP). The updating of the matrix of the normal equations is therefore efficient, and the least-square solution for the new triangulation after an edge swap can be computed by using the CG scheme taking the previous solution as an initial guess. We do not present experiments with noisy data in this paper.

We have tested the COMPRESS scheme with the Delaunay, LS, and ABN criteria (presented in §2) for optimizing a triangulation. The Delaunay criterion, used as the only criterion for the basic scheme in [5] and [11], usually results in locally optimal triangulations that are worse, i.e., the corresponding mean/max errors are larger, than triangulations corresponding to the LS or ABN criteria. However, at least one case was found, namely, test function $F1$, where the Delaunay criterion led to a better triangulation than the LS and ABN criteria.

The LS criterion often leads to a very good locally optimal triangulation, i.e., the interpolating/least-square surface defined over the corresponding locally optimal triangulation provides a good approximation to the test functions. However, this criterion has obvious limitations: Suppose that T and T' are two triangulations that differ only in the way a convex quadrilateral Q is triangulated. If no data point from V lies inside Q, then no triangulation is preferred by the criterion. Thus it is likely that the criterion performs well only when there are enough data points inside each triangle of the triangulation. In the numerical experiments, the LS criterion sometimes seems to produce locally optimal triangulations containing several "badly" shaped triangles, i.e., they are long and thin and have their long side in the direction of high curvature of the underlying test function. The number of these triangles is small, but they effect the quality of approximation, especially when the errors are measured in the max norm; see, e.g., functions $F1$ and $F5$ in Table 4. In Fig. 14, the triangulation of a subset of 50 points generated by the COMPRESS scheme with the LS criterion for data sampled from test function $F5$ is displayed. Figure 14 also displays the level curves of the piecewise linear interpolating surface defined on this triangulation. The effect of the long thin triangles on the quality of the approximation is clearly understood: The resulting locally optimal triangulation contains some long thin triangles resulting in a very poor approximation to the test function in the neighborhood of these triangles. The badly shaped triangles, once created, could not be optimized in subsequent iterations since they did not contain any data point. This behavior of the LS criterion is the reason we do not recommend, in general, the use of the LS criterion for optimizing a triangulation in the LOP step of the COMPRESS algorithm.

The ABN criterion performs nicely in most cases. Exceptions are some cases when the number of points in the subset is too small to properly reflect the characteristics of the data set; see, e.g., some cases in Tables 1 and 2. This phenomenon usually happens in the first COMPRESS iterations, and is corrected as more points are added to the subset.

An idea to improve the performance of the scheme is to use a hybrid scheme that switches, in various stages of the COMPRESS iterations, between the Delaunay triangulation TD and the ABN triangulations TA, selecting the one for which the corresponding interpolating (or least-square) surface has the least deviation from the data set (e.g., the ABN triangulation is selected if $E(F_{TA,f})$ is smaller than $E(F_{TD,f})$). The hybrid criterion was found to perform rather well overall and is thus recommended for use in practical applications.

The results of the numerical experiments with the COMPRESS algorithm using the Delaunay, LS, ABN, and hybrid triangulation criteria are displayed in Tables 1–4. The tables show, for each of these variants, the deviation of the interpolating surface $F_{T,f}$ and the least-square surface $U_{T,f}$ (for the same final $V^{(M)}$-triangulation T) from the test function. The variants of the COMPRESS algorithm displayed in the tables use the interpolating surface $F_{T,f}$ as the approximating surface $W_{T,f}$ in all intermediate steps of the algorithm.

Figures 5–13 display the triangulations of subsets containing 50 points that were generated by the COMPRESS scheme. The COMPRESS scheme was applied with two triangulation criteria: (a) the Delaunay criterion, resulting in a Delaunay triangulation TD of the subset; and (b) the hybrid criterion, resulting in the hybrid triangulation TH. The figures clearly demonstrate the success of the COMPRESS scheme in adapting to the behavior of the underlying test function, i.e., more points are spread by the scheme in areas of high curvature of the test function. It is clear that the hybrid triangulation results in a better approximation to the test functions than the Delaunay triangulation. We note that the hybrid triangulations produce sometimes long and thin triangles, which are traditionally considered bad for approximation. Such triangles, which also appear in the experiments in [3], are very desirable if their long side is positioned in the direction of low curvature of the underlying function [13], [18]. The resulting triangulations provide better approximation to the test functions than the nearly equiangular triangulations provided by the Delaunay triangulation.

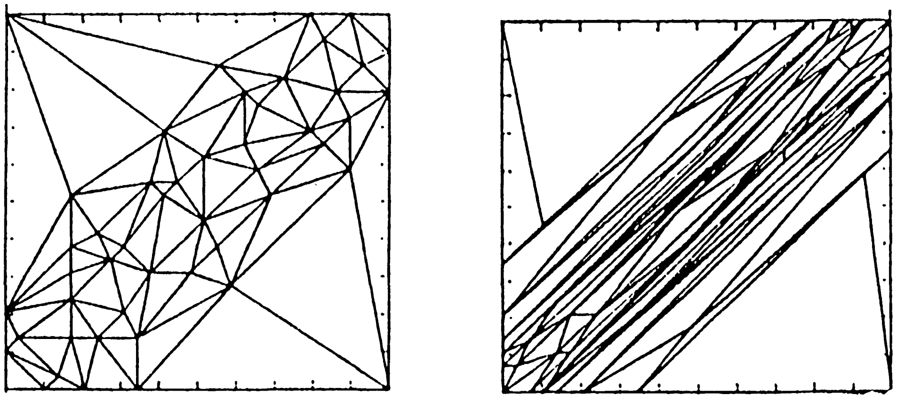


FIG. 5. *Subset triangulation: Delaunay TD (left) and hybrid TH (right), data vector f sampled from test function F2.*

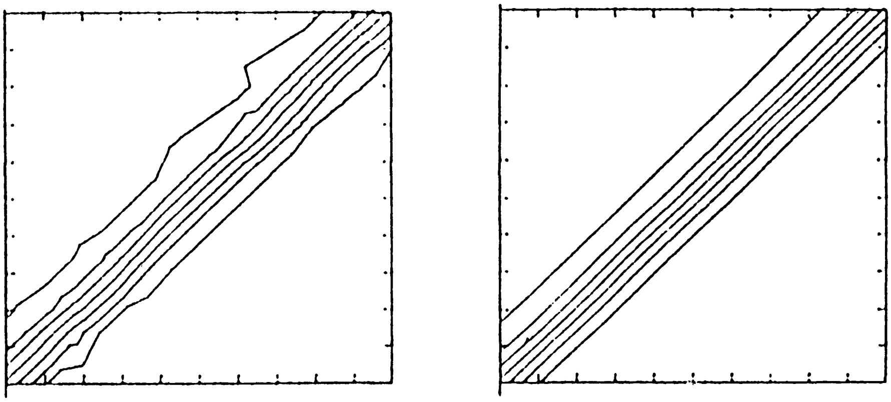


FIG. 6. *Level curves of* $F_{TD,f}$ *(left) and* $F_{TH,f}$ *(right).*

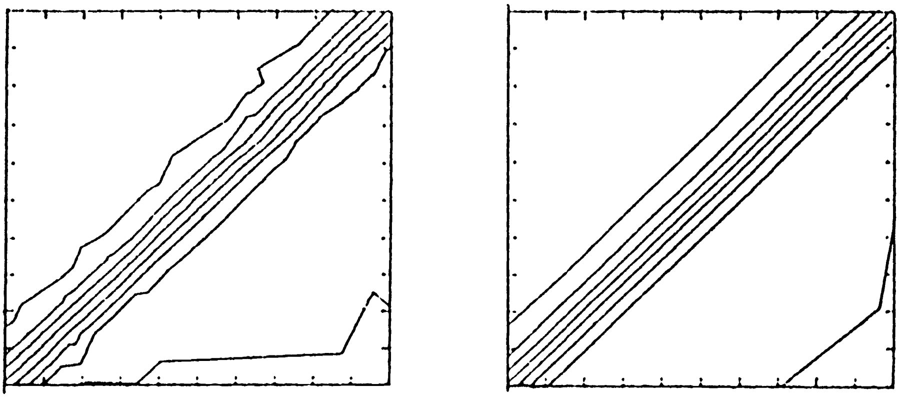


FIG. 7. *Level curves of* $U_{TD,f}$ *(left) and* $U_{TH,f}$ *(right).*

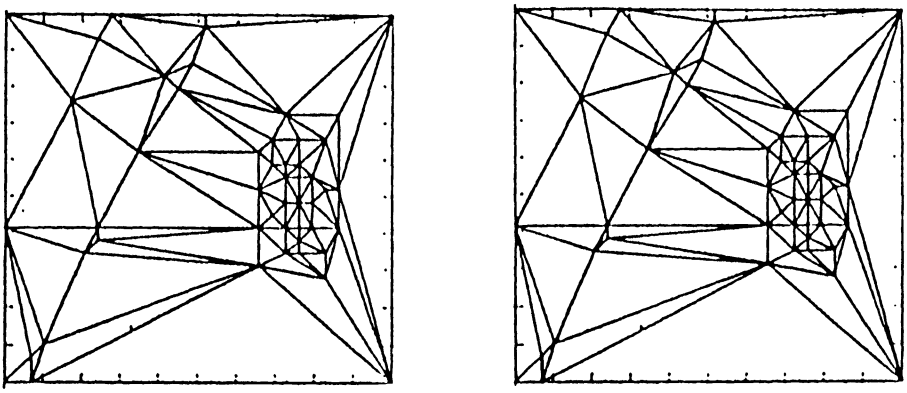


FIG. 8. *Subset triangulation: Delaunay TD (left) and hybrid TH (right), data vector f sampled from test function F3.*

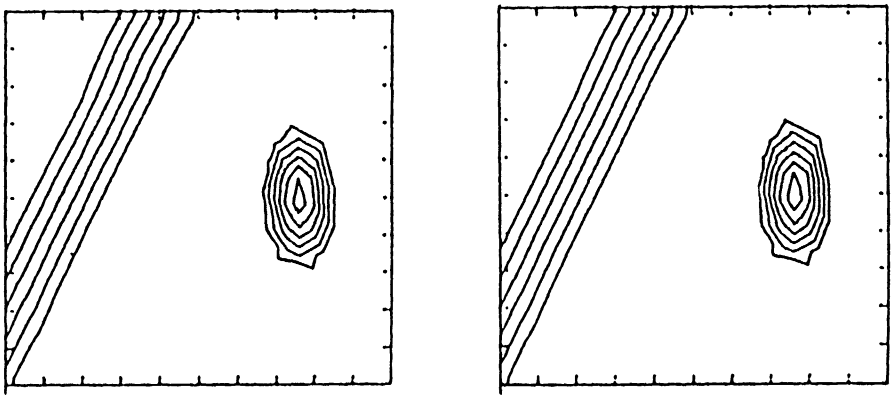


FIG. 9. *Level curves of* $F_{TD,f}$ *(left) and* $F_{TH,f}$ *(right).*

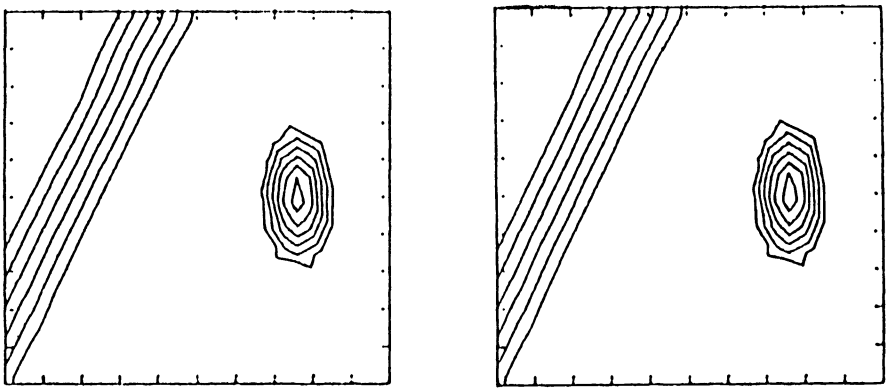


FIG. 10. *Level curves of* $U_{TD,f}$ *(left) and* $U_{TH,f}$ *(right).*

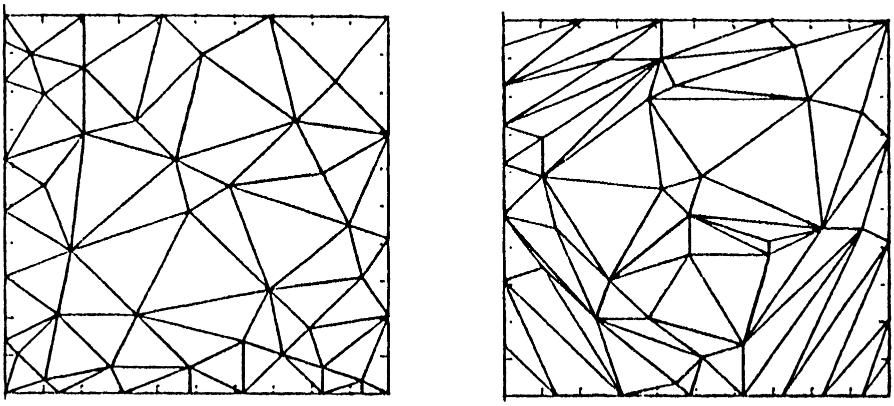


FIG. 11. *Subset triangulation: Delaunay* TD *(left) and hybrid* TH *(right), data vector* f *sampled from test function* $F5$.

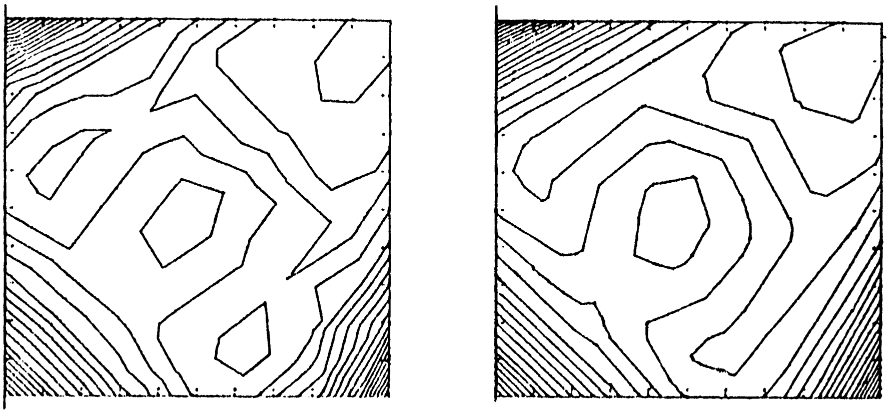


FIG. 12. *Level curves of* $F_{TD,f}$ *(left) and* $F_{TH,f}$ *(right).*

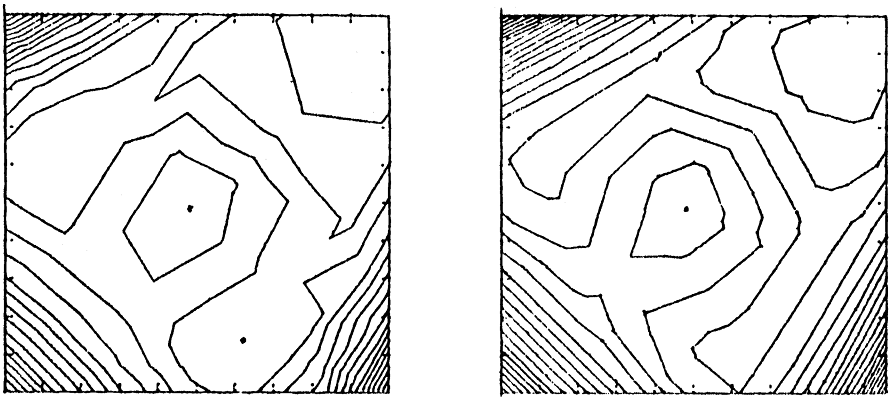


FIG. 13. *Level curves of* $U_{TD,f}$ *(left) and* $U_{TH,f}$ *(right).*

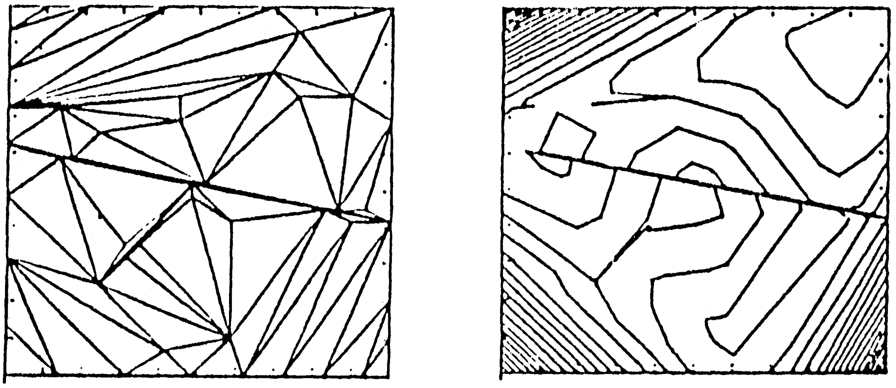


FIG. 14. *Subset triangulation* TL *and level curves of* $F_{TL,f}$*, where* f *is sampled from test function* $F5$.

5. Conclusions. In this paper, we have considered a scheme for generating a piecewise linear surface defined on a triangulation of a chosen subset of the set V of data points. It is demonstrated by a variety of numerical examples that the basic scheme, suggested in [5] and [11], can be improved significantly when the triangulation is optimized using a data-dependent criterion, and when least-square surfaces are used for approximation of the data values on the final triangulation. In practical terms, the improved scheme uses less data points from the set V to construct an approximating surface satisfying a prescribed error tolerance at all the data points. It is also proved that the condition number of the normal equations for the solution of the least-square problem is relatively low, so the solution of the normal equations can be considered a practical way to obtain the least-square surface. The ideas presented in this paper were formulated for piecewise linear surfaces but most of them can be extended to piecewise polynomials of higher degree. The use of piecewise polynomial spaces for the approximating surfaces in this scheme is currently under investigation.

Acknowledgments. I thank my advisors, Nira Dyn and David Levin of the School of Mathematical Sciences, Tel Aviv University, for their useful discussions and suggestions.

REFERENCES

[1] R. E. BARNHILL, *Representation and approximation of surfaces*, in Mathematical Software III, J. R. Rice, ed., Academic Press, New York, 1977, pp. 68–119.

[2] L. P. CHEW, *Constrained Delaunay triangulation*, Algorithmica, 4 (1989), pp. 97–108.

[3] N. DYN, D. LEVIN, AND S. RIPPA, *Data dependent triangulations for piecewise linear interpolation*, IMA J. Numer. Anal., 10 (1990), pp. 137–154.

[4] ———, *Algorithms for the Construction of Data Dependent Triangulations*, in Algorithms for Approximation II, J. C. Mason and M. G. Cox, eds., Chapman and Hall, London, 1990, pp. 185–192.

[5] L. DE. FLORIANI, B. FALCIDIENO, AND C. PIENOVI, *Delaunay-based representation of surfaces defined over arbitrarily shaped domains*, Comput. Vision, Graphics and Image Process., 32 (1985), pp. 127–140.

[6] R. FRANKE, *Scattered data interpolation: Tests of some methods*, Math. Comp., 38 (1982), pp. 181–200.

[7] ———, *A critical comparison of some methods for interpolation of scattered data*, Rep. NPS-53-79-003, Naval Postgraduate School, Monterey, CA, 1979.

[8] ———, *Recent advances in the approximation of surfaces from scattered data*, in Topics in Multivariate Approximation, C. K. Chui, L. L. Schumaker, and F. I. Utreras, eds., Academic Press, New York, 1987, pp. 79–98.

[9] C. R. JOHNSON, *A Gersgorin-type lower bound for the smallest singular value,* Linear Algebra Appl., 112 (1989), pp. 1–7.

[10] C. L. LAWSON, *Software for C^1 interpolation*, in Mathematical Software III, J. R. Rice, ed., Academic Press, New York, 1977, pp. 161–194.

[11] D. T. LEE AND B. J. SCHACHTER, *Two algorithms for constructing a Delaunay triangulation*, Internat. J. Comp. Inf. Sci., 9 (1980), pp. 219–242.

[12] T. LYCHE AND K. MORKEN, *Knot removal for parametric B-spline curves and surfaces*, Comput. Aided Geom. Design, 4 (1987), pp. 217–230.

[13] E. J. NADLER, *Piecewise linear approximation on triangulations of a planar region*, Ph.D. thesis, Division of Applied Mathematics, Brown University, Providence, RI, May 1985.

[14] E. QUAK AND L. L. SCHUMAKER, *Cubic spline fitting using data dependent triangulations*, Comput. Aided Geom. Design, 7 (1990), pp. 293–301.

[15] ———, *Least squares fitting by linear splines on data dependent triangulations*, in Curves and Surfaces, P. J. Laurent, A. Le Méhauté and L. L. Schumaker, eds., Academic Press, Boston, 1991, pp. 387-390.

[16] M. J. D. POWELL, *Radial basis functions for multivariable interpolation: A review*, in Algorithms for Approximation, J. C. Mason and M. G. Cox, eds., Clarendon Press, Oxford, U.K., 1987, pp. 143–167.

[17] S. I. M. RITCHIE, *Surface representation by finite elements*, Master's thesis, Department of Mathematics and Statistics, University of Calgary, Calgary, Alberta, Canada, 1978.

[18] S. RIPPA, *Piecewise linear interpolation and approximation schemes over data dependent triangulations*, Ph.D. thesis, School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel, 1990.

[19] L. L. SCHUMAKER, *Fitting surfaces to scattered data*, in Approximation Theory II, G. G. Lorentz, C. K. Chui, and L. L. Schumaker, eds., Academic Press, New York, 1976, pp. 203–268.

[20] ———, *Numerical aspects of spaces of piecewise polynomials on triangulations*, in Algorithms for Approximation, J. C. Mason and M. G. Cox, eds., Clarendon Press, Oxford, U.K., 1987, pp. 373–406.

[21] ———, *Triangulation Methods*, in Topics in Multivariate Approximation, C. K. Chui, L. L. Schumaker, and F. I. Utreras, eds., Academic Press, New York, 1987, pp. 219–232.

[22] ———, *Constructive aspects of spaces of bivariate piecewise polynomials*, in The Mathematics of Finite Elements and Application VI, J. H. Whiteman, ed., Academic Press, London, 1988, pp. 513–520.

[23] J. STOER AND R. BULIRSCH, *Introduction to Numerical Analysis*, Springer-Verlag, New York, Heidelberg, Berlin, 1980.

[24] D. F. WATSON AND G. M. PHILIP, *Survey–systematic triangulation*, Comput. Vision, Graphics and Image Process., 26 (1984), pp. 217–223.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 5, pp. 1142–1150, September 1992

006

THE MODIFIED TRUNCATED SVD METHOD FOR REGULARIZATION IN GENERAL FORM*

PER CHRISTIAN HANSEN[†], TAKASHI SEKII[‡], AND HIROMOTO SHIBAHASHI[§]

Abstract. The truncated singular value decomposition (SVD) method is useful for solving the standard-form regularization problem: $\min \|\mathbf{x}\|_2$ subject to $\min \|A\mathbf{x} - \mathbf{b}\|_2$. This paper presents a modification of the truncated SVD method, which solves the more general problem: $\min \|L\mathbf{x}\|_2$ subject to $\min \|A\mathbf{x}-\mathbf{b}\|_2$, where L is a general matrix with full row rank. The extra work, associated with the introduction of the matrix L, is dominated by a QR-factorization of a matrix with dimensions smaller than those of L. In order to determine the optimal solution, it is often necessary to compute a sequence of regularized solutions, and it is shown how this can be accomplished with little extra computational effort. Finally, the new method is illustrated with an example from helioseismology.

Key words. truncated SVD, discrete ill-posed problems, regularization

AMS(MOS) subject classifications. 65F30, 65F20, 65R20

1. Introduction.

A variety of problems in astronomy, geodesy, image and signal processing, and statistics lead to so-called discrete ill-posed problems when they are solved numerically. By a *discrete ill-posed problem* we mean either a square or an overdetermined system of linear algebraic equations, i.e., $A\mathbf{x} = \mathbf{b}$ or $\min \|A\mathbf{x} - \mathbf{b}\|_2$, whose coefficient matrix A is very ill conditioned in such a way that its singular values decay rapidly to zero. Such discrete ill-posed problems often arise when one wants to solve an underlying, continuous, ill-posed problem, typically a Fredholm integral equation of the first kind. See [1, §26] and [9] for surveys of numerical aspects of discrete ill-posed problems.

Discrete ill-posed problems cannot be solved by standard methods in numerical linear algebra such as LU- or QR-factorizations due to the large condition number of the matrix A (note that the condition number is the ratio between the largest and the smallest singular value). However, the large condition number does not imply that the problem cannot be solved, it merely implies that the standard methods are not suited for solving the problem. Instead, a regularization method must be used to "filter out" the parts of the solution corresponding to all the small singular values.

A widely used regularization method is the *truncated singular value decomposition* (TSVD). This method amounts to truncating the singular value expansion of the coefficient matrix A in such a way that the smallest singular values of A are discarded, and then solving this modified least squares problem. The major advantage of using the TSVD method is that the singular value decomposition (SVD) itself provides important information about, and insight into, the discrete ill-posed problem. TSVD is equivalent to Tikhonov regularization in standard form, $\min \{\|A\mathbf{x}-\mathbf{b}\|_2^2+\lambda^2\|\mathbf{x}\|_2^2\}$, in the sense that for any truncation parameter in TSVD there exists a λ such that the solutions obtained by the two methods are close. Hence TSVD implicitly seeks to minimize the norm of the solution. See [7] for more details.

* Received by the editors January 21, 1991; accepted for publication July 10, 1991.

[†] UNI•C (Danish Computing Center for Research and Education), Building 305, Technical University of Denmark, DK-2800 Lyngby, Denmark (unipch@wuli.uni-c.dk).

[‡] Institute of Astronomy, University of Cambridge, The Observatories, Madingley Road, Cambridge CB3 0HA, United Kingdom (sekii@starlink.astronomy.cambridge.ac.uk).

[§] Department of Astronomy, Faculty of Science, University of Tokyo, Bunkyo-ku, Tokyo 113, Japan (shibahashi@dept.astron.s.u-tokyo.ac.jp).

Although TSVD works well in some applications, there are other applications that explicitly require the minimization of a seminorm of the solution $\|L\,\mathbf{x}\|_2$ instead of minimization of $\|\mathbf{x}\|_2$. The matrix $L \in \Re^{p\times n}$ (with $p \leq n$ and with full rank) is usually a discrete approximation to some derivative operator. Since the SVD of the matrix A reveals so much information about the underlying ill-posed problem, we are interested in an SVD-based method for treating regularization problems with a general matrix L.

In this paper, we describe a new *modified* TSVD (MTSVD) *method.* This method is an extension of the TSVD method and, in addition to the computation of the SVD of the coefficient matrix A, it only requires a standard QR-factorization of a matrix that appears during the algorithm. The basic idea was originally outlined in [12]. Here we derive an efficient and numerically stable algorithm for implementing the MTSVD method that relies solely on standard linear algebra computations.

Our paper is organized as follows. In §2 we briefly summarize the TSVD method and define the new MTSVD method. Sections 3 and 4 give important computational aspects of computing the MTSVD solution and determining of the optimal truncation parameter. Finally, in §5 we give a practical example from helioseismology in astrophysics, which illustrates the use of the method.

2. The MTSVD method. Let us first briefly summarize the properties of the SVD of an $m \times n$ matrix A (we assume for simplicity that $m \geq n$). The SVD consists of the following decomposition:

$$A = \sum_{i=1}^{n} \mathbf{u}_i\,\sigma_i\,\mathbf{v}_i^T\,,$$

where the singular vectors are orthonormal, $\mathbf{u}_i^T\mathbf{u}_j = \mathbf{v}_i^T\mathbf{v}_j = \delta_{ij}$ for $i,j = 1,\cdots,n$, and the singular values σ_i are nonnegative and appear in nonincreasing order: $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. The TSVD of A is then defined as the rank-k matrix

$$(1) \qquad A_k \equiv \sum_{i=1}^{k} \mathbf{u}_i\,\sigma_i\,\mathbf{v}_i^T\,, \qquad k < n$$

obtained from A by neglecting its $n-k$ smallest singular values. The TSVD solution $\mathbf{x}_k$ is defined in terms of A_k as the solution with minimum two-norm to the least squares problem $\min \|A_k\mathbf{x} - \mathbf{b}\|_2$. In other words, $\mathbf{x}_k$ solves the problem

$$(2) \qquad \min_{\mathbf{x}\in\mathcal{S}} \|\mathbf{x}\|_2, \qquad \mathcal{S} = \{\,\mathbf{x}\,|\,\|A_k\,\mathbf{x} - \mathbf{b}\|_2 = \min\,\}.$$

It is straightforward to show that the TSVD solution defined by (2) can be written in terms of the SVD as

$$(3) \qquad \mathbf{x}_k = A_k^+\mathbf{b}, \qquad A_k^+ \equiv \sum_{i=1}^{k} \mathbf{v}_i\,\sigma_i^{-1}\,\mathbf{u}_i^T\,.$$

Here A_k^+ denotes the pseudo-inverse of the matrix A_k [1, §4]. The TSVD solution $\mathbf{x}_k$ is a regularized solution [7]. The integer k is usually called the *truncation parameter*, and the larger k is the larger $\|\mathbf{x}_k\|_2$ will be.

Let us now introduce the MTSVD solution $\mathbf{x}_{L,k}$ as the solution to (2) with $\|\mathbf{x}\|_2$ replaced by $\|L\,\mathbf{x}\|_2$, i.e., $\mathbf{x}_{L,k}$ solves the problem

$$(4) \qquad \min_{\mathbf{x}\in\mathcal{S}} \|L\,\mathbf{x}\|_2, \qquad \mathcal{S} = \{\,\mathbf{x}\,|\,\|A_k\,\mathbf{x} - \mathbf{b}\|_2 = \min\,\}.$$

Equation (4) is a natural generalization of (2), and the MTSVD solution $\mathbf{x}_{L,k}$ is obviously also a regularized solution.

In order to derive a formal as well as a computational expression for the MTSVD solution, we first need to define the following two matrices:

$$V_k \equiv [\mathbf{v}_{k+1}, \cdots, \mathbf{v}_n], \qquad U_k \equiv [\mathbf{u}_{k+1}, \cdots, \mathbf{u}_n]. \tag{5}$$

It is also convenient to define the $(n-k)\times(n-k)$ diagonal matrix

$$\Sigma_k \equiv \mathrm{diag}\,(\sigma_{k+1}, \cdots, \sigma_n) \tag{6}$$

consisting of the $n-k$ smallest singular values of A. Then the MTSVD solution $\mathbf{x}_{L,k}$ can be expressed as follows.

THEOREM 2.1. *Let A_k denote the* TSVD *of A, given by* (1)*, and let V_k be given by* (5)*. Then the* MTSVD *solution to* (4) *can be written as*

$$\mathbf{x}_{L,k} = (I_n - V_k\,(L\,V_k)^+ L)\,A_k^+\mathbf{b} = \mathbf{x}_k - V_k\,(L\,V_k)^+ L\,\mathbf{x}_k, \tag{7}$$

where $\mathbf{x}_k$ is the ordinary TSVD *solution* (3).

Proof. Following Eldén [4, Thm. 2.1], the MTSVD solution can be written formally as

$$\mathbf{x}_{L,k} = (I_n - (L\,(I_n - A_k^+ A_k))^+ L)\,A_k^+\mathbf{b}.$$

If we insert (1), (5), and (6) into this equation, use the relation $I_n - A_k^+ A_k = V_k V_k^T$, and also use the fact that $(L\,V_k V_k^T)^+ = V_k\,(L\,V_k)^+$, then we arrive at (7). The latter equation is proved by verifying the four Penrose conditions [1, Remark 4.1] and using the fact that $V_k^T V_k = I_{n-k}$. □

Remark. In (7), the matrix $(I - V_k\,(L\,V_k)^+ L)\,A_k^+$ is the so-called *L-weighted pseudo-inverse* of A_k (cf. [4]).

3. Computational aspects. We emphasize that (7) should not be used for computational purposes. However, it gives insight into the MTSVD solution. We see that $\mathbf{x}_{L,k}$ consists of two components: the ordinary TSVD solution $\mathbf{x}_k$ (3) minus a "correction" vector in the null space of A_k given by $V_k\,\mathbf{z}_k = V_k\,(L\,V_k)^+ L\,\mathbf{x}_k$. We also see that the vector $\mathbf{z}_k = (L\,V_k)^+ L\,\mathbf{x}_k$ is the solution to the following least squares problem:

$$\min \|(L\,V_k)\,\mathbf{z} - L\,\mathbf{x}_k\|_2. \tag{8}$$

This problem is most conveniently solved by means of a QR-factorization of the matrix $L\,V_k$, which ensures that $\mathbf{z}_k$ is computed in a numerically stable manner.

As we shall see in the next section, it is sometimes necessary to compute the MTSVD solution $\mathbf{x}_{L,k}$ for several values of k, in order to determine the optimal k. It is therefore important to solve the least squares problem (8) efficiently when stepping through several values of k. Note that decreasing k simply adds more columns to the left of $L\,V_k$, while a QR-factorization is easier to update when columns are appended to the right. Hence we want to operate with the matrix $L\,V_k\,E_k$, where $E_k \equiv \mathrm{antidiag}\,(1, \cdots, 1)$ is the $(n-k)\times(n-k)$ exchange matrix that reverts the columns of $L\,V_k$. Now let $k_{\min}$ denote the smallest desired truncation parameter and partition the QR-factorization of $L\,V_{k_{\min}}\,E_{k_{\min}}$ such that for any $k > k_{\min}$ we have that

$$\begin{pmatrix} L\,V_k E_k & L\,[\mathbf{v}_k, \cdots, \mathbf{v}_{k_{\min}+1}] \end{pmatrix} = \begin{pmatrix} Q_1^{(k)} & Q_2 \end{pmatrix} \begin{pmatrix} R_{11}^{(k)} & R_{12} \\ 0 & R_{22} \end{pmatrix}, \tag{9}$$

where $Q_1^{(k)} \in \Re^{p\times(n-k)}$ and $R_{11}^{(k)} \in \Re^{(n-k)\times(n-k)}$. Then it is easy to see that the QR-factorization of $L\,V_k\,E_k$ for $k > k_{\min}$ is given by

$$L\,V_k\,E_k = Q_1^{(k)} R_{11}^{(k)},$$

such that $\mathbf{z}_k$ is given by $\mathbf{z}_k = E_k\,(R_{11}^{(k)})^{-1}(Q_1^{(k)})^T\mathbf{x}_k$. Using this "trick," it is straightforward to step through a sequence of different truncation parameters k without having to compute a new QR-factorization for each k.

The complete algorithm for computing the MTSVD solution, including determination of the optimal truncation parameter k via inspection of the residual norm and the solution seminorm, thus takes the following form:

Algorithm MTSVD

```
1     Compute the SVD of $A$ and store $U^T\mathbf{b}$, $\Sigma$, and $V$.
2     Form $L\,V_{k_{\min}}E_{k_{\min}}$ and compute its QR-factorization.
3     For all the required values of $k$:
3.1       Compute $\mathbf{x}_k$ and $L\,\mathbf{x}_k$.
3.2       Compute $\mathbf{z}_k = E_k\,(R_{11}^{(k)})^{-1}(Q_1^{(k)})^T L\,\mathbf{x}_k$.
3.3       Compute $\|L\,\mathbf{x}_{L,k}\|_2$ and $\|A\,\mathbf{x}_{L,k} - \mathbf{b}\|_2$ (see §4).
3.4       Use this information to choose the optimal $k$.
4     Compute $\mathbf{x}_{L,k} = \mathbf{x}_k - V_k\,\mathbf{z}_k$.
```

The computational overhead associated with computation of the MTSVD solution, compared to that of computing the TSVD solution, is dominated by the QR-factorization, which requires approximately $2\,p\,(2\,n^2 - 3\,n\,k_{\min} + k_{\min}^2)$ flops, including formation of the matrix $L\,V_{k_{\min}}$. This overhead should be compared to the computational effort of computing the SVD of A, which amounts to approximately $(2\,m + 11\,n)\,n^2$ flops [6, §5.4.5]. Hence even for moderate values of m, the overhead associated with computing the MTSVD solution is only a fraction of that of computing the SVD of A.

Whenever there is a distinct gap in the singular value spectrum of A, it is natural to set the truncation parameter k in (1) equal to the numerical rank of A, i.e., such that the gap appears between singular values σ_k and σ_{k+1}. In such situations a search through several singular values of k is not required, and it is not even necessary to compute the SVD of A. Instead, one can take advantage of the computationally much less expensive *rank-revealing QR-factorization* of A, which is guaranteed to capture the numerical rank k of A. By means of a rank-revealing QR-factorization, we can easily compute a minimum-norm least squares solution as well as null-vectors of A (analogous to the vectors in V_k). We will not pursue this aspect further here, but instead refer to [2] and [3, §3] for theoretical and computational details.

4. Methods for choosing the regularization parameter. We now turn to the determination of the appropriate truncation parameter k. Often the optimal value of k is evident from a simple plot of the singular values σ_i and the corresponding "Fourier coefficients" $\mathbf{u}_i^T\mathbf{b}$ for $i = 1, \cdots, n$. See [8] for more details on this approach and why it works. If such a plot does not reveal the optimal truncation parameter, we suggest two alternative methods. One is the generalized cross-validation method discussed below, which requires computation of the norm of the residual vector corresponding to $\mathbf{x}_{L,k}$, i.e., $\|A\,\mathbf{x}_{L,k} - \mathbf{b}\|_2$. In the other method, one plots the seminorm of the solution $\|L\,\mathbf{x}_{L,k}\|_2$ versus the residual norm, and then chooses the k that corresponds to a characteristic L-shaped "corner" of the $(\|A\,\mathbf{x}_{L,k} - \mathbf{b}\|_2\,,\ \|L\,\mathbf{x}_{L,k}\|_2)$-curve.

For more details about this method for choosing k, as well as some alternative methods, we refer to [9, §6].

Both the solution seminorm $\|L\,\mathbf{x}_{L,k}\|_2$ and the residual norm $\|A\,\mathbf{x}_{L,k}-\mathbf{b}\|_2$ can be computed efficiently by means of the following expressions.

THEOREM 4.1. *With the notation from* (5), (6), *and* (9), *the solution seminorm and the residual norm are given by:*

$$\|L\,\mathbf{x}_{L,k}\|_2 = \|(I_p - L\,V_k\,(L\,V_k)^+)L\,\mathbf{x}_k\|_2 = \|Q_2^T L\,\mathbf{x}_k\|_2, \tag{10}$$

$$\|A\,\mathbf{x}_{L,k}-\mathbf{b}\|_2 = (\|U_k^T\mathbf{b}+\Sigma_k\,\mathbf{z}_k\|_2^2+\delta_o^2)^{1/2}, \tag{11}$$

where we have defined $\delta_o \equiv \left(\sum_{i=n+1}^m(\mathbf{u}_i^T\mathbf{b})^2\right)^{1/2}$.

Proof. It follows from (7) that $\|L\,\mathbf{x}_{L,k}\|_2$ is equal to the norm of the residual vector in (8). Using the factorization (9) we then obtain (10). The residual norm is given by $\|A\,\mathbf{x}_{L,k}-\mathbf{b}\|_2 = \|A\,\mathbf{x}_k-\mathbf{b}-A\,V_k\,\mathbf{z}_k\|_2$. Since $\mathbf{x}_k=\sum_{i=1}^k\mathbf{v}_i\sigma_i^{-1}\mathbf{u}_i^T\mathbf{b}\Rightarrow A\,\mathbf{x}_k=\sum_{i=1}^k\mathbf{u}_i\mathbf{u}_i^T\mathbf{b}$, we can write $A\,\mathbf{x}_k-\mathbf{b}=-U_kU_k^T\mathbf{b}-\sum_{i=n+1}^m\mathbf{u}_i\mathbf{u}_i^T\mathbf{b}$. Moreover, $A\,V_k\,\mathbf{z}_k=U_k\Sigma_k\,\mathbf{z}_k$. The vector $\sum_{i=n+1}^m\mathbf{u}_i\mathbf{u}_i^T\mathbf{b}$ is obviously orthogonal to the vectors $U_kU_k^T\mathbf{b}$ and $U_k\Sigma_k\,\mathbf{z}_k$, and the residual norm is therefore given by $\|A\,\mathbf{x}_{L,k}-\mathbf{b}\|_2^2 = \|U_k^T\mathbf{b}+\Sigma_k\,\mathbf{z}_k\|_2^2+\delta_o^2$. □

Since the seminorm of the MTSVD solution $\|L\,\mathbf{x}_{L,k}\|_2$ is simply the norm of the residual vector in (8), it can be computed with little overhead. The quantities $\mathbf{u}_i^T\mathbf{b}$ are available from most SVD programs such that $\|A\,\mathbf{x}_{L,k}-\mathbf{b}\|_2$ can be computed in only $O(m)$ flops. Both norms can therefore be computed very efficiently, so that it is easy to monitor the $(\|A\,\mathbf{x}_{L,k}-\mathbf{b}\|_2\,,\,\|L\,\mathbf{x}_{L,k}\|_2)$-curve during Algorithm MTSVD.

Next we consider the *generalized cross-validation* (GCV) method [5], [15]. To summarize this method in general terms, let $\mathbf{x}_\lambda = A_\lambda^I\,\mathbf{b}$ denote the solution computed by means of some regularization method with regularization parameter λ. Then the GCV method amounts to finding the regularization parameter λ that minimizes the GCV-function $G(\lambda)$, defined by

$$G(\lambda)\equiv\frac{\|A\,\mathbf{x}_\lambda-\mathbf{b}\|_2^2}{(\mathrm{trace}\,(I_m-A\,A_\lambda^I))^2}.$$

Here the matrix $A\,A_\lambda^I$ is called the influence matrix. For MTSVD we obtain the following simple expression.

THEOREM 4.2. *The* GCV-*function for* MTSVD *is given by*

$$G(k)=\frac{\|A\,\mathbf{x}_{L,k}-\mathbf{b}\|_2^2}{(m-k)^2}. \tag{12}$$

Proof. In the MTSVD solution the regularization parameter is $\lambda=k$, and from (7) we see that A_λ^I is the L-weighted pseudo-inverse of A, i.e., $A_\lambda^I=(I_n-V_k\,(L\,V_k)^+L)\,A_k^+$. Hence if we define $\bar\Sigma_k\equiv\mathrm{diag}\,(\sigma_1,\cdots,\sigma_k)$, $\bar V_k\equiv[\mathbf{v}_1,\cdots,\mathbf{v}_k]$, and $\bar U_k\equiv[\mathbf{u}_1,\cdots,\mathbf{u}_k]$, then the influence matrix for MTSVD is given by

$$\begin{aligned}A\,A_\lambda^I &= A\,(I_n-V_k\,(L\,V_k)^+L)\,A_k^+ = A\,A_k^+ - A\,V_k\,(L\,V_k)^+L\,A_k^+\\ &= \bar U_k\bar U_k^T - U_k\Sigma_k(L\,V_k)^+L\,\bar V_k\bar\Sigma_k^{-1}\bar U_k^T = \bar U_n\begin{pmatrix} I_k & 0\\ \Sigma_k(L\,V_k)^+L\,\bar V_k\bar\Sigma_k^{-1} & 0\end{pmatrix}\bar U_n^T.\end{aligned}$$

From this relation, it immediately follows that $\operatorname{trace}(I_m - A A_\lambda^I) = m - k$, and we therefore obtain (12). □

We suggest that this easily computed function be used in the above Algorithm MTSVD, in addition to monitoring the $(\|A\mathbf{x}_{L,k} - \mathbf{b}\|_2, \|L\mathbf{x}_{L,k}\|_2)$-curve as a criterion for choosing the optimal k.

5. An application from helioseismology. In this final section, we illustrate the use of the MTSVD method by applying it to an inverse problem from helioseismology in astrophysics. The sun is oscillating in millions of global eigenmodes, and by using information from these eigenmodes we are able to explore the internal structure of the sun. See [13] for an introductory review of helioseismology and [14] for more details. The fundamental equation in helioseismology is the following Fredholm integral equation of the first kind:

$$\int_0^R \int_0^\pi K_{nlm}(r,\theta)\, \Omega(r,\theta)\, d\theta\, dr = \delta\omega_{nlm}. \tag{13}$$

Here the unknown function $\Omega(r,\theta)$ is the solar internal rotational frequency at the spherical coordinate (r,θ). Moreover, the kernel $K_{nlm}(r,\theta)$ is a function that depends on the solar internal structure and the modes n, l, and m, which are quantum numbers that describe the eigenoscillations of the sun. Finally, the right-hand side quantities $\delta\omega_{nlm}$ are the so-called frequency splittings, which are essentially observed quantities. The number of observable modes is as many as a few thousand.

Since our purpose here is to illustrate the MTSVD method, rather than to discuss the physical meaning of the computed results, we assume that the function $\Omega(r,\theta)$ is a function of only r, in order to make the following discussion simple. The full two-dimensional inversion is discussed in other papers [10], [11]. In this numerical experiment, we prepare first an artificial (but plausible) function $\Omega(r)$ and calculate the resultant frequency splittings $\delta\omega_{nlm}$. We then treat them as observed values and solve the integral equation (13) to see how well we can reproduce $\Omega(r)$ as the solution of the integral equation.

We discretize the radial coordinate into nonequidistant mesh points r_j, which become denser towards the surface of the sun. We then compute approximations Ω_j to $\Omega(r_j)$ by substituting the integral in (13) with a simple summation. The integral equation is thus transformed into a discrete ill-posed problem, where the right-hand side is $\mathbf{b} \equiv \{\delta\omega_i\}$, the coefficient matrix A is given by $A \equiv \{K_i(r_j)\,(r_j - r_{j-1})\}$, and the solution is $\mathbf{x} \equiv \{\Omega_j\}$. The suffix i denotes the mode number, that is, the set of the quantum numbers $\{nlm\}$, and the suffix j denotes the mesh number. In the following examples, we use 250 mesh points and 1836 different modes. We impose "smoothness" on the computed solution by choosing the regularization matrix L as an approximation to the second derivative operator.

Figure 1 shows all the singular values σ_i of the matrix A. We see that they are clustered in two distinct groups with a narrow transition region between large and small σ_i. It is natural to choose the truncation parameter k somewhere in this transition region and thus avoid a search through many values of k. We have chosen $k = 82$, which gives an effective condition number σ_1/σ_k of A_k of the order 10^5. We then solve the system for two different cases of the function $\Omega(r)$.

Example 1. First we deal with slightly differential rotation. The TSVD and MTSVD solutions are shown in Fig. 2, and we see that the TSVD solution is unrealistically oscillatory. We also see that the MTSVD solution reproduces the original

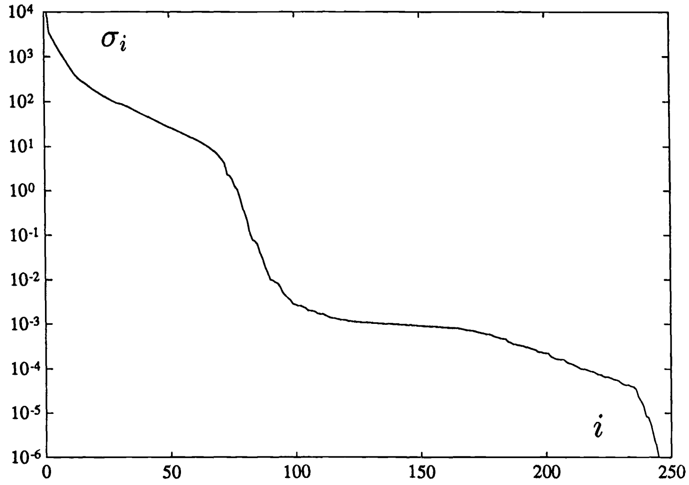


FIG. 1. *The singular values of the* 1836 × 250 *matrix A obtained from discretization of the integral equation* (13).

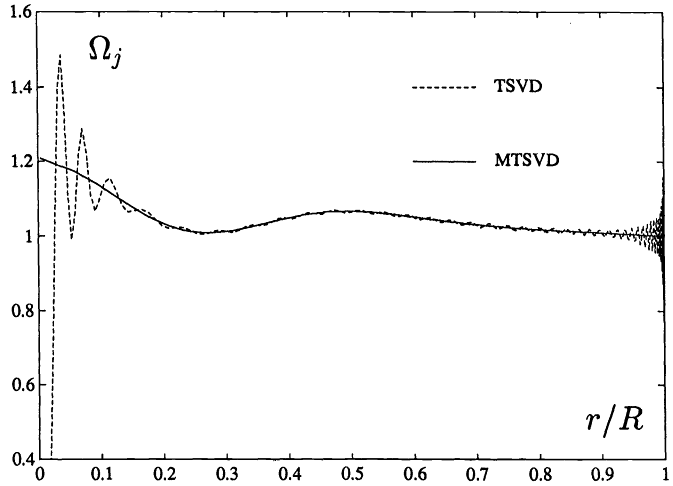


FIG. 2. *The computed solutions to Example* 1, *where $\Omega(r)$ varies slowly.*

function $\Omega(r)$ much better than the TSVD solution does, in fact, the MTSVD is indistinguishable from the exact $\Omega(r)$ in this plot.

Example 2. The second example features differential rotation with some steep gradients and a discontinuity. Figure 3 shows the TSVD and the MTSVD solutions. Again, the TSVD solution is unsatisfactory because it includes unrealistic oscillations. The MTSVD solution is much better, although (as expected) it cannot completely reproduce the step in $\Omega(r)$ at $r/R = 0.8$.

6. Conclusion. We have presented an efficient algorithm for regularization of discrete ill-posed problems in general form, i.e., for solving the problem min $\|L\mathbf{x}\|_2$

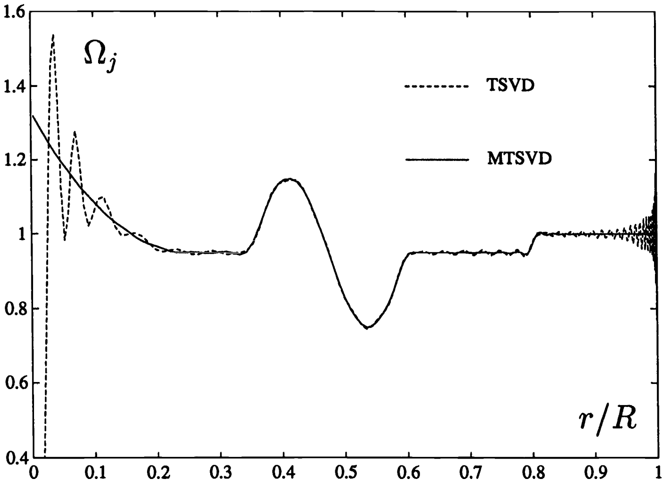


FIG. 3. *The computed solutions to Example* 2, *where* $\Omega(r)$ *has steep gradients and a discontinuity.*

subject to $\min \|A\mathbf{x} - \mathbf{b}\|_2$. The algorithm extends the well-known TSVD method (which corresponds to $L = I$) to the case with a general matrix L. The computational overhead associated with an $L \neq I$ is dominated by a QR-factorization, and this overhead is usually much smaller than the effort involved in computing the SVD of the matrix A.

We have also shown how the norms $\|L\mathbf{x}\|_2$ and $\|A\mathbf{x} - \mathbf{b}\|_2$ can be computed efficiently. This is important in connection with methods such as generalized cross-validation, which automatically seek to compute the optimal truncation parameter.

Finally, we have illustrated the use of our algorithm in connection with the numerical treatment of the inverse helioseismic problem in astrophysics. Our examples show the superiority of our new method with $L \neq I$, as compared to the case $L = I$, for this particular problem.

REFERENCES

[1] A. BJÖRCK, *Least Squares Methods*, in Handbook of Numerical Analysis Vol. I, P. G. Ciarlet and J. L. Lions, eds., Elsevier, Amsterdam, 1990.

[2] T. F. CHAN AND P. C. HANSEN, *Computing truncated* SVD *least squares solutions by rank revealing* QR*-factorizations*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 519–530.

[3] ———, *Some applications of the rank revealing* QR*-factorization*, CAM Report 90-09, Department of Mathematics, University of California, Los Angeles, CA, May 1990; SIAM J. Sci. Statist. Comput., 13 (1992), pp. 727–741.

[4] L. ELDÉN, *A weighted pseudoinverse, generalized singular values, and constrained least squares problems*, BIT, 22 (1982), pp. 487–502.

[5] G. H. GOLUB, M. T. HEATH, AND G. WAHBA, *Generalized cross validation as a method for choosing a good ridge parameter*, Technometrics, 21 (1979), pp. 215–224.

[6] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, Second Edition, The Johns Hopkins University Press, Baltimore, MD, 1989.

[7] P. C. HANSEN, *Truncated* SVD *solutions to discrete ill-posed problems with ill-determined numerical rank*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 503–518.

[8] ———, *The discrete Picard condition for discrete ill-posed problems*, BIT, 30 (1990), pp. 658–672.

[9] ———, *Analysis of discrete ill-posed problems by means of the L-curve*, Report MCS-P157-0690,

Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, July 1990; SIAM Review, to appear.

[10] T. Sekii, *Two-dimensional inversion of rotational splitting data*, in Progress of Seismology of the Sun and Stars, Y. Osaki and H. Shibahashi, eds., Springer-Verlag, Berlin, New York, 1990, pp. 337–340.

[11] ———, *Two-dimensional inversion for solar internal rotation*, Publ. Astron. Soc. Japan, 43 (1991), pp. 381–411.

[12] T. Sekii and H. Shibahashi, *An inversion method based on the Moore–Penrose generalized inverse matrix*, in Seismology of the Sun and Sun-Like Stars, E. J. Rolfe, ed., ESA SP-286, European Space Agency Publications Division, Noordwijk, the Netherlands, 1988, pp. 521–523.

[13] H. Shibahashi, *Introductory review of solar and stellar oscillations*, in Progress of Seismology of the Sun and Stars, Y. Osaki and H. Shibahashi, eds., Springer-Verlag, Berlin, New York, 1990, pp. 3–19.

[14] W. Unno, Y. Osaki, H. Ando, H. Saio, and H. Shibahashi, *Nonradial Oscillations of Stars*, Second Edition, University of Tokyo Press, Tokyo, 1989.

[15] G. Wahba, *Spline Models for Observational Data*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 5, pp. 1151–1172, September 1992 007

HIGHLY PARALLEL SPARSE CHOLESKY FACTORIZATION*

JOHN R. GILBERT[†] AND ROBERT SCHREIBER[‡]

Abstract. This paper develops and compares several fine-grained parallel algorithms to compute the Cholesky factorization of a sparse matrix. The experimental implementations are on the Connection Machine, a distributed-memory SIMD machine whose programming model conceptually supplies one processor per data element. In contrast to special-purpose algorithms in which the matrix structure conforms to the connection structure of the machine, this paper focuses on matrices with arbitrary sparsity structure.

The most promising alternative is a supernodal, multifrontal algorithm whose inner loop performs several dense factorizations simultaneously on a two-dimensional grid of processors. The key subroutine is a fine-grained parallel, dense-factorization algorithm. The sparse code attains execution rates comparable to those of the dense subroutine. Although at present architectural limitations prevent the dense factorization from realizing its potential efficiency, it is concluded that a regular data parallel architecture can be used efficiently to solve arbitrarily structured sparse problems.

A performance model is also presented and used to analyze these algorithms. Asymptotic analysis combined with experimental measurement of parameters is accurate enough to be useful in choosing among alternative algorithms for a complicated problem.

Key words. sparse matrix algorithms, Cholesky factorization, supernodal factorization, multifrontal factorization, systems of linear equations, parallel computing, data parallel algorithms, chordal graphs, clique trees, Connection Machine, performance analysis

AMS(MOS) subject classifications. 05C50, 15A23, 65F05, 65F50, 68M20

1. Introduction.

1.1. Data parallelism.

Highly parallel, local-memory computer architectures promise to achieve high performance inexpensively by assembling a large amount of simple hardware in a way that scales without bottlenecks. A data parallel programming model simplifies the programming of local memory parallel architectures by associating a processor with every data element in a computation (at least conceptually), thus hiding from the programmer the details of distribution of data and work to the processors.

Some major challenges accompany these promises. Communication is expensive relative to computation, so an algorithm must minimize communication. Locality is important in communication, so it pays to substitute communication with nearby processors for more general patterns where possible. The sequential programmer tunes the inner loop of an algorithm for high performance, but simple data parallel algorithms tend to have "everything in the inner loop" because a sequential loop over the data is typically replaced by a parallel operation. (From this point of view, the advantage of the more complicated of our two algorithms, Grid Cholesky, is that it removes the general-pattern communication from the inner loop.)

* Received by the editors May 23, 1990; accepted for publication (in revised form) August 16, 1991.

[†] Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, California 94304. Copyright © 1990, 1991 Xerox Corporation. All rights reserved.

[‡] Research Institute for Advanced Computer Science, MS T045-1, National Aeronautics and Space Administration Ames Research Center, Moffett Field, California 94035. This author's work was supported by the Numerical Aerodynamic Simulation Systems Division of the National Aeronautics and Space Administration and by the Advanced Research Projects Agency via Cooperative Agreement NCC 2-387 between National Aeronautics and Space Administration and the Universities Space Research Association.

Algorithms for data parallel architectures must make different trade-offs than sequential algorithms: They must exploit regularity in the data. For efficiency on single instruction, multiple data (SIMD) machines, they must also be highly regular in the time dimension. In some cases entirely new approaches may be appropriate; examples of experiments with such approaches include particle-in-box flow simulation, knowledge-base maintenance [5], and the entire field of neural computation [20]. On the other hand, the same kind of regularity in a problem or an algorithm can often be exploited in a wide range of architectures; therefore, many ideas from sequential computation turn out to be surprisingly applicable in the highly parallel domain. For example, block-oriented matrix operations are useful in sequential machines with hierarchical storage and conventional vector supercomputers [3]; we shall see that they are also crucial to efficient data parallel matrix algorithms.

1.2. Goals of this study. Data parallel algorithms are natural for computations on matrices that are dense or have regular nonzero structures arising from, for example, regular finite-difference discretizations. The main goal of this research is to determine whether data parallelism is useful in dealing with irregular, arbitrarily structured problems. Specifically, we consider computing the Cholesky factorization of an arbitrary sparse, symmetric, positive definite matrix. We make no assumptions about the nonzero structure of the matrix besides symmetry. We present evidence that arbitrary sparse problems can be solved nearly as efficiently as dense problems by carefully exploiting regularities in the nonzero structure of the triangular factor that come from the clique structure of its chordal graph.

A second goal is to perform a case study in analysis of parallel algorithms. The analysis of sequential algorithms and data structures is a mature and useful science that has contributed to sparse matrix computation for many years. By contrast, the study of complexity of parallel algorithms is in its infancy, and it remains to be seen how useful parallel complexity theory will be in designing efficient algorithms for real parallel machines. We argue by example that, at least within a particular class of parallel architectures, asymptotic analysis combined with experimental measurement of parameters is accurate enough to be useful in choosing among alternative algorithms for a single fairly complicated problem.

1.3. Outline. The structure of the remainder of the paper is as follows. In §2 we review the definitions we need from numerical linear algebra and graph theory, sketch the architecture of the Connection Machine, and present a timing model for a generalized data parallel computer that abstracts that architecture.

In §3 we present the first of two parallel algorithms for sparse Cholesky factorization. The algorithm, which we call Router Cholesky, is based on a theoretically efficient algorithm in the parallel random access machine (PRAM) model of parallel computation. We analyze the algorithm and point out two reasons that it fails to be practical, one having to do with communication and the other with processor utilization.

In §4 we present a second algorithm, which we call Grid Cholesky. Grid Cholesky is a data parallel implementation of a supernodal, multifrontal method that draws on the ideas of Duff and Reid [7] and Ashcraft et al. [1]. It improves on Router Cholesky by using a two-dimensional grid of processors to operate on dense submatrices, thus replacing most of the slow generally routed communication of Router Cholesky with faster grid communication. It also solves the processor utilization problem by assigning different data elements to the working processors at different stages of the computation. We present an analysis and experimental results for a pilot implementation of Grid Cholesky on the Connection Machine.

The pilot implementation of Grid Cholesky is approximately as efficient as a dense Cholesky factorization algorithm, but is still slow compared to the theoretical peak performance of the machine. Several steps necessary to improve the absolute efficiency of the algorithm, most of which concern efficient Cholesky factorization of dense matrices, are described. Finally, we draw some conclusions and discuss avenues for further research.

2. Definitions.

The first two subsections below summarize the linear algebra and graph theory needed to study sparse Cholesky factorization. Most of this material is covered in more detail by George and Liu [13], [23], [24]. The remainder of the section outlines the data parallel programming model and its implementation on the Connection Machine, and describes our parametric model of a data parallel computer.

2.1. Linear algebra.

Let $A = (a_{ij})$ be an $n \times n$ real, symmetric, positive definite sparse matrix. There is a unique $n \times n$ lower triangular matrix $L = (l_{ij})$ with positive diagonal such that $A = LL^T$. This is the Cholesky factorization of A. We seek to compute L; with it we solve the linear system $Ax = b$ by solving $Ly = b$ and $L^T x = y$. We discuss algorithms for computing L below. In general, L is less sparse than A. The nonzero positions of L that were zero in A are called *fill* or *fill-in*. For any matrix X, we write $\eta(X)$ to denote the number of nonzero elements of X.

The rows and columns of A may be symmetrically reordered so that the system solved is $PAP^T(Px) = Pb$, where P is a permutation matrix. We assume that such a reordering, chosen to reduce $\eta(L)$ and the number of operations required to compute L, has been done. We further assume that the structure of L has been determined by a symbolic factoring process. We ignore these preliminary computations in this study because the cost of actually computing L typically dominates. (In many cases, several identically structured matrices may be factored using the same ordering and symbolic factorization.) A study of the implementation of appropriate reordering and symbolic factorization procedures on data parallel architectures is in preparation [18].

If the matrix A is such that its Cholesky factor L has no more nonzeros than the lower triangle of A, i.e., there is no fill, then A is a *perfect elimination matrix.* If PAP^T is a perfect elimination matrix for some permutation matrix P, we call the ordering corresponding to P a *perfect elimination ordering* of A.

Let R and S be subsets of $\{1, \cdots, n\}$. Then $A(R, S)$ is the $|R| \times |S|$ matrix whose elements are $a_{r,s}$ for $r \in R$ and $s \in S$. (For any set S, we write $|S|$ to denote its cardinality.)

2.2. Graph theory.

2.2.1. Vertex elimination.

We associate two ordered, undirected graphs with the sparse, symmetric matrix A. First, $G(A)$, the graph of A, is the graph with vertices $\{1, 2, \cdots, n\}$ and edges $E(A) = \{(i, j) \mid a_{ij} \neq 0\}$. (Note that $E(A)$ is a set of unordered pairs.) Next, we define the *filled graph* $G^*(A)$ with vertices $\{1, 2, \cdots, n\}$ and edges $E^*(A) = \{(i, j) \mid l_{ij} \neq 0\}$, so that $G^*(A)$ is $G(L + L^T)$. The edges in $G^*(A)$ that are not edges of $G(A)$ are called *fill edges.* The output of a symbolic factorization of A is a representation of $G^*(A)$.

For every fill edge (i, j) in $E^*(A)$ there is a path in $G(A)$ from vertex i to vertex j whose vertices all have numbers lower than both i and j; moreover, for every such path in $G(A)$ there is an edge in $G^*(A)$ [28]. Consider renumbering the vertices of $G^*(A)$. With another numbering, this last property may or may not hold. If it does, then the new ordering is a perfect elimination ordering of $G^*(A)$; Liu [24] calls it an *equivalent reordering* of $G^*(A)$.

2.2.2. Chordal graphs. Every cycle of more than three vertices in $G^*(A)$ has an edge between two nonconsecutive vertices (a *chord*). Such a graph is said to be *chordal.* Not only is $G^*(A)$ chordal for every A, but every chordal graph is the filled graph of some matrix [27].

Let $G = G(V, E)$ be any undirected graph. A *clique* is a subset X of V such that for all $u, v \in X$, $(u, v) \in E$. A clique is *maximal* if it is not a proper subset of any other clique. For any $v \in V$, the *neighborhood* of v, written $\operatorname{adj}(v)$, is the set $\{u \in V \mid (u, v) \in E\}$. The *monotone neighborhood* of v, written $\operatorname{madj}(v)$, is the smaller set $\{u \in V \mid u > v, (u, v) \in E\}$. We extend adj and madj to sets of vertices in the usual way.

A vertex v is *simplicial* if $\operatorname{adj}(v)$ is a clique. Two vertices, u and v, are *indistinguishable* if $\{u\} \cup \operatorname{adj}(u) = \{v\} \cup \operatorname{adj}(v)$. Two vertices are *independent* if there is no edge between them. A set of vertices is independent if every pair of vertices in it is independent; two sets A and B are independent if no vertex of A is adjacent to a vertex of B.

It is immediate that any two simplicial vertices are either independent or indistinguishable. A set of indistinguishable simplicial vertices thus forms a clique, though not generally a maximal clique. The equivalence relation of indistinguishability partitions the simplicial vertices into pairwise independent cliques. We call these the *simplicial cliques* of the graph.

2.2.3. Elimination trees. A fundamental tool in studying sparse Gaussian elimination is the *elimination tree.* This structure was defined by Schreiber [30]; Liu [24] gives a survey of its many uses. Let A have the Cholesky factor L. The elimination tree $T(A)$ is a rooted spanning forest of $G^*(A)$ defined as follows. If vertex u has a higher-numbered neighbor v, then the parent $p(u)$ of u in $T(A)$ is the smallest such neighbor; otherwise u is a root. In other words, the first off-diagonal nonzero element in the uth column of L is in row $p(u)$. It is easy to show that $T(A)$ is a forest consisting of one tree for each connected component of $G(A)$. For simplicity we assume in what follows that A is irreducible, so that vertex n is the only root, though our algorithms do not assume this.

There is a monotone increasing path in $T(A)$ from every vertex to the root. If (u, v) is an edge of $G^*(A)$ with $u < v$ (that is, if $l_{vu} \neq 0$) then v is on this unique path from u to the root. This means that when $T(A)$ is considered a spanning tree of $G^*(A)$, there are no "cross edges" joining vertices in different subtrees. It implies that, if we think of the vertices of $T(A)$ as columns of A or L, any given column of L depends only on columns that are its descendants in the tree.

2.2.4. Clique trees. The elimination tree describes a Cholesky factorization in terms of operations on single columns. A description in terms of operations on full blocks can yield algorithms with better locality of reference, which is an advantage either on a machine with a memory hierarchy (registers, cache, main memory, disk) or on a distributed-memory parallel machine. The Connection Machine falls into both of these categories.

Describing symmetric factorization in terms of operations on full submatrices is the key idea in both Duff and Reid's "multifrontal" algorithm [7] and the "supernodal" algorithm of Ashcraft et al. [1], which can be traced back to the so-called element model of factorization [15], [33]. A full submatrix of L is a clique in the chordal graph $G^*(A)$. The clique structure of chordal graphs has been explored extensively in the combinatorial literature; representations of chordal graphs as trees of cliques date back at least to 1972 [10] and continue to be used [16], [25], [26].

A *clique tree* for matrix A is a tree whose nodes are sets that partition the vertices of $G^*(A)$ into cliques in such a way that all the vertices of a node N are indistinguishable simplicial vertices in the graph that results by deleting from $G^*(A)$ all vertices in nodes that are proper descendants of N. An equivalent definition is to think of starting with an elimination tree and collapsing vertices that are indistinguishable from their parents. (This definition differs slightly from that of Peyton [26], whose tree nodes are overlapping maximal cliques of $G^*(A)$.)

A clique that is a node in a clique tree is also called a *supernode.* If all vertices of $G^*(A)$ in proper descendants of some supernode are deleted, the supernode becomes a simplicial clique in the resulting graph. The clique tree is sometimes called a *supernode tree* or *supernodal elimination tree* [2]. A matrix may have many different clique trees—indeed, the elimination tree itself is one. Our numerical factorization algorithm Grid Cholesky can actually use any clique tree; the symbolic factorization we describe in §4.1 uses a block Jess–Kees algorithm to compute a shortest possible clique tree.

2.3. The Connection Machine.

2.3.1. Architecture. The Connection Machine (model CM-2) is a local-memory, SIMD parallel computer. The description we present here corresponds to the machine architecture presented by the assembly language instruction set Paris [34]. We programmed the CM in *lisp, which is compiled into Paris.

A full-sized CM has $2^{16} = 65{,}536$ processors, each of which can directly access 65,536 bits of memory. (Since this work was completed, larger memories have become available.) The processors are connected by a communication network called the *router*, which is configured by a combination of microcode and hardware to be a 16-dimensional hypercube.

The essential feature of the CM programming model is the *parallel variable* or pvar. A pvar is an array of data in which every processor stores and manipulates one element. The size of a pvar may be a multiple of the number of physical machine processors, p. If there are V times as many elements in the pvar X as there are processors, then (through microcode) each physical processor simulates V virtual processors; thus the programmer's view remains "one processor per datum." The ratio V is called the *virtual processor* (VP) *ratio.* The CM requires that V must be a power of two. Thus we find useful the notation $\lceil x \rceil$, meaning the smallest power of two not smaller than the real number x.

The *geometry* of each set of virtual processors (and its associated pvars) is also determined by the programmer, who may choose to view it as any multidimensional array with dimensions that are powers of two. The VP sets and their pvars are embedded in the machine (using Gray codes) so that neighboring virtual processors are simulated by the same or neighboring physical processors.

The Paris instruction set corresponds fairly well to the abstract model of data parallel programming that the CM attempts to present to the programmer, but it does not correspond closely to the actual hardware of the CM. Largely for this reason, it is hard to get high performance when programming in Paris or in a language that is compiled into Paris [31]. We shall discuss this point in detail later. Other ways to view and to program the hardware of the CM-2 to provide better performance are now becoming available to users.

2.3.2. Connection Machine programming. Parallel computation on the CM is expressed through elementwise binary operations on pairs of pvars that reside in the same VP set—that is, that have the same VP ratio and layout on the machine. (Optionally, we may specify a boolean mask that selects only certain virtual processors to be active.) These operations take time proportional to V, since the actual processors must loop over their simulated virtual processors. This remains true even when the set of selected processors is very sparse.

Interprocessor communication is expressed and accomplished in three ways, which we discuss in order of increasing generality but decreasing speed.

Communication with virtual processors at nearest-neighbor grid cells is most efficient. A pvar may be shifted along any of its axes using this type of communication. The shift may be circular or end-off at the programmer's discretion.

A second communication primitive, *scan*, allows broadcast of data. For example, if x is a one-dimensional pvar with the value [1, 2, 3, 4, 5, 6, 7, 8] then a scan of x yields [1, 1, 1, 1, 1, 1, 1, 1]. Scans are implemented using the hypercube connections. The time for a scan of length n is linear in $\log n$. Scans can also be used to broadcast along either rows or columns of a two-dimensional array. Scans that perform parallel prefix arithmetic operations are also available, but we do not use them.

Scans of subarrays are possible. In a *segmented scan*, the programmer specifies a boolean pvar, the *segment pvar*, congruent to x. The segments of x between adjacent T values in the segment pvar are scanned independently. Thus, for example, if we use the segment pvar [T F F F T F F T] and x is as above, then a segmented scan returns [1, 1, 1, 1, 5, 5, 5, 8].

The third and most general form of communication allows a virtual processor to access data in the memory of any other virtual processor. These operations go by several different names even within the CM environment; we refer to them in terms already familiar in sparse matrix computation: *gather* and *scatter*.

A gather allows processors to read data in the memory of other processors. The CM term for a gather is `pref!!` (for the *lisp programmer) or `get` (for the Paris programmer). In a gather, three pvars are involved: the source, the destination, and the address. The address of the processor whose memory is to be read is taken from the integer address pvar. Suppose that the source is the one-dimensional pvar $[15, 14, 13, \cdots, 2, 1, 0]$ and the address pvar is $[0, 1, 2, 0, 1, 2, \cdots, 0, 1, 2, 0]$. Then the data stored in the destination is $[15, 14, 13, 15, 14, 13, \cdots, 15, 14, 13, 15]$. The Fortran-90 or Matlab statement that accomplishes this is `dest = source(address)`; it performs the assignment $dest(i) \leftarrow source(address(i))$ for $1 \leq i \leq \text{length}(dest)$.

A scatter allows processors to write data to the memory of other processors. The CM term for a scatter is `*pset` (for the *lisp programmer) or `send` (for the Paris programmer). Again, the three pvars are a source, a destination, and an integer address. The Fortran-90 or Matlab version is `dest(address) = source`, and the effect is $dest(address(i)) \leftarrow source(i)$ for $1 \leq i \leq \text{length}(source)$.

In a scatter, when the address pvar has duplicate values, data from several source processors are sent to the same destination processor. (When this happens we say that a *collision* has occurred.) The programmer can select one of several different ways to combine colliding values by specifying a *combining operator*. For example, if the source and address are as above and the destination initially has the value $[1, 1, \cdots, 1]$, then after a scatter-with-add, the destination has the value $[45, 40, 35, 1, 1, 1, \cdots, 1]$. The sum of elements $source(j)$, such that $address(j) = k$, is stored in $dest(k)$ if there are any such elements; otherwise, $dest(k)$ is unchanged. Other combining operators

TABLE 1
Parameters of CM *model.*

Connection Machine parametric model		
Parameter	Description	Measured CM-2 value
V	Virtual processor ratio	
μ	Memory reference time	$4.8 \cdot V$ μsec
ϕ	Multiply or add time $\div\, \mu$	7
σ	Scan time $\div\, \phi$	$6.2 + 1.2 \log_2$ scan-distance
ν	News time $\div\, \phi$	3
ρ	Route time $\div\, \phi$	scatter (no collisions): 64 scatter-add (4 collisions): 110 scatter-add (100 collisions): 200 gather (many collisions): 430

include product, maximum, minimum, and "error." We have found combining scatter to be a powerful aid to data parallel programming.

2.3.3. Measured CM performance. We now develop a model of performance on data parallel architectures and use it to analyze performance of several algorithms for sparse Cholesky factorization. The essential machine characteristics in the model are described by five parameters:

μ: The memory reference time for a 32-bit word.
ϕ: The 32-bit floating-point multiply time, in units of μ.
ν: The 32-bit grid communication time, in units of ϕ.
σ: The 32-bit scan time, in units of ϕ.
ρ: The 32-bit router time, in units of ϕ.

(Floating-point addition takes the same time as multiplication: $\phi\mu$.) In our model, execution time scales linearly with VP ratio, which is essentially correct for the Connection Machine. Therefore, μ is proportional to VP ratio, and the other parameters are independent of VP ratio. In Table 1, we give measured values for these parameters obtained by experiment on the CM-2. We observe that router times range over a factor of four depending on the number of collisions; it is possible to design pathological routing patterns that perform even worse than this. For any given pattern, gather usually takes just twice as long as scatter, presumably because it is implemented by sending a request and then sending a reply. Therefore in our approximate analyses we generally choose a value of ρ for scatter corresponding to the number of collisions observed, and model gather as taking 2ρ floating-point times.

3. Router Cholesky.

Our first parallel Cholesky factorization algorithm is a parallel implementation of a standard column-oriented sparse Cholesky; it is based closely on Gilbert and Hafsteinsson's theoretically efficient algorithm [17] for the PRAM model of computation. Its communication requirements are too unstructured for it to be very efficient on a fine-grained multiprocessor like the CM, but we implemented and analyzed it to use as a basis for comparison and to help tune our performance model of the CM.

3.1. The Router Cholesky algorithm. Router Cholesky uses the elimination tree $T(A)$ to organize its computation. For the present, assume that both the tree and the symbolic factorization $G^*(A)$ are available. (In our experiments we computed the symbolic factorization sequentially; Gilbert and Hafsteinsson [17] describe a parallel algorithm.) Each vertex of the tree corresponds to a column of the matrix.

Following George et al. [12], we express a sequential column-oriented Cholesky factorization in terms of two primitive operations, *cdiv* and *cmod*.

```
procedure Sequential Cholesky (matrix $A$);
    for $j \leftarrow 1$ to $n$ do
        for each edge $(i, j)$ of $G^*(A)$ with $i < j$ do
            cmod $(j, i)$ od;
        cdiv $(j)$ od
end Sequential Cholesky;
```

Routine *cdiv* (j) divides the subdiagonal elements of column j by the square root of the diagonal element in that column, and routine *cmod* (j, i) modifies column j by subtracting a multiple of column i. This is called a *left-looking* algorithm because column j accumulates all necessary updates *cmod* (j, i) from columns to its left just before the *cdiv* (j) that completes its computation. By contrast, a *right-looking* algorithm would perform all the updates *cmod* (j, i) using column i immediately after the *cdiv* (i).

Now consider the elimination tree $T(A)$. A given column (vertex) is modified only by columns (vertices) that are its descendants in the tree [30]. Therefore a parallel left-looking algorithm can compute all the leaf vertex columns at once.

```
procedure Router Cholesky (matrix $A$);
    for $h \leftarrow 0$ to $height(n)$ do
        for each edge $(i, j)$ with $height(i) < height(j) = h$ pardo
            cmod $(j, i)$ od;
        for each vertex $j$ with $height(j) = h$ pardo
            cdiv $(j)$ od
        od
end Router Cholesky;
```

Here $height(j)$ is the length of the longest path in $T(A)$ from vertex j to a leaf. Thus the leaves have height 0, vertices whose children are all leaves have height 1, and so forth. The outer loop of this algorithm works sequentially from the leaves of the elimination tree up to the root. At each step, an entire level's worth of *cmod*'s and *cdiv*'s are done.

A processor is assigned to every nonzero of the triangular factor (or, equivalently, to every edge and vertex of the filled graph G^*). Suppose processor P_{ij} is assigned to the nonzero that is initially a_{ij} and will eventually become l_{ij}. (If l_{ij} is a fill, then a_{ij} is initially zero; recall that we assume that the symbolic factorization is already done, so we know which l_{ij} will be nonzero.) In the parallel *cdiv* (j), processor P_{jj} computes l_{jj} as the square root of its element, and sends l_{jj} to processors P_{ij} for $i > j$, which then divide their own nonzeros by l_{jj}. In the parallel *cmod* (j, i), processor P_{ji} sends the multiplier l_{ji} to the processors P_{ki} with $k > j$. Each such P_{ki} then computes the update $l_{ki}l_{ji}$ locally and sends it to P_{kj} to be subtracted from l_{kj}.

We call this a *left-initiated* algorithm because the multiplications in *cmod* (j, i) are performed by the processors in column i who then, on their own initiative, send these updates to a processor in column j. Each column i is involved in at most one *cmod* at a time because every column modifying j is a descendant of j in $T(A)$, and the subtrees rooted at vertices of any given height are disjoint. Therefore each processor participates in at most one *cmod* or *cdiv* at each parallel step. If we ignore the time taken by communication (including the time to combine updates to a single P_{kj} that may come from different P_{ki_1}, P_{ki_2}, $\cdots$), then each parallel step takes a constant amount of time and the parallel algorithm runs in time proportional to the height of the elimination tree $T(A)$.

3.2. CM implementation of Router Cholesky. To implement Router Cholesky on the CM, we must specify how to assign data to processors, and then describe how to do the communication in *cdiv* and *cmod.*

We use one (virtual) processor for each nonzero in the triangular factor L. We lay out the nonzeros in a one-dimensional array in column major order, as in many standard sequential sparse matrix algorithms [13]. This makes operations within a single column efficient because they can use the CM scan instructions. Each column is represented by a processor for its diagonal element followed by a processor for each subdiagonal nonzero. The symmetric upper triangle is not stored. We can also think of this processor assignment as a processor for each vertex j of the filled graph, followed by a processor for each edge (i, j) with $i > j$.

Router Cholesky, like many data parallel algorithms, is profligate of parallel variable storage. Each processor contains some working storage and the following pvars:

`l`: Element of factor matrix L, initially A.
`i`: Row number of this element.
`j`: Column number of this element.
`j_ht`: *height*(j) in $T(A)$.
`i_ht`: *height*(i) in $T(A)$.
`diagonal_p`: Boolean: Is this a diagonal element?
`e_parent`: In processor P_{ij}, a pointer to $P_{i,p(j)}$.
`next_update`: Pointer to next element this one may update.

(Recall that $p(j) > j$ is the elimination tree parent of vertex $j < n$.)

At each stage of the sequential outer loop, each processor uses `i_ht` and `j_ht` to decide whether it participates in a *cdiv* or *cmod.* By comparing the local processor's `i_ht` or `j_ht` to the current value of the outer loop index, a processor can determine if its element is in a column that is involved in a *cdiv* or a *cmod.*

The *cdiv* uses a scan operation to copy the diagonal element to the rest of the active column.

The *cmod* uses a similar scan to copy the multiplier l_{ji} down to the rest of column i. The actual update is done by a scatter-with-add, which uses the router to send the update to its destination.

To determine where to send the update, each element maintains a pointer called `next_update` to a later element in its row. The nonzero positions in each row are a connected subgraph of the elimination tree, and are linked together in this tree structure by the `e_parent` pointers. Each nonzero updates only elements in columns that are its ancestors in the elimination tree. At each stage, `next_update` is moved one step up the tree using the `e_parent` pointers.

3.3. Router Cholesky performance: Theory. Each stage of Router Cholesky does a constant number of router operations, scans, and arithmetic operations. The number of stages is $h+1$, where h is the height of the elimination tree. In terms of the parameters of the machine model in §2.3.2, its running time is

$$(c_1\rho + c_2\sigma + c_3)\phi\mu h.$$

Recall that the memory reference time μ is proportional to the virtual processor ratio, which in this case is $\lceil \eta(L)/p \rceil$. The c_i are constants.

The most time-consuming step is incrementing the `next-update` pointer; the router is used twice in this operation. The dominant term is the router term $c_1 \rho \phi \mu h$. Notice that we do not explicitly count time for combining updates to the same element from different sources, since this is handled within the router and is thus included in ρ.

To get a feeling for this analysis consider the model problem, a five-point finite-difference mesh in two dimensions ordered by nested dissection [11]. If the mesh has k points on a side, then the graph is a k by k square grid, and we have $n = k^2$, $h = O(k)$, and $\eta(L) = O(k^2 \log k)$. The number of arithmetic operations in the Cholesky factorization is $O(k^3)$, in either the sequential or parallel algorithms. Router Cholesky's running time is $O(\rho k^3 \log k/p)$. If we define performance as the ratio of the number of operations in the sequential algorithm to parallel time, we find that the performance is $O(p/\log k)$ (taking ρ to be a constant independent of p or k; this is approximately correct for the Connection Machine, although theoretically ρ should grow at least with $\log p$). This analysis points out two weak points of Router Cholesky. First, the performance on the model problem drops with increasing problem size. (This depends on the problem, of course; for a three-dimensional model problem a similar analysis shows that performance is $O(p)$ regardless of problem size.) Second, and more seriously, the constant in the leading term of the complexity is proportional to the router time ρ, because every step uses general communication.

This analysis can be extended to any two-dimensional finite-element mesh with bounded-node degree, ordered by nested dissection [17]. The asymptotic analysis is the same but the values of the constants will be different.

3.4. Router Cholesky performance: Experiments. In order to validate the timing model and analysis, we experimented with Router Cholesky on a variety of sparse matrices. We present one example here in detail. The original matrix is 2,500 × 2,500 with 7,400 nonzeros (counting symmetric entries only once), representing a 50 × 50 five-point square mesh. It is preordered by Sparspak's automatic nested dissection heuristic [13], which gives orderings very similar to the ideal nested dissection ordering used in the analysis of the model problem above. The Cholesky factor has $\eta(L) =$ 48,608 nonzeros, an elimination tree of height $h = 144$, and takes 1,734,724 arithmetic operations to compute.

We ran this problem on CM-2 computers at the Xerox Palo Alto Research Center and the NASA Ames Research Center. The results quoted here are from $p =$ 8,192 processors, with floating-point coprocessors, of the machine at NASA. The VP ratio was therefore $V = \lceil \eta(L)/p \rceil = 8$. (Rounding up to a power of two has considerable cost here, since we use only 48,608 of the 65,536 virtual processors.) We observed a running time of 53 seconds, of which about 41 seconds was due to gathers and scatters. Substituting into the analysis above (using $\rho = 200$ since there were generally many collisions), we would predict router time $c_1 \rho \phi \mu h = 39$ seconds and other time $(c_2 \sigma + c_3) \phi \mu h = 1.5$ seconds. This is not a bad fit for router time; it is not clear why the remaining time is such a poor fit, but the expensive square root and the data movement involved in the pointer updates contribute to it, and it seems that I/O may have affected the measured 53 seconds.

The observation, in any case, is that router time completely dominates.

3.5. Remarks on Router Cholesky. Router Cholesky is too slow as it stands to be a cost-effective way to factor sparse matrices. Each stage does two gathers and a scatter with exactly the same communication pattern. More careful use of the router could probably speed it up by a factor of two to five. However, this would not be enough to make it practical; something more like a hundredfold improvement in router speed would be needed.

The one advantage of Router Cholesky is the extreme simplicity of its code. It is no more complicated than the numeric factorization routine of a conventional sequential sparse Cholesky package [13]. It is interesting to note that column-oriented sparse Cholesky codes on multiple instruction, multiple data (MIMD) message-passing multiprocessors [12], [14], [35] are more complex. They exploit MIMD capability to implement dynamic scheduling of the *cmod* and *cdiv* tasks. They allow arbitrary assignment of columns to processors and therefore are required to use indirect addressing of columns. Finally, they are written with low-level communication primitives, the explicit "send" and "receive."

Router Cholesky's simplicity comes dearly. Flexibility in scheduling allows MIMD implementations to gain a modest performance advantage over any possible SIMD implementation. More important, we employ $\eta(L)$ virtual processors, regardless of the number of physical processors. It is essential that these virtual processors not all sit idle, consuming physical processor time slices, when there is nothing for them to do. As implemented by the Paris instruction set, they do sit idle.

We described Router Cholesky as a left-initiated, left-looking algorithm. In a right-initiated algorithm, processor P_{ij} would perform the updates to l_{ij}. In a right-looking algorithm, updates would be applied as soon as the updating column of L was computed instead of immediately before the updated column of L was to be computed. Router Cholesky is thus one of four cousins. It is the only one of the four that maps operations to processors evenly; the other three alternatives require an inner sequential loop of some kind. All four versions require at least h router operations.

4. Grid Cholesky.

In this section we present a parallel supernodal, multifrontal Cholesky algorithm and its implementation on the CM. Multifrontal methods, introduced by Duff and Reid [7], compute a sparse Cholesky factorization by performing symmetric updates on a set of dense submatrices. We follow Liu [23] in referring to an algorithm that uses rank-1 updates as "multifrontal" and the block version that uses rank-k updates as "supernodal multifrontal." The idea of using block methods to operate on supernodes has been used in many different sparse factorization algorithms [1], [7]. Parallel supernodal or multifrontal algorithms have been used on MIMD message-passing and shared-memory machines [2], [6], [32].

The algorithm uses a two-dimensional VP set (which we call the "playing field") to partially factor, in parallel, a number of dense principal submatrices of the partially factored matrix. By working on the playing field, we may use the fast grid and scan mechanisms for all the necessary communication during the factorization of the dense submatrices. Only when we need to move these dense submatrices back and forth to the playing field do we need to use the router. In this way we drastically reduce the use of the router: for the model problem on a $k \times k$ grid, we reduce the number of uses from $h = 3k - 1$ to $2 \log_2 k - 1$. The playing field can also operate at a lower VP ratio, in general, because it does not need to store the entire factored matrix at once.

This method, like all multifrontal methods, is in essence an "out of core" method in that the Cholesky factor is kept in a data structure that is not referred to while doing arithmetic, all of which is done on dense submatrices. The novelty here is the factorization of many of these dense problems in parallel; the simultaneous exploitation of the parallelism available *within* each of the dense factorizations; the use of a two-dimensional grid of processors for these parallel dense factorizations; the use of the machine's router for parallel transfers from the matrix storage data structure; and the use of the combining scatter operations for parallel update of the matrix storage data structure.

4.1. The Grid Cholesky algorithm.

4.1.1. Block Jess and Kees reordering. First, we describe an equivalent reordering of the chordal graph $G^* = G^*(A)$ that we call the *block Jess/Kees ordering.* Block Jess/Kees is a perfect elimination ordering that has two properties that make it the best equivalent reordering for our purposes: It eliminates vertices with identical monotone neighborhoods consecutively, and it produces a clique tree of minimum height.

Our reordering eliminates all the simplicial vertices of G^* simultaneously as one major step. In the process, it partitions all the vertices of G^* into supernodes. Each of these supernodes is a clique in G^*, and is a simplicial clique when its component vertices are about to be eliminated. Each vertex is labeled with the *stage*, or major step number, at which it is eliminated. In more detail, the reordering algorithm is as follows.

```
procedure Reorder (graph $G^*$)
    $active\_stage \leftarrow -1$;
    while $G^*$ is not empty do
        $active\_stage \leftarrow active\_stage + 1$;
        Number all the simplicial vertices in $G^*$, assigning
            consecutive numbers within each supernode;
        $stage(v) \leftarrow active\_stage$ for all simplicial vertices $v$;
        Remove all the simplicial vertices from $G^*$ od;
    $\hbar \leftarrow active\_stage$
end Reorder;
```

The cliques are the nodes of a clique tree whose height is $\hbar$, one less than the number of major elimination steps. The parent of a given clique is the lowest-stage clique adjacent to the given clique.

The name "block Jess/Kees" indicates a relationship with an algorithm due to Jess and Kees [21] that finds an equivalent reordering for a chordal graph so as to minimize the height of its elimination tree. The original (or "point") Jess/Kees ordering eliminates just one vertex from each simplicial clique at each major step. (This is a maximum-size independent set of simplicial vertices.) Each step of point Jess/Kees produces one level of an elimination tree, from the leaves up. The resulting elimination tree has minimum height over all perfect elimination orders on G^*. Our block Jess/Kees eliminates all the simplicial vertices at each major step, producing a clique tree one level at a time from the leaves up. This ordering may not minimize the height of the elimination tree. However, as Blair and Peyton [4] have shown, it does produce a clique tree of minimum height over all perfect elimination orders on G^*.

Every vertex is included in exactly one supernode. We number the supernodes as $\{S_1, \cdots, S_m\}$ in such a way that if $i < j$ then the vertices in S_i have lower numbers than the vertices in S_j. The *stage* at which a supernode S is eliminated is the iteration of the *while* loop at which its vertices are numbered and eliminated; thus for all $v \in S$, $stage(v) = stage(S)$ is the height of node S in the clique tree.

4.1.2. Parallel supernodal multifrontal elimination. Let C be a supernode. It is immediate that $K = \mathrm{adj}(C) \cup C$ is a clique, and that it is maximal. Our factorization algorithm works by forming the principal submatrices of A corresponding to vertices in the maximal cliques generated by supernodes. Let $\gamma = |C|$ and $\delta = |\,\mathrm{adj}(C)|$. Write $A(K, K)$ for the principal submatrix of order $|K| = \gamma + \delta$ consisting of elements $a_{i,j}$ with $i, j \in K$. It is natural to partition the principal submatrix $A(K, K)$ of A as

$$A(K,K) = \begin{pmatrix} X & E \\ E^T & Y \end{pmatrix},$$

where $X = A(C, C)$ is $\gamma \times \gamma$, $Y = A(\mathrm{adj}(C), \mathrm{adj}(C))$ is $\delta \times \delta$, and E is $\gamma \times \delta$.

In the terminology of the previous section, Grid Cholesky is a "block right-looking" algorithm. The details are as follows.

```
procedure Grid Cholesky(matrix $A$)
    for active_stage $\leftarrow 0$ to $\hbar$ do
        for each supernode $C$ with stage(C) = active_stage pardo
            Move $A(K, K)$ to the playing field,
                where $K = C \cup \mathrm{adj}(C)$;
            Set $Y$ to zero on the playing field;
            Perform $\gamma = |C|$ steps of parallel Gaussian elimination
                to compute the Cholesky factor $L$ of $X$,
                the updated submatrix $E' = L^{-1}E$,
                and the Schur complement $Y' = -E^T X^{-1} E$,
                where $X$, $E$, and $Y$ partition $A(K, K)$ as above;
            $A(C, C) \leftarrow L$;
            $A(\mathrm{adj}(C), C) \leftarrow E'^T$;
            $A(\mathrm{adj}(C), \mathrm{adj}(C)) \leftarrow A(\mathrm{adj}(C), \mathrm{adj}(C)) + Y'$ od
    od
end Grid Cholesky;
```

4.2. Multiple dense partial factorization. In order to make this approach useful, we need a fast dense matrix factorization on two-dimensional VP sets. To that end, we discuss an implementation of *LU* decomposition without pivoting. (We use *LU* instead of Cholesky here because we can see no efficient way to exploit symmetry with a two-dimensional machine; moreover, *LU* avoids the square root at each step and so is a bit faster.) We analyzed and implemented two methods: a systolic algorithm that uses only nearest-neighbor communication on the grid, and a rank-1 update algorithm that uses row-and-column broadcast. With either of these methods, all the submatrices $A(K, K)$ corresponding to supernodes at a given stage are distributed about the two-dimensional playing field simultaneously (each as a separate "baseball diamond"), and the partial factorization is applied to all the submatrices at once.

We describe the rank-1 algorithm in terms of its effect on a single submatrix $A(K, K)$, with a supernode of size γ and a Schur complement of size δ. A single rank-1 update consists of a division to compute the reciprocal of the diagonal element, a scan down the columns to copy the pivot row to the remaining rows, a parallel multiplication to compute the multiplier for each row, another scan to copy the multiplier across each row, and finally a parallel multiply and subtract to apply the update. The number of rank-1 updates is γ, the size of the supernode.

An entire stage of partial factorizations is performed at once, using segmented scans to keep each factorization within its own "diamond." The number of steps on the playing field at stage s is γ_s, the maximum value of γ over all supernodes at stage s. Then a stage of rank-1 partial factorization takes time $(c_3\sigma + c_4)\phi\mu$. Here c_3 is $2\gamma_s$, and c_4 is proportional to γ_s as well.

The relative cost of the various parts of the rank-1 update code are summarized below for a complete factorization (that is, one in which $\delta = 0$). The bookkeeping includes nearest-neighbor communication to move three one-bit tokens that control which processors perform reciprocals, multiplications, and so on at each step.

Scans (row and column broadcast): 79.7%.

News (moving the tokens): 5.5%.

Multiply (computing multipliers): 2.7%.

Divide (reciprocal of pivot element): 7.1%.

Multiply-subtract (Gaussian elimination): 4.8%.

Unlike the rank-1 implementation, the systolic implementation never uses a scan; all communication is between grid neighbors. Thus its communication terms are proportional to ν rather than σ. This advantage is more than offset by the fact that $3\gamma_s + 2\delta_s$ sequential iterations are necessary, while the rank-1 method only needs γ_s.

4.2.1. Remarks on dense partial factorization. Theoretically, systolic factorization should be asymptotically more efficient as machine size and problem size grow without bound, because scans must become more expensive as the machine grows. Realistically, however, the CM happens to have $\sigma \approx 6\nu$; for a full factorization ($\delta = 0$) a sixfold decrease in communication time per step more than balances the threefold increase in number of steps, and so the systolic method *is* somewhat faster. But for a partial factorization, the rank-1 algorithm is the clear winner. For example, for the two-dimensional model problem the average Schur complement size δ_s is about $4\gamma_s$, so the rank-1 code has an 11-to-1 advantage in number of steps. This more than makes up for the fact that scan is much slower than grid communication.

It is interesting to note that the only arithmetic that matters in a sequential algorithm, the multiply-subtract, accounts for only 1/21 of the total time in the rank-1 parallel algorithm. Moreover, only 1/3 of the multiply-subtract operations actually do useful work, since the active part of the matrix occupies a smaller and smaller subset of the playing field as the computation progresses. This gives the code an overall efficiency of one part in 63 for *LU* and one part in 126 for Cholesky. We have found this to be typical in *lisp codes for matrix operations, especially with high VP ratios. The reasons are these: scan is slow relative to arithmetic; the divide and multiply operations occur on very sparse VP sets; and the VP ratio remains constant as the active part of the matrix gets smaller.

Significant performance improvements could come from several possible sources. A more efficient implementation of virtual processors could improve performance substantially: the loop over V virtual processors should be restricted to those virtual processors that are active. Sometimes a determination that this is going to happen can readily be made at compile time. As an alternative, we could have rewritten the code: the VP set could shrink as the matrix shrinks, and the divides and the multiplies could be performed in a sparser VP set.

The scans could be sped up considerably within the hypercube connection structure of the CM. Ho and Johnsson [19] have developed an ingenious algorithm that takes $O(b/d + d)$ time to broadcast b bits in a d-dimensional hypercube, in contrast to the scan, which takes $O(b+d)$. The copy scans could also be implemented so that each physical processor gets only one copy of the data rather than V copies.

More efficient use of the low-level floating-point architecture of the CM-2 is possible. The Paris instruction set hides the fact that every 32 physical processors share one vector floating-point arithmetic chip. Performing 32 floating-point operations implies moving 32 numbers in bit-serial fashion into a transposer chip, then moving them in parallel into the vector chip, then reversing the process to store the result. While this mode of operation conforms to the one-processor-per-data-element programming model, it wastes time and memory bandwidth when only a few processors are actually active (such as computing multipliers or diagonal reciprocals in LU), since 32 memory cycles are required just to access one floating-point number.

This mode also requires intermediate results to be stored back to main memory, precluding the use of block algorithms [3] that could store intermediate results in the registers in the transposer. Thus the computation rate is limited by the bandwidth between the transposer and memory (about 3.5 gigaflops for a 64K processor CM) instead of by the operation rate of the vector chip (about 27 gigaflops total).

A more efficient dense matrix factorization can be achieved by treating each 32-processor-plus-transposer-and-vector-chip unit as a single processor, and representing 32-bit floating-point numbers "slicewise," with one bit per physical processor, so that they do not need to be moved bit-serially into the arithmetic unit. (Viewed this way, we were working with just 256 real processors!) Also, if virtualization is handled efficiently, we need only keep 256 processors busy. At the time this work was done (late in 1988) the tools for programming on this level were not easily usable. While CM Fortran allows this model, it does not allow scans and scatter with combining, so we still (early 1991) cannot use it.

4.3. CM implementation of Grid Cholesky. Grid Cholesky uses two VP sets with different geometries: the *matrix storage* stores the nonzero elements of A and L (doing almost no computation), and the *playing field* is where the dense partial factorizations are done. The top-level factorization procedure is just a loop that moves the active submatrices to the playing field, factors them, and moves updates back to the main matrix.

4.3.1. Matrix storage. The matrix storage VP set is a one-dimensional array of virtual processors that stores all of A and L in essentially the same form as Router Cholesky. Each of the following pvars has one element for each nonzero in L:

`l_value`: Elements of L, initially those of A.
`grid_i`: The playing field row in which this element sits.
`grid_j`: The playing field column in which this element sits.
`active_stage`: The stage at which j occurs in a supernode.
`updates`: Working storage for sum of incoming updates.

We use a scatter to move the active columns from matrix storage to the playing field. The supernodes C are disjoint, but their neighboring sets adj(C) may overlap; that is, more than one Y_C may be computing updates to the same element of L at the same stage. Therefore, we use scatter-with-add to move the partially factored matrix from the playing field back to matrix storage.

4.3.2. The playing field. The second VP set, called the playing field, is the two-dimensional grid on which the supernodes are factored. In our implementation it is large enough to hold all the principal submatrices for all maximal cliques at any stage, although it could actually use different VP ratios at different stages for more efficiency. Its size is determined as part of the symbolic factorization and reordering. The pvars used in this VP set are

- `dense_a`: The playing field for matrix elements.
- `update_dest`: The matrix storage location (processor) that holds this matrix element; an integer pvar array indexed by stage.

Some boolean flags are also used to coordinate the simultaneous partial factorization of all the maximal cliques.

The processors of the playing field compute LU factorizations by simultaneously doing rank-1 updates (see §4.2) of all the dense submatrices stored there, using segmented scans to distribute the pivot rows and columns within all submatrices at the same time. The number of rank-1 update steps is the size of the largest supernode at the current stage. The submatrices may be different sizes; each matrix does only as many rank-1 updates as the size of its supernode.

In order to use this procedure, we need to find a placement of all the submatrices $A(K, K)$ for all the maximal cliques K at every stage. This is a two-dimensional bin packing problem. In order to minimize CM computation time, we want to pack these square arrays onto the smallest possible rectangular playing field (whose borders must be powers of two). Optimal two-dimensional bin packing is, in general, an NP-hard problem, though various efficient heuristics exist [9]. Our experiments use a simple "first fit by levels" heuristic. This layout is done during the sequential symbolic factorization, before the numeric factorization is begun.

4.4. Grid Cholesky performance: Theory. We separate the running time for Grid Cholesky into time in the matrix storage VP set and time on the playing field. The former includes all the router traffic, and essentially nothing else. (There is one addition per stage to add the accumulated updates to the matrix.) There is a fixed number of router uses per stage, so the matrix storage time is $T_{MS} = c_5 \rho \phi \mu_{MS} \hbar$ for some constant c_5. The subscript MS indicates that the value of μ is taken in the matrix storage VP set, whose VP ratio set is $V_{MS} = \lceil \eta(L)/p \rceil$. In the current implementation $c_5 = 4$, since two scatters are used to move the dense matrices to the playing field at the beginning of a stage, and then two scatters are used to move the completely computed columns and the Schur complements back to matrix storage.

We express the playing field time as a sum over levels. At each level s the number of rank-1 updates is the size of the largest supernode at that level, which is γ_s. According to the analysis in §4.2,

$$T_{PF} = (c_6 \sigma + c_7) \sum_{s=0}^{\hbar} \gamma_s \phi \mu_s,$$

where c_6 and c_7 are constants (in fact, $c_6 = 2$), and the subscript s indicates that the value of μ is taken in the playing field VP set at stage s. The VP ratio in this VP set could be approximately the ratio of the total size of the dense submatrices at stage s to the number of processors, changing at each stage as the number and size of the maximal cliques vary. However, in our implementation, it is simply fixed at the maximum of this value over all stages.

Again, to get a feeling for this analysis, let us consider the model problem, a five-point finite-difference mesh on a $k \times k$ grid ordered by nested dissection. For this problem $n = k^2$, $\hbar = O(\log k)$, and $\eta(L) = O(k^2 \log k)$. The factorization requires $O(k^3)$ arithmetic operations. Table 2 summarizes the number and sizes of the cliques that occur at each stage. The columns in the table are as follows:

TABLE 2

Subproblem counts and playing field size for the model problem.

Stage s	$R(s)$	γ_s	$\gamma_s + \delta_s$	$\sum(\gamma_C + \delta_C)^2$
$\hbar$	1	k	k	k^2
$\hbar - 1$	2	$k/2$	$3k/2$	$4.5k^2$
$\hbar - 2$	4	$k/2$	$3k/2$	$9k^2$
$\hbar - 3$	8	$k/4$	$3k/2$	$18k^2$
$\hbar - 4$	16	$k/4$	$5k/4$	$25k^2$
$\hbar - 5$	32	$k/8$	$7k/8$	$24.5k^2$
$\hbar - 6$	64	$k/8$	$5k/8$	$25k^2$
$\hbar - 7$	128	$k/16$	$7k/16$	$24.5k^2$
$\hbar - 2r$	2^{2r}	$k/2^r$	$5k/2^r$	$25k^2$
$\hbar - 2r - 1$	2^{2r+1}	$k/2^{r+1}$	$7k/2^{r+1}$	$24.5k^2$

$R(s)$: Number of supernodes at stage s.
γ_s: Size of largest supernode at stage s.
$\gamma_s + \delta_s$: Size of largest maximal clique $C \cup \operatorname{adj}(C)$ at stage s.
$\sum(\gamma_C + \delta_C)^2$: Total area of all dense submatrices $A(K, K)$ at stage s.

The VP ratio in matrix storage for the model problem is $O(\eta(L))/p = O(k^2 \log k/p)$, so the matrix storage time is $O(k^2 \log^2 k/p)$. Our pilot implementation uses the same size playing field at every stage. According to Table 2, a playing field of size $\lceil 25k^2 \rceil$ suffices if the problems can be packed in without too much loss. The VP ratio is $O(k^2/p)$. The sum over all stages of γ_s is $O(k)$ (in fact, it is $3k + O(1)$), so the playing field time is $O(k^3/p)$. In summary, the total running time of Grid Cholesky for the model problem is

$$O\left(\frac{k^2 \log^2 k}{p}\rho + \frac{k^3}{p}\sigma\right).$$

Two things are notable about this. First, the performance, or ratio of sequential arithmetic operations to time, is $O(p)$; the $\log k$ inefficiency of Router Cholesky has vanished. This is because the playing field, where the bulk of the computation is done, has a lower VP ratio than the matrix storage structure. Second, and much more important in practice, the router speed ρ appears only in the second-order term. This is because the playing field computations are done on dense matrices with more efficient grid communication. This means that the router time becomes less important as the problem size grows, whether or not the number of processors grows with the problem size.

One way of looking at this analysis is to think of increasing both problem size and machine size so that the VP ratio remains constant. Then the model problem requires $O(k)$ total parallel operations, but only $O(\log k)$ router operations. The analysis of the model problem carries through (with different constant factors) for any two-dimensional finite-element problem ordered by nested dissection; a similar analysis carries through for any three-dimensional finite-element problem. Thus Grid Cholesky is a "scalable" parallel algorithm.

4.5. Grid Cholesky performance: Experiments. Here we present experimental results from a fairly small model problem, the matrix arising from the five-point discretization of the Laplacian on a square 63 × 63 mesh, ordered by nested dissection. This matrix has n = 3,969 columns and 11,781 nonzeros (counting symmetric entries only once). The Cholesky factor has $\eta(L)$ = 86,408 nonzeros, a clique tree with $\hbar = 11$ stages of supernodes, and takes 3,589,202 arithmetic operations to compute.

The matrix storage VP set requires 128K VPs. The fixed-size playing field requires 256×512 VPs. The results quoted here are from 8,192 processors, with floating-point coprocessors, of the machine at NASA. Both VP sets therefore had a VP ratio of 16. (A larger problem would need a higher VP ratio in the matrix storage than in the playing field.)

We observed a running time of 6.13 seconds. Of this, 4.09 seconds was playing field time (3.12 for the scans, 0.15 for nearest-neighbor moves of one-bit tokens, and 0.82 for local computation). The other 2.04 seconds was matrix storage time, consisting mostly of the four scatters at each stage. Our analytic model predicts playing field time to be about $3k \cdot (2\sigma + 4)\phi\mu_{PF}$. This comes to 4.0 seconds, which is in agreement with the experiment. The model predicts matrix storage time of about $\hbar \cdot 4\rho\phi\mu_{MS}$. This comes to between 1.5 and 4.7 seconds, depending on which value we choose for ρ. In fact three-fourths of the router operations are scatters with no collisions, and the other one-fourth are scatter-with-add, typically with two to four collisions. The fit to the model is therefore quite close.

4.6. Remarks on Grid Cholesky. On this small example, Grid Cholesky is about 20 times as fast as Router Cholesky. It is, however, only running at 0.586 megaflops on 8K processors, which would scale to 4.68 megaflops on a full 64K machine. A larger problem would run somewhat faster, but it is clear that making Grid Cholesky attractive will require major improvements. Some of these were sketched in §4.2.1.

A first question is whether Grid Cholesky is a router-bound code, as Router Cholesky is. For the small sample problem, the relative times for router and nonrouter computations are as follows:

Moving data to the playing field: 12%.
Factoring on the playing field: 67%.
Moving Schur complements back to matrix storage: 21%.

Evidently, the Grid Cholesky code is not router bound for this problem. For larger (or structurally denser) problems this situation gets better still: for a machine of fixed size, the time spent using the router grows like $O(\log^2 k)$, while the time on the playing field grows like $O(k^3)$ for a $k \times k$ grid, as we showed above. If we solved the same problem on a full-sized 64K processor machine, the relative times would presumably be the same as above; but if we solved a problem 8 times as large, the operation count would increase by a factor of about 22 while the number of stages, and router operations, would increase only by a factor of about 1.3.

Next, we ask whether our use of the playing field is efficient or not. The number of parallel elimination steps on the playing field is given by

$$\sum_{s=1}^{\hbar} \gamma_s,$$

which is 177 for the example. On the playing field of 131,072 processors, this allows us to do 22.8×10^6 multiply-adds or 45.6×10^6 flops. The number of flops actually done is 3.69×10^6, so the processor utilization is 7.9 percent. There are several reasons for this loss of efficiency.

- The algorithm does both halves of the symmetric dense subproblems (factor of 2).

TABLE 3
Factors affecting efficiency of Grid Cholesky.

Impediment removed	Time	Mflops	Speedup
None	6.13	4.8	
Virtualization	0.60	49.	10.
Slow router	0.33	89.	1.8
Slow scans	0.065	450.	5.0
SIMD	0.021	1,400.	3.1
Paris	0.0042	7,000.	5.0

- The implementation uses the same playing field size at every level (factor of about 4/3).
- The architecture forces the dimensions of the playing field to be powers of two (factor of about 4/3).
- The playing field is not fully covered by active processors initially, and as the dense factorization progresses, processors in the supernodes fall idle (factor of about 7/2).

These effects account for a factor of roughly 12.4, which is consistent with our measured result; $1/.079 = 12.6$. In other words, every step must use all 131,072 virtual processors, but on average, we only have work for about 10,500 of them. Thus the Paris virtualization method costs us a factor of roughly $(131{,}072/10{,}500) = 12.5$.

A similar analysis shows that virtualization slows the use of the router by a factor of 7.6.

We summarize the reasons that our achieved performance is so far below the CM's peak as follows:

- Virtualization. We used 2^{17} virtual processors. On average, we make use of 10,500 on the playing field and 5,600 in matrix storage. In actuality, there are 256 physical (floating-point) processors in the machine we used.
- The slow router.
- Communication costs for scans on the playing field, using the built-in suboptimal algorithm.
- The SIMD restriction. This causes us to have to wait for the divides and multiplies. (Since there are very few active virtual processors during these tasks, most of this effect could also be removed by efficient virtualization.)
- The Paris instruction set, which makes reuse of data in registers impossible, thus exposing the low memory bandwidth of the CM.

Table 3 gives an upper bound on the improvement that removal of each of these impediments to performance would provide. In each case, we hypothesize an "ideal" machine in which the corresponding cost is completely removed. Thus for example, the statistics for the third row of the table are for a marvelous machine in which a router operation takes no time whatever.

Clearly, good efficiency is possible, even on an SIMD machine with a router. The chief problems are the Paris virtualization method; the lack of a fast broadcast in the grid; and the memory-to-memory instruction set. All of these are peculiarities of Paris on the CM-2; none is fundamental to SIMD, data parallel computing.

5. Conclusions. We have compared two highly parallel general-purpose sparse Cholesky factorization algorithms, implemented for a data parallel computer. The first, Router Cholesky, is concise and elegant and takes advantage of the parallelism present in the elimination tree, but because it pays little attention to the cost of communication it is not practical.

We therefore developed a parallel algorithm, Grid Cholesky, that does most of its work with grid communication on dense submatrices. Analysis shows that the requirement for expensive general-purpose communication grows only logarithmically with problem size; even for modest problems the code is not limited by the CM router. Experiment shows that Grid Cholesky is about 20 times as fast as Router Cholesky for a moderately small sample problem.

Our pilot implementation of Grid Cholesky is not fast enough to make the Connection Machine cost effective for solving generally structured sparse-matrix problems. We believe, however, that our experiments and analysis lead to the conclusion that a parallel supernodal, multifrontal algorithm can be made to perform efficiently on a highly parallel SIMD machine. We showed in detail that Grid Cholesky could run two to three orders of magnitude faster with improvements in the instruction set of the Connection Machine. These suggestions for improvement make the chasm separating the five-megaflop performance of our pilot implementation from the 27-gigaflop theoretical peak performance of a 64K processor CM yawn somewhat less dauntingly.

Most of these improvements are below the level of the Paris virtual processor abstraction, which is to say, below the level of the assembly-language architecture of the machine. Although TMC has recently released a low-level language called CMIS in which a user can program below the virtual-processor level, we believe that ultimately most of these optimizations should be applied by high-level language compilers. Future compilers for highly parallel machines, while they will support the data parallel virtual processor abstraction at the user's level, will generate code at a level below that abstraction.

Although Grid Cholesky is more complicated than Router Cholesky, we are still able to use the data parallel programming paradigm to express it in a straightforward way. The high-level scan and scatter-with-add communication primitives substantially simplified the programming. The simplicity of our codes speaks well for this data parallel programming model.

In summary, even though our pilot implementation is not fast, we are nonetheless encouraged about the Grid Cholesky algorithm, and about the potential of data parallel programming and highly parallel architectures for solving unstructured problems. We believe that future generations of highly parallel machines may allow efficient parallel programs for complex tasks to be written nearly as easily as sequential programs. To get to that point, there will have to be improvements in compilers, instruction sets, and router technology. Virtualization will have to be implemented without sacrificing efficiency.

We mention four avenues for further research. The first is scheduling the dense partial factorizations efficiently. The tree of supernodes identifies a precedence relationship among the various partial factorizations. Our simple approach of scheduling these one level at a time onto a fixed-size playing field is not the only possible one. There is, in general, no need to perform all the partial factorizations at a single level simultaneously. It should be possible to use more sophisticated heuristics to schedule these factorizations onto a playing field of varying VP ratio, or even (for the Connection Machine) onto a playing field considered as a mesh of individual vector floating-point chips. Some theoretical work has been done on scheduling "rectangular" tasks onto a square grid of processors efficiently [8]. Kratzer [22] has experimented with sparse QR factorization on the Connection Machine using a dataflow model to schedule individual row operations.

The second avenue is improving the time spent in the matrix storage VP set. Of course, as problems get larger, this time becomes a smaller fraction of the total. At present, matrix storage time is not very significant even for a small problem, but it will become more so as the playing field time is improved.

Third, we mention the possibility of an out-of-main-memory version of Grid Cholesky for very large problems. Here the clique tree would be used to schedule transfers of data between the high-speed parallel disk array connected to the CM and the processors themselves.

Fourth and finally, we mention the possibility of performing the combinatorial preliminaries to the numerical factorization in parallel. Our pilot implementation uses a sequentially generated ordering, symbolic factorization, and clique tree. We are currently designing data parallel algorithms to do these three steps [18].

We conclude by extracting one last moral from Grid Cholesky. We find it interesting and encouraging that the key idea of the algorithm, namely partitioning the matrix into dense submatrices in a systematic way, has also been used to make sparse Cholesky factorization more efficient on vector supercomputers [32], and even on workstations [29]. In the former case, the dense submatrices vectorize efficiently; in the latter, the dense submatrices are carefully blocked to minimize traffic between cache memory and main memory. We expect that more experience will show that many techniques for attaining efficiency on sequential machines with hierarchical storage will turn out to be useful for highly parallel machines.

REFERENCES

[1] C. Ashcraft, R. Grimes, J. Lewis, B. Peyton, and H. Simon, *Recent progress in sparse matrix methods for large linear systems*, Internat. J. Supercomput. Appl., (1987), pp. 10–30.

[2] C. C. Ashcraft, *The domain/segment partition for the factorization of sparse symmetric positive definite matrices*, Tech. Report ECA–TR–148, Boeing Computer Services, Engineering, Computing and Analysis Division, Seattle, WA, 1990.

[3] C. H. Bischof and J. J. Dongarra, *A project for developing a linear algebra library for high-performance computers*, Tech. Report MCS–P105–0989, Argonne National Laboratory, Argonne, IL, 1989.

[4] J. R. S. Blair and B. W. Peyton, *On finding minimum-diameter clique trees*, Tech. Report ORNL/TM–11850, Oak Ridge National Laboratory, Oak Ridge, TN, 1991.

[5] M. Dixon and J. de Kleer, *Massively parallel assumption-based truth maintenance*, in Proc. Nat. Conf. Artificial Intelligence, ACM, New York, 1988, pp. 199–204.

[6] I. S. Duff, *Multiprocessing a sparse matrix code on the Alliant FX/8*, Tech. Report CSS 210, Computer Science and Systems Division, AERE Harwell, Oxfordshire, U.K., 1988.

[7] I. S. Duff and J. K. Reid, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Trans. Math. Software, 9 (1983), pp. 302–325.

[8] A. Feldmann, J. Sgall, and S.-H. Teng, *Dynamic scheduling on parallel machines*, in 31st Annual Symposium on Foundations of Computer Science, San Juan, October 1991, pp. 111–120.

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, San Francisco, CA, 1979.

[10] F. Gavril, *Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph*, SIAM J. Comput., 1 (1972), pp. 180–187.

[11] A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.

[12] A. George, M. T. Heath, J. Liu, and E. Ng, *Sparse Cholesky factorization on a local-memory multiprocessor*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 327–340.

[13] A. George and J. W. H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[14] A. GEORGE AND E. NG, *On the complexity of sparse* QR *and* LU *factorization of finite-element matrices*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 849–861.

[15] J. A. GEORGE AND D. MCINTYRE, *On the application of the minimum degree algorithm to finite element systems*, SIAM J. Numer. Anal., 15 (1978), pp. 90–112.

[16] J. R. GILBERT, *Some nested dissection order is nearly optimal*, Inform. Process. Lett., 26 (1988), pp. 325–328.

[17] J. R. GILBERT AND H. HAFSTEINSSON, *Parallel solution of sparse linear systems*, in SWAT 88: Proceedings of the First Scandinavian Workshop on Algorithm Theory, Lecture Notes in Computer Science 318, Springer-Verlag, Berlin, 1988, pp. 145–153.

[18] J. R. GILBERT, C. LEWIS, AND R. SCHREIBER, *Parallel preordering for sparse matrix factorization*, in preparation.

[19] C.-T. HO AND S. L. JOHNSSON, *Spanning balanced trees in Boolean cubes*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 607–630.

[20] J. J. HOPFIELD, *Neural networks and physical systems with emergent collective computational abilities*, Proc. Nat. Acad. Sci., 79 (1982), pp. 2554–2558.

[21] J. A. G. JESS AND H. G. M. KEES, *A data structure for parallel L/U decomposition*, IEEE Trans. Comput., C-31 (1982), pp. 231–239.

[22] S. G. KRATZER, *Massively parallel sparse matrix computations*, Tech. Report SRC–TR–90–008, Supercomputer Research Center, Bowie, MD, 1990.

[23] J. W. H. LIU, *The multifrontal method for sparse matrix solution: Theory and practice*, Tech. Report CS–90–04, Computer Science Department, York University, York, England, 1990.

[24] J. W. H. LIU, *The role of elimination trees in sparse factorization*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 134–172.

[25] J. NAOR, M. NAOR, AND A. J. SCHÄFFER, *Fast parallel algorithms for chordal graphs*, SIAM J. Comput., 18 (1989), pp. 327–349.

[26] B. W. PEYTON, *Some Applications of Clique Trees to the Solution of Sparse Linear Systems*, Ph.D. thesis, Clemson University, Clemson, SC, 1986.

[27] D. J. ROSE, *A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations*, in Graph Theory and Computing, R. C. Read, ed., 1972, pp. 183–217.

[28] D. J. ROSE, R. E. TARJAN, AND G. S. LUEKER, *Algorithmic aspects of vertex elimination on graphs*, SIAM J. Comput., 5 (1976), pp. 266–283.

[29] E. ROTHBERG AND A. GUPTA, *Fast sparse matrix factorization on modern workstations*, Tech. Report STAN–CS–89–1286, Stanford University, Stanford, CA, 1989.

[30] R. SCHREIBER, *A new implementation of sparse Gaussian elimination*, ACM Trans. Math. Software, 8 (1982), pp. 256–276.

[31] R. SCHREIBER, *An assessment of the connection machine*, in Scientific Applications of the Connection Machine, H. Simon, ed., World Scientific, Singapore, 1991.

[32] H. SIMON, P. VU, AND C. YANG, *Performance of a supernodal general sparse solver on the Cray Y-MP*, Tech. Report SCA–TR–117, Boeing Computer Services, Seattle, WA, 1989.

[33] B. SPEELPENNING, *The generalized element method*, Tech. Report UIUCDCS–R–78–946, University of Illinois, Urbana, IL, 1978.

[34] THINKING MACHINES CORPORATION, *Paris reference manual, version* 5.0, Cambridge, MA, 1988.

[35] E. ZMIJEWSKI, *Sparse Cholesky Factorization on a Multiprocessor*, Ph.D. thesis, Cornell University, Ithaca, NY, 1987.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 5, pp. 1173–1193, September 1992

008

PARALLEL BLOCK-PARTITIONING OF TRUNCATED NEWTON FOR NONLINEAR NETWORK OPTIMIZATION*

STAVROS A. ZENIOS† AND MUSTAFA Ç. PINAR‡

Abstract. Using the primal truncated Newton algorithm for the solution of nonlinear network optimization problems gives rise to very large and sparse systems of linear equations. These systems are solved iteratively with conjugate gradient methods. Using the structural characteristics of the network basis the system of equations is partitioned into independent blocks. Each can be solved in a fraction of the time required for the original equations and can also be solved in parallel.

Partitioning schemes are developed for both pure and generalized network problems. Empirical results using problems with up to 15,000 nodes and 37,588 arcs demonstrate the efficiency of the block-partitioning techniques, both for serial and parallel computing. Details of the parallel implementation on a shared-memory multiprocessor, the Alliant FX/4, are given together with computational results on a CRAY X–MP vector supercomputer.

Key words. nonlinear programming, network optimization, truncated Newton, parallel computing

AMS(MOS) subject classifications. 90B10, 90C35, 90C06, 90C30

1. Introduction. In this paper we consider linearly constrained nonlinear network (NLNW) programs of the form

$$\min_{x\in\Re^n} F(x) \tag{1}$$

$$\text{s.t. } Ax = b \tag{2}$$

$$l \le x \le u \tag{3}$$

where $F : \Re^n \longmapsto \Re$ is convex and twice continuously differentiable; A is an $m \times n$ node–arc incidence matrix; b, l and $u \in \Re^n$ are given vectors; and $x \in \Re^n$ is the vector of decision variables. The node–arc incidence matrix A specifies conservation of flow constraints (2) on some network $G = (N, E)$ with $|N| = m$ and $|E| = n$. It can be used to represent pure networks, in which case each column has two nonzero entries: a "+1" and a "−1." Generalized networks are also represented by matrices with two nonzero entries in each column: a "+1" and a real number that represents the arc multiplier. The structure of the constraint matrices of network problems is described in [13] and [17].

Nonlinear network problems appear in several applications in operations research, transportation, economics, finance, engineering design, and so on. These problems are usually characterized by their very large size. Several theoretical and numerical studies have developed algorithms and software for NLNW problems. As a result of their form, NLNW problems are some of the largest nonlinear optimization problems solved in practice today. A recent survey of models and methods for network optimization is given in [8].

* Received by the editors November 8, 1989; accepted for publication (in revised form) July 1, 1991.

† Decision Sciences Department, University of Pennsylvania, Philadelphia, Pennsylvania 19104. The research of this author was funded in part by National Science Foundation grant CCR-8811135 and Air Force Office of Scientific Research grant 91-0168.

‡ Department of Systems Engineering, University of Pennsylvania, Philadelphia, Pennsylvania 19104. The research of this author was funded in part by National Science Foundation grant ECS-8718971.

One of the most efficient algorithms for solving large instances of NLNW programs is the primal truncated Newton algorithm (PTN) of Dembo and Steihaug [9], implemented within the active set framework of Murtagh and Saunders [15]. In this paper we develop techniques for partitioning Newton's equations, which comprise the primary computational step of PTN, into blocks of equations of reduced size. These blocks of equations can be solved in a fraction of the time required by the original system. (For example, solving k systems of size n/k requires $O(1/k^2)$ fewer operations than solving the original system of size n.) Furthermore, the blocks can be solved in parallel both on message-passing and shared-memory architectures. The block-partitioning techniques exploit the special structure of the basis of network problems. Hence they can be executed efficiently even for very large problems. The partitioning schemes for pure networks formalize and generalize the procedures designed by Rosenthal [18] and Escudero [10], [11] for the partitioning of replicated pure network problems. The technique used for partitioning generalized networks extends the scheme proposed in Clark and Meyer [4] for the solution of linear generalized networks to nonlinear problems. Some related work on the partitioning of Newton's algorithm for unconstrained optimization is given in [16].

This paper makes three contributions. First, it develops the block-partitioning methods based on the structure of the network basis. Second, it evaluates the efficacy of the block-partitioned algorithm for solving large-scale problems on both serial and parallel computers. Third, it compares the efficiency of these methods with more standard partitioning techniques that do not exploit the basis structure. As a by-product, this study compares two alternative methods for parallel computing within PTN: the block-partitioning developed here and the implementation of Zenios and Mulvey [20], which emphasizes parallel execution of the linear algebra operations within PTN.

Our notation is as follows: B^T denotes the transpose of matrix B; $B_{.t}$ and $B_{t.}$ denote the tth column and row, respectively, of B. With the tth variable x_t we associate the tuple (i,j) where i is the row with "+1" in the tth column of A, and j is the row with the arbitrary real value in the tth column of A. In network terminology, i and j are called the incident nodes of arc (i,j). We use $t \sim (i,j)$ to indicate this association.

The remainder of this paper is organized as follows: §2 reviews concepts from PTN and active set methods that are relevant to our work. Section 3 reveals the relation between the structure of Newton's equations and the basis of network models and develops the block-partitioning schemes. Section 4 discusses implementation issues and reports results from numerical experimentation on serial computers, a shared-memory multiprocessor, the Alliant FX/4, and a vector supercomputer, the CRAY X-MP/48. Concluding remarks are given in §5.

2. The truncated Newton algorithm and active sets.

The primal truncated Newton algorithm is a feasible direction method within an active set framework. We describe the algorithm in two steps: First, we give a model Newton's algorithm for unconstrained optimization problems. Second, we discuss the active set method, which reduces a constrained optimization problem into a sequence of (locally) unconstrained problems in lower dimension. There exists an extensive literature on both techniques. Our desktop reference is [12]. The development of active set methods for large-scale constrained optimization is given in [15]. The truncated Newton algorithm for unconstrained optimization is given in [9]. The combination of both techniques for pure network problems is given in [6], and for generalized networks in [1].

2.1. Model truncated Newton algorithm for unconstrained optimization.

Consider the unconstrained problem

$$\min_{x\in\Re^n} F(x), \tag{4}$$

where $F(x)$ has the same properties as in (1). The PTN algorithm starts from an arbitrary feasible point $x^0 \in \Re^n$ and generates a sequence $\{x^k\}$, $k = 1, 2, 3, \cdots$, such that

$$\lim_{k\to\infty} x_k = x^*,$$

where x^* belongs to the set of optimal solutions to (4) (i.e., $x^* \in X^* = \{x | F(x) \leq F(y), \forall\, y \in \Re^n\}$). The iterative step of the algorithm is the following:

$$x^{k+1} = x^k + \alpha^k p^k. \tag{5}$$

$\{p^k\}$ is a sequence of descent directions computed by solving the system of (Newton's) equations:

$$\nabla^2 F(x^k) p^k = -\nabla F(x^k) + \eta^k. \tag{6}$$

$\nabla^2 F(x^k)$ and $\nabla F(x^k)$ denote the Hessian matrix and gradient vector of $F(x)$ evaluated at point x^k. The sequence η^k is a measure of accuracy in solving (6). A scale-independent measure of the residual error is

$$r^k = \frac{\|\nabla^2 F(x^k) p^k + \nabla F(x^k)\|_2}{\|\nabla F(x^k)\|_2}. \tag{7}$$

The step direction is computed from (6) such that the condition $r^k \leq \eta^k$ is satisfied and the sequence $\{\eta^k\} \to 0$ as $k \to \infty$. $\{\alpha^k\}$ is a sequence of step sizes computed by solving

$$\alpha^k = \arg\min_{\alpha\in\Re_+} \{F(x^k + \alpha p^k)\} \tag{8}$$

(i.e., at iteration k the scalar α^k is the step size that minimizes the function $F(x)$ along the direction p^k starting from point x^k). Computing α^k from (8) corresponds to an exact minimization calculation that may be expensive for large-scale problems. It is possible to use an inexact line search. The global convergence of the algorithm is preserved if the step length computed by inexact solution of (8) produces a sufficient descent of $F(x)$, satisfying Goldstein–Armijo-type conditions.

2.2. Model active set algorithm for constrained optimization.

Consider now the transformation of (1)–(3) into a locally unconstrained problem. Following [15], we partition the matrix A into the form

$$A = [\,B \;\; S \;\; N\,]. \tag{9}$$

B is a nonsingular matrix of dimension $m \times m$ whose columns form a basis. S is a matrix of dimension $m \times r$ and N is a matrix of dimension $m \times (n - m - r)$. We also use $\mathcal{B}$, $\mathcal{S}$, and $\mathcal{N}$ to denote the sets of basic, superbasic, and nonbasic variables, respectively. Similarly, we partition x^k into

$$x^k = [\, x_B^k \;\; x_S^k \;\; x_N^k \,]. \tag{10}$$

$x_B^k \in \Re^m$ are the basic variables, $x_S^k \in \Re^r$ are the superbasic variables, and $x_N^k \in \Re^{n-m-r}$ denote nonbasic variables. Nonbasic variables, for a given partitioning (9)–(10), are kept fixed to one of their bounds. If we now partition the step direction p as

$$p^k = [\, p_B^k \;\; p_S^k \;\; p_N^k \,], \tag{11}$$

then we require $p_N^k \equiv 0$ (i.e., nonbasic variables remain fixed) and furthermore, p^k should belong to the nullspace of A (i.e., $Ap^k = 0$), so that p^k is a feasible direction. Hence p^k must satisfy

$$Bp_B^k + Sp_S^k = 0 \quad \text{or} \quad p_B^k = -(B^{-1}S)p_S^k. \tag{12}$$

If the superbasic variables are strictly between their bounds and the basis B is maximal as defined in [7] (i.e., a nonzero step in the basic variables x_B is possible for any choice of direction $(p_B^k \;\; p_S^k \;\; 0)$), then the problem is locally unconstrained with respect to the superbasic variables. Hence a descent direction for p_S^k can be obtained by solving the (projected) Newton's equations:

$$\left(Z^T\nabla^2 F(x^k)Z\right) p_S^k = -Z^T\nabla F(x^k) + \eta^k, \tag{13}$$

where Z is a basis for the nullspace of A defined as

$$Z = \begin{bmatrix} -B^{-1}S \\ I \\ 0 \end{bmatrix}. \tag{14}$$

The primary computational requirement of the algorithm is in solving the system of equations (13) of dimension $r \times r$. This system is solved using conjugate gradient with a preconditioner matrix equal to the inverse of the diagonal of the reduced Hessian matrix $Z^T\nabla^2 F(x^k)Z$. Calculation of p_B^k from (12) involves only a matrix–vector product and is an easy computation. The partitioning of the variables and the matrix A into basic, superbasic, and nonbasic elements is also, in general, very fast. For some of the bigger test problems the solution of system (13) takes as much as 99 percent of the overall solution time. For example, solving problem MULTH2 of Table 1 gives rise to a system of equations of dimension $22,588 \times 22,588$. In spite of its very large size, system (13) is usually very sparse. In §3 we look at its sparsity pattern, and determine ways to partition it into smaller systems that can be solved independently and also in parallel.

3. Block-partitioning of Newton's equations.

We return now to (13), and try to identify a partitioning of the matrix $\left(Z^T\nabla^2 F(x^k)Z\right)$. Recall that

$$Z = \begin{bmatrix} -B^{-1}S \\ I \\ 0 \end{bmatrix}, \tag{15}$$

and assume that the function $F(x) = \sum_{t=1}^n F_t(x_t)$ is separable so that the Hessian matrix is diagonal. (This assumption is relaxed in §3.4.) If we momentarily ignore the dense submatrix $(B^{-1}S)$ and assume that

$$Z \doteq \hat{Z} = \begin{bmatrix} I \\ 0 \end{bmatrix} \tag{16}$$

(the identity I and null matrix 0 chosen such that Z is conformable to $\nabla^2 F(x^k)$), then the product $\hat{Z}^T \nabla^2 F(x^k) \hat{Z}$ is a matrix of the form H_I where H_I is a diagonal matrix with the tth diagonal element given by

$$\frac{\partial^2 F_t(x_t^k)}{\partial x_t^2}.$$

Hence the complication in partitioning (13) is the presence of the submatrix $(B^{-1}S)$. The structure of this submatrix is examined next.

3.1. The structure of $(B^{-1}S)$. The matrix B is a basis for the network flows of problem (1)–(3). It is well known (see, e.g., [5] or [13]) that the basis of a pure network problem is a lower triangular matrix. The graph associated with this basis matrix is a rooted tree. The basis of a generalized network is characterized by the following theorem (see, e.g., [5, p. 421]).

THEOREM 3.1. *Any basis B of a generalized network problem can be put in the form*

$$B = \begin{bmatrix} B^1 & & & & & \\ & B^2 & & & & \\ & & \cdot & & & \\ & & & B^\ell & & \\ & & & & \cdot & \\ & & & & & B^L \end{bmatrix},$$

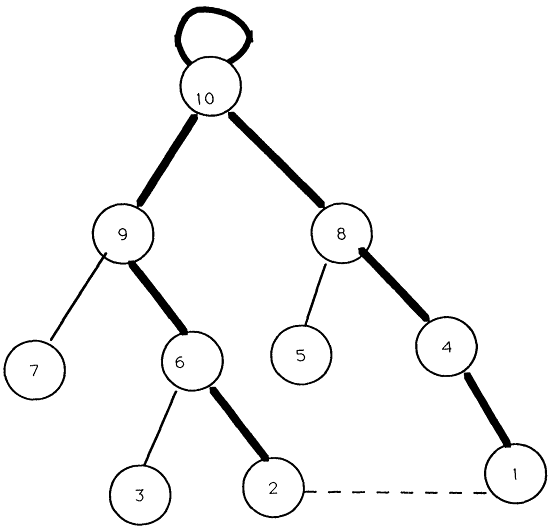
where each square submatrix B^ℓ is lower triangular with at most one element above the diagonal.

The graph associated with each submatrix B^ℓ is a tree with one additional arc, making it either a rooted tree or a tree with exactly one cycle, and is called a quasi tree (abbreviated: q-tree). The graph associated with a generalized network basis is a forest of q-trees.

To describe the structure of $(B^{-1}S)$ we first define the *basic equivalent path* (BEP) for a superbasic variable x_t with incident nodes (i, j). For pure network problems, it is the set of arcs on the basis tree that lead from node j to node i. The BEP together with arc $t \sim (i, j)$ creates a loop.

In the case of a generalized network, it is the set of arcs that lead from nodes i and j to a cycle; the BEP includes all arcs on the cycle. The tth column of $(B^{-1}S)$ has nonzero entries corresponding to the BEP of the tth superbasic variable. The numerical values of $(B^{-1}S)$ are ±1 for pure network problems and arbitrary real numbers for generalized networks; the numerical values are of no consequence to our development. To illustrate the preceding discussion, Fig. 1 shows the basis of a pure network problem together with the BEP for a superbasic arc and the corresponding column of $(B^{-1}S)$. Figure 2 illustrates the same definitions for generalized network problems.

The matrix $(B^{-1}S)$ can be partitioned into submatrices with nonoverlapping rows if the columns of each submatrix have a BEP with no basic arcs in common with the columns of any other submatrix.

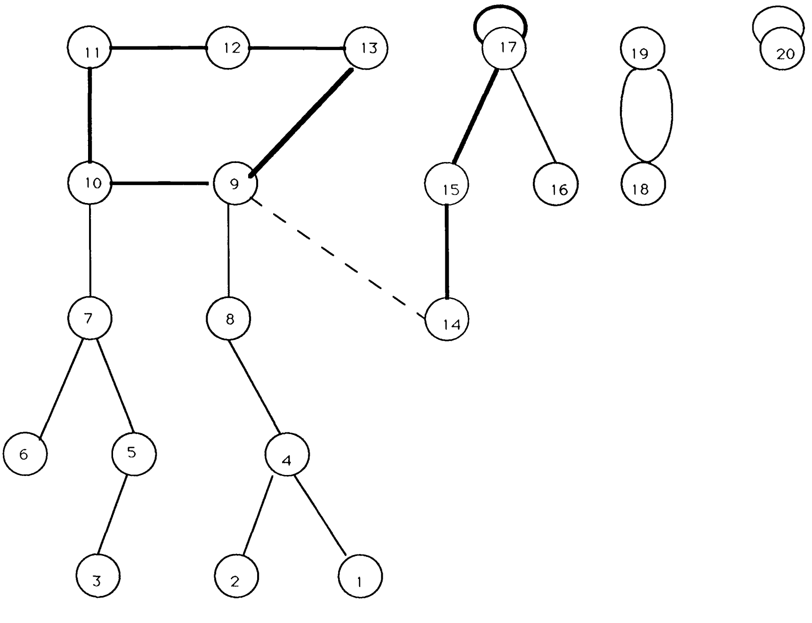


	1	2	3	4	5	6	7	8	9	10
1	*									
2		*								
3			*							
4	*			*						
5					*					
6		*	*			*				
7							*			
8				*	*			*		
9						*	*		*	
10								*	*	*

Sparsity pattern of $(B^{-1}S)$ corresponding to superbasic $(1,2)$:

Row no.		Corresponding Basic Arc
1	*	(1,4)
2	*	(2,6)
3	0	
4	*	(4,8)
5	0	
6	*	(6,9)
7	0	
8	*	(8,10)
9	*	(9,10)
10	*	(10,10)

FIG. 1. *Pure network basis: Matrix and graph representation, and an example of a basic equivalent path. Basic equivalent path* (BEP) *for arc with incident nodes* $(1,2)$: $\{(2,6),(6,9),(9,10),(10,10),(10,8),(8,4),(4,1)\}$.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	*																			
2		*																		
3			*																	
4	*	*		*																
5			*		*															
6						*														
7					*	*	*													
8				*				*												
9								*	*				*							
10							*		*	*										
11										*	*									
12											*	*								
13												*	*							
14														*						
15														*	*					
16																*				
17															*	*	*			
18																		*	*	
19																		*	*	
20																				*

Sparsity pattern of $B^{-1}S$ corresponding to superbasic arc (9,14):

Row no.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	0	0	0	0	0	0	0	0	*	*	*	*	*	*	*	0	*	0	0	0

FIG. 2. *Generalized network basis: Matrix and graph representation, and an example of a basic equivalent path. Basic equivalent path* (BEP) *for arc* (9, 14): {(9, 10), (10, 11), (11, 12), (12, 13), (13, 9), (14, 15), (15, 17), (17, 17)}.

3.2. Partitioning of $(B^{-1}S)$ for pure networks. Let β_t denote an ordered set of binary indices such that $(\beta_t)_l = 1$ if the lth arc $(i,j) \in \mathcal{B}$ is in the BEP of the tth arc $(i',j') \in \mathcal{S}$, and $(\beta_t)_l = 0$ otherwise. We seek a partitioning of the set $\mathcal{S}$ into K disjoint independent subsets, say $\mathcal{S}^k$, $k \in \mathcal{K} = \{1,2,\cdots,K\}$ such that

$$\mathcal{S} = \bigcup_{k=1}^{K} \mathcal{S}^k \tag{17}$$

and

$$t,u \in \mathcal{S}^k \quad \text{if} \quad \beta_t \vee \beta_u \neq 0 \quad \forall\, k \in \mathcal{K} \tag{18}$$

(i.e., superbasic arcs t and u belong to the same subset, say $\mathcal{S}^k$, if the sets β_t and β_u have at least one overlapping nonzero element, and hence there is at least one common basic arc in the BEPs of the tth and uth superbasic arcs, respectively. Otherwise, t and u belong to two distinct subsets).

Escudero [11] was the first to propose the partitioning of $\mathcal{S}$ into independent superbasic subsets $\mathcal{S}^k$ according to (17)–(18), for replicated pure networks. These replicated networks consist of subnetworks with identical structure and are connected by linking arcs. (These linking arcs represent inventory flow for his problems that are multiperiod networks.) In the same reference Escudero gives a procedure for identifying the independent superbasic sets $\mathcal{S}^k$.

In this section we formulate the problem of identifying independent superbasic sets as problems from graph theory. Define the graph $G_{\mathcal{S}} = \{\mathcal{V}, \mathcal{E}_{\mathcal{S}}\}$ where the node and edge sets are defined as $\mathcal{V} = \{1,2,3,\cdots,|\mathcal{S}|\}$ and $\mathcal{E}_{\mathcal{S}} = \{(t,u) | t,u \in \mathcal{V}, \beta_t \vee \beta_u \neq 0\}$. A node is associated with each superbasic variable $t \in \mathcal{S}$, and an edge is incident to two nodes if and only if the BEP of the corresponding superbasic variables overlap. We call $G_{\mathcal{S}}$ the *connectivity graph* of the superbasic set $\mathcal{S}$. It can be constructed as the adjacency graph of the matrix $Z^T Z$.

The first partitioning scheme simply formalizes Escudero's procedures.

Partitioning Scheme I. Find the connected components of $G_{\mathcal{S}}$, say $\mathcal{V}^k \subseteq \mathcal{V}$, $k = 1,2,\cdots,K$. Then $\mathcal{S}^k = \{t | t \in \mathcal{V}^k\}$ will satisfy conditions (17)–(18), by definition of connected components, and hence $\mathcal{S}^k$, $k = 1,2,\cdots,K$ are independent superbasic sets. Finding connected components of $G_{\mathcal{S}}$ can be achieved with the algorithm of Tarjan [19]. Unfortunately it is not always possible to find more than one connected component of $G_{\mathcal{S}}$. For the case of replicated networks such a partitioning usually exists since the interaction among superbasics in different subnetworks is weak. This explains the success of Escudero's method for the multiperiod networks. The second partitioning scheme we propose here allows us to partition the superbasic set $\mathcal{S}$ for a broader class of problems.

Partitioning Scheme II. Find the articulation points of $G_{\mathcal{S}}$. Let $\mathcal{C} \subset \mathcal{V}$ be the set of articulation points and $\mathcal{S}_{\mathcal{C}}$ the set of superbasic variables corresponding to the set $\mathcal{C}$. Update the set of superbasic variables by $\mathcal{S}' = \mathcal{S} \setminus \mathcal{C}$ and let $\mathcal{N} = \mathcal{N} \cup \mathcal{C}$. The new superbasic set $\mathcal{S}'$ now consists of at least $|\mathcal{C}|$ independent subsets. The fact that identifying articulation points leads to partitioning of the superbasic sets can easily be seen by the definition of articulation points (see, e.g., [2]). An algorithm with linear time and space complexity for finding articulation points of a graph is given in [19], and was used in our implementation of this partitioning scheme. The variables in the set $\mathcal{S}_{\mathcal{C}}$ that have been eliminated from the superbasic set are kept fixed at their current value. The new superbasic set $\mathcal{S}'$, which leads to a block-diagonal form for the projected Hessian matrix, can be used to compute a preconditioner for the reduced Hessian matrix of the original superbasic set $\mathcal{S}$. An alternative approach, which has been followed in our implementation, is to optimize the objective function over the superbasic set $\mathcal{S}'$ while the variables in $\mathcal{S}_C$ remain nonbasic. Of course, the elements of $\mathcal{S}_C$ may change for different major iterations. However, this scheme is not guaranteed to produce an optimal solution since some eligible superbasic variables (i.e., those in $\mathcal{S}_C$) are kept fixed. Once the block-partitioned algorithm terminates, we start a new major iteration without using the block-partitioning scheme. Hence the algorithm reverts to the original PTN. We observed that, for our test problems, one major iteration without block-partitioning would suffice. Furthermore, the objective value was only improved by less than 0.5 percent when the last, non-block-partitioned major iteration was completed. (See also the discussion in §4.3.) In essence, the block-partitioned truncated Newton algorithm is used to compute an approximate solution. Higher accuracy can then be achieved by applying the original truncated Newton algorithm directly.

3.3. Partitioning of $(B^{-1}S)$ for generalized networks. The graph-partitioning schemes discussed in §3.2 for pure network problems can also be applied in the case of generalized networks. We refer to these procedures as scheme GN-I (for generalized networks-I). Here we develop alternative techniques to partition the superbasic set of generalized network problems that take advantage of the block structure of the generalized network basis.

In §3.1 we observed that the graph associated with the basis of a generalized network problem is a collection of quasi trees. Suppose the basis matrix B consists of submatrices B^ℓ, $\ell = 1, \cdots, L$. We denote the graph (q-tree) associated with B^ℓ by $G_\ell = (N_\ell, E_\ell)$. The superbasic set $\mathcal{S}$ can be partitioned into subsets $\mathcal{S}^\ell$ defined by

$$\mathcal{S}^\ell = \{(i,j) \in \mathcal{S} | i,j \in N_\ell\} \quad \forall\, \ell = 1, 2, \cdots, L, \tag{19}$$

with $\cup_{\ell=1}^{L} \mathcal{S}^\ell \subseteq \mathcal{S}$. This partitioning scheme will ignore any superbasic variables that connect basis submatrices. A partitioning scheme that includes additional superbasic variables is the following: Given indices k, $p(k) \leq L$ and $q(k) \leq L$, $p(k) \neq q(k)$, choose $B^{p(k)}$ and $B^{q(k)}$ and define

$$\mathcal{S}^{p(k)q(k)} = \{(i,j) \in \mathcal{S} | i \in N_{p(k)}, j \in N_{q(k)}\}, \tag{20}$$

$$\mathcal{S}^{p(k)} = \{(i,j) \in \mathcal{S} | i,j \in N_{p(k)}\}, \tag{21}$$

$$\mathcal{S}^{q(k)} = \{(i,j) \in \mathcal{S} | i,j \in N_{q(k)}\}, \tag{22}$$

and finally, $\mathcal{S}^k = \mathcal{S}^{p(k)q(k)} \cup \mathcal{S}^{p(k)} \cup \mathcal{S}^{q(k)}$. To ensure that two sets $\mathcal{S}^{k_1}, \mathcal{S}^{k_2}$ are independent we require that

$$B^{p(k_1)} \neq B^{p(k_2)} \neq B^{q(k_2)}, \qquad B^{q(k_1)} \neq B^{p(k_2)} \neq B^{q(k_2)}.$$

We now describe a procedure to identify these independent subsets $\mathcal{S}^k$ of superbasic arcs. The procedure works as follows: at each iteration, a pair of basis subgraphs connected with the largest number of superbasic arcs is identified; these superbasic arcs, together with the superbasic arcs that connect nodes in each subgraph, form a subset $\mathcal{S}^k$. To ensure the independence of the subsets thus formed, the basis subgraphs used in constructing the subsets are marked and not considered in subsequent iterations. A complete description of the procedure is given below.

```
$\mathcal{U} = \{1, 2, \cdots, L\}$; $k = 0$; $scount = 0$; $\rho \in [0.5, 1)$.
Repeat until $scount \geq \rho|\mathcal{S}|$ or $|\mathcal{U}| \leq 1$.
    1. $k = k + 1$.
    2. Find $r(k)$ and $s(k)$ such that $r(k) \neq s(k)$ and $|\mathcal{S}^{r(k)s(k)}| = \max_{p,q \in \mathcal{U}} |\mathcal{S}^{pq}|$.
    3. $scount = scount + |\mathcal{S}^{r(k)s(k)}| + |\mathcal{S}^{r(k)}| + |\mathcal{S}^{s(k)}|$.
    4. $\mathcal{S}^k = \mathcal{S}^{r(k)s(k)} \cup \mathcal{S}^{r(k)} \cup \mathcal{S}^{s(k)}$.
    5. $\mathcal{U} = \mathcal{U} \setminus \{r(k), s(k)\}$.
```

We refer to the procedure described above as partitioning scheme GN-II. We note that scheme GN-II may place some candidate superbasic arcs into the nonbasic set when creating the independent subsets $\mathcal{S}^k$. It is also advisable to choose ρ as close to 1 as possible. Some preliminary experimentation is needed to find a suitable value of ρ. We also remark that alternative procedures that take into account the interaction of more than two basis subgraphs can be designed. However, this scheme achieved a satisfactory partitioning of the superbasic set with the generalized networks problems used in this study.

To illustrate the partitioning procedure on the basis illustrated in Fig. 2, let us assume that the superbasic set is as follows:

$$\begin{aligned}\mathcal{S} = \{&(6,11), (3,8), (9,14), (14,17), (13,19), (7,19),\\ &(1,18), (16,20), (18,20), (14,16), (17,20)\}.\end{aligned}$$

Scheme GN-II will identify

$$\mathcal{S}^1 = \{(1,18), (7,19), (13,19), (6,11), (3,8)\}$$

and

$$\mathcal{S}^2 = \{(14,17), (14,16), (16,20), (17,20)\}.$$

In this example, superbasic arcs $(9,14), (16,20)$ are placed into the nonbasic set to ensure independence of $\mathcal{S}^1$ and $\mathcal{S}^2$.

3.4. Extensions to nonseparable problems. We can develop a partitioning of the system $\left(Z^T \nabla^2 F(x) Z\right)$ for the case when the function $F(x)$ is nonseparable. In this case the Hessian matrix is not diagonal. The partitioning condition (18) from §3.2 has to be modified as follows. Let $t, u \in \mathcal{S}$. Then

$$t \in \mathcal{S}^{k_1} \quad \text{and} \quad u \in \mathcal{S}^{k_2} \quad \text{iff} \quad \beta_t \vee \beta_u = 0 \quad \text{and} \quad \frac{\partial^2 F(x)}{\partial x_t x_u} = 0. \tag{23}$$

The connectivity graph associated with the superbasic set $\mathcal{S}$ is now defined as the graph $G_{\mathcal{S}} = \{\mathcal{V}, \mathcal{E}_{\mathcal{S}}\}$ where $\mathcal{V} = \{1, 2, 3, \cdots, |\mathcal{S}|\}$, as in §3.2, and

$$\mathcal{E}_{\mathcal{S}} = \left\{(t,u) |\, t, u \in \mathcal{V},\ \beta_t \vee \beta_u \neq 0 \text{ and } \frac{\partial^2 F(x)}{\partial x_t x_u} \neq 0\right\}$$

(i.e., an edge is incident to two nodes if and only if the BEP of the corresponding superbasics overlap and changes in the value of one variable, x_t, change the objective value for the second, x_u). The connectivity graph can be obtained from the adjacency graph of $(Z^T\nabla^2F(x)Z)$. With this definition of connectivity graph, we can now apply either partitioning scheme from §3.2. Similarly, we can extend the partitioning technique of §3.3 to handle nonseparable generalized network problems as well.

4. Computational experiments.

The partitioning techniques discussed earlier were implemented in the network optimizer GENOS of Mulvey and Zenios [14].[1] The modified code, which we call GENOS/PCG, was used to solve a collection of nonlinear problems, both pure and generalized. (PCG stands for partitioned conjugate gradient, since GENOS is using conjugate gradient to solve the partitioned systems of Newton's equations.) The objective of the numerical experiments is to establish the performance of PTN when Newton's equations are solved in block-partitioned form. Furthermore, we solved the test problems on a shared-memory vector multiprocessor to study the performance of the block-partitioned PTN with parallel computing. Finally, we compare the performance of the parallel implementation of GENOS/PCG to an alternative parallel implementation proposed by Zenios and Mulvey [20].

GENOS and the modifications in GENOS/PCG are written in Fortran 77. Experiments on serial computer were carried out on a VAX 8700 at the Wharton School, running VMS. The programs were compiled with the default compiler optimization option. Parallel computing experiments were carried out on an Alliant FX/4 of the HERMES Laboratory for Financial Modeling and Simulation at the Wharton School, running Ultrix. The level of optimization used was at least $-Og$ (i.e., global scalar optimizations). The flag $-O$ was used for experiments using the vector and parallel features of the Alliant. All times are reported in CPU seconds, exclusive of input and output. The termination tolerance η^k is adjusted dynamically using a forcing sequence; its final value is 10^{-2}.

4.1. Test problems.

The characteristics of the test problems are summarized in Table 1. In addition to the size of the problems, we give the number of superbasic arcs at optimality; this is the dimension of the system of equations we had to solve at the last iteration of the algorithm. The MULTxn problems were generated by the network problem generator of Chang and Engquist [3]. They are multiperiod networks: a basic generalized network structure is replicated for several time periods and inventory-type links connect the replicated components. All problems have a quadratic objective function of the form $\sum_t a_t x_t^2$ with $a_t \in [1, 100]$. The number at the end of each problem name indicates the number of replications. For example, MULTC2 is a two-period model; each single period network has 1000 nodes. MULTC8 is an eight-period model with a total of 8000 nodes. The STICKn and PTNn problems were obtained from [1].

4.2. Solving pure network problems.

We implemented both Partitioning Schemes I and II, discussed in §3.2. First, we must construct the connectivity graph G_S. This is the adjacency graph of the matrix Z^TZ. Due to the large size of the matrix Z, it is more efficient to construct the connectivity graph G_S based on the structural nonzeros of Z than to form the product Z^TZ. For example, finding the adjacency

[1] GENOS is a library of algorithms for solving network problems. It includes network simplex, primal truncated Newton, and simplicial decomposition. The partitioning techniques were implemented within the primal truncated Newton solver. Details on the implementation of this solver within GENOS using sparse graph data structures are given in [1].

TABLE 1
Test problem characteristics.

Test problem	**Size**	**No. of basis submatrices**	**Free arcs at opt.**	**Obj. value**
MULTA4	400–1002	9–10	602	0.32713×10^7
MULTA8	800–2000	15–22	1200	0.66535×10^7
MULTA12	1200–3603	23–27	2403	0.10615×10^8
MULTB4	2000–4010	8–31	2010	0.130059×10^7
MULTB8	4000–8022	21–28	4022	0.267280×10^8
MULTB12	6000–15,039	35–36	9039	0.175176×10^8
MULTB15	7500–18,000	40–50	10,500	0.3779514×10^7
MULTC2	2000–5008	4–5	3008	0.53512×10^7
MULTC4	4000–10,027	5–8	6027	0.111013×10^8
MULTC8	8000–20,047	40–50	12,047	0.201423×10^8
MULTH1	11,000–27,571	40–60	16,571	0.566242×10^7
MULTH2	15,000–37,588	50–60	22,588	0.795584×10^7
PTN150	150–196	1	44	-0.481973×10^5
PTN660	666–906	1	240	-0.206107×10^6
STICK1	209–454	1	246	6.934392
STICK2	650-1412	1	763	3.124563
STICK3	782–1686	1	905	0.111797×10^2
STICK4	832–2264	1	1433	1.566195

graph of $Z^T Z$ takes 21.6 seconds for PTN660 and 18.5 seconds for STICK1. Working on the matrix Z instead, the same graphs are constructed in 4.1 and 0.8 seconds, respectively. The most efficient implementation was adopted in all experiments.

Partitioning Scheme I was implemented using an algorithm for connected components due to Tarjan [19]. For the test problems we have available, the connectivity graphs G_S are very dense and in all cases there is only one connected component.

Partitioning Scheme II was implemented using an articulation point algorithm due to Tarjan [19]. This algorithm was then applied to the connectivity graph of the superbasic set for the pure network problems STICK1–4 and PTN150–660 at each major iteration of GENOS/PCG. The results are summarized in Table 2. Under the column "Size of components" we do not list the (usually large) number of components corresponding to a single superbasic variable. The difference between the size of the subspace and the total of the sizes of the disconnected components is the number of single-variable sets. We give the total CPU seconds spent in the graph-partitioning procedure during the execution of the primal truncated Newton algorithm under the heading "Partition. time."

We observe from the table that these test problems do not produce independent superbasic sets of (approximately) equal sizes. The largest independent set dominates the computations, and solution time for the block–partitioned Newton's equations is only marginally better than the solution time required in solving the equations over the original subspace. In addition some overhead is incurred in creating the connectivity graph and the subsequent partitioning. The connectivity graph of the superbasic sets tend to be dense, and hence very large. For example, the connectivity graph for the superbasic set of PTN660 has 236 nodes and 5768 arcs. The articulation point algorithm identified four articulation points in 0.1 seconds. However, creating the connectivity graph from the matrix $(B^{-1}S)$ takes 0.48 seconds.

We conclude that for pure network problems the block-partitioning techniques

TABLE 2
Articulation point analysis: Results for PTN *and* STICK *problems.*

Problem	Major Iter.	Subspace dimension	Number of artic. points	Number of Comp.	Size of Components	Partition. time
PTN150						0.53
	1	45	4	4	3–2–4–36	
	2	43	2	4	2–1–34 –3	
	3	44	2	4	2–1–35–3	
PTN660						0.68
	1	236	5	6	2–2–2–2–1–224	
	2	238	4	5	2–2–2–2–226	
STICK1						0.33
	1	246	7	8	24–6–1–1–23–3–8–107	
	2	246	11	12	14–10–6–22–105–8–2	
	3	246	8	9	14–10–5–131–7–3	
STICK2						0.66
	1	763	11	12	5–4–590–11–8	
	2	763	14	15	517–32–31–11–9	
STICK3						0.27
	1	905	10	11	124–237–19–56–12	
	2	905	9	10	247–180–12–11–9	
STICK4						1.41
	1	1433	9	10	1341–28–26–12–2–2	

do not offer any computational savings. However, for problems that have additional structures, such as Escudero's multiperiod networks, the partitioning could be very effective. Nevertheless, since an attempt to partition the network can be executed very efficiently, it is included as an optional preprocessing phase in GENOS/PCG.

4.3. Solving generalized network problems. We implemented both partitioning schemes GN-I and GN-II. We observed that the scheme GN-II based on the structure of the basis graph is more efficient and produces better partitioning of the superbasic set $\mathcal{S}$. For example, problem MULTA4 is solved by GENOS in 40 seconds without partitioning. Using partitioning scheme GN-I takes 54 seconds, whereas with partitioning scheme GN-II the problem is solved in 15 seconds. We adopted the scheme GN-II in solving all the test problems, as shown in Table 3 together with the results from GENOS[2]. The speedup factor indicates the ratio of total solution times of GENOS by that of GENOS/PCG. For smaller problems where the network basis does not partition evenly, the savings are reduced due to overhead in creating the subspaces. Improvements in performance increase with the problem size. We observe an improvement in performance by a factor of 1.20 to 5.25.

One final observation on these experiments: We noted earlier that with the partitioning procedure described in §3.3 some superbasic arcs may be placed in the nonbasic set in order to achieve independence of the blocks. This usually affects the accuracy of the solution. However, as the optimality tolerance is sufficiently decreased, one iteration without partitioning the superbasic set is sufficient to achieve high accuracy. But this step is computationally expensive, particularly for larger problems. Ignoring

[2] **PTN** indicates the overhead in forming the network basis and other initializations, **SB** stands for subspace selection, i.e., the selection of the superbasic variables. This is the step where partitioning scheme GN-II is incorporated into the program. **CG** and **LS** stand for the conjugate gradient and linesearch procedures, respectively.

TABLE 3
Solution times with GENOS *and* GENOS/PCG *on the* VAX 8700.

	GENOS					GENOS/PCG					Ratio
Problem	PTN	SB	CG	LS	Total	PTN	SB	CG	LS	Total	
MULTA4	2.32	2.20	35.38	0.50	40.42	2.23	1.83	9.64	1.57	15.27	2.64
MULTA8	4.83	4.39	112.47	1.03	122.73	6.72	5.26	41.21	6.85	60.00	2.04
MULTA12	7.94	8.97	241.58	1.86	260.35	13.35	11.70	77.49	12.85	115.47	2.25
MULTB4	14.60	14.04	506.27	2.73	537.64	22.66	16.69	228.32	12.37	280.0	1.92
MULTB8	29.00	29.69	1227.99	4.48	1291.16	64.92	45.56	601.06	28.42	739.97	1.74
MULTB12	57.91	60.16	3130.59	9.73	3258.40	91.98	85.00	1537.10	72.95	1787	1.82
MULTB15	96.41	97.44	15092.50	17.32	15303.72	114.35	101.87	4398.57	98.41	4713.20	3.24
MULTC2	21.41	29.25	696.56	2.50	749.74	27.51	29.25	557.90	7.24	621.99	1.20
MULTC4	43.7	52.33	2233.40	6.95	2336.40	83.51	65.85	1372.41	22.54	1544.00	1.51
MULTC8	123.94	106.70	21659.00	14.99	21904.70	214.86	146.41	3666.11	144.23	4171.62	5.25
MULTH1	102.29	100.88	29908.23	14.54	30125.95	186.56	163.82	6425.61	161.08	6937.08	4.34
MULTH2	226.37	205.33	80902.46	40.59	81374.76	352.59	265.59	26009.06	350.60	26977.00	3.03

TABLE 4
Comparing serial GENOS *with parallel* GENOS/PCG.

Test problem	GENOS (1CE)	GENOS/PCG (4CE)	Speedup
MULTB12	2316.00	753.68	3.07
MULTB15	6362.59	1172.81	5.42
MULTC8	6723.11	2245.15	2.99
MULTH1	17045.85	2009.73	8.48
MULTH2	25336.70	3982.50	6.35

this step results in significant savings in performance. The objective value at termination in this case is within 0.5 percent of the objective value given by GENOS. This observation is illustrated in Fig. 3. In all experiments the last step is performed and hence the problem is solved to optimality.

4.3.1. Parallel implementation.

Solving Newton's equations. First we tested the parallel implementation of GENOS/PCG in solving the set of blocks in a particular instance of Newton's equation. Using problem MULTB12, we isolated one block of dimension 750 × 750, replicated it 20 times, and solved it in parallel (i.e., we assumed that the system of equations in the case of MULTB12 would partition into 20 blocks of equal size. This would be the ideal situation.). Figure 4 shows the solution time when solving this set of 20 blocks using 1–4 processors. Significant speedup factors are observed when using parallel processors ranging from 1.6 on two computational elements (CE) to 3.4 on four. The speedup is not perfect due to the overhead incurred in concurrent subroutine calls. In Fig. 4 we also show the solution times when solving the actual set of 20 blocks, with dimensions ranging from 10 × 10 to 900 × 900. While we still observe significant speedup between the serial and parallel implementation, the speedup factors are lower than those observed in the previous experiment. The difference is attributed to the uneven load balancing among processors that is due to the varying sizes of the blocks.

The linesearch procedure. Parallel CEs were also used in the line search procedure of GENOS/PCG. The line search routine of GENOS (see [1]) is a quadratic interpolation with safeguards. At every iteration of the linesearch algorithm the function value, gradient vector, and Hessian matrix are evaluated for all the arcs. In GENOS/PCG we need only to evaluate this information for those basic and superbasic variables that appear in the current block. Furthermore, multiple processors can compute in parallel the required information for multiple arcs in the block. Figure 5 illustrates the speedup factor of both the routine that provides function, gradient, and Hessian values, as well as the speedup of the overall line search procedure.

4.3.2. Comparing GENOS with Parallel GENOS/PCG. As a concluding test on the Alliant, we run GENOS on one CE of the Alliant with the fully parallelized GENOS/PCG running on four CEs. The observed speedup factor for some of the bigger problems is shown in Table 4. The speedup factor in most cases exceeds four, which is the number of parallel CEs used. This is due to the combined effect of solving a sequence of smaller problems and the effect of multiprocessing. These two effects were analyzed separately in the results of Table 3 and in §4.3.1, respectively.

4.4. Comparison with alternative parallel implementations. In [20] we proposed an alternative parallel implementation of PTN based on the row-wise dis-

TABLE 5
Comparison of alternative parallel implementation methods.

Problem structure		Tightly coupled multiprocessor	Loosely coupled multiprocessor
Networks	Partitioning		
Pure	Poor	MPTN	MPTN
	Good	MPTN	BPTN
Generalized		MPTN	BPTN

tribution of the nullspace matrix among processors. This implementation produced significant speedups when implemented on the CRAY X–MP. We tested the same implementation on the Alliant FX/4. Hence we now have two alternative parallel implementations on two different architectures and can draw some conclusions on the relative merits of the two methods. Table 5 indicates which method should be preferred for different problem structures and computer systems (MPTN indicates the microtasked implementation of Zenios and Mulvey, BPTN indicates the block-partitioned methods developed here).

The advantage of MPTN is that its implementation does not require any additional computing, and it results in even load balancing among processors. The disadvantage is that the granularity of the parallel tasks is very small; the overhead in spawning tasks on some computers could be significant compared to the amount of computation performed by the task.

The advantage of BPTN is that it produces tasks of large granularity. The disadvantage is that it may produce uneven load balancing among processors. Furthermore, BPTN needs to execute the partitioning algorithms and this overhead can add significantly to the total solution time.

Hence for tightly coupled systems where the overhead of spawning a task is only a few machine cycles, as in the case of CRAY X–MP, MPTN should be preferred. From Fig. 6 we observed that MPTN achieves a speedup of 2.6 on three CPUs for both pure and generalized networks. BPTN does not achieve any speedups for the pure network test problems and a speedup of 1.41 is achieved for generalized networks.

For more loosely coupled systems, which may also include distributed-memory architectures, the BPTN is preferred unless the problems do not partition well. For example, MPTN achieves a speedup of 1.75 on a four-processor Alliant FX/4. BPTN achieves a speedup of 2.21 on the same system for generalized network problems that partition well.

4.5. Solving the test problems on a CRAY X–MP/48. Although we were able to achieve significant improvements both on sequential and parallel implementations, we observed that solving the multiperiod problems was still taking a considerable amount of time. As a final test, we conducted some experiments on the CRAY X–MP/48 to test the effectiveness of the partitioning schemes on a different parallel architecture. The results are summarized in Table 6. The tests were conducted with the default vector option of the CRAY FORTRAN compiler. The results are stated in CPU seconds. The last column gives the ratio of GENOS/PCG solution times on the CRAY to GENOS solution time on the VAX. As can be observed from the table, significant gains in computing resources were possible with a vector supercomputer like the CRAY X–MP/48.

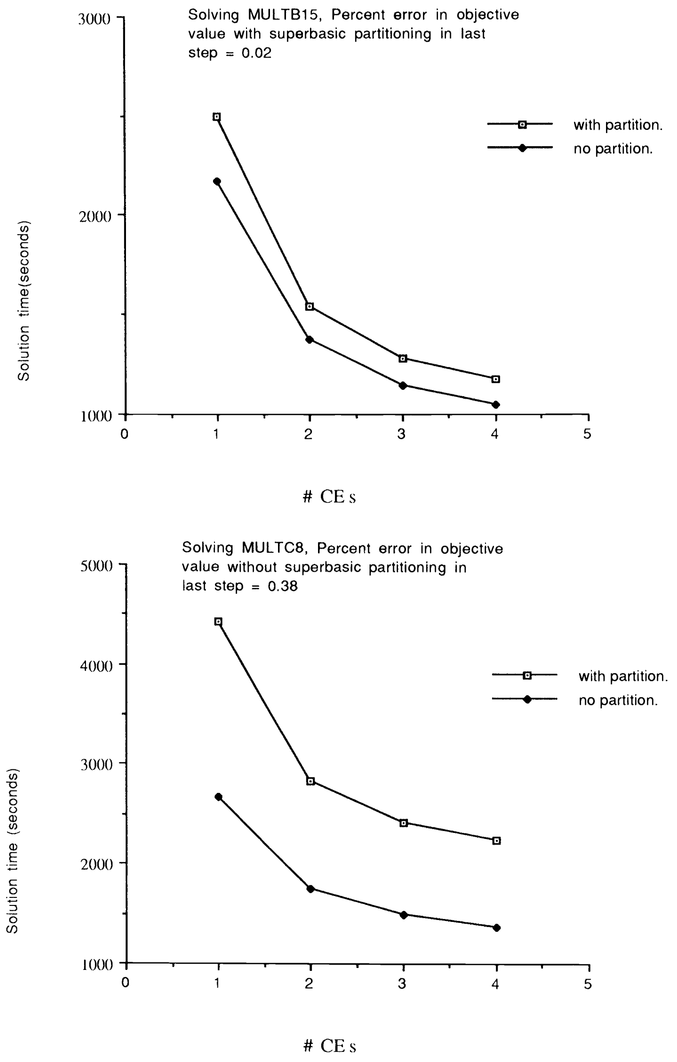


FIG. 3. *Solution times with and without the superbasic partitioning step at the last iteration of* PTN.

TABLE 6

Comparing GENOS *with* GENOS/PCG *on the* CRAY X-MP/48.

Test problem	GENOS	GENOS/PCG	Ratio of GENOS/PCG to GENOS on the VAX
MULTB8	101.92	49.90	25.87
MULTB12	179.36	110.79	29.41
MULTB15	795.29	291.64	52.47
MULTC4	141.05	102.23	22.85
MULTC8	869.66	531.18	41.23

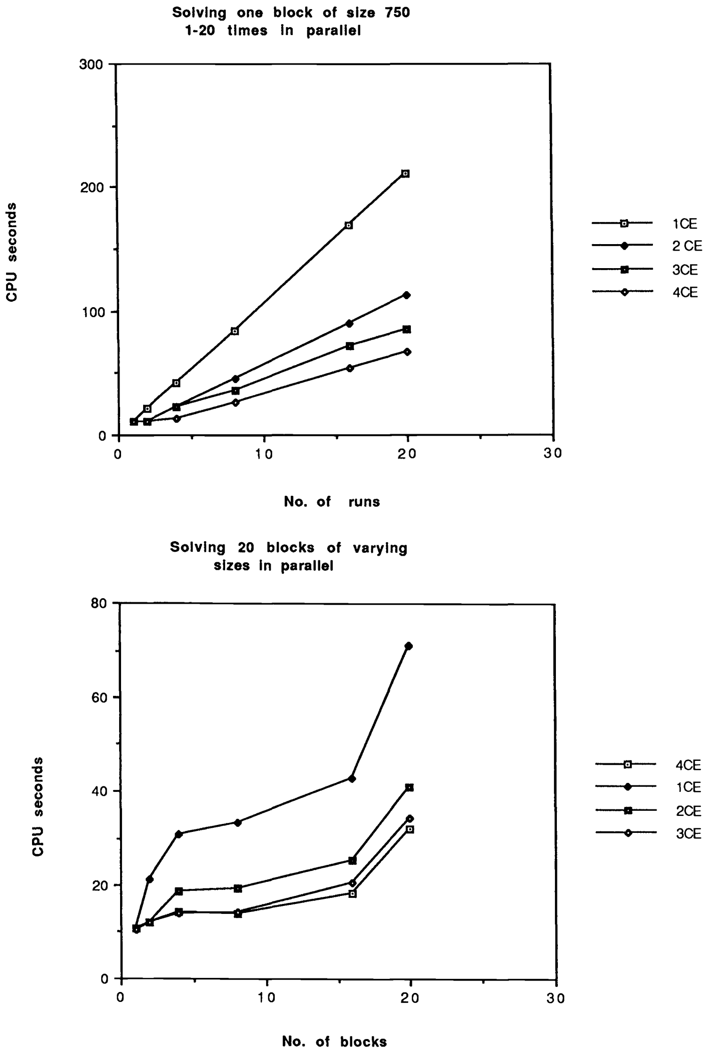


FIG. 4. *Parallel solution of the block-partitioned equations.*

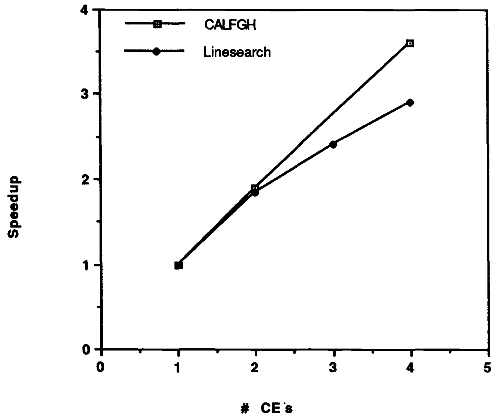
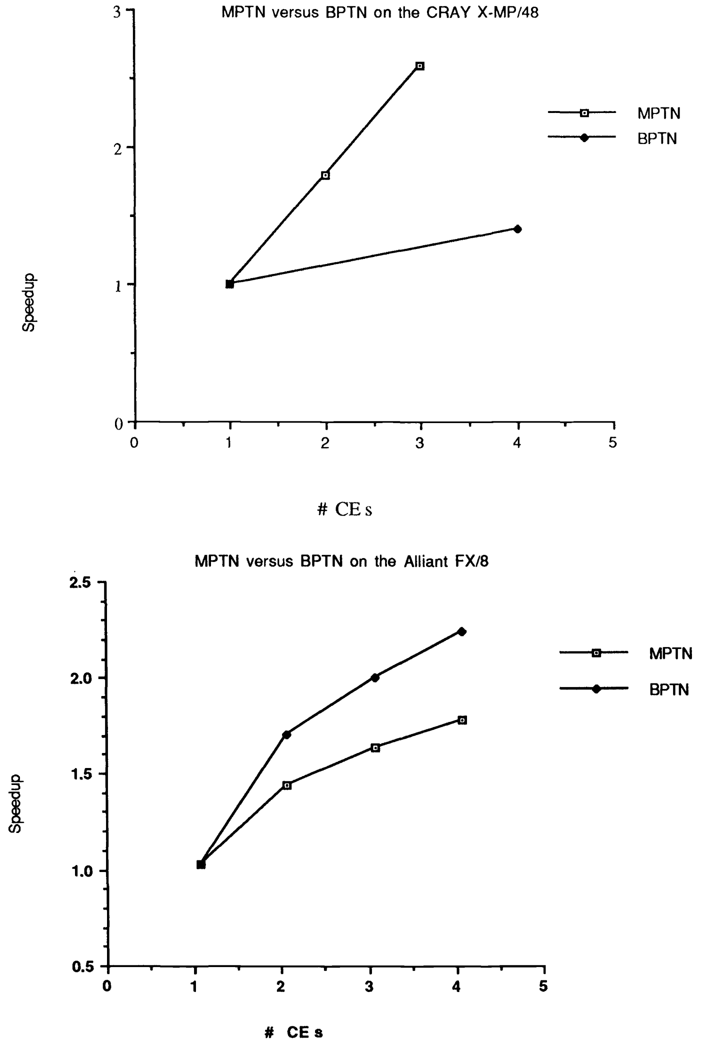


FIG. 5. *Speedup factors of the line search.*


MPTN versus BPTN on the CRAY X-MP/48

MPTN versus BPTN on the Alliant FX/8

FIG. 6. *Comparison of microtasked* PTN *algorithm of Zenios and Mulvey* [20] (MPTN) *and the block-partitioned* PTN (BPTN) *on different parallel architectures.*

5. Concluding observations. We developed here techniques for partitioning Newton's equations in the context of solving nonlinear network problems. The techniques appear to be quite effective and efficient for generalized network problems and are also well suited for parallel computations. In Fig. 7 we use our generalized network problems to illustrate the combined effect of both the partitioning schemes and the computer architecture. As observed in the figure, the reduction in the solution time increases as the problem size gets larger. The partitioning techniques can also be applied efficiently for pure network problems. However, our current collection of pure network test problems does not partition well. Our study also provides guidelines on

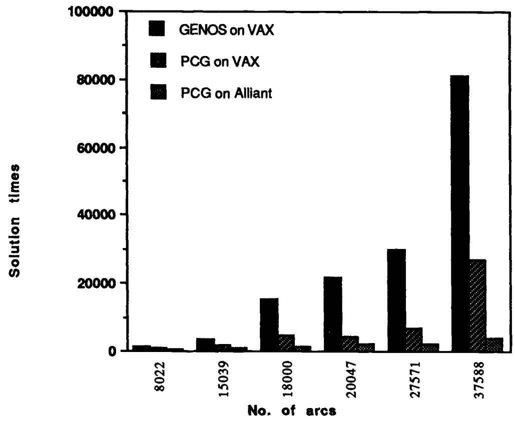


FIG. 7. *Comparative chart of solution times.*

choosing between two alternative parallel implementations depending on characteristics of the problem and the parallel computing platform.

Acknowledgments. The authors benefited from discussions with R. S. Dembo and L. F. Escudero. The assistance of J. W. Gregory in the experiments with the CRAY X-MP is gratefully acknowledged. The manuscript benefited from the comments of two anonymous referees.

REFERENCES

[1] D. P. AHLFELD, J. M. MULVEY, R. S. DEMBO, AND S. A. ZENIOS, *Nonlinear programming on generalized networks*, ACM Trans. Math. Software, 13 (1987), pp. 350–367.

[2] A. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison–Wesley, Reading, MA, 1974.

[3] M. CHANG AND M. ENGQUIST, GTGEN: *A generator for generalized transportation problems*, Res. Report CCS 540, Center for Cybernetic Studies, The University of Texas, Austin, 1986.

[4] R. H. CLARK AND R. R. MEYER, *Parallel arc-allocation algorithms for optimizing generalized networks*, Ann. Oper. Res., 22 (1990), pp. 126–160.

[5] G. B. DANTZIG, *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, 1963.

[6] R. S. DEMBO, *A primal truncated Newton algorithm with application to nonlinear network optimization*, Math. Programming Stud., 31 (1987), pp. 43–72.

[7] R. S. DEMBO AND J. G. KLINCEWICZ, *Dealing with degeneracy in reduced gradient algorithms*, Math. Programming, 31 (1985), pp. 357–363.

[8] R. S. DEMBO, J. M. MULVEY, AND S. A. ZENIOS, *Large-scale nonlinear network models and their application*, Oper. Res., 37 (1989), pp. 353–372.

[9] R. S. DEMBO AND T. STEIHAUG, *Truncated Newton algorithms for large-scale unconstrained optimization*, Math. Programming, 26 (1983), pp. 190–212.

[10] L. F. ESCUDERO, *A motivation for using the truncated Newton approach in a very large scale network problem*, Math. Programming Stud., 26 (1986), pp. 240–244.

[11] ———, *Performance evaluation of independent superbasic sets on nonlinear replicated networks*, European J. Oper. Res., 23 (1986), pp. 343–355.

[12] P. Gill, W. Murray, and M. Wright, *Practical Optimization*, Academic Press, London, New York, 1981.

[13] J. L. Kennington and R. V. Helgason, *Algorithms for Network Programming*, Wiley–Interscience, New York, 1980.

[14] J. M. Mulvey and S. A. Zenios, *User's guide to* GENOS 1.0: *A generalized network optimization system*, Department of Decision Sciences Report 87–12–03, University of Pennsylvania, Philadelphia, 1987.

[15] B. Murtagh and M. Saunders, *Large-scale linearly constrained optimization*, Math. Programming, 14 (1978), pp. 41–72.

[16] S. G. Nash and A. Sofer, *Block truncated Newton methods for parallel optimization*, Math. Programming, 45 (1989), pp. 529–546.

[17] R. T. Rockafellar, *Network Flows and Monotropic Optimization,* Wiley–Interscience, New York, 1984.

[18] R. E. Rosenthal, *A nonlinear network flow algorithm for maximizing the benefits in a hydroelectric power system*, Oper. Res., 29 (1981), pp. 763–786.

[19] R. Tarjan, *Depth-first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.

[20] S. A. Zenios and J. M. Mulvey, *Vectorization and multitasking of nonlinear network programming algorithms*, Math. Programming, 43 (1988) pp. 449-470.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 5, pp. 1194–1217, September 1992

009

NEW BRANCH-AND-BOUND RULES FOR LINEAR BILEVEL PROGRAMMING*

PIERRE HANSEN†, BRIGITTE JAUMARD‡, AND GILLES SAVARD§

Abstract. A new branch-and-bound algorithm for linear bilevel programming is proposed. Necessary optimality conditions expressed in terms of tightness of the follower's constraints are used to fathom or simplify subproblems, branch and obtain penalties similar to those used in mixed-integer programming. Computational results are reported and compare favorably to those of previous methods. Problems with up to 150 constraints, 250 variables controlled by the leader, and 150 variables controlled by the follower have been solved.

Key words. bilevel programming, Stackelberg game, variable elimination, branch and bound

AMS(MOS) subject classifications. 90C27, 90D05

1. Introduction. In many situations, multiple decision makers with divergent objectives intervene in the decisions to be made (Wendell [38]). The simplest such case, in which there are only two decision makers, has long been studied in game theory. If there is some asymmetry between the decision makers in that one of them, called the *leader*, makes his decisions first, anticipating the reaction of the other one, called the *follower*, and cooperation is ruled out a priori, one has a Stackelberg game. Adding joint constraints on the strategies of the leader and the follower makes the model more realistic. This leads to *bilevel programming*, a topic which has attracted much attention recently, mostly in the linear case (see Fortuny-Amat and McCarl [19]; Candler and Townsley [16]; Papavassilopoulos [34]; Bard [5]; Bialas and Karwan [10], [11]; Ünlü [37]; Dempe [17]; Bard and Moore [9]; Judice and Faustino [27], [28]; Ben-Ayed and Blair [12]; Haurie, Savard, and White [24]; Anandalingam and White [4]). The nonlinear case and extensions to three or more levels are treated in a few papers (e.g., Aiyoshi and Shimizu [1], [2]; Bard [6], [7]; Bard and Falk [8]; Bard and Moore [9]; Al-Khayyal, Horst, and Pardalos [3]). Related bilevel and multilevel decision models and algorithms have also been extensively studied in control theory.

Applications of bilevel programming have been made in many domains; these include economic development policy (Candler and Norton [15]), agricultural economics (Candler, Fortuny-Amat, and McCarl [14]), road network design (Marcotte [30], Leblanc and Boyce [29]), and cogeneration of electrical energy (Savard [35], Haurie, Loulou, and Savard [23]). Mathematical programs with optimization problems in the constraints (e.g., Bracken and McGill [13]), arising in weapons allocation problems, can also be viewed as (nonlinear) bilevel programs.

The paper is organized as follows. In §2, the linear bilevel programming problem is stated mathematically. It is shown to be strongly NP-hard in §3. Our algorithm is

* Received by the editors May 1, 1989; accepted for publication (in revised form) July 2, 1991.

† Rutgers Center for Operations Research, Rutgers University, Hill Center for the Mathematical Sciences, New Brunswick, New Jersey 08903. The work of this author was supported in part by Air Force Office of Scientific Research grants 0271 and 0066 to Rutgers University, and done in part during two visits to GERAD, Montréal.

‡ Groupe d'Études et de Recherche en Analyse des Décisions and École Polytechnique de Montréal, Case Postale 6079, Succursale "A," Montréal, Canada, H3C 3A7. The work of this author was supported by a Natural Sciences and Engineering Research Council of Canada grant GP0036426 and a Fonds pour la Formation de Chercheurs et l'Aide à la Recherche of Québec grant 90NC0305.

§ Groupe d'Études et de Recherche en Analyse des Décisions and Collège Militaire Royal de Saint-Jean, Saint-Jean-sur-Richelieu, Canada, J0J 1R0. The work of this author was supported by Programme de Recherche Universitaire du Ministère Canadien de la Défense grant FUHBO.

stated in §4 and illustrated in an example in §5. Computational experience is reported in §6.

2. Formulation. The linear bilevel program (LBP) may be written in the following general form:

$$\text{(1)} \qquad \underset{x}{\text{maximize}} \quad z_1(x, y(x)) = c^1x + d^1y(x)$$

subject to:

$$\text{(2)} \qquad A^1x + B^1y(x) \leq b^1,$$

$$\text{(3)} \qquad x \geq 0;$$

$$\text{(4)} \qquad \underset{y}{\text{maximize}} \quad z_2(x, y) = c^2x + d^2y,$$

subject to:

$$\text{(5)} \qquad A^2x + B^2y \leq b^2,$$

$$\text{(6)} \qquad y \geq 0,$$

where

$$\begin{array}{ccc} c^{1T}, c^{2T}, x \in \mathbb{R}^{n_1}; & d^{1T}, d^{2T}, y \in \mathbb{R}^{n_2}; & A^1 \in \mathbb{R}^{m_1} \times \mathbb{R}^{n_1}; \\ A^2 \in \mathbb{R}^{m_2} \times \mathbb{R}^{n_1}; & B^1 \in \mathbb{R}^{m_1} \times \mathbb{R}^{n_2}; & B^2 \in \mathbb{R}^{m_2} \times \mathbb{R}^{n_2}; \\ b^1 \in \mathbb{R}^{m_1}; & b^2 \in \mathbb{R}^{m_2}. & \end{array}$$

For simplicity, we assume the polyhedra defined by (3), (5), and (6) and by (2), (3), (5), and (6) to be nonempty and bounded. The constraints on $y(x)$ implicitly defined by (4)–(6) can be represented as

$$\text{(7)} \qquad y(x) \in \arg\max\{z_2(x, y) \, ; \, y \in \Omega^2(x)\},$$

where

$$\text{(8)} \qquad \Omega^2(x) = \{y \, : \, B^2y \leq b^2 - A^2x, y \geq 0\}.$$

A solution $(x, y(x))$ such that $y(x)$ satisfies (7) and (8) is called *rational.* The set of rational solutions of LBP is, in general, not convex and, in the presence of first-level constraints (2), may be empty (in which case LBP has no solution). If the set of rational solutions is nonempty, at least one optimal solution of LBP is obtained at an extreme point of the polytope defined by (2), (3), (5), and (6). This important property was proven by Candler and Townsley [16], Bialas and Karwan [10], and Bard [5]. (Their proofs are for the case in which there are no first-level constraints (2), but readily extend to the case where such constraints are present.)

The leader controls the variables of x, and the follower, those of y, which are chosen after those of x; i.e., by solving the follower's subproblem (4)–(6). The model (1)–(6), due to Aiyoshi and Shimizu [1], contains first-level constraints that are binding only for the leader but depend also on the decisions of the follower. Such constraints arise, for instance, when the leader, a central agency, decides upon public investments and has to bear the cost of their maintenance, which depends on the level of use, decided upon by the follower, i.e., the users. Another instance is the problem of cogeneration of electricity by the leader, a central agency, and by the follower, i.e., independent firms (Savard [35], Haurie, Loulou, and Savard [23]). Electricity produced by the latter is bought at marginal cost. If the constraint on demand satisfaction is introduced at the second level, the optimal decision of the central agency is to produce no electricity at all, which is clearly unrealistic. Introducing the demand constraint at the first level requires the leader to produce a sufficient amount of electricity for the remaining demand to be satisfied through voluntary decisions of the follower.

Some recent algorithms for LBP, e.g., those of Bard and Moore [9], which do not allow for y variables in the first-level constraints, can easily be modified to handle them (see §6).

Algorithms for linear bilevel programming may be grouped as follows:

(i) Branch-and-bound algorithms, in which branching is done on complementarity conditions (Fortuny-Amat and McCarl [19], Bard and Falk [8], Bard and Moore [9]).

(ii) Extreme point-ranking methods (Papavassilopoulos [34], Bialas and Karwan [11]).

(iii) Algorithms based on finding efficient solutions for the leader's and follower's objective functions (Bard [5], Ünlü [37]). These algorithms should be viewed as heuristics, as the set of efficient solutions may contain no optimal solution (see, e.g., Haurie, Savard, and White [24]).

(iv) Algorithms using complementary pivoting (Candler and Townsley [16]; Bialas and Karwan [11]; Judice and Faustino [27], [28]). The algorithm of Bialas and Karwan [11] does not always provide an optimal solution, as shown in Judice and Faustino [27]; the latter authors' parametric method provides an ε-optimal solution for any given ε.

(v) Penalty function methods (Dempe [17], Anandalingam and White [4]).

(vi) Reverse convex programming (Al-Khayyal, Horst, and Pardalos [3]).

The best computational results to date appear to be those of Bard and Moore [9] and of Judice and Faustino [27], [28].

In this paper we propose a new algorithm of class (i). It exploits necessary conditions for subsets of the follower's constraints to contain at least one constraint that is tight at the optimum. These conditions are new in linear bilevel programming, although similar ones have been used in global optimization in design. They allow fathoming of subproblems for which they cannot be satisfied and lead to better bounds, as well as penalties, similar to those used in mixed-integer programming. We also investigate many branching rules, some of them based on logical relations expressing the conditions on the tightness of constraints that have been detected.

For fixed $\overline{x}$ the follower's subproblem (4)–(6) has a single optimal solution $\overline{y}$ in the absence of dual degeneracy. Otherwise there is an infinity of optimal $\overline{y}$'s, not all of which need to be feasible or of equal value in the leader's problem. Then some further assumptions have to be made on the willingness of the follower to cooperate with the leader in order for the problem to be well defined. From the point of view of the leader, the best case is when the follower accepts the leader's preferences regarding the y_j's for which he (the follower) is indifferent (the tie cooperative case) and the worst case is when the follower adopts the opposite of the leader's preferences regarding these y_j's (the tie noncooperative case). In this paper, we present an algorithm for the tie cooperative case, and indicate briefly how it can be modified to solve the tie noncooperative case.

As cooperation between the leader and the follower is ruled out, except in the case of ties for the follower's objective function, the optimal solution of LBP need not be an efficient one. In other words, there might be another feasible solution with higher value for both objective functions.

The LBP includes as particular cases the *linear max-min* (LMM) problem and the bilinear programming problem and is equivalent to some cases of concave programming and of nonconvex quadratic programming (see Pardalos and Rosen [33] for definitions and references).

3. Complexity.

3. Complexity. Jeroslow [26] has shown that LBP is NP-hard, and a shorter proof has recently been given by Ben-Ayed and Blair [12]. We next show that this result can be strengthened; we consider the LMM problem, a more restricted model than LBP. LMM is obtained from LBP by omitting constraints (2) and setting $c^2 = -c^1$, $d^2 = -d^1$, as pointed out by Bard and Falk [8]. For definitions, see Garey and Johnson [20].

THEOREM 3.1. LMM *is strongly* NP*-hard.*

Proof. We use reduction to KERNEL, which has been proved to be NP-hard by Chvâtal (see Garey and Johnson [20, p. 204]). Recall that a kernel K of a graph $G = (V, E)$ is a vertex set that is stable (no two vertices of K are adjacent) and absorbing (any vertex not in K is adjacent to a vertex of K). Then consider the LMM

$$\max_x \min_y z = -\sum_{j=1}^{n} y_j, \tag{9}$$

$$\text{s.t.} \quad x_j + x_k \leq 1 \qquad \forall\, j,k|\{v_j,\, v_k\} \in E, \tag{10}$$

$$y_j \leq 1 - x_j \quad \forall\, j, \tag{11}$$

$$y_j \leq 1 - x_k \quad \forall\, j,k|\{v_j,\, v_k\} \in E, \tag{12}$$

$$x_j, y_j \geq 0 \qquad \forall\, j, \tag{13}$$

where variables x_j and y_j are both associated with the vertex v_j of G, for $j = 1, 2, \cdots, n$. We claim that G has a kernel if and only if the optimal solution to (9)–(13) has a value $z^* = 0$. Assume first that G has a kernel K. Then set $x_j = 1$ for all $j \,|\, v_j \in K$, $x_j = 0$ otherwise. This solution satisfies (10) as K is stable. It imposes $y_j = 0$ for all j through (11) and (12) as K is absorbing; so $z^* = 0$.

Assume next that G has no kernel. Then any optimal solution in which all x_j^*'s are integers defines a stable set $S = \{v_j \,|\, x_j^* = 1\}$. As this stable set cannot be absorbing, at least one y_j^* must be equal to 1 and $z^* \leq -1$. Moreover, any nonintegral optimal solution must contain at least one x_j^* taking a positive fractional value, and because of (10) no x_k^* such that $(v_j,\, v_k) \in E$ can be equal to 1. So y_j^* takes a positive value and $z^* < 0$. As KERNEL is not a number problem, this completes the proof. □

COROLLARY 3.2. LBP *is strongly* NP*-hard.*

Recall that a *fully polynomial approximation scheme* provides an ε-optimal solution to a given problem in time polynomial in the problem size and in ε. It follows from the above corollary that no fully polynomial approximation scheme can be found for LBP unless $P = NP$.

4. Algorithm.

4.1. Necessary conditions for optimality and penalties.

We next derive necessary optimality conditions, expressed in terms of tightness of constraints in the follower's subproblem, i.e., constraints (5) and (6). To this effect we associate with each such constraint a boolean variable α_i equal to 1 if the constraint is tight, and equal to 0 otherwise. These conditions will play a basic role in the branch-and-bound algorithm described in §4.2

THEOREM 4.1. *In any rational solution to* LBP *the tightness of the constraints in the follower's subproblem is such that*

$$\sum_{i|B_{ij}^2>0} \alpha_i \geq 1 \quad \textit{if } d_j^2 > 0, \tag{14}$$

$$\sum_{i|B_{ij}^2<0} \alpha_i + \alpha_{m_2+j} \geq 1 \quad \textit{if } d_j^2 < 0 \tag{15}$$

for $j = 1, 2, \cdots, n_2$.

Proof. Assume that by contradiction there is a rational solution $(\hat{x}, \hat{y})$ to LBP such that (14) does not hold for some $j \in \{1, 2, \cdots, n_2\}$ such that $d_j > 0$ or (15) does not hold for some $j \in \{1, 2, \cdots, n_2\}$ such that $d_j < 0$. In the former case, increasing the value of $\hat{y}_j$ by

$$\Delta\hat{y}_j = \min_{i|B_{ij}^2>0} \left\{ \frac{1}{B_{ij}^2} \cdot \left(b_i^2 - \sum_{k=1}^{n_1} A_{ik}^2 \hat{x}_k - \sum_{\substack{k=1 \\ k \neq j}}^{n_2} B_{ik}^2 \hat{y}_k \right) \right\} \tag{16}$$

yields a solution satisfying constraints (5) for $x = \hat{x}$ and (6) with a value larger by $d_j \Delta\hat{y}_j$ than that of $(\hat{x}, \hat{y})$. This contradicts $(\hat{x}, \hat{y})$ being rational. In the latter case, decreasing the value of $\hat{y}_j$ by

$$\Delta\hat{y}_j = \min \left[\hat{y}_j, \min_{i|B_{ij}^2<0} \left\{ \frac{1}{B_{ij}^2} \left(\sum_{k=1}^{n_1} A_{ik}^2 \hat{x}_k + \sum_{\substack{k=1 \\ k \neq j}}^{n_2} B_{ik}^2 \hat{y}_k - b_i^2 \right) \right\} \right] \tag{17}$$

yields a solution satisfying constraints (5) for $x = \hat{x}$ and (6) with a value larger by $-d_j \Delta\hat{y}_j$ than that of $(\hat{x}, \hat{y})$. This again contradicts $(\hat{x}, \hat{y})$ being rational. □

COROLLARY 4.2. *In any optimal solution to* LBP *the tightness of the constraints in the follower's subproblem is such that conditions* (14) *and* (15) *are satisfied for all* $j \in \{1, 2, \cdots, n_2\}$ *such that* $d_j > 0$ *and* $d_j < 0$, *respectively.*

Proof. This follows immediately from the fact that optimal solutions to LBP are rational. □

A weaker condition involving tightness of constraints was obtained by Falk [18] for the LMM problem. Falk indeed noted that at least one constraint in the follower's subproblem must be tight at the optimum, i.e., that

$$\sum_{i=1}^{m_2} \alpha_i + \sum_{j=1}^{n_2} \alpha_{m_2+j} \geq 1. \tag{18}$$

Falk [18] showed that using this relation at the first node while solving by a branch-and-bound algorithm does reduce computational effort, but he did not apply it for subproblems obtained by branching. We next discuss branching for the LBP. This leads us to show how further relations of type (14) or (15) can be obtained for all subproblems.

In all rules studied, branching is done by fixing some binary variable(s) α_i at 0 or at 1. Either a single variable α_i will be chosen for that purpose (dichotomous branching), or a logical relation (14) or (15), e.g., $\alpha_{i_1} + \alpha_{i_2} + \alpha_{i_3} + \cdots \geq 1$ will be chosen and branching will be done according to the rule $\alpha_{i_1} = 1$ or ($\alpha_{i_1} = 0$ and $\alpha_{i_2} = 1$) or ($\alpha_{i_1} = \alpha_{i_2} = 0$ and $\alpha_{i_3} = 1$) and so on (multiple branching).

If $\alpha_i = 1$, the ith constraint (in (5) or (6)) in the follower's subproblem becomes an equality. It can then be used to eliminate one of the follower's variables y_j (which is y_{i-m_2} in case of a constraint of type (6)). New logical relations of type (14) and (15) can then be derived.

Of course, variable elimination does not reduce the number of structural constraints in that the nonnegativity constraint $y_j \geq 0$ for the eliminated variable is replaced by an inequality involving several variables. If there are several possible choices for y_j, the variable with the smallest fill-in, i.e., which least augments the number of nonzero coefficients, is chosen.

If a subproblem is obtained in which no more y variables remain, its optimal solution is found by solving the problem obtained by deleting the objective function (4) of the follower (called leader relaxation below and denoted LR), as the leader controls all remaining variables. However, it will be necessary to check whether this solution is rational or not. This will be the case if the value of the follower's objective function, when the x variables are fixed at their values in the optimal solution of LR, is the same for the y_j values obtained from the tight constraints used to eliminate them, and for the optimal y_j values for the follower's subproblem.

If $\alpha_i = 0$, the ith constraint (15) in the follower's subproblem becomes a strict inequality, and from the complementary slackness theorem of linear programming, the ith variable in the dual of the follower's subproblem must be equal to 0. As it is not easy to handle constraints stating that a variable is strictly positive in linear programming, the dual of the follower's subproblem will be solved instead of the primal in one test of the algorithm described below. When many variables α_i are fixed at 0, this dual problem may be infeasible, in which case, from the duality theorem, the primal follower's subproblem together with the strict positivity constraints is also infeasible, as by assumption it is bounded.

In practice, variable elimination can be performed by pivoting in a simplex tableau and fixing the eliminated variable, which leaves the basis at 0. This is easily implemented in a linear programming package such as Marsten's XMP [31].

In the branch-and-bound algorithm described in the next subsection, depth-first search is used and the subset of logical relations (denoted by R) is updated after either branching or backtracking.

When a branch corresponding to $\alpha_i = 0$ is explored, all logical relations involving α_i are simplified by deleting α_i. When a branch corresponding to $\alpha_i = 1$ is explored, all logical relations involving α_i are deleted as they are trivially satisfied. Then new relations (14) or (15) are obtained after a variable y_j has been eliminated. Finally, redundant relations are eliminated from R. (Recall that a logical relation $r_k \equiv \sum_{i\in I_k} \alpha_i \geq 1$, where I_k denotes the set of indices of the logical variables in r_k, of R is redundant if R contains another logical relation $r_\ell \equiv \sum_{i\in I_\ell} \alpha_i \geq 1$ such that $I_\ell \subseteq I_k$, i.e., satisfaction of $r_\ell \geq 1$ implies that $r_k \geq 1$ is satisfied as all variables α_i appearing in r_ℓ appear also in r_k.) When backtracking occurs, we revert to the set R corresponding to the last explored node for which one branch is unexplored and then explore this branch, updating R accordingly.

When branching is done on the values of the α_i, at most $2^{m_2+n_2+1}-1$ subproblems will be generated. If multiple branching is used the number of subproblems will be less, as subproblems with $\alpha_i = 0$ for all $i \in I_k$ for relations $r_k \in R$ used for branching are excluded.

As in many other algorithms for LBP, linear programming will be used to obtain bounds on the optimal value. To this effect, the objective functions (4) of the follower will be deleted (in the original problem or in the current subproblem in which some variables y_j have been eliminated). Solving the so-obtained linear program LR gives such an upper bound (denoted z_L^*). Then the effect of fixing a variable α_i at 1, i.e., satisfying a constraint as an equality, can be anticipated to some extent by computing a penalty as in mixed-integer programming (see, e.g., Taha [36]). Consider the equations corresponding to an optimal tableau of the LR of the current subproblem

$$z = z_L^* - \sum_{j \in N} (z_j^* - c_j)x_j,$$

$$x_i = b_i^* - \sum_{j \in N} A_{ij}^* x_j, \qquad i \in B,$$

where B denotes the index set of basic and N the index set of nonbasic variables. Then if x_i is the slack variable of the ith constraint the down penalty for setting x_i at 0 is

$$p_i = b_i^* \min_{j \in N | A_{ij}^* > 0} \left\{ \frac{z_j^* - c_j}{A_{ij}^*} \right\}.$$

It corresponds to the decrease in the value of z_L during the first dual-simplex iteration after adding the constraint $x_i \leq 0$. Moreover, if $r_k \equiv \sum_{i \in I_k} \alpha_i \geq 1$ is a logical relation of type (14) or (15) that must be satisfied by any rational solution, then at least one of the slack variables x_i $(i \in I_k)$ must be set at 0. It follows that

$$z_L^{*'} = z_L^* - \min_{i \in I_k} p_i$$

is an upper bound on the optimal value of the current subproblem. Finally, taking into account all logical relations of type (14) or (15) leads to a stronger upper bound:

$$z_L^{*''} = z_L^{*'} - \max_{k | r_k \in R} \min_{i \in I_k} p_i.$$

4.2. Tests. Before giving the rules of the new algorithm, we recall and illustrate a classification of tests for branch-and-bound methods (Hansen, Jaumard, and Lu [21], [22]). Viewed abstractly, any test of a branch-and-bound algorithm consists in examining whether a sufficient condition for some proposition about the current subproblem to be true is satisfied or not, and making use of this proposition when it is the case to fathom or simplify the subproblem.

A *direct test* is such that the information provided by the proposition when the condition holds is all that is required for the solution of the current subproblem. This is the case when it can be shown: (i) that a known solution x^* is the globally optimal solution of the subproblem (*direct resolution test*); or (ii) that the subproblem has no solution better than the best one known, called *incumbent* (*direct optimality test*); or (iii) that the subproblem has no feasible solution (*direct feasibility test*).

One resolution test for LBP, which applies to subproblems in which all variables y_j have been eliminated, consists in solving LR for x_L^* and then checking whether the solution (x_L^*, y_L^*), where y_L^* is obtained by using the equations of elimination backwards, is rational or not. If it is rational, (x_L^*, y_L^*) is the optimal solution of the current subproblem. A stronger version of this test, in which some y_j may remain in LR, is given below. One direct optimality test for LBP consists of solving LR and checking if its optimal value z_L^* exceeds the value z_{opt} of the best solution found so far. One direct feasibility test for LBP, discussed above, consists in checking whether the dual of the follower's relaxation (FR) is feasible or not.

A *conditional test* is such that the information provided by the proposition when the condition holds is all that is required for the solution of the current subproblem when a variable is fixed at a given value (here a boolean variable α_i fixed at the value 1).

One conditional optimality test for LBP consists in solving LR, computing the penalties p_i for loose constraints (i.e., those with strictly positive slack variables in the basis) and checking if $z_L^* - p_i \leq z_{\text{opt}}$. If this last inequality holds, no better solution than the incumbent one can be found for the current subproblem with the ith constraint being tight (i.e., $\alpha_i = 1$), so α_i may be fixed at 0.

A *relational test* is such that the information provided by the proposition when the condition holds is all that is required for the solution of the current subproblem if some algebraic or (here) logical relation is not satisfied.

Applying Theorem 4.1 yields logical relations (14) and (15) for all y_j remaining in the current subproblem; if all α_i in one of these relations are equal to 0, the current subproblem contains no rational solution and is fathomed.

We next describe a depth-first branch-and-bound algorithm. The current subproblem is characterized by: (i) objective functions and constraints of type (1)–(6) in x and in the remaining y_j variables; (ii) the vector α specifying which of the initial constraints are tight, loose, or of unknown tightness; (iii) the logical relations obtained from monotonicity; and (iv) the list of eliminated variables and the equalities defining their values. This subproblem may be written as follows:

$$\max_x \quad \tilde{z}_1(x,\tilde{y}) = \tilde{c}^1 x + \tilde{d}^1 \tilde{y}, \tag{19}$$

subject to:

$$\tilde{A}^1 x + \tilde{B}^1 \tilde{y} \leq \tilde{b}^1, \tag{20}$$

$$x \geq 0; \tag{21}$$

$$\max_{\tilde{y}} \quad \tilde{z}_2(x,\tilde{y}) = \tilde{c}^2 x + \tilde{d}^2 \tilde{y}, \tag{22}$$

subject to:

$$\tilde{A}^2 x + \tilde{B}^2 \tilde{y} \leq \tilde{b}^2, \tag{23}$$

$$\tilde{y} \geq 0, \tag{24}$$

where $\tilde{y}$ denotes the vector of remaining y_j variables and the tilde indicates that the coefficients in (19)–(24) are those obtained by eliminating the variables of $y \setminus \tilde{y}$. In addition, the current subproblem is defined by $\alpha = (\alpha_1, \cdots, \alpha_{m_2+n_2})$, where the α_i can be fixed at 0 or at 1 or be free, and $R = \{r_k, k \in K\}$, where $r_k \equiv \sum_{i \in I_k} \alpha_i \geq 1$ with $I_k \subseteq \{1, 2, \cdots, m_2 + n_2\}$, as well as the linear equations used to eliminate the variables of $y \setminus \tilde{y}$.

As discussed above, elimination of variables can be done by pivoting and fixation of nonbasic variables at 0. For simplicity of exposition we will not distinguish between eliminated and noneliminated variables when stating the rules of the algorithm given below.

We use two relaxations of the subproblem: the *leader's relaxation* (LR) obtained, as explained above, by omitting (22), and the *follower's relaxation* (FR), which consists of (22)–(24) for x fixed to $\overline{x}$. We will also consider the *follower's subproblem* (FS), which consists of (4)–(6), for $x = \overline{x}$.

ALGORITHM HJS

a. Initialization.
Obtain an initial solution (x_h, y_h) with a heuristic (the choice of which is discussed below). Initialize the incumbent solution $(x_{\text{opt}}, y_{\text{opt}})$ to (x_h, y_h) and the incumbent value z_{opt} to $c^1 x_{\text{opt}} + d^1 x_{\text{opt}}$. If no heuristic solution can be found, initialize $(x_{\text{opt}}, y_{\text{opt}})$ to an arbitrary value and set $z_{\text{opt}} = -\infty$.
Consider all variables α_i $(i = 1, 2, \cdots, m_2 + n_2)$ to be free. Set $R = \emptyset$.

b. First direct optimality test.
Solve LR; let (x_L^*, y_L^*) denote an optimal solution.
If $z_L^* = c^1 x_L^* + d^1 y_L^* \leq z_{\text{opt}}$, go to step m (backtracking).

c. First direct feasibility test.
Solve the dual of FR(x). If it has no feasible solution, go to step m.

d. Direct resolution test, first part.
Consider again the optimal solution (x_L^*, y_L^*) of LR.
Check if (x_L^*, y_L^*) is rational for the current subproblem: solve FR(x_L^*), and let y_F^* be an optimal solution. If $d^2 y_L^* = d^2 y_F^*$ then (x_L^*, y_L^*) is rational; otherwise go to step f.

e. Direct resolution test, second part.
Consider again the optimal solution (x_L^*, y_L^*) of LR.
Check if (x_L^*, y_L^*) is rational for the initial problem: solve FS(x_L^*), and let y_{FS}^* be an optimal solution.
If $d^2 y_L^* = d^2 y_{FS}^*$ then (x_L^*, y_L^*) is rational; otherwise go to step f.
Update z_{opt} and $(x_{\text{opt}}, z_{\text{opt}})$ if $z_{\text{opt}} < c^1 x_L^* + d^1 y_L^*$ and go to step m.

f. Second direct optimality test.
Compute all penalties p_i associated with strictly positive slack variables in the optimal tableau of LR. Set the other p_i equal to 0. Then for all k such that $r_k \in R$, compute

$$\pi_k = \min_{i \in I_k} p_i,$$

and set

$$\Pi = \max_k \pi_k.$$

If $z_{\text{opt}} \geq z_L^* - \Pi$, go to step m.

g. Second direct feasibility test.
If LR is infeasible, go to step m.

h. First conditional optimality test.
Consider again LR with the penalties p_i. For all i such that $z_{\text{opt}} \geq z_L^* - p_i$, fix α_i at 0 and update R.

i. Third direct optimality test.
If R contains a relation r_k such that $\alpha_j = 0$ for all $j \in I_k$, go to step m.

j. Relational optimality test.
For all remaining y_j appearing in $z_2(x, y)$, add to R the logical relations (14) or (15) on the α_i, if they are nonredundant. Eliminate from R those relations that have become redundant.

k. Second conditional optimality test.
If R contains a relation r_k such that $\alpha_j = 0$ for all $j \in I_k$ except for one index i, set the corresponding α_i to 1. Eliminate from the subproblem a variable y_j remaining in the ith constraint such that fill-in is minimum and return to step b.

l. Branching.
Apply the selected branching rule (the choice of which is discussed in §5) to choose either a free variable α_i or a relation $r_k \in R$ for which all variables α_i with $i \in I_k$ are free. In the former case, branch by fixing α_i at 1. In the latter case, branch by fixing the first variable α_i in r_k equal to 1. Eliminate a variable y_j remaining in the ith constraint such that fill-in is minimum. Return to step b.

m. Backtracking.
If branching took place on a variable, find the last α_i branched upon and equal to 1, set this $\alpha_i = 0$ and free the α_j fixed at 0 after α_i was fixed at 1. Otherwise consider the last logical relation r_k for which less than $|I_k|$ branches have been explored; consider the next branch. If there is no such variable or relation, stop. Otherwise update the current subproblem and return to step b.

Various heuristics may be used to obtain a first rational solution in the absence of first-level constraints. One of them, used in our implementation, consists of solving LR with the objective function $\lambda c^1 x + (1-\lambda) d^2 y$, i.e., a weighted sum of the leader's objective function for the x variables and of the follower's objective function for the y variables. The weight λ was chosen to be equal to

$$\frac{n_1 + n_2}{n_1 + 2n_2}.$$

Better heuristic solutions could be obtained at higher computational cost, e.g., by varying λ between 0 and 1 in the following objective function $(1-\lambda)(c^1x + d^1y) + \lambda d^2 y$ and keeping the first rational solution obtained as in Bard's algorithm [5], or by using Judice and Faustino's algorithm as they suggest in [28].

THEOREM 4.3. *Algorithm* HJS *solves* LBP *in a finite time.*

Proof. As mentioned in §2, LBP has an optimal solution at an extreme point of Ω, or has no solution. Branching on the tightness of the follower's constraints implies an implicit enumeration of all bases of FS, which are finite in number, as is the number of values for the vector α. Moreover, all steps take a finite time and the algorithm cannot cycle, as they are done in sequence unless simplification or branching takes place, which can happen only a finite number of times. □

As mentioned above, Algorithm HJS is for the tie cooperative case. Therein, ties in the values of the follower's objective function are automatically resolved as the leader wishes. This is ensured by LR: the objective function is that of the leader, so that if in tests d and e the solution (x_L^*, y_L^*) is found to be rational, it is the best from the leader's point of view among all rational solutions with $\overline{x} = x_L^*$. Algorithm HJS is easily adapted to solve the tie noncooperative case. Indeed, it suffices to add a secondary objective function in the follower's subproblem equal to $-d^1 y$ and activated only in case of ties for the objective function $d^2 y$, in a similar way as in preemptive goal programming (see, e.g., Ignizio [25]).

This modification, which affects tests d and e, has two consequences: first, rational solutions in case of ties for $d^2 y$ will usually differ from (x_L^*, y_L^*), so more branches will have to be explored; second, again in case of ties for $d^2 y$, the optimal solution may be modified.

5. An example. Consider the following two-level linear program (from Candler and Townsley [16], with an additional first-level constraint). The α_i are written in parentheses next to the corresponding constraints:

$$\max_x z_1 = 8x_1 + 4x_2 - 4y_1 + 40y_2 + 4y_3, \tag{25}$$

$$\text{subject to: } x_1 + 2x_2 - y_3 \le 1.3, \tag{26}$$

$$x_1 \ge 0, \quad x_2 \ge 0, \tag{27}$$

$$\max_y z_2 = -2y_1 - y_2 - 2y_3, \tag{28}$$

$$\text{subject to:}$$

$$(\alpha_1) \qquad -y_1 + y_2 + y_3 \le 1, \tag{29}$$

$$(\alpha_2) \qquad 4x_1 - 2y_1 + 4y_2 - y_3 \le 2, \tag{30}$$

$$(\alpha_3) \qquad 4x_2 + 4y_1 - 2y_2 - y_3 \le 2, \tag{31}$$

$$(\alpha_4) \qquad y_1 \ge 0, \tag{32}$$

$$(\alpha_5) \qquad y_2 \ge 0, \tag{33}$$

$$(\alpha_6) \qquad y_3 \ge 0. \tag{34}$$

Solve the problem:

$$\max_{x,\,y} z_R = \lambda c^1 x + (1-\lambda) d^2 y \quad \text{with } \lambda = \frac{n_1 + n_2}{n_1 + 2n_2} = \frac{5}{7},$$

i.e., $z_R = 5x_1 + 2.5x_2 - 1.5y_1 - 15y_2 + 1.5y_3$ subject to constraints (26)–(27) and (29)–(34) (step a). Its optimal solution $(x_R^*, y_R^*) = (1.5, 0, 1, 0, 2)$, with value $z_R^* = 9$, is feasible for problem (25)–(34). Therefore, $(x_{\text{opt}}, y_{\text{opt}})$ is initialized to (x_R^*, y_R^*) and z_{opt} to 16.

Solving the LR leads to $(x_L^*, y_L^*) = (0, 0, 1.5, 1.5, 1)$ and $z_L^* = 58 > z_{\text{opt}}$ (step b). As no constraint has yet been imposed on the dual variables of the follower's problem, it is feasible (step c). Setting $\overline{x}$ to x_L^*, we obtain $y_F^* = (0, 0, 0)$ and $z_2(y_F^*) = 0 \neq z_2(y_L^*) = -6.5$; hence (x_L^*, y_L^*) is not rational (step d).

The penalties p_i associated with constraints (29)–(34) are equal to:

$$p_1 = 0\,,\ p_2 = 0\,,\ p_3 = 0\,,\ p_4 = 28.8\,,\ p_5 = 42, \text{and } p_6 = 22 \qquad \text{(steps f and h)}.$$

The set R of logical conditions

$$\begin{aligned} \alpha_1 + \alpha_2 + \alpha_4 &\ge 1 \quad (\text{w.r.t. } y_1),\\ \alpha_3 + \alpha_5 &\ge 1 \quad (\text{w.r.t. } y_2),\\ \alpha_2 + \alpha_3 + \alpha_6 &\ge 1 \quad (\text{w.r.t. } y_3) \end{aligned}$$

are obtained (step j). We branch on the second relation and first consider $\alpha_5 = 1$ as $p_5 > p_3$. This leads to $y_2 = 0$ and the problem reduces to

$$\max_x z_1 = 8x_1 + 4x_2 - 4y_1 + 4y_3,$$

$$\text{subject to: } x_1 + 2x_2 - y_3 \le 1.3,$$

$$x_1 \ge 0, \qquad x_2 \ge 0;$$

$$\max_{y_1,\,y_3} z_2 = -2y_1 - 2y_3,$$

$$\text{subject to:}$$

$$\begin{aligned} -y_1 + y_3 &\le 1,\\ 4x_1 - 2y_1 - y_3 &\le 2,\\ 4x_2 + 4y_1 - y_3 &\le 2,\\ y_1 \ge 0, \qquad y_3 &\ge 0. \end{aligned}$$

The optimal solution $(x_L^*,\, y_L^*)$ of LR is (1.5, 0, 1, 2) with a value $z_L^* = 16 \le z_{\text{opt}}$ (step b).

The incumbent solution remains: $z_{\text{opt}} = 16$, $(x_{\text{opt}},\, y_{\text{opt}}) = (1.5,\ 0,\ 1,\ 0,\ 2)$.

Backtracking (step m) leads to the branch $\alpha_5 = 0$, $\alpha_3 = 1$. Variable y_1 is eliminated, leading to the subproblem

$$\begin{aligned} &\max_x z_1 = 8x_1 + 8x_2 + 38y_2 + 3y_3 - 2,\\ &\text{subject to: } x_1 + 2x_2 - y_3 \ge 1.3,\\ &\qquad x_1 \ge 0, \qquad x_2 \ge 0,\\ &\qquad \max_{y_2,\, y_3} z_2 = 2x_2 - 2y_2 - 2.5y_3 - 1,\\ &\qquad\quad \text{subject to:}\\ &\qquad\quad (\alpha_1) \qquad\qquad x_2 + 0.5y_2 + 0.75y_3 \le 1.5,\\ &\qquad\quad (\alpha_2) \qquad\qquad 2x_1 + x_2 + 1.5y_2 - 0.75y_3 \le 1.5,\\ &\qquad\quad (\alpha_4) \qquad\qquad x_2 - 0.5y_2 - 0.25y_3 \le 0.5,\\ &\qquad\quad (\alpha_5) \qquad y_2 > 0,\\ &\qquad\quad (\alpha_6) \qquad y_3 \ge 0. \end{aligned}$$

The current set of logical relations R (step j) is equal to $\{\alpha_4 \ge 1\}$ (with respect to y_2 in the above subproblem). Indeed the constraint $\alpha_2 + \alpha_4 + \alpha_6 \ge 1$, corresponding to y_3 in the current subproblem, is redundant and the three previous constraints of R are eliminated, two of them because they contain $\alpha_3 = 1$ and the third one because $\alpha_4 = 1$ makes it redundant. Tightness of the fourth constraint (step k), i.e., $\alpha_4 = 1$, allows us to eliminate y_2, leading to the subproblem

$$\begin{aligned} &\max_x z_1 = 8x_1 + 84x_2 - 16y_3 - 40,\\ &\text{subject to: } x_1 + 2x_2 - y_3 \le 1.3,\\ &\qquad x_1 \ge 0, \qquad x_2 \ge 0,\\ &\qquad \max_{y_3} z_2 = -2x_2 - 1.5y_3 + 1,\\ &\qquad\quad \text{subject to:}\\ &\qquad\quad (\alpha_1) \qquad 2x_2 + 0.5y_3 \le 2,\\ &\qquad\quad (\alpha_2) \qquad 2x_1 + 4x_2 - 1.5y_3 \le 3,\\ &\qquad\quad (\alpha_5) \qquad -2x_2 + 0.5y_3 \le -1,\\ &\qquad\quad (\alpha_6) \qquad y_3 \ge 0. \end{aligned}$$

A new logical relation is found : $\alpha_2 + \alpha_6 \ge 1$. Computing the optimal solution $(x_L^*,\, y_L^*)$ of LR, its value z_L^* and the penalties p_i yield $(x^*,\, y_L^*) = (0,\ .833,\ .466)$, $z_L^* = 26.73 > z_{\text{opt}}$, and $p_1 = 0$, $p_2 = 8.333$, $p_6 = 12.333$. Setting $\overline{x}$ to x_L^* in FR yields the solution $y_{FR}^* = (.3555)$ and $z_2(y_F^*) = -1.3 \ne z_2(y_L) = -1.466$; hence $(x_L^*,\, y_L^*)$ is not rational.

Penalty $p_6 = 12.33$ is such that $z_{\text{opt}} > z_L^* - p_6$; hence $\alpha_6 = 0$ (step h). The set R reduces to $\{\alpha_2 \ge 1\}$. Tightness of the second constraint allows us to eliminate y_3, the last of the follower's variables. This gives the following subproblem

$$\begin{aligned} \max_x z_1 &= -13.33x_1 + 41.33x_2 - 8 \\ \text{subject to: } & -x_1 - 2x_2 \le -2.1 \\ & x_1 \ge 0, \qquad x_2 \ge 0; \\ & \max_y \ z_2 = -2x_1 - 6x_2 + 4 \\ & \quad \text{subject to:} \\ & \qquad (\alpha_1) \quad 2x_1 + 10x_2 \le 9, \\ & \qquad (\alpha_5) \quad x_1 - x_2 \le 0, \\ & \qquad (\alpha_6) \quad -2x_1 - 4x_2 \le -3. \end{aligned}$$

Solving LR yields $x_L^* = (0.5, 0.8)$ and $z_L^* = 18.4$, which is trivially rational for the current subproblem. Moreover, setting $\overline{x}$ to x_L^* in FS yields the solution $y_{FS}^* = (0, .2, .8)$ with $z_2(y_{FS}^*) = z_2(y_L^*) = -1.8$. Thus the solution (0.5, 0.8, 0, 2, 0.8) is rational for the initial problem, and the incumbent vector and value are updated. Backtracking brings the algorithm to stop. The problem is thus solved after examining three nodes only (see Fig. 1).

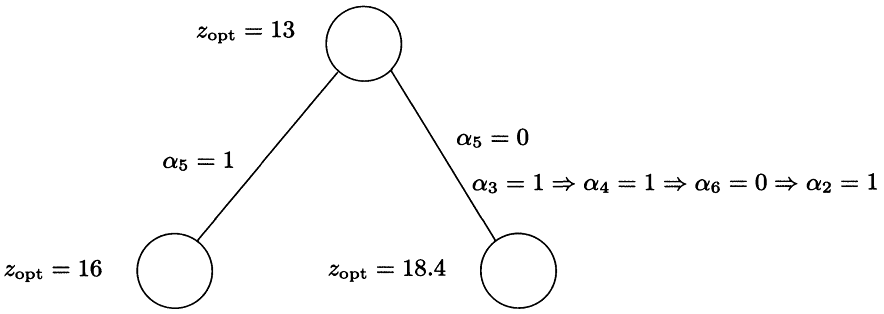


FIG. 1. *Branch-and-bound tree for the example.*

6. Computational experience. The algorithm of §4 has been coded in FORTRAN 77 and extensively tested on a SUN 3/50 and, for one series of tests, on a SUN SPARC computer. We also solved some test problems with the program of Bard and Moore [9] (referred to as BM below) on the same computer. Algorithm BM is similar to that of Fortuny-Amat and McCarl [19] in that both use branching on the complementary slackness conditions associated with the follower's subproblem. Fortuny-Amat and McCarl convert the complementary conditions into mixed-integer linear constraints before branching. The relaxation of the problem so obtained is less tight than in the BM algorithm. Bard and Moore consider the relaxation of LBP obtained by deleting the follower's objective function (i.e., program LR), then add the constraints of the dual of the follower's subproblem, solve in both the primal and dual variables, and branch on the complementary slackness conditions until they are satisfied. The program BM uses Marsten's code XMP [31] as a subroutine to solve the linear programs. We used the same code for that purpose.

TABLE 1
Comparison of branching criteria.

	n_1	42		42		42		42		60		50		70		60		70	
	n_2	28		28		28		28		40		50		50		60		60	
	m_1	0		0		0		7		0		0		0		0		0	
	m_2	28		28		28		21		40		40		48		48		52	
	d	40%		17%		8%		8%		8%		8%		8%		8%		8%	
		nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu
BR1	μ	4959.8	7830.4	416.6	250.3	37.5	18.7	3433.2	1754.6	1182.6	1045.1	2772.1	3984.5	4800.5	9098.4	—	—	—	—
	σ	4479.2	7576.9	1244.8	727.5	78.5	29.2	7823.8	3841.3	3089.3	2603.2	4046.2	6095.0	4293.5	7623.2	—	—	—	—
	m	2439	1922.9	3	6.7	9	7.7	122	90.0	11	27.4	191	480.5	5150	15000.0	—	—	—	—
BR2	μ	4290.8	6380.2	377.6	227.7	37.3	18.0	3210.2	1669.8	261.7	284.7	1631.4	2288.1	4518.0	9109.6	—	—	—	—
	σ	4675.4	7464.1	1132.0	663.4	78.6	29.7	7761.4	3953.9	481.3	463.8	3046.3	4602.9	4095.8	7611.4	—	—	—	—
	m	733	635.7	3	6.4	7	5.6	29	27.6	9	20.3	115	206.3	5113	15000.4	—	—	—	—
BR3	μ	2917.4	4575.7	649.0	653.5	917.8	402.1	3699.9	3111.0	752.8	925.4	2512.2	4367.0	3722.8	8478.0	—	—	—	—
	σ	4574.6	7194.8	1565.9	1595.5	1910.2	826.9	5820.0	4801.3	881.2	1144.7	2966.4	5428.2	3109.0	7175.6	—	—	—	—
	m	73	100.5	7	13.0	13	11.23	235	172.9	215	325.5	1081	2226.5	3715.0	7352.3	—	—	—	—
BR4	μ	34.6	50.1	6.0	8.7	5.2	4.7	13.4	15.5	6.2	17.6	56.4	138.0	30.0	113.2	29.4	157.4	65.4	341.2
	σ	48.4	66.0	7.0	7.5	3.5	2.3	18.9	17.2	3.5	9.5	129.3	257.0	32.2	103.4	45.2	212.1	112.8	520.0
	m	9	15.3	3	6.5	7	5.5	5	8.0	5	14.4	13	44.4	19	69.3	11	71.4	11	68.7
BR5	μ	32.6	47.5	5.4	7.9	5.4	4.6	10.8	13.0	6.4	18.0	17.6	56.8	30.4	112.3	24.0	126.0	26.2	154.8
	σ	44.1	59.9	5.2	5.7	3.8	2.4	10.2	9.8	3.6	9.4	12.8	35.7	31.3	98.5	29.4	126.6	26.2	140.8
	m	9	15.1	3	6.4	7	5.3	5	8.0	5	14.2	11	41.1	19	83.4	11	69.5	11	68.7
BR6	μ	72.2	96.7	5.6	8.2	5.4	4.6	14.4	15.8	7.6	20.6	22.6	67.0	34.2	121.3	36.4	174.6	22.6	133.7
	σ	108.3	132.7	5.3	5.7	3.8	2.3	19.1	17.6	5.0	13.0	14.9	42.8	29.4	95.6	50.6	205.9	20.3	109.4
	m	9	14.9	3	6.4	7	5.3	5	8.0	5	14.1	21	59.0	25	85.2	15	76.8	11	70.8
BR7	μ	1065.4	1317.1	6.0	8.6	6.0	5.0	76.8	73.2	8.0	20.6	28.2	76.1	56.8	193.3	46.2	206.9	33.6	185.3
	σ	1660.8	2094.3	6.5	6.9	4.5	2.5	160.9	151.6	5.7	13.0	22.2	53.5	66.3	200.6	62.4	253.1	39.9	201.5
	m	13	19.7	3	6.4	7	5.6	13	13.8	5	14.1	17	45.7	21	84.3	27	117.5	19	120.9

Problems were generated randomly with the same characteristics as in Bard and Moore [9] for 40 percent and 17 percent density and as in Savard [35] for sparser (8 percent and 6 percent density) ones. To avoid empty columns, coefficients were generated column after column with these densities. To avoid unbounded optimal solutions in the follower's subproblem if all coefficients in a column corresponding to a variable y_j with $d_j^2 > 0$ are negative, the first one is multiplied by -1; then boundedness is checked and unbounded problems deleted.

The main parameters considered are the numbers n_1 and n_2 of leader and follower variables, m_1 and m_2 of first-level and second-level constraints and density d of the coefficient matrix. For each set of these values in each series of tests, 10 problems are solved. Mean (μ) and median (m) values as well as standard deviations (σ) for the number of nodes in the branch-and-bound tree and for the cpu times are given.

A first class of experiments was aimed at streamlining the algorithm and assessing which of its features are the most useful. Numerous branching rules were tested. In Table 1 we present results for the seven best rules, which are described next. We denote by s_i the slack variable associated with constraint i of type (5) or (6) and by u_i the corresponding dual variable for $i = 1, 2, \cdots, m_2 + n_2$.

BR1. *Multiple branching.* (i) Select the logical relation $r_k \in R$ with smallest cardinality. (ii) Break ties by choosing a relation with the largest number of slack variables s_i in basis. (iii) Order the variables α_i in r_k by decreasing values of the penalties p_i.

BR2. Same as BR1, except for (iii). Order the variables α_i by decreasing values of the products $s_i u_i$.

BR3. *Binary branching.* Select the boolean variable α_i associated with the largest penalty p_i.

BR4. Select the α_i associated with the largest product $s_i u_i$ (same rule as in Algorithm BM).

BR5. *Multiple or binary branching.* (i) Select the relation $r_k \in R$ with two variables α_i and both the corresponding slack variables s_i in basis, such that $\sum_{i \in I_k} u_i s_i$ is maximum. (ii) If there is no such relation, use BR4.

BR6. Same as BR5 except that in (ii), branch on the variable α_i with largest penalty p_i among those for which $s_i u_i > 0$.

BR7. Same as BR5 except that in (ii), branch on the variable with smallest product $s_i u_i > 0$.

It appears that:

(i) numbers of nodes generated and computation times are extremely sensitive to the chosen branching rule;

(ii) multiple branching on logical relations involving few α_i is less efficient than dichotomous branching on the α_i;

(iii) best results are obtained by a hybrid rule in which multiple branching is first used and one switches to dichotomous branching when no more logical relations with few α_i are available;

(iv) with all branching rules, difficulty of resolution varies greatly from problem to problem: often $\sigma > m$ and $m << \mu$. Many problems are easy to solve, with one or a few nodes in the branch-and-bound tree, while a few take several thousand nodes.

As branching rules BR5 and BR6 gave the best results, they were adopted in the remaining experiments.

In a second series of tests the effect of heuristics was assessed by obtaining an upper bound on the reduction in computing effort they can provide in the following

TABLE. 2a
No heuristic.

	n_1	50		70		60		70	
	n_2	50		50		60		60	
	m_2	40		48		48		52	
	d	8%		8%		8%		8%	
		nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu
BM	μ	2863.2	2504.8	2111.7	3628.3	1981.6	4238.4	3658.3	7302.3
	σ	4968.3	4569.9	3114.2	4999.1	2844.7	5853.9	3462.3	6332.3
	m	396	330.2	518	1208.2	662	1420.5	2138.0	4040.9
BR5	μ	18.0	59.4	30.4	112.6	24.6	132.4	26.2	159.3
	σ	12.6	37.7	31.3	98.3	29.0	129.5	26.2	144.8
	m	13	53.2	19	83.3	11	69.0	11	73.2
BR6	μ	23.2	70.8	34.2	123.3	37.0	181.6	22.6	136.1
	σ	15.0	47.1	29.4	97.4	50.2	205.8	20.3	111.5
	m	21	63.7	25	85.6	15	77.8	11	75.0

TABLE 2b
HJS *heuristic.*

	n_1	50		70		60		70	
	n_2	50		50		60		60	
	m_2	40		48		48		52	
	d	8%		8%		8%		8%	
		nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu
BM	μ	2175.0	1991.2	1959.2	3421.9	1953.7	4209.6	3357.4	7303.1
	σ	4828.4	4525.3	3114.7	5010.3	2859.3	5872.3	2989.2	6332.5
	m	342	250.3	518	1205.6	466	1153.5	2138	4040.6
BR5	μ	17.6	56.8	30.4	112.3	24.0	126.0	26.2	154.8
	σ	12.8	35.7	31.3	98.5	29.4	126.6	26.2	140.8
	m	11	41.1	19	83.4	11	69.5	11	68.7
BR6	μ	22.6	67.0	34.2	121.3	36.4	174.6	22.6	133.7
	σ	14.9	42.8	29.4	95.6	50.6	205.9	20.3	109.4
	m	21	59.0	25	85.2	15	76.8	11	70.8

TABLE 2c
Using the optimal solution as first heuristic solution.

	n_1	50		70		60		70	
	n_2	50		50		60		60	
	m_2	40		48		48		52	
	d	8%		8%		8%		8%	
		nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu
BM	μ	876.0	833.6	1299.7	2700.1	1722.5	3719.7	2032.6	4675.5
	σ	1935.6	1822.3	2190.6	4574.7	2914.1	6011.2	2452.6	5643.5
	m	138	98.0	304.0	598.7	160	652.1	944	2068.6
BR5	μ	12.4	42.7	15.2	64.9	18.8	99.2	17.0	97.7
	σ	11.2	30.8	14.3	46.8	30.1	129.7	21.0	97.1
	m	7	27.6	9	424	3	31.1	9	57.7
BR6	μ	16.4	51.5	17.4	70.6	25.0	119.1	13.8	85.4
	σ	13.8	39.4	16.5	53.7	48.6	196.5	11.5	66.5
	m	11	38.2	9	42.4	5	33.9	9	57.6

way: a series of test problems solved previously was solved again, first using no heuristic and then assuming the optimal value to be known a priori. Results are given in Tables 2a, 2b, and 2c (lines corresponding to BM will be discussed later) and are to be compared with those of Table 1. It appears that:

(i) a priori knowledge of the best solution significantly decreases computational effort (average cpu times are reduced by 21.3–42.2 percent);

TABLE 3
Influence of penalties.

	n_1	42				42				42				60			
	n_2	28				28				28				40			
	m_2	28				28				28				40			
	d	40%				17%				8%				8%			
				fixations				fixations				fixations				fixations	
		nodes	cpu	rlog	pen	nodes	cpu	rlog	pen	nodes	cpu	rlog	pen	nodes	cpu	rlog	pen
BR6	μ	68.8	60.6	1.5	—	7.2	7.0	5.4	—	7.6	4.2	15.9	—	11.4	15.9	12.4	—
	σ	87.6	76.1	1.7	—	7.3	5.0	2.7	—	5.8	2.2	5.0	—	8.3	8.6	4.0	—
	m	29	24.1	1.0	—	5	5.7	5.0	—	7	4.1	15.0	—	7.0	10.9	12.0	—
BR6	μ	33.0	43.0	4.3	30.5	5.4	7.4	6.4	6.7	5.4	4.4	16.9	7.6	6.4	16.9	21.5	23.3
	σ	44.9	54.6	5.3	30.0	5.2	5.2	2.7	8.3	3.9	2.1	5.6	6.9	3.7	8.5	9.5	24.4
	m	9	13.7	3.0	20.0	3	6.1	7.0	3	7	5.0	14.0	8.0	5.0	13.3	20.0	14

	n_1	50				70				60				70			
	n_2	50				50				60				60			
	m_2	40				48				48				52			
	d	8%				8%				8%				8%			
				fixations				fixations				fixations				fixations	
		nodes	cpu	rlog	pen	nodes	cpu	rlog	pen	nodes	cpu	rlog	pen	nodes	cpu	rlog	pen
BR6	μ	41.8	64.4	30.4	—	53.2	102.8	22.3	—	44.2	126.6	19.0	—	82.6	268.7	24.9	—
	σ	37.5	48.7	16.4	—	50.2	85.7	12.9	—	48.2	124.2	7.2	—	105.9	331.4	19.4	—
	m	23	37.8	19.0	—	41	83.9	22.0	—	23	66.2	16.0	—	21	78.5	14	—
BR6	μ	17.6	53.6	51.3	49.6	30.4	103.9	48.0	75.9	24.0	119.0	40.3	66.2	26.2	145.7	36.7	57.10
	σ	12.8	33.3	25.8	38.0	31.3	90.4	33.1	57.6	29.4	119.2	28.6	56.0	26.2	132.0	25.1	55.21
	m	11	38.7	46.0	42.0	19	77.5	43.0	56.0	11	66.5	26.0	50.0	11.0	65.4	30.0	30.00

(ii) some problems remain difficult to solve;

(iii) the heuristic described in §5 only entails small reductions in numbers of nodes and cpu time.

So, there is room for improved heuristics (e.g., as discussed above, those of Bard [5] and Judice and Faustino [27], [28]) but heuristics alone will not always render the solution of LBP easy.

In the next series of tests the effect of penalties was assessed by solving the same series of problems without and with them. Results, given in Table 3, show that:

(i) using penalties significantly reduces the number of nodes in the branch-and-bound tree (the average reduction in number of nodes is between 25.0 and 68.3 percent);

(ii) cpu times are not much affected (sometimes slightly increased and sometimes slightly decreased) except for the largest test problems, in which case using penalties reduces mean cpu time by 45.8 percent.

Table 3 also gives statistics on the number of times a variable α_i is fixed using a logical relation (rlog) and using a test involving penalties (pen). Note that the number of fixations due to logical relations increases when penalties are used, as these penalties can show that some α_i must be equal to 0 and hence sharpen the logical relations in which these α_i appear.

A second class of experiments aims at studying which parameters influence the difficulty of resolution of LBP. Results of a sensitivity analysis on the right-hand side vector are given in Table 4. Each component of this vector is chosen to be equal to s times the fraction of the right-hand side minus the sum of the negative coefficients over the sum of the absolute values of the coefficients in the corresponding constraint; s is varied from 0.4 to 0.8. No clear-cut tendency appears to be detectable. Contrasting with this result, very large differences in problem difficulty are observed when the amount by which the leader's and follower's objective functions are antagonistic varies (this amount depends on the ranges and the signs of the coefficients in both objective functions). Table 5 shows results obtained when coefficients d_j^1 in the leader's objective function and coefficients d_j^2 in the follower's objective function are randomly chosen in a uniform distribution on $[\ell, 20]$ where ℓ varies from -20 to $+12$. It appears that:

(i) computational effort regularly decreases when the lower bound ℓ increases, i.e., when the leader's and follower's objective functions become closer to being parallel.

(ii) computational difficulty of the hardest and easiest problems is very different (averages 33.3–56.3 times more nodes and 12.3–19.6 times more cpu time in the former than in the latter case).

This suggests that in reporting future experiments on algorithms for LBP, a parameter such as ℓ specifying the amount of antagonism between the two objective functions should be systematically specified.

In another series of experiments the number of first-level variables was varied while the numbers of second-level variables and of constraints were kept constant. Results of Table 6 show that computational effort significantly *decreases* when the number of first-level variables *increases.* It thus appears that the number of follower's variables is not in itself a sufficient measure of computational difficulty for LBP.

A third class of experiments was aimed at comparing our algorithm with the best previous ones, those of Bard and Moore [9] and of Judice and Faustino [27], [28]. A direct comparison with the algorithm of Bard and Moore was possible, as they kindly made their code available to us. Results are reported in Table 2 and Tables 7a and

TABLE 4
Sensitivity analysis on the right-hand side vector.

		$n_1=50$, $n_2=50$, $m_1=0$, $m_2=40$, $d=8\%$						$n_1=60$, $n_2=60$, $m_1=0$, $m_2=48$, $d=8\%$					
		$s=.4$		$s=.6$		$s=.8$		$s=.4$		$s=.6$		$s=.8$	
		nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu
BR5	μ	53.4	137.3	47.0	111.7	64.2	161.3	188.2	706.6	213.0	787.7	196.6	754.4
	σ	76.0	156.9	80.9	147.7	111.8	258.3	471.4	1695.4	447.3	1534.3	336.4	1227.1
	m	19	61.9	19.0	57.1	19	58.5	19	101.4	55	284.3	45	231.0
BR6	μ	44.5	124.6	4.0	108.7	62.3	152.2	249.1	868.6	220.2	790.3	297.4	1047.4
	σ	45.9	118.8	52.8	116.4	99.4	249.0	466.2	1443.4	324.9	1016.5	533.4	1617.5
	m	19	75.3	19	57.5	19	75.7	43	196.0	95	413.8	43	212.1

TABLE 5
Sensitivity analysis on the coefficients of the objective function.

	$n_1=50$ $n_2=50$ $m_1=0$ $m_2=40$ $d=8\%$	$c\in[-20,20]$		$c\in[-12,20]$		$c\in[-4,20]$		$c\in[0,20]$		$c\in[4,20]$		$c\in[12,20]$	
		nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu
BR5	μ	63.3	149.0	25.1	66.7	13.4	40.0	3.8	16.4	3.3	15.2	1.9	12.1
	σ	63.6	135.9	29.9	71.1	29.2	69.9	5.5	14.6	4.1	12.8	1.5	4.8
	m	29	82.5	9	27.5	5	20.2	3	14.6	1	10.3	1	9.7
BR6	μ	106.5	237.3	24.1	65.1	10.3	33.1	4.2	17.2	2.9	13.9	1.9	12.1
	σ	155.1	329.2	28.3	69.2	19.9	48.6	7.0	17.7	2.8	8.7	1.5	4.8
	m	31	90.9	9	33.2	5	20.1	3	14.5	1	10.2	1	9.6

TABLE 6
Increasing the number of first-level variables.

	n_1	50		75		100		150		200		250		300		350	
	n_2	60		60		60		60		60		60		60		60	
	m_2	100		100		100		100		100		100		100		100	
	d	8%		8%		8%		8%		8%		8%		8%		8%	
		nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu
BR5	μ	619.8	908.5	64.8	126.9	141.0	286.7	12.0	35.5	20.2	86.5	9.0	37.1	7.8	38.0	4.0	27.5
	σ	966.4	1377.1	83.9	156.2	232.1	466.9	10.1	26.7	23.6	131.1	15.1	48.6	7.4	27.6	4.2	18.4
	m	221	333.6	15	30.4	49	112.9	7	25.6	13	51.7	3	21.9	5	25.1	1	19.0

7b. It appears that:

(i) Algorithm HJS with branching rule BR5 always outperforms Algorithm BM and does the same with branching rule BR6 with two exceptions;

(ii) The ratios of computation times for Algorithms BM and HJS with branching rule BR5 are between 1.4 and 9.6 when $d = 40$ percent, 3.5 and 49.0 when $d = 17$ percent, 23 and 47.1 when $d = 8$ percent, but vary widely in each of these cases;

(iii) This ratio increases with problem size;

(iv) Similar conclusions hold even if no heuristic is used or if the optimum value of LBP is assumed to be known *a priori* (see Table 2).

Reasons for which Algorithm HJS is faster than Algorithm BM appear to be the following (probably in decreasing order of importance):

(i) use of logical relations (14) and (15); for sparse problems some of these logical relations may have only one α_i and thus lead to problem simplification and possibly to complete solution without branching;

TABLE 7a
Comparison of HJS *and* BM *algorithms on problems without first-level constraints.*

	n_1	20		50		42		35		60		45	
	n_2	30		20		28		35		30		45	
	m_2	20		28		28		28		36		36	
		nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu
40%	μ	175.2	130.8	67.6	148.2	158.4	306.9	579.0	1080.7	203.0	932.6	766.2	2957.5
BM	σ	239.0	169.8	60.6	116.7	199.3	337.6	1244.5	2219.2	165.6	719.3	877.7	3469.5
	m	68	59.2	30	75.0	82	172.2	146	306.2	148	733.1	364	1259.8
40%	μ	105.8	92.4	10.2	13.0	32.6	47.5	211.2	339.7	60.2	136.2	441.8	1292.3
BR5	σ	195.8	159.0	20.5	19.3	44.1	59.9	449.4	692.2	58.2	127.2	520.2	1513.0
	m	5	9.6	1	3.6	9	15.1	41	80.5	43	92.7	81	244.3
40%	μ	271.2	223.7	14.2	16.7	72.2	96.7	398.6	614.9	110.2	235.4	1637.9	4662.4
BR6	σ	638.5	506.6	31.6	29.1	108.3	132.7	659.8	988.9	116.3	243.5	2274.0	6486.7
	m	5	9.6	1	3.6	9	14.9	71	122.6	61	129.7	117	345.5
17%	μ	162.4	46.7	34.6	42.7	180.0	192.8	753.8	788.6	86.0	249.7	551.8	1097.9
BM	σ	165.2	45.2	24.2	22.4	252.7	265.0	1079.8	1057.2	137.4	350.6	1459.9	2659.3
	m	28	12.3	20	28.2	32	39.5	252	275.7	36	116.8	80	267.2
17%	μ	18.2	13.3	8.6	12.2	5.9	7.9	81.4	104.5	4.8	12.4	20.8	58.2
BR5	σ	21.3	14.4	10.1	12.9	5.2	5.7	96.2	114.7	7.6	13.4	32.1	79.0
	m	5	5.9	3	5.2	3	6.4	17	30.4	1	5.9	5	17.5
17%	μ	24.6	16.4	4.2	4.9	5.6	8.2	53.8	74.4	6.6	15.2	27.6	70.8
BR6	σ	34.4	20.9	3.9	3.3	5.3	5.7	62.6	80.4	12.4	20.8	51.3	114.6
	m	5	5.9	3	3.8	3	6.4	15	26.7	1	5.9	5	17.5
8%	μ							76.6	14.4	58.0	40.6	2290.0	1317.2
BM	σ							98.2	14.4	60.6	44.2	4522.5	2511.0
	m							40	9.6	24	20.0	600	359.1
8%	μ							5.2	6.4	6.0	9.0	30.4	58.6
BR5	σ							4.5	4.0	5.8	6.5	42.1	65.2
	m							3	4.4	3	5.9	19	45.4
8%	μ							5.4	6.3	6.0	8.6	17.6	38.4
BR6	σ							4.9	4.0	5.7	6.2	15.3	27.9
	m							3	4.4	3	6.0	9	27.0

(ii) solution of LR, and the dual of FR separately instead of jointly, thus reducing the size of the LPs solved in various tests (Algorithm BM includes the dual variables in the master program and hence contains m_2 more variables than does Algorithm HJS);

(iii) use of hybrid branching rules;

(iv) use of penalties.

In Table 8 we consider problems similar to those considered in the previous series of experiments but transfer some of the second-level constraints to the first level. These problems are solved by Algorithm HJS and a version of Algorithm BM, slightly modified to allow for first-level constraints with y variables. It appears that:

(i) computational difficulty sometimes increases and sometimes decreases when constraints are transferred from second to first level;

(ii) increase in computing time when there are first-level constraints does not exceed a factor of 3 with Algorithm HJS;

(iii) Algorithm HJS always outperforms Algorithm BM, and the ratios of computing times are higher when there are first-level constraints (from 2.83 to 53.3) than when there are none.

Comparison of Algorithm HJS with Algorithm JF of Judice and Faustino [27], [28] is more difficult, as a code for the latter is not generally available at this time.

Judice and Faustino [28] present computational results for several large sparse

TABLE 7b
Comparison of HJS *and* BM *algorithms on problems without first-level constraints.*

	n_1	70		60		50		70		60		70	
	n_2	30		40		50		50		60		60	
	m_2	40		40		40		48		48		52	
		nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu
40%	μ	77.4	519.5	128.8	434.4								
BM	σ	53.9	342.5	97.2	293.8								
	m	54	383.0	92	302.2								
40%	μ	17.2	54.2	48.6	160.8								
BR5	σ	22.4	55.8	72.7	215.0								
	m	3	29.3	7	32.2								
40%	μ	18.0	60.3	97.8	315.5								
BR6	σ	30.8	96.4	134.5	422.4								
	m	7	29.3	33	97.5								
17%	μ	260.6	926.2	1032.6	1739.2								
BM	σ	506.8	1559.2	2497.9	3920.4								
	m	62	290.8	44	119.6								
17%	μ	9.4	25.1	10.0	35.5								
BR5	σ	10.4	21.3	11.9	29.5								
	m	5	14.1	1	12.9								
17%	μ	9.6	25.7	8.4	30.8								
BR6	σ	9.7	20.0	11.5	28.7								
	m	5	16.6	1	12.8								
8%	μ	77.4	84.7	211.4	153.4	2863.2	2496.7	2114.0	3622.9	1984.7	4235.4	3663.9	7294.5
BM	σ	62.8	67.2	147.7	118.7	4968.3	4559.3	3119.9	4997.7	2850.9	5855.2	3469.5	6333.0
	m	46	50.0	168	100.1	396	329.2	518	1204.5	662	1417.1	2138	4022.6
8%	μ	5.8	10.8	6.4	18.0	17.6	56.8	30.4	112.3	24.0	126.0	26.2	154.8
BR5	σ	5.3	9.3	3.6	9.4	12.8	35.7	31.3	98.5	29.4	126.6	26.2	140.8
	m	3	6.1	5	14.2	11	41.1	19	83.4	11	69.5	11	68.7
8%	μ	5.2	9.5	7.6	20.6	22.6	67.0	34.2	121.3	36.4	174.6	22.6	133.7
BR6	σ	4.0	6.7	5.0	13.0	14.9	42.8	29.4	95.6	50.6	205.9	20.3	109.4
	m	3	6.1	5	14.1	21	59.0	25	85.2	15	76.8	11	70.8

problems with up to 150 constraints, 250 variables controlled by the leader, and 150 variables controlled by the follower, on a CYBER 180-830 computer. These problems belong to two classes: "nonconflicting" ones in which all coefficients in the leader's and follower's objective functions are positive, and "conflicting" ones in which a few coefficients in these objective functions differ in sign. We have generated similar problems with all coefficients of the objective functions taken randomly in $[0, 20]$ in the first case: 80 percent of them are taken in that way and 20 percent are taken in $[-20, 0]$ in the second case. Results obtained on a SUN SPARC computer are presented in Table 9 (which also contains results for a few slightly denser problems). It appears that computation times are roughly similar to those of JF, but the latter have been obtained on a much more powerful computer (CYBER) and with a tolerance of 1 or 2 percent of the optimal value, which is obtained by Algorithm HJS.

Acknowledgment. The authors are grateful to Jonathan F. Bard for making available to them the code for linear bilevel programming he recently wrote with J. T. Moore. They also thank Alain Haurie for several fruitful discussions of preliminary drafts, and one anonymous referee for his comments.

TABLE 8
Comparison of HJS *and* BM *algorithms on problems without first-level constraints.*

	n_1	35		60		45		70		60		50		70		60		70	
	n_2	35		30		45		30		40		50		50		60		60	
	m_1	7		9		9		10		10		10		12		12		13	
	m_2	21		27		27		30		30		30		36		36		39	
	d	8%		8%		8%		8%		8%		8%		8%		8%		8%	
		nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu
BM	μ	108.8	20.7	211.4	119.5	572.8	340.4	392.0	511.0	745.4	834.2	5116.5	2810.1	3033.6	5846.5	4719.0	7322.5	3862.8	8393.2
	σ	103.2	20.4	362.9	196.6	646.0	415.4	1042.9	1413.1	1097.4	1391.7	9031.6	4399.7	2909.9	5755.2	4244.3	6210.9	3258.5	5813.2
	m	72	14.4	36	27.8	310	201.1	56	54.2	280	392.2	2162	1499.3	1784	3169.5	2958	5114.7	2036	5000.9
BR5	μ	8.2	7.3	17.4	19.8	24.6	40.7	8.6	14.2	16.0	30.4	95.6	197.1	33.8	109.6	369.2	1133.7	301.2	1070.9
	σ	4.8	3.4	26.3	27.2	22.1	32.6	8.3	11.4	11.6	17.6	71.3	139.4	22.1	64.8	579.5	1669.9	447.7	1329.9
	m	7	6.2	5	7.7	17	30.3	5	9.2	11	22.9	83	171.9	27	81.1	57	240.4	101	448.4
BR6	μ	7.0	6.8	13.4	15.1	24.6	42.5	9.0	14.8	17.4	32.5	135.6	273.0	59.0	174.4	332.6	1052.0	285.4	1082.5
	σ	4.6	3.5	18.5	18.3	22.6	36.7	9.8	13.5	13.2	21.5	118.0	231.1	50.0	137.8	386.3	1226.1	306.7	1107.9
	m	5	6.2	5	7.7	17	34.1	5	9.3	9	20.1	71	142.2	27	78.6	61	218.8	173	657.6

TABLE 9
Results for some larger problems (SUN SPARC).

	n_1	70		80		80*		300**		250**		300***		250***	
	n_2	70		70		80		100		150		100		150	
	m_2	56		60		64		100		150		100		150	
	d	8%		8%		8%		6%		6%		6%		6%	
		nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu	nodes	cpu
BR5	μ	133.4	195.8	161.4	262.1	1379.1	2420.5	2.6	37.4	5.8	238.9	5.0	49.9	31.8	698.8
	σ	147.2	211.6	225.6	336.9	2866.0	4834.0	3.5	28.6	7.8	104.7	3.53	26.8	65.9	1052.9
	m	85	120.6	55	126.2	117.0	256.9	1	27.2	1	184.9	5	38.0	5	259.2

* one problem was not solved within 15000 seconds.

** $c \in [0, 20]$

*** 20 percent of the variables have conflicting coefficients, i.e., for 80 percent of them $(c^1, d^1), d^2 \in [0, 20]$, and for 20 percent of them, $(c^1, d^1) \in [0, 20], d^2 \in [-20, 0]$.

REFERENCES

[1] E. AIYOSHI AND K. SHIMIZU, *Hierarchical decentralized systems and its new solution by a barrier method*, IEEE Trans. Systems Man Cybernet., SMC–11 (1981), pp. 444–449.

[2] ———, *A solution method for the static constrained Stackelberg problem via penalty method*, IEEE Trans. Automat. Control, AC–29 (1984), pp. 1111–1114.

[3] F. A. AL-KHAYYAL, R. HORST, AND P. M. PARDALOS, *Global optimization of concave functions subject to quadratic constraints: An application in nonlinear bilevel programming*, Ann. Oper. Res., to appear.

[4] G. ANANDALINGAM AND D. J. WHITE, *A penalty function approach for solving bilevel linear programs*, Res. Report, Department of Systems Engineering, University of Pennsylvania, Philadelphia, November 1989.

[5] J. F. BARD, *An efficient point algorithm for a linear two-stage optimization problem*, Oper. Res., 31 (1983), pp. 670–684.

[6] ———, *An algorithm for solving the general bilevel programming problem*, Math. Oper. Res., 8 (1983), pp. 260–272.

[7] ———, *Convex two-level programming*, Math. Programming, 40 (1988), pp. 15–28.

[8] J. F. BARD AND J. E. FALK, *An explicit solution to the multi-level programming problem*, Comput. Oper. Res., 9 (1982), pp. 77–100.

[9] J. F. BARD AND J. T. MOORE, *A branch and bound algorithm for the bilevel programming problem*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 281–292.

[10] W. F. BIALAS AND M. H. KARWAN, *On two-level linear optimization*, IEEE Trans. Automat. Control, AC-27 (1982), pp. 211–214.

[11] ———, *Two-level linear programming*, Management Sci., 30 (1984), pp. 1004–1020.

[12] O. BEN-AYED AND C. E. BLAIR, *Computational difficulties of bilevel linear programming*, Oper. Res., 38 (1990), pp. 556–559.

[13] J. BRACKEN AND J. MCGILL, *Mathematical programs with optimization problems in the constraints*, Oper. Res., 21 (1973), pp. 37–44.

[14] W. CANDLER, J. FORTUNY-AMAT, AND B. MCCARL, *The potential role of multilevel programming in agricultural economics*, Amer. J. Agricultural Econ., 63 (1981), pp. 521–531.

[15] W. CANDLER AND R. NORTON, *Multilevel programming and development policy*, No. 258, IBRD, World Bank Staff Working Paper, Washington DC, 1977.

[16] W. CANDLER AND R. TOWNSLEY, *A linear two-level programming problem*, Comput. Oper. Res., 9 (1982), pp. 59–76.

[17] S. DEMPE, *A simple algorithm for the linear bilevel programming problem*, Optimization, 18 (1987), pp. 373–385.

[18] J. E. FALK, *A linear max-min problem*, Math. Programming, 5 (1973), pp. 169–188.

[19] J. FORTUNY-AMAT AND B. MCCARL, *A representation and economic interpretation of a two-level programming problem*, J. Oper. Res. Soc., 32 (1981), pp. 783–792.

[20] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of* NP-*Completeness*, W.H. Freeman, San Francisco, 1979.

[21] P. HANSEN, B. JAUMARD, AND S.-H. LU, *A framework for algorithms in globally optimal design*, J. Mech. Trans. Automat. Design, 111 (1989), pp. 353–360.

[22] ———, *An analytical approach to global optimization*, Math. Programming, 52 (1991), pp. 227–254.

[23] A. HAURIE, R. LOULOU, AND G. SAVARD, *A two-player model of power cogeneration in New England*, IEEE Trans. Automat. Control, 1992, to appear.

[24] A. HAURIE, G. SAVARD, AND J. C. WHITE, *A note on an efficient point algorithm for a linear two-stage optimization problem*, Oper. Res., 38 (1990), pp. 553–555.

[25] J. P. IGNIZIO, *Goal Programming and Extensions*, Lexington Books, Lexington, MA, 1976.

[26] R. G. JEROSLOW, *The polynomial hierarchy and a simple model for competitive analysis*, Math. Programming, 32(1985), pp. 146–164.

[27] J. J. JUDICE AND A. M. FAUSTINO, *The solution of the linear bilevel programming problem by using the linear complementarity problem*, Investigação Oper., 8 (1988), pp. 77–95.

[28] ———, *A sequential* LCP *method for bilevel linear programming*, Res. Report, Department of Mathematics, University of Coïmbra, Coïmbra, Portugal, 1989.

[29] L. J. LEBLANC AND D. E. BOYCE, *A bilevel programming algorithm for exact solution of the network design problem with user-optimal flows*, Transportation Res., 20B (1986), pp. 259–265.

[30] P. MARCOTTE, *Network design problem with congestion effects: A case of bilevel programming*, Math. Programming, 34 (1986), pp. 142–162.

[31] R. E. MARSTEN, *The design of the* XMP *linear programming library*, Trans. Math. Software,

7 (1981), pp. 481–497.

[32] K. G. Murty, *Solving the fixed-charge problem by ranking the extreme points*, Oper. Res., 16 (1968), pp. 268–279.

[33] P. M. Pardalos and J. B. Rosen, *Constrained Global Optimization: Algorithms and Applications*, Lecture Notes in Computer Science 268, Springer-Verlag, Berlin, 1987.

[34] G. Papavassilopoulos, *Algorithms for static Stakelberg games with linear costs and polyhedral constraints*, in Proc. 21st IEEE Conference on Decisions and Control, 1982, pp. 647–652.

[35] G. Savard, *Contributions à la programmation mathématique à deux niveaux*, Ph.D. Thesis, École Polytechnique de Montréal, Montréal, Canada, April 1989.

[36] H. A. Taha, *Integer Programming, Theory, Applications and Computations*, Academic Press, New York, 1975.

[37] G. Ünlü, *A linear bilevel programming algorithm based on bicriteria programming*, Comput. Oper. Res., 14 (1987), pp. 173–179.

[38] R. E. Wendell, *Multiple criteria decision making with multiple decision makers*, Oper. Res., 28 (1980), pp. 1100–1111.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 5, pp. 1218–1235, September 1992

010

A FAMILY OF BLOCK PRECONDITIONERS FOR BLOCK SYSTEMS*

RAYMOND H. CHAN† AND XIAO-QING JIN†

Abstract. The solution of block system $A_{mn}x = b$ by the preconditioned conjugate gradient method where A_{mn} is an m-by-m block matrix with n-by-n Toeplitz blocks is studied. The preconditioner $c_F^{(1)}(A_{mn})$ is a matrix that preserves the block structure of A_{mn}. Specifically, it is defined as the minimizer of $\|A_{mn} - C_{mn}\|_F$ over all m-by-m block matrices C_{mn} with n-by-n circulant blocks. We prove that if A_{mn} is positive definite, then $c_F^{(1)}(A_{mn})$ is positive definite too. We also show that $c_F^{(1)}(A_{mn})$ is a good preconditioner for solving separable block systems with Toeplitz blocks and quadrantally symmetric block Toeplitz systems. We then discuss some of the spectral properties of the operator $c_F^{(1)}$. In particular, we show that the operator norms $\|c_F^{(1)}\|_2 = \|c_F^{(1)}\|_F = 1$.

Key words. Toeplitz matrix, circulant matrix, circulant operator, preconditioned conjugate gradient method

AMS(MOS) subject classifications. 65F10, 65F15

1. Introduction.

Preconditioned conjugate gradient methods have been used efficiently in solving large matrix problems. The idea of using the method with circulant preconditioners for solving symmetric positive definite Toeplitz systems $T_n x = b$ was proposed by Strang [16] and Olkin [15] independently. The number of operations per iteration is $O(n \log n)$ as circulant systems can be solved efficiently by the fast Fourier transform (FFT) and the matrix-vector multiplication $T_n v$ can also be computed by the FFT by first embedding T_n into a $2n$-by-$2n$ circulant matrix. The convergence rate of the preconditioned conjugate gradient method depends on the whole spectrum of the preconditioned matrix. In general, the more clustered the eigenvalues are, the faster the convergence rate will be.

There are many circulant preconditioners that can produce clustered spectra; see Chan and Yeung [5]. One good example is Chan's [9] circulant preconditioner, which is defined to be the minimizer of $\|T_n - C_n\|_F$ in Frobenius norm over all circulant matrices C_n. One can consider this circulant preconditioner from the operator point of view. Given any arbitrary n-by-n matrix A_n, we define an operator c_F which maps A_n to the matrix $c_F(A_n)$ that minimizes $\|A_n - C_n\|_F$ over all circulant matrices C_n. This circulant operator c_F has been studied in Chan, Jin, and Yeung [3].

In this paper, we generalize the idea to the case of block matrices. Our interest is in solving systems $T_{mn}x = b$ where T_{mn} is an m-by-m block matrix with n-by-n Toeplitz blocks. This kind of system occurs in a variety of applications, such as two-dimensional digital signal processing and the discretization of two-dimensional partial differential equations. Given such T_{mn}, we can use the mn-by-mn point-circulant matrix $c_F(T_{mn})$ as a circulant approximation to T_{mn}; see Chan and Olkin [10] and Chan and Chan [8]. In this paper, however, we consider another approximation to T_{mn} that preserves the block structure. The approximation extends the one proposed by Chan and Olkin [10]. We define the matrix $c_F^{(1)}(T_{mn})$ to be the minimizer of

*Received by the editors April 23, 1991; accepted for publication (in revised form) November 12, 1991.

†Department of Mathematics, University of Hong Kong, Hong Kong.

$\|T_{mn} - C_{mn}\|_F$ over all m-by-m block matrices C_{mn} with n-by-n circulant blocks. We will show that the operator $c_F^{(1)}$ is well defined for all mn-by-mn complex matrices A_{mn}. Some properties of $c_F^{(1)}$ are then discussed. In particular, we prove that if A_{mn} is positive definite, then $c_F^{(1)}(A_{mn})$ is also positive definite. We also show that the operator $c_F^{(1)}$ has operator norms $\|c_F^{(1)}\|_2 = \|c_F^{(1)}\|_F = 1$.

We then consider the cost of using the preconditioned conjugate gradient method with the preconditioner $c_F^{(1)}(A_{mn})$ for solving block systems $A_{mn}x = b$. The convergence rate of the method is then analyzed for two specific types of block systems. The first is the quadrantally symmetric block Toeplitz systems. We show that in this case, if the generating sequence of the matrices is absolutely summable, then the method converges in at most $O(\min\{m, n\})$ steps. Next we consider block matrices that are of the form $A_m \otimes T_n$ where A_m is nonsingular and T_n is a Toeplitz matrix with a positive 2π-periodic continuous generating function. We show that the resulting preconditioned system has spectrum clustered around 1 and hence the method converges superlinearly. Our numerical experiments have shown that $c_F^{(1)}(A_{mn})$ is indeed a good preconditioner for solving these block systems—the number of iterations is roughly a constant in both cases.

The outline of the paper is as follows. In §2, we first recall some properties of the point-circulant operator c_F. Then we introduce three different possible block preconditioners that preserve the block structure of the given matrix. In §3, we consider the cost of using $c_F^{(1)}(A_{mn})$ as a preconditioner for solving block systems $A_{mn}x = b$. The convergence rate of the method is analyzed in §4 and numerical results are then given in §5.

2. Operators for block matrices. Let us begin by introducing the operator for point matrices. Given an n-by-n unitary matrix U, let

$$\mathcal{M}_U = \{U^*\Lambda_n U \mid \Lambda_n \text{ is an } n\text{-by-}n \text{ complex diagonal matrix}\},$$

where "$*$" denotes the conjugate transposition. We note that when U is equal to the Fourier matrix F, $\mathcal{M}_F$ is the set of all circulant matrices (see Davis [11]). Let $\delta(A_n)$ denote the diagonal matrix whose diagonal is equal to the diagonal of the matrix A_n. The following lemma was first proved by Chan, Jin, and Yeung [3] for the case $U = F$ and was extended to the general unitary case by Huckle [13].

LEMMA 1. *Let A_n be an arbitrary n-by-n matrix and $c_U(A_n)$ be the minimizer of $\|W_n - A_n\|_F$ over all $W_n \in \mathcal{M}_U$. Then*

(i) *$c_U(A_n)$ is uniquely determined by A_n and is given by*

$$c_U(A_n) = U^*\delta(UA_nU^*)U \ . \tag{1}$$

(ii) *If A_n is Hermitian, then so is $c_U(A_n)$. Furthermore, if $\lambda_{\min}(\cdot)$ and $\lambda_{\max}(\cdot)$ denote the largest and the smallest eigenvalues, respectively, then we have*

$$\lambda_{\min}(A_n) \le \lambda_{\min}\big(c_U(A_n)\big) \le \lambda_{\max}\big(c_U(A_n)\big) \le \lambda_{\max}(A_n) \ .$$

In particular, if A_n is positive definite, then $c_U(A_n)$ is also positive definite.

(iii) *The operator c_U is a linear projection operator from the set of all n-by-n complex matrices into $\mathcal{M}_U$ and has the operator norms*

$$\|c_U\|_2 = \sup_{\|A_n\|_2=1} \|c_U(A_n)\|_2 = 1$$

and

$$||c_U||_F = \sup_{||A_n||_F=1} ||c_U(A_n)||_F = 1.$$

(iv) *When U is the n-by-n Fourier matrix F,*

$$c_F(A_n) = \sum_{j=0}^{n-1} \left(\frac{1}{n} \sum_{p-q\equiv j \pmod n} a_{pq} \right) Q^j \ , \tag{2}$$

where Q is the n-by-n circulant matrix

$$Q \equiv \begin{bmatrix} 0 & & & & 1 \\ 1 & 0 & & & \\ 0 & 1 & \ddots & & \\ \vdots & & \ddots & \ddots & \\ 0 & & & 1 & 0 \end{bmatrix} . \tag{3}$$

The circulant matrix $c_F(A_n)$, first proposed by Chan [9], is a good preconditioner for solving some Toeplitz systems by the preconditioned conjugate gradient method (see Chan [6]). In the following, we call c_U the point operator in order to distinguish it from the block operators that we now introduce.

2.1. Block operator $c_U^{(1)}$. Let us now consider a general system $A_{mn}x = b$ where A_{mn} is an mn-by-mn matrix partitioned as

$$A_{mn} = \begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,m} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,m} \\ \vdots & \ddots & \ddots & \vdots \\ A_{m,1} & A_{m,2} & \cdots & A_{m,m} \end{bmatrix} . \tag{4}$$

Here the blocks $A_{i,j}$ are square matrices of order n. We emphasize that we are interested in solving block systems where the blocks $A_{i,j}$ are Toeplitz matrices. In view of the point case, a natural choice of preconditioner for A_{mn} is

$$E_{mn} = \begin{bmatrix} c_F(A_{1,1}) & c_F(A_{1,2}) & \cdots & c_F(A_{1,m}) \\ c_F(A_{2,1}) & c_F(A_{2,2}) & \cdots & c_F(A_{2,m}) \\ \vdots & \ddots & \ddots & \vdots \\ c_F(A_{m,1}) & c_F(A_{m,2}) & \cdots & c_F(A_{m,m}) \end{bmatrix} ,$$

where the blocks $c_F(A_{i,j})$ are just the point-circulant approximations to $A_{i,j}$ (see (2)). We will show in §§4 and 5 that E_{mn} is a good preconditioner for solving some block systems. First, however, we study some of the spectral properties of the matrix E_{mn}.

Let $\delta^{(1)}(A_{mn})$ be defined by

$$\delta^{(1)}(A_{mn}) \equiv \begin{bmatrix} \delta(A_{1,1}) & \delta(A_{1,2}) & \cdots & \delta(A_{1,m}) \\ \delta(A_{2,1}) & \delta(A_{2,2}) & \cdots & \delta(A_{2,m}) \\ \vdots & \ddots & \ddots & \vdots \\ \delta(A_{m,1}) & \delta(A_{m,2}) & \cdots & \delta(A_{m,m}) \end{bmatrix} , \tag{5}$$

where each block $\delta(A_{i,j})$ is the diagonal matrix of order n whose diagonal is equal to the diagonal of the matrix $A_{i,j}$. The following lemma gives the relation between $\sigma_{\max}(A_{mn})$ and $\sigma_{\max}(\delta^{(1)}(A_{mn}))$, where $\sigma_{\max}(\cdot)$ denotes the largest singular value.

LEMMA 2. *Given any mn-by-mn complex matrix A_{mn} partitioned as in* (4), *we have*

$$\sigma_{\max}\left(\delta^{(1)}(A_{mn})\right) \le \sigma_{\max}(A_{mn}). \tag{6}$$

Furthermore, when A_{mn} is Hermitian, we have

$$\lambda_{\min}(A_{mn}) \le \lambda_{\min}\left(\delta^{(1)}(A_{mn})\right) \le \lambda_{\max}\left(\delta^{(1)}(A_{mn})\right) \le \lambda_{\max}(A_{mn}) \ . \tag{7}$$

In particular, if A_{mn} is positive definite, then $\delta^{(1)}(A_{mn})$ is also positive definite.

Proof. Let $(A_{mn})_{i,j;k,l} = (A_{k,l})_{ij}$ be the (i,j)th entry of the (k,l)th block of A_{mn}. Let P be the permutation matrix that satisfies

$$(P^*A_{mn}P)_{k,l;i,j} = (A_{mn})_{i,j;k,l}, \quad 1 \le i,j \le n, \quad 1 \le k,l \le m. \tag{8}$$

Then it is easy to see that $B_{mn} \equiv P^*\delta^{(1)}(A_{mn})P$ is of the form

$$B_{mn} = \begin{bmatrix} B_{1,1} & 0 & \cdots & 0 \\ 0 & B_{2,2} & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & B_{n,n} \end{bmatrix} .$$

Clearly, the matrices B_{mn} and $\delta^{(1)}(A_{mn})$ have the same singular values and eigenvalues. For each k, since $B_{k,k}$ is a principal submatrix of the matrix A_{mn}, it follows that

$$\sigma_{\max}(B_{k,k}) \le \sigma_{\max}(A_{mn});$$

see, for instance, Thompson [17]. Hence we have

$$\sigma_{\max}\left(\delta^{(1)}(A_{mn})\right) = \sigma_{\max}(B_{mn}) = \max_k \left(\sigma_{\max}(B_{k,k})\right) \le \sigma_{\max}(A_{mn}) \ .$$

When A_{mn} is Hermitian, by Cauchy's Interlace Theorem (see Golub and Van Loan [12]) we then have

$$\begin{aligned}\lambda_{\min}(A_{mn}) \le & \min_k \left(\lambda_{\min}(B_{k,k})\right) = \lambda_{\min}\left(\delta^{(1)}(A_{mn})\right) \\ \le & \lambda_{\max}\left(\delta^{(1)}(A_{mn})\right) = \max_k \left(\lambda_{\max}(B_{k,k})\right) \le \lambda_{\max}(A_{mn}) \ . \qquad \square\end{aligned}$$

In the following, we use $\mathcal{D}^{(1)}_{m,n}$ to denote the set of all m-by-m block matrices where each block is a complex diagonal matrix of order n, i.e., $\mathcal{D}^{(1)}_{m,n}$ is the set of all matrices of the form given in (5). Let

$$\mathcal{M}^{(1)}_U = \{(I \otimes U)^*\Lambda^{(1)}_{mn}(I \otimes U) \mid \Lambda^{(1)}_{mn} \in \mathcal{D}^{(1)}_{m,n}\} \ ,$$

where I is the m-by-m identity matrix and U is any given n-by-n unitary matrix. We then define the operator $c^{(1)}_U$ to be the mapping that maps every mn-by-mn matrix A_{mn} to the minimizer of $\|W_{mn} - A_{mn}\|_F$ over all $W_{mn} \in \mathcal{M}^{(1)}_U$. Some of the properties of this operator are given in the following theorem.

THEOREM 1. *For any arbitrary mn-by-mn complex matrix A_{mn} partitioned as in (4), let $c_U^{(1)}(A_{mn})$ be the minimizer of $\|W_{mn} - A_{mn}\|_F$ over all $W_{mn} \in \mathcal{M}_U^{(1)}$. Then*

(i) *$c_U^{(1)}(A_{mn})$ is uniquely determined by A_{mn} and is given by*

$$c_U^{(1)}(A_{mn}) = (I \otimes U)^* \delta^{(1)}\left[(I \otimes U) A_{mn} (I \otimes U)^*\right](I \otimes U). \tag{9}$$

(ii) *$c_U^{(1)}(A_{mn})$ is also given by*

$$c_U^{(1)}(A_{mn}) = \begin{bmatrix} c_U(A_{1,1}) & c_U(A_{1,2}) & \cdots & c_U(A_{1,m}) \\ c_U(A_{2,1}) & c_U(A_{2,2}) & \cdots & c_U(A_{2,m}) \\ \vdots & \ddots & \ddots & \vdots \\ c_U(A_{m,1}) & c_U(A_{m,2}) & \cdots & c_U(A_{m,m}) \end{bmatrix}, \tag{10}$$

where c_U is the point operator defined by (1).

(iii) *We have*

$$\sigma_{\max}\big(c_U^{(1)}(A_{mn})\big) \le \sigma_{\max}(A_{mn}). \tag{11}$$

(iv) *If A_{mn} is Hermitian, then $c_U^{(1)}(A_{mn})$ is also Hermitian and*

$$\lambda_{\min}(A_{mn}) \le \lambda_{\min}\big(c_U^{(1)}(A_{mn})\big) \le \lambda_{\max}\big(c_U^{(1)}(A_{mn})\big) \le \lambda_{\max}(A_{mn})\ .$$

In particular, if A_{mn} is positive definite, then $c_U^{(1)}(A_{mn})$ is also positive definite.

(v) *The operator $c_U^{(1)}$ is a linear projection operator from the set of all mn-by-mn complex matrices into $\mathcal{M}_U^{(1)}$ and has the operator norms*

$$\|c_U^{(1)}\|_2 \equiv \sup_{\|A_{mn}\|_2=1} \|c_U^{(1)}(A_{mn})\|_2 = 1$$

and

$$\|c_U^{(1)}\|_F \equiv \sup_{\|A_{mn}\|_F=1} \|c_U^{(1)}(A_{mn})\|_F = 1.$$

Proof. The following proves Theorem 1.

(i) Let $W_{mn} \in \mathcal{M}_U^{(1)}$ be given by

$$W_{mn} = (I \otimes U)^* \Lambda_{mn}^{(1)} (I \otimes U)\ ,$$

where $\Lambda_{mn}^{(1)} \in \mathcal{D}_{m,n}^{(1)}$. Since the Frobenius norm is unitary invariant, we have

$$\begin{aligned}\|W_{mn} - A_{mn}\|_F =& \|(I \otimes U)^* \Lambda_{mn}^{(1)} (I \otimes U) - A_{mn}\|_F \\ =& \|\Lambda_{mn}^{(1)} - (I \otimes U) A_{mn} (I \otimes U)^*\|_F\ .\end{aligned}$$

Thus the problem of minimizing $\|W_{mn} - A_{mn}\|_F$ over $\mathcal{M}_U^{(1)}$ is equivalent to the problem of minimizing $\|\Lambda_{mn}^{(1)} - (I \otimes U) A_{mn} (I \otimes U)^*\|_F$ over $\mathcal{D}_{m,n}^{(1)}$. Since $\Lambda_{mn}^{(1)}$ can only affect the diagonal of each block of $(I \otimes U) A_{mn} (I \otimes U)^*$, we see that the solution for the latter problem is $\Lambda_{mn}^{(1)} = \delta^{(1)}\left[(I \otimes U) A_{mn} (I \otimes U)^*\right]$. Hence

$$c_U^{(1)}(A_{mn}) = (I \otimes U)^* \delta^{(1)}\left[(I \otimes U) A_{mn} (I \otimes U)^*\right](I \otimes U)$$

is the minimizer of $\|W_{mn} - A_{mn}\|_F$. It is clear that $\Lambda_{mn}^{(1)}$ and hence $c_U^{(1)}(A_{mn})$ are uniquely determined by A_{mn}.

(ii) Since

$$\delta^{(1)}\left[(I \otimes U)A_{mn}(I \otimes U)^*\right] = \begin{bmatrix} \delta(UA_{1,1}U^*) & \delta(UA_{1,2}U^*) & \cdots & \delta(UA_{1,m}U^*) \\ \delta(UA_{2,1}U^*) & \delta(UA_{2,2}U^*) & \cdots & \delta(UA_{2,m}U^*) \\ \vdots & \ddots & \ddots & \vdots \\ \delta(UA_{m,1}U^*) & \delta(UA_{m,2}U^*) & \cdots & \delta(UA_{m,m}U^*) \end{bmatrix},$$

by (1) and (9) we see that $c_U^{(1)}(A_{mn})$ is also given by (10).

(iii) For general mn-by-mn matrix A_{mn}, we have by (9) and (6),

$$\begin{aligned}\sigma_{\max}\left(c_U^{(1)}(A_{mn})\right) =& \sigma_{\max}\left[\delta^{(1)}\left((I \otimes U)A_{mn}(I \otimes U)^*\right)\right] \\ \le& \sigma_{\max}\left[(I \otimes U)A_{mn}(I \otimes U)^*\right] = \sigma_{\max}(A_{mn}) \ .\end{aligned}$$

(iv) If A_{mn} is Hermitian, then it is clear from (10) and Lemma 1 (ii) that $c_U^{(1)}(A_{mn})$ is also Hermitian. Moreover, by (7) and (9), we have

$$\begin{aligned}\lambda_{\min}(A_{mn}) =& \lambda_{\min}\left[(I \otimes U)A_{mn}(I \otimes U)^*\right] \\ \le& \lambda_{\min}\left[\delta^{(1)}\left((I \otimes U)A_{mn}(I \otimes U)^*\right)\right] \\ =& \lambda_{\min}\left(c_U^{(1)}(A_{mn})\right) \le \lambda_{\max}\left(c_U^{(1)}(A_{mn})\right) \\ =& \lambda_{\max}\left[\delta^{(1)}\left((I \otimes U)A_{mn}(I \otimes U)^*\right)\right] \\ \le& \lambda_{\max}\left[(I \otimes U)A_{mn}(I \otimes U)^*\right] = \lambda_{\max}(A_{mn}) \ .\end{aligned}$$

(v) By (11), we have

$$\|c_U^{(1)}(A_{mn})\|_2 = \sigma_{\max}[c_U^{(1)}(A_{mn})] \le \sigma_{\max}(A_{mn}) = \|A_{mn}\|_2.$$

However, for the mn-by-mn identity matrix I_{mn}, we have $\|c_U^{(1)}(I_{mn})\|_2 = \|I_{mn}\|_2 = 1$. Hence $\|c_U^{(1)}\|_2 = 1$. For the Frobenius norm, we also have

$$\begin{aligned}\|c_U^{(1)}(A_{mn})\|_F =& \|\delta^{(1)}\left[(I \otimes U)A_{mn}(I \otimes U)^*\right]\|_F \\ \le& \|(I \otimes U)A_{mn}(I \otimes U)^*\|_F = \|A_{mn}\|_F \ .\end{aligned}$$

Since

$$\left\|c_U^{(1)}\left(\frac{1}{\sqrt{mn}}I_{mn}\right)\right\|_F = \frac{1}{\sqrt{mn}}\|I_{mn}\|_F = 1,$$

it follows that $\|c_U^{(1)}\|_F = 1$. □

2.2. Block operator $\tilde{c}_V^{(1)}$. For matrices A_{mn} partitioned as in (4), we can define another block approximation to them. Let $\tilde{\delta}^{(1)}(A_{mn})$ be defined by

$$\tilde{\delta}^{(1)}(A_{mn}) \equiv \begin{bmatrix} A_{1,1} & 0 & \cdots & 0 \\ 0 & A_{2,2} & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & A_{m,m} \end{bmatrix}. \tag{12}$$

In the following, we use $\tilde{\mathcal{D}}_{m,n}^{(1)}$ to denote the set of all m-by-m block diagonal matrices where each block is a complex matrix of order n, i.e., $\tilde{\mathcal{D}}_{m,n}^{(1)}$ is the set of all matrices of the form given by (12). Let

$$\tilde{\mathcal{M}}_V^{(1)} = \{(V \otimes I)^* \tilde{\Lambda}_{mn}^{(1)} (V \otimes I) \mid \tilde{\Lambda}_{mn}^{(1)} \in \tilde{\mathcal{D}}_{m,n}^{(1)}\} ,$$

where V is any given m-by-m unitary matrix and I is the n-by-n identity matrix.

We define the operator $\tilde{c}_V^{(1)}$ to be the mapping that maps every mn-by-mn matrix A_{mn} to the minimizer of $\|W_{mn} - A_{mn}\|_F$ over all $W_{mn} \in \tilde{\mathcal{M}}_V^{(1)}$. We have the following theorem, which is similar to Theorem 1.

THEOREM 2. *For any arbitrary mn-by-mn complex matrix A_{mn} partitioned as in* (4), *let $\tilde{c}_V^{(1)}(A_{mn})$ be the minimizer of $\|W_{mn} - A_{mn}\|_F$ over all $W_{mn} \in \tilde{\mathcal{M}}_V^{(1)}$. Then*

(i) *$\tilde{c}_V^{(1)}(A_{mn})$ is uniquely determined by A_{mn} and is given by*

$$\tilde{c}_V^{(1)}(A_{mn}) = (V \otimes I)^* \tilde{\delta}^{(1)} \left[(V \otimes I) A_{mn} (V \otimes I)^*\right] (V \otimes I). \tag{13}$$

(ii) *We have*

$$\sigma_{\max}\left(\tilde{c}_V^{(1)}(A_{mn})\right) \le \sigma_{\max}(A_{mn}).$$

(iii) *If A_{mn} is Hermitian, then $\tilde{c}_V^{(1)}(A_{mn})$ is also Hermitian and*

$$\lambda_{\min}(A_{mn}) \le \lambda_{\min}\left(\tilde{c}_V^{(1)}(A_{mn})\right) \le \lambda_{\max}\left(\tilde{c}_V^{(1)}(A_{mn})\right) \le \lambda_{\max}(A_{mn}) \ .$$

In particular, if A_{mn} is positive definite, then $\tilde{c}_V^{(1)}(A_{mn})$ is also positive definite.

(iv) *The operator $\tilde{c}_V^{(1)}$ is a linear projection operator from the set of all mn-by-mn complex matrices into $\tilde{\mathcal{M}}_V^{(1)}$ and has the operator norms*

$$\|\tilde{c}_V^{(1)}\|_2 = \|\tilde{c}_V^{(1)}\|_F = 1.$$

The proof of Theorem 2 is quite similar to that of Theorem 1, so we omit it here. We note, however, that Theorem 2 (ii)–(iv) can be proved easily by using the following relationship between $c_U^{(1)}$ and $\tilde{c}_V^{(1)}$.

LEMMA 3. *Let U be any given unitary matrix and P be the permutation matrix defined in* (8). *Then for any arbitrary mn-by-mn complex matrix A_{mn} partitioned as in* (4), *we have*

$$\delta^{(1)}(A_{mn}) = P \tilde{\delta}^{(1)}(P^* A_{mn} P) P^*$$

and

$$c_U^{(1)}(A_{mn}) = P \tilde{c}_U^{(1)}(P^* A_{mn} P) P^*.$$

Proof. To prove the first equality, we note that by the definition of $\tilde{\delta}^{(1)}$ and (8), we have

$$\begin{aligned}
[\tilde{\delta}^{(1)}(P^* A_{mn} P)]_{k,l;i,j} &= \begin{cases} (P^* A_{mn} P)_{k,l;i,j}, & i = j \ , \\ 0, & i \neq j \ , \end{cases} \\
&= \begin{cases} (A_{mn})_{i,j;k,l}, & i = j \ , \\ 0, & i \neq j \ . \end{cases}
\end{aligned}$$

Hence

$$[P\tilde{\delta}^{(1)}(P^*A_{mn}P)P^*]_{i,j;k,l} = [\tilde{\delta}^{(1)}(P^*A_{mn}P)]_{k,l;i,j} = \begin{cases} (A_{mn})_{i,j;k,l}, & i = j\ , \\ 0, & i \neq j\ , \end{cases}$$

which by definition is equal to $[\delta^{(1)}(A_{mn})]_{i,j;k,l}$.

To prove the second equality, we first note that

$$(I \otimes U)P = P(U \otimes I)$$

for any matrix U. Hence by (13) and (9), we have

$$\begin{aligned} P\tilde{c}_U^{(1)}(P^*A_{mn}P)P^* &= P(U \otimes I)^*\tilde{\delta}^{(1)}[(U \otimes I)P^*A_{mn}P(U \otimes I)^*](U \otimes I)P^* \\ &= (I \otimes U)^*P\tilde{\delta}^{(1)}[P^*(I \otimes U)A_{mn}(I \otimes U)^*P]P^*(I \otimes U) \\ &= (I \otimes U)^*\delta^{(1)}[(I \otimes U)A_{mn}(I \otimes U)^*](I \otimes U) = c_U^{(1)}(A_{mn}). \end{aligned}$$ □

2.3. Operator $c_{V,U}^{(2)}$. Intuitively, $c_U^{(1)}(A_{mn})$ and $\tilde{c}_V^{(1)}(A_{mn})$ resemble the diagonalization of A_{mn} along one specific direction. It is then natural to consider the matrix that results from diagonalization along both directions. Thus let $c_{V,U}^{(2)}$ denote the composite of the two operators, i.e., $c_{V,U}^{(2)} \equiv \tilde{c}_V^{(1)} \circ c_U^{(1)}$. The following lemma will be used to derive the properties of the operator $c_{V,U}^{(2)}$.

LEMMA 4. *For any given A_{mn} partitioned as in* (4), *we have*

$$(14) \qquad (I \otimes U)^*\tilde{\delta}^{(1)}(A_{mn})(I \otimes U) = \tilde{\delta}^{(1)}[(I \otimes U)^*A_{mn}(I \otimes U)]$$

and

$$(15) \qquad (V \otimes I)\delta^{(1)}(A_{mn})(V \otimes I)^* = \delta^{(1)}[(V \otimes I)A_{mn}(V \otimes I)^*]\ .$$

Furthermore,

$$(16) \qquad \tilde{\delta}^{(1)} \circ \delta^{(1)}(A_{mn}) = \delta(A_{mn}) = \delta^{(1)} \circ \tilde{\delta}^{(1)}(A_{mn})\ .$$

The proof of Lemma 4 is straightforward, so we omit it here. By using Lemma 4, we can prove the following theorem, which states that the operator $c_{V,U}^{(2)}$ is just a particular case of the point operator.

THEOREM 3. *For any given A_{mn} partitioned as in* (4), *we have*

$$c_{V,U}^{(2)}(A_{mn}) = c_{V\otimes U}(A_{mn})\ ,$$

where $c_{V\otimes U}$ is the point operator defined in Lemma 1.

Proof. For any given A_{mn}, by definitions of $c_U^{(1)}$ and $\tilde{c}_V^{(1)}$, we have

$$\begin{aligned} c_{V,U}^{(2)}(A_{mn}) =& \tilde{c}_V^{(1)}[c_U^{(1)}(A_{mn})] \\ =& (V \otimes I)^*\tilde{\delta}^{(1)}\big\{(V \otimes I)\big[(I \otimes U)^*\delta^{(1)}[(I \otimes U)A_{mn}(I \otimes U)^*](I \otimes U)\big] \\ & \times (V \otimes I)^*\big\}(V \otimes I) \\ =& (V \otimes I)^*\tilde{\delta}^{(1)}\big\{(I \otimes U)^*(V \otimes I)\delta^{(1)}[(I \otimes U)A_{mn}(I \otimes U)^*] \\ & \times (V \otimes I)^*(I \otimes U)\big\}(V \otimes I). \end{aligned}$$

Hence by (14), (15), and (16), we have

$$\begin{aligned} c^{(2)}_{V,U}(A_{mn}) &= (V \otimes U)^* \tilde{\delta}^{(1)}\{\delta^{(1)}[(V \otimes U)A_{mn}(V \otimes U)^*]\}(V \otimes U) \\ &= (V \otimes U)^* \delta[(V \otimes U)A_{mn}(V \otimes U)^*](V \otimes U) = c_{V\otimes U}(A_{mn}) \,. \qquad \Box \end{aligned}$$

Since $c^{(2)}_{V,U}$ is just another point operator, we will concentrate our discussion on $c^{(1)}_U$ and $\tilde{c}^{(1)}_V$ in the remainder of the paper. We remark that $c^{(2)}_{V,U}(A_{mn})$ is an approximation of A_{mn} in two directions, whereas $c^{(1)}_U(A_{mn})$ and $\tilde{c}^{(1)}_V(A_{mn})$ are approximations in one direction only (with the other direction being approximated exactly). Thus we expect that the $c^{(1)}_U(A_{mn})$ and $\tilde{c}^{(1)}_V(A_{mn})$ are better preconditioners than $c^{(2)}_{V,U}(A_{mn})$. This is confirmed by the numerical results in §5.

We now give two simple formulae for finding $c^{(1)}_U(A_{mn})$ and $\tilde{c}^{(1)}_V(A_{mn})$ in the case where U and V are just the Fourier matrix F. When $U = F$, we have by (10),

$$(17) \qquad c^{(1)}_F(A_{mn}) = \begin{bmatrix} c_F(A_{1,1}) & c_F(A_{1,2}) & \cdots & c_F(A_{1,m}) \\ c_F(A_{2,1}) & c_F(A_{2,2}) & \cdots & c_F(A_{2,m}) \\ \vdots & \ddots & \ddots & \vdots \\ c_F(A_{m,1}) & c_F(A_{m,2}) & \cdots & c_F(A_{m,m}) \end{bmatrix},$$

where each block $c_F(A_{i,j})$ is Chan's circulant preconditioner for $A_{i,j}$.

Next we find $\tilde{c}^{(1)}_F(A_{mn})$ by using Lemma 3. We first let $A_{mn} = P^* B_{mn} P$ and partition B_{mn} into n^2 blocks with each block $B_{i,j}$ an m-by-m matrix. Then by Lemma 3 and (17), we have

$$[\tilde{c}^{(1)}_F(A_{mn})]_{i,j;k,l} = [P^* c^{(1)}_F(B_{mn})P]_{i,j;k,l} = [c^{(1)}_F(B_{mn})]_{k,l;i,j} = (c_F(B_{i,j}))_{kl},$$

where $B_{i,j}$ is the (i,j)th block of the matrix B_{mn}. By (2), we see that the (k,l)th entry of the circulant matrix $c_F(B_{i,j})$ is given by

$$(c_F(B_{i,j}))_{kl} = \frac{1}{m} \sum_{p-q \equiv k-l \pmod m} (B_{i,j})_{pq}.$$

Since $(B_{i,j})_{pq} = (A_{p,q})_{ij}$, we have

$$[\tilde{c}^{(1)}_F(A_{mn})]_{i,j;k,l} = \frac{1}{m} \sum_{p-q \equiv k-l \pmod m} (A_{p,q})_{ij}, \quad 1 \le i,j \le n, \quad 1 \le k,l \le m.$$

Thus the (k,l)th block of $\tilde{c}^{(1)}_F(A_{mn})$ is given by $\frac{1}{m}\sum_{p-q\equiv k-l \pmod m}(A_{pq})$. Since it depends only on $k-l \pmod m$, we see that $\tilde{c}^{(1)}_F(A_{mn})$ is a block-circulant matrix. Using the definition of the matrix Q in (3), we have

$$\tilde{c}^{(1)}_F(A_{mn}) = \frac{1}{m} \sum_{j=0}^{m-1} \left(Q^j \otimes \sum_{p-q \equiv j \pmod m} A_{p,q} \right).$$

3. Block preconditioners for block systems. In this section, we consider the cost of solving block systems $A_{mn}x = b$ by the preconditioned conjugate gradient method with preconditioner $c^{(1)}_F(A_{mn})$. The analysis for $\tilde{c}^{(1)}_F(A_{mn})$ is similar. We first recall that in each iteration of the preconditioned conjugate gradient method, we have to compute the matrix-vector multiplication $A_{mn}v$ for some vector v and solve the system

$$(18) \qquad c^{(1)}_F(A_{mn})y = d$$

for some vector d (see Golub and Van Loan [12]).

3.1. General matrices. Let A_{mn} be a general mn-by-mn matrix. We note that by (9), the solution to (18) is given by

$$y = (I \otimes F)^* \left[\delta^{(1)}\left((I \otimes F)A_{mn}(I \otimes F)^*\right)\right]^{-1}(I \otimes F)d \, . \tag{19}$$

Hence before we start the iteration, we should form the matrix

$$\Delta \equiv \delta^{(1)}\left((I \otimes F)A_{mn}(I \otimes F)^*\right)$$

and compute its inverse. We note that by (17), the (i,j)th block of Δ is just $Fc_F(A_{i,j})F^*$. By (1), $Fc_F(A_{i,j})F^* = \delta(FA_{i,j}F^*)$ and hence it can be computed in n^2 operations and one FFT (see Chan, Jin, and Yeung [3]). Thus the cost of obtaining Δ is $O(m^2n^2)$ operations. Next we compute its inverse.

We first permute the matrix Δ by P to obtain

$$B_{mn} = P^*\Delta P = \begin{bmatrix} B_{1,1} & 0 & \cdots & 0 \\ 0 & B_{2,2} & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & B_{n,n} \end{bmatrix} .$$

We then compute the LU decompositions for all diagonal blocks $B_{k,k}$. That will take $O(nm^3)$ operations. It requires a total of $O(n^2m^2 + nm^3)$ operations in the initialization step.

After obtaining the LU factors of Δ, we start the iteration. For a general dense matrix A_{mn}, $A_{mn}v$ can be computed in $O(n^2m^2)$. To get the vector y in (19), we note that by using the FFT, vectors of the form $(I \otimes F)d$ can be computed in $O(mn \log n)$ operations. Using the LU factors of Δ, $O(nm^2)$ operations are need to compute $\Delta^{-1}d$ for any vector d. The total cost per iteration is $O(mn \log n) + O(nm^2)$ operations.

Thus the algorithm for solving system $A_{mn}x = b$ for the general matrix A_{mn} requires $O(n^2m^2 + nm^3)$ operations in the initialization step and $O(n^2m^2)$ operations per iteration. Clearly, if A_{mn} is sparse, the cost can be reduced. In the next two subsections, we will consider two types of block systems where the cost can be drastically reduced.

Finally, we note that some of the block operations mentioned above can be done in parallel. For instance, the diagonal $\delta(FA_{i,j}F^*)$ of the blocks $c_F(A_{i,j})$ can be obtained in $O(n^2)$ parallel steps with $O(m^2)$ processors and the LU decompositions of the blocks B_{kk} in B_{mn} can also be computed in parallel. This can further reduce the cost per iteration.

3.2. Quadrantally symmetric block Toeplitz matrices. Let us consider the family of block Toeplitz systems $T_{mn}x = b$ where T_{mn} is of the form

$$T_{mn} = \begin{bmatrix} T_{1,1} & T_{1,2} & \cdots & T_{1,m} \\ T_{2,1} & T_{2,2} & \cdots & T_{2,m} \\ \vdots & \ddots & \ddots & \vdots \\ T_{m,1} & T_{m,2} & \cdots & T_{m,m} \end{bmatrix} = \begin{bmatrix} T_{(0)} & T_{(1)} & \cdots & T_{(m-1)} \\ T_{(1)} & T_{(0)} & \cdots & T_{(m-2)} \\ \vdots & \ddots & \ddots & \vdots \\ T_{(m-1)} & T_{(m-2)} & \cdots & T_{(0)} \end{bmatrix} . \tag{20}$$

Here the blocks $T_{i,j} = T_{(|i-j|)}$ are themselves symmetric Toeplitz matrices of order n. Such T_{mn} are called quadrantally symmetric block Toeplitz matrices.

By (17), the blocks of $c_F^{(1)}(T_{mn})$ are just $c_F(T_{(k)})$. Hence by (2) and the fact that $T_{(k)}$ is Toeplitz, the diagonal $\delta(FT_{(k)}F^*)$ can be computed in $O(n \log n)$ operations. Therefore, we need $O(mn \log n)$ operations to form $\Delta = \delta^{(1)}\big((I \otimes F)T_{mn}(I \otimes F)^*\big)$. We emphasize that in this case, there is no need to compute the LU factors of Δ. In fact,

$$P^* \Delta P = \begin{bmatrix} \tilde{T}_{1,1} & 0 & \cdots & 0 \\ 0 & \tilde{T}_{2,2} & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \tilde{T}_{n,n} \end{bmatrix},$$

where

$$(\tilde{T}_{k,k})_{ij} = \big(\delta(FT_{i,j}F^*)\big)_{kk} = \big(\delta(FT_{(|i-j|)}F^*)\big)_{kk}, \quad 1 \le i,j \le m, \quad 1 \le k \le n.$$

Hence we see that the diagonal blocks $\tilde{T}_{k,k}$ are still symmetric Toeplitz matrices of order m. Therefore, it requires only $O(m \log^2 m)$ operations to compute $\tilde{T}_{k,k}^{-1}v$ for any vector v (see Ammar and Gragg [1]). Thus the system $c_F^{(1)}(T_{mn})y = d$ can be solved in $O(nm \log^2 m)$ operations.

Next we consider the cost of the matrix-vector multiplication $T_{mn}v$. We recall that for any Toeplitz matrix $T_{(k)}$, the matrix-vector multiplication $T_{(k)}w$ can be computed by the FFT by first embedding $T_{(k)}w$ into a $2n$-by-$2n$ circulant matrix and extending w to a $2n$-vector by zeros. For the matrix-vector product $T_{mn}v$, we can use the same trick. We first embed T_{mn} into a (blockwise) $2m$-by-$2m$ block-circulant matrix where each block itself is a $2n$-by-$2n$ circulant matrix. Then we extend v to a $4mn$-vector by putting zeros in the appropriate places. Using FFT, or more precisely using $(F_{2m} \otimes F_{2n})$ to diagonalize the $2m$-by-$2m$ block-circulant matrix, we see that $T_{mn}v$ can be obtained in $O(mn(\log m + \log n))$ operations.

Thus we conclude that the initialization cost in this case is $O(mn \log n)$ and the cost per iteration is $O(nm \log^2 m + mn \log n)$. We emphasize that if $m > n$, then one should consider using $\tilde{c}_F^{(1)}(A_{mn})$ as preconditioner instead.

3.3. Separable matrices. Consider the following system $(A_m \otimes B_n)x = b$ where A_m is an m-by-m nonsingular matrix and B_n is an n-by-n Hermitian positive definite matrix. This system arises in solving the inverse heat problem in two dimensions (see Chan [7]). Since $\delta^{(1)}(A_m \otimes B_n) = A_m \otimes \delta(B_n)$, it follows that

$$c_F^{(1)}(A_m \otimes B_n) = A_m \otimes c_F(B_n).$$

Thus the preconditioned system becomes

$$(A_m \otimes c_F(B_n))^{-1}(A_m \otimes B_n)x = (A_m \otimes c_F(B_n))^{-1}b$$

or

$$(I \otimes c_F(B_n)^{-1}B_n)x = (A_m^{-1} \otimes c_F^{-1}(B_n))b.$$

For general B_n, $c_F(B_n)$ can be obtained in $O(n^2)$ operations and $c_F(B_n)^{-1}y$ can be obtained in $O(n \log n)$ operations for any vector y. By decomposing A_m into its LU factors first, we can then generate the new right-hand side vector

$$(A_m^{-1} \otimes c_F^{-1}(B_n))b = (A_m^{-1} \otimes I)(I \otimes c_F^{-1}(B_n))b$$

in $O(m^3+m^2n+mn\log n+n^2)$ operations. In each subsequent iteration, the matrix-vector multiplication $(I\otimes c_F(B_n)^{-1}B_n)v$ can be done in $O(mn\log n+mn^2)$ operations.

When B_n is a Hermitian positive definite Toeplitz matrix, $c_F(B_n)$ can be obtained in $O(n)$ operations. Hence the initialization cost is reduced to $O(m^3+m^2n+mn\log n)$. Moreover, since the cost of multiplying B_ny becomes $O(n\log n)$, we see that the cost per iteration decreases to $O(mn\log n)$.

4. Convergence rate.

In this section, we analyze the convergence rate of the preconditioned conjugate gradient method when applied to solving some special block systems.

4.1. Quadrantally symmetric block Toeplitz matrices.

Let us consider the system $T_{mn}x=b$ where T_{mn} is a quadrantally symmetric block Toeplitz matrix given by (20). Let the entries of the block $T_{(j)}$ be denoted by $t_{pq}^{(j)}=t_{|p-q|}^{(j)}$ for $1\le p,q\le n, 0\le j<m$. We assume that the generating sequence $t_k^{(j)}$ of T_{mn} is absolutely summable, i.e.,

$$\sum_{j=0}^{\infty}\sum_{k=0}^{\infty}|t_k^{(j)}|\le K<\infty\ .$$

In order to analyze the distribution of the eigenvalues of $T_{mn}-c_F^{(1)}(T_{mn})$, we need to introduce Strang's circulant preconditioner. For each $T_{(j)}$, Strang's preconditioner $s_F(T_{(j)})$ is defined to be the circulant matrix obtained by copying the central diagonals of $T_{(j)}$ and bringing them around to complete the circulant. More precisely, the entries $s_{pq}^{(j)}=s_{|p-q|}^{(j)}$ of $s_F(T_{(j)})$ are given by

$$s_k^{(j)}=\begin{cases} t_k^{(j)}, & 0\le k\le r,\\ t_{n-r}^{(j)}, & r\le k<n.\end{cases} \tag{21}$$

Here, for simplicity, we have assumed that $n=2r$. Define

$$s_F^{(1)}(T_{mn})=\begin{bmatrix} s_F(T_{(0)}) & s_F(T_{(1)}) & \cdots & s_F(T_{(m-1)})\\ s_F(T_{(1)}) & s_F(T_{(0)}) & \cdots & s_F(T_{(m-2)})\\ \vdots & \ddots & \ddots & \vdots\\ s_F(T_{(m-1)}) & s_F(T_{(m-2)}) & \cdots & s_F(T_{(0)})\end{bmatrix}. \tag{22}$$

We prove below that the matrices $c_F^{(1)}(T_{mn})$ and $s_F^{(1)}(T_{mn})$ are asymptotically the same.

LEMMA 5. *Let T_{mn} be given by* (20) *with an absolutely summable generating sequence. Then for all $m>0$,*

$$\lim_{n\to\infty}\|s_F^{(1)}(T_{mn})-c_F^{(1)}(T_{mn})\|_1=0.$$

Proof. Let $B_{mn}\equiv s_F^{(1)}(T_{mn})-c_F^{(1)}(T_{mn})$. By (17) and (22), we see that the block $B_{(j)}$ of B_{mn} are given by $s_F(T_{(j)})-c_F(T_{(j)})$. Hence by (2) and (21) they are circulant, with entries $b_{pq}^{(j)}=b_{|p-q|}^{(j)}$ given by

$$b_k^{(j)}=\begin{cases} \dfrac{k}{n}(t_k^{(j)}-t_{n-k}^{(j)}), & 0\le k\le r,\\ \dfrac{n-k}{n}(t_{n-k}^{(j)}-t_k^{(j)}), & r\le k<n.\end{cases}$$

Thus

$$\|B_{mn}\|_1 \le 2\sum_{j=0}^{m-1}\|B_{(j)}\|_1 \le 2\sum_{j=0}^{m-1}\sum_{k=0}^{n-1}|b_k^{(j)}| \le 4\sum_{j=0}^{m-1}\sum_{k=1}^{r}\frac{k}{n}|t_k^{(j)}| + 4\sum_{j=0}^{m-1}\sum_{k=r+1}^{n-1}|t_k^{(j)}|.$$

For all $\varepsilon > 0$, since the generating sequence is absolutely summable, we can always find an $N_1 > 0$ and an $N_2 > 2N_1$ such that

$$\sum_{j=0}^{\infty}\sum_{k=N_1}^{\infty}|t_k^{(j)}| < \varepsilon \quad\text{and}\quad \frac{1}{N_2}\sum_{j=0}^{\infty}\sum_{k=1}^{N_1}k|t_k^{(j)}| < \varepsilon.$$

Thus for all $n > N_2$,

$$\|B_{mn}\|_1 \le \frac{4}{N_2}\sum_{j=0}^{\infty}\sum_{k=1}^{N_1}k|t_k^{(j)}| + 4\sum_{j=0}^{\infty}\sum_{k=N_1+1}^{r}|t_k^{(j)}| + 4\sum_{j=0}^{\infty}\sum_{k=r+1}^{\infty}|t_k^{(j)}| < 12\varepsilon\,. \qquad \square$$

In view of Lemma 5 and the following equality,

$$T_{mn} - c_F^{(1)}(T_{mn}) = \left(s_F^{(1)}(T_{mn}) - c_F^{(1)}(T_{mn})\right) + \left(T_{mn} - s_F^{(1)}(T_{mn})\right)\,,$$

we see that the spectra of $T_{mn} - c_F^{(1)}(T_{mn})$ and $T_{mn} - s_F^{(1)}(T_{mn})$ are asymptotically the same. However, it is easier to obtain spectral information about the second matrix, as the following lemma shows.

LEMMA 6. *Let T_{mn} be given by* (20) *with an absolutely summable generating sequence. Then for all $\varepsilon > 0$, there exists an $N_3 > 0$ such that for all $n > N_3$ and $m > 0$,*

$$s_F^{(1)}(T_{mn}) - T_{mn} = W_{mn}^{(N_3)} + U_{mn}^{(N_3)},$$

where $\|W_{mn}^{(N_3)}\|_1 \le \varepsilon$ and $\operatorname{rank}(U_{mn}^{(N_3)}) \le 2N_3m$.

Proof. Define $W_{mn} \equiv s_F^{(1)}(T_{mn}) - T_{mn}$. It is clear from (21) that its blocks $W_{(j)} \equiv s_F(T_{(j)}) - T_{(j)}$ are symmetric Toeplitz matrices with entries $w_{pq}^{(j)} = w_{|p-q|}^{(j)}$ given by

$$w_k^{(j)} = \begin{cases} 0, & 0 \le k \le r\,, \\ t_{n-k}^{(j)} - t_k^{(j)}, & r < k < n\,. \end{cases}$$

For all $\varepsilon > 0$, since the generating sequence is absolutely summable, there exists an $N_3 > 0$ such that $\sum_{j=0}^{\infty}\sum_{k=N_3}^{\infty}|t_k^{(j)}| < \varepsilon$. Corresponding to this N_3, we define for each block $W_{(j)}$, the n-by-n matrix

$$W_{(j)}^{(N_3)} = \begin{bmatrix} \tilde{W}_{(j)} & 0 \\ 0 & 0 \end{bmatrix}\,,$$

where $\tilde{W}_{(j)}$ is the $(n - N_3)$-by-$(n - N_3)$ principal submatrix of $W_{(j)}$. Clearly, each $\tilde{W}_{(j)}$ is a Toeplitz matrix. Let $U_{(j)}^{(N_3)} \equiv W_{(j)} - W_{(j)}^{(N_3)}$ for all j. We note that $U_{(j)}^{(N_3)}$ is nonzero only in the last N_3 rows and N_3 columns; therefore, $\operatorname{rank}(U_{(j)}^{(N_3)}) \le 2N_3$.

Let

$$W_{mn}^{(N_3)} = \begin{bmatrix} W_{(0)}^{(N_3)} & W_{(1)}^{(N_3)} & \cdots & W_{(m-1)}^{(N_3)} \\ W_{(1)}^{(N_3)} & W_{(0)}^{(N_3)} & \cdots & W_{(m-2)}^{(N_3)} \\ \vdots & \ddots & \ddots & \vdots \\ W_{(m-1)}^{(N_3)} & W_{(m-2)}^{(N_3)} & \cdots & W_{(0)}^{(N_3)} \end{bmatrix} \tag{23}$$

and

$$U_{mn}^{(N_3)} = \begin{bmatrix} U_{(0)}^{(N_3)} & U_{(1)}^{(N_3)} & \cdots & U_{(m-1)}^{(N_3)} \\ U_{(1)}^{(N_3)} & U_{(0)}^{(N_3)} & \cdots & U_{(m-2)}^{(N_3)} \\ \vdots & \ddots & \ddots & \vdots \\ U_{(m-1)}^{(N_3)} & U_{(m-2)}^{(N_3)} & \cdots & U_{(0)}^{(N_3)} \end{bmatrix} .$$

Then $s_F^{(1)}(T_{mn}) - T_{mn} = W_{mn}^{(N_3)} + U_{mn}^{(N_3)}$. Since each block $U_{(j)}^{(N_3)}$ in $U_{mn}^{(N_3)}$ is an n-by-n matrix where the leading $(n - N_3)$-by-$(n - N_3)$ principal submatrix is a zero matrix, it is easy to see that $\text{rank}(U_{mn}^{(N_3)}) \le 2N_3 m = O(m)$. For $W_{mn}^{(N_3)}$, we have by (23) that

$$\begin{aligned} \|W_{mn}^{(N_3)}\|_1 &\le 2\sum_{j=0}^{m-1} \|W_{(j)}^{(N_3)}\|_1 = 2\sum_{j=0}^{m-1} \|\tilde{W}_{(j)}\|_1 \\ &= 2\sum_{j=0}^{m-1}\sum_{k=r+1}^{n-N_3-1} |w_k^{(j)}| = 2\sum_{j=0}^{m-1}\sum_{k=r+1}^{n-N_3-1} |t_{n-k}^{(j)} - t_k^{(j)}| \\ &\le 2\sum_{j=0}^{m-1}\sum_{k=N_3+1}^{n-N_3-1} |t_k^{(j)}| \le 2\sum_{j=0}^{\infty}\sum_{k=N_3}^{\infty} |t_k^{(j)}| < 2\varepsilon . \end{aligned}$$

□

Let $N = \max\{N_2, N_3\}$, where N_2 and N_3 are given in the proofs of Lemmas 5 and 6. Then for all $n > N$ and $m > 0$, we have

$$T_{mn} - c_F^{(1)}(T_{mn}) = M_{mn} + L_{O(m)} ,$$

where $M_{mn} = s_F^{(1)}(T_{mn}) - c_F^{(1)}(T_{mn}) + W_{mn}^{(N)}$ with $\|M_{mn}\|_1 < \varepsilon$ and $L_{O(m)} = U_{mn}^{(N)}$ with rank $(L_{O(m)}) = O(m)$. Since M_{mn} is symmetric, we have

$$\|M_{mn}\|_2 \le (\|M_{mn}\|_1 \|M_{mn}\|_\infty)^{1/2} = \|M_{mn}\|_1 < \varepsilon .$$

By using Cauchy's Interlace Theorem, we then have the following theorem.

THEOREM 4. *Let T_{mn} be given by* (20) *with an absolutely summable generating sequence. Then for all $\varepsilon > 0$, there exists an $N > 0$ such that for all $n > N$ and $m > 0$, at most $O(m)$ eigenvalues of $c_F^{(1)}(T_{mn}) - T_{mn}$ have absolute values exceeding ε.*

If T_{mn} is positive definite with the smallest eigenvalue $\lambda_{\min}(T_{mn}) \ge \delta > 0$, where δ is independent of m and n, then by Theorem 1 (iv), $\lambda_{\min}\big(c_F^{(1)}(T_{mn})\big) \ge \delta > 0$. Hence $\|\big(c_F^{(1)}(T_{mn})\big)^{-1}\|_2$ is uniformly bounded. By noting that

$$\big(c_F^{(1)}(T_{mn})\big)^{-1} T_{mn} = I - \big(c_F^{(1)}(T_{mn})\big)^{-1}\big(c_F^{(1)}(T_{mn}) - T_{mn}\big),$$

we then have the following immediate corollary.

COROLLARY 1. *Let T_{mn} be given by* (20) *with an absolutely summable generating sequence. If T_{mn} are positive definite for all m and n and $\lambda_{\min}(T_{mn}) \geq \delta > 0$, then for all $\varepsilon > 0$, there exists an $N > 0$ such that for all $n > N$ and $m > 0$, at most $O(m)$ eigenvalues of $\left(c_F^{(1)}(T_{mn})\right)^{-1} T_{mn} - I$ have absolute value larger than ε.*

As a consequence, the spectrum of $\left(c_F^{(1)}(T_{mn})\right)^{-1} T_{mn}$ is clustered around 1 except for at most $O(m)$ outlying eigenvalues. When the preconditioned conjugate gradient method is applied to solving the system $T_{mn}x = b$, Corollary 1 shows that the number of iterations will grow to a maximum similar to $O(m)$. We recall that in §3.2, the algorithm requires $O(mn \log n)$ operations in the initialization step and $O(mn \log^2 m + mn \log n)$ operations in each iteration. Thus the total complexity of the algorithm is bounded above by $O(m^2 n \log^2 m + m^2 n \log n)$.

We emphasize that for the quadrantally symmetric block Toeplitz systems we tested in §5, the number of iterations is independent of m and n and the complexity of the method is therefore $O(nm \log^2 m + nm \log n)$.

We remark again that when $m > n$, one should consider using the preconditioner $\tilde{c}_F^{(1)}(T_{mn})$ instead. Then, by repeating the whole argument we used, we can show that the preconditioned conjugate gradient method will converge in at most $O(n)$ steps for m sufficiently large. Hence the total complexity of the algorithm in this case is bounded above by $O(n^2 m \log^2 n + n^2 m \log m)$.

Before we close this subsection, we would like to point out that for the quadrantally symmetric block Toeplitz matrix T_{mn}, we can define the matrix $\tilde{s}_F^{(1)}(T_{mn})$ analogously to $\tilde{c}_V^{(1)}(T_{mn})$:

$$\tilde{s}_F^{(1)}(T_{mn}) = P^* s_F^{(1)}(P T_{mn} P^*) P,$$

where P is defined by (8). Then, as in §2.3, we can further define the doubly circulant block preconditioner $\tilde{s}_F^{(1)} \circ s_F^{(1)}(T_{mn})$. As remarked after the proof of Theorem 3, $\tilde{s}_F^{(1)} \circ s_F^{(1)}(T_{mn})$ is the approximation of T_{mn} in two directions. Therefore, it will not be a good preconditioner compared to either $s_F^{(1)}(T_{mn})$ or, in view of Lemma 5, $c_F^{(1)}(T_{mn})$. We finally remark that if Chan's preconditioner [6] is used in (22) instead of Strang's circulant preconditioner, then the corresponding doubly-circulant block preconditioner is the block preconditioner considered in Ku and Kuo [14].

4.2. Separable matrices. Next we consider the system $(A_m \otimes T_n)x = b$ where T_n is a Toeplitz matrix with generating function f, i.e., the diagonals of T_n are given by the Fourier coefficients $a_j(f)$ of f. More precisely, we have

$$(T_n)_{jk} = a_{j-k}(f), \qquad j, k = 1, 2, \cdots .$$

We assume that f is positive, 2π-periodic, and continuous, and denote T_n by $T_n(f)$. For such $T_n(f)$, we have the following result (see Chan and Yeung [4]).

LEMMA 7. *Let f be a positive, 2π-periodic, and continuous function. Then for all $\epsilon > 0$, there exist N and $M > 0$, such that for all $n > N$, at most M eigenvalues of the matrices $c_F^{-1}(T_n(f))T_n(f) - I_n$ have absolute values larger than ϵ.*

Since the preconditioned matrix is given by

$$\left[A_m \otimes c_F(T_n(f))\right]^{-1} \left(A_m \otimes T_n(f)\right) = I_m \otimes \left[c_F^{-1}(T_n(f)) T_n(f)\right],$$

it is clear that the number of distinct eigenvalues of the preconditioned matrix is the same as the number of distinct eigenvalues of $c_F^{-1}(T_n(f))T_n(f)$. In view of Lemma 7, we then see that for all $\epsilon > 0$, there exist N, $M > 0$, such that for all $n > N$ and $m > 0$, at most M distinct eigenvalues of the matrices $\{I_m \otimes [c_F^{-1}(T_n(f))T_n(f)]\} - I$ have absolute values large than ϵ. Thus the eigenvalues of the preconditioned matrix are clustered around 1 and hence the number of iterations required for convergence is a constant independent of n and m. Recalling the operation count in §3.3, the total complexity of the algorithm in this case is equal to $O(m^3 + nm^2 + mn \log n)$.

5. Numerical results.

In this section, we apply the preconditioned conjugate gradient method to the block systems we considered in §4. The stopping criteria for the method is set at

$$\frac{\|r_k\|_2}{\|r_0\|_2} < 10^{-7},$$

where r_k is the residual vector at the kth iteration. The right-hand side vector b is chosen to be the vector of all ones and the zero vector is the initial guess.

5.1. Quadrantally symmetric block Toeplitz matrices.

We consider T_{mn} of the form given in (20) with the diagonals of the blocks $T_{(j)}$ being given by $t_i^{(j)}$. Four different generating sequences were tested. They are

$$\text{(i)}\quad t_i^{(j)} = \frac{1}{(j+1)(|i|+1)^{1+0.1\times(j+1)}}, \qquad j \geq 0, \quad i = 0, \pm1, \pm2, \cdots,$$

$$\text{(ii)}\quad t_i^{(j)} = \frac{1}{(j+1)^{1.1}(|i|+1)^{1+0.1\times(j+1)}}, \qquad j \geq 0, \quad i = 0, \pm1, \pm2, \cdots,$$

$$\text{(iii)}\quad t_i^{(j)} = \frac{1}{(j+1)^{1.1} + (|i|+1)^{1.1}}, \qquad j \geq 0, \quad i = 0, \pm1, \pm2, \cdots,$$

$$\text{(iv)}\quad t_i^{(j)} = \frac{1}{(j+1)^{2.1} + (|i|+1)^{2.1}}, \qquad j \geq 0, \quad i = 0, \pm1, \pm2, \cdots.$$

The generating sequences (ii) and (iv) are absolutely summable while (i) and (iii) are not. Tables 1 and 2 show the number of iterations required for convergence. In all cases, we see that as $m = n$ increases, the number of iterations remains roughly a constant or increases very slowly for the preconditioned system with preconditioner $c_F^{(1)}(T_{mn})$ while it increases with other choices of preconditioners.

TABLE 1
Preconditioners used and the number of iterations.

		Sequence (i)			Sequence (ii)		
$n = m$	mn	None	$c_F^{(1)}(T_{mn})$	$c_{F,F}^{(2)}(T_{mn})$	None	$c_F^{(1)}(T_{mn})$	$c_{F,F}^{(2)}(T_{mn})$
8	64	20	6	12	19	5	12
16	256	35	6	18	32	6	17
32	1024	43	6	21	41	6	20
64	4096	51	7	25	47	7	22
128	16384	54	7	26	50	7	24

TABLE 2
Preconditioners used and the number of iterations.

		Sequence (iii)			Sequence (iv)		
$n=m$	mn	None	$c_F^{(1)}(T_{mn})$	$c_{F,F}^{(2)}(T_{mn})$	None	$c_F^{(1)}(T_{mn})$	$c_{F,F}^{(2)}(T_{mn})$
8	64	18	7	16	14	7	12
16	256	40	8	30	22	8	20
32	1024	63	9	49	30	9	26
64	4096	101	11	80	36	9	33
128	16384	144	12	123	42	8	38

5.2. Separable matrices. We consider the separable block Toeplitz system $(\tilde{T}_m \otimes T_n)x = b$ where the diagonals of $\tilde{T}_m$ and T_n are given by $\tilde{t}_i = (|i|+1)^{-1}$ and $t_j = (|j|+1)^{-1.1}$, respectively, for $i, j = 0, \pm1, \pm2, \cdots$. We note that $\tilde{T}_m \otimes T_n$ is also a quadrantally symmetric block Toeplitz matrix with the generating sequence given by

$$t_j^{(i)} = \frac{1}{(i+1)(|j|+1)^{1.1}}, \quad i \geq 0, \quad j = 0, \pm1, \pm2, \cdots .$$

The preconditioner $c_F^{(1)}(\tilde{T}_m \otimes T_n)$ is given by $\tilde{T}_m \otimes c_F(T_n)$. Table 3 shows the number of iterations required for convergence. We notice that as $n = m$ increases, the number of iterations stays almost the same for the preconditioned system with preconditioner $c_F^{(1)}(\tilde{T}_m \otimes T_n)$ while it increases with other choices of preconditioners. We remark that since $\tilde{T}_m$ is a Toeplitz matrix, its inverse can be obtained in $O(m \log^2 m)$. Hence the total complexity of the algorithm is reduced to $O(mn \log^2 m + mn \log n)$.

TABLE 3
Preconditioners used and the number of iterations.

$n=m$	mn	None	$c_F(\tilde{T}_m T_m) \otimes c_F(T_n)$	$\tilde{T}_m \otimes I_n$	$\tilde{T}_m \otimes c_F(T_n)$
8	64	20	7	5	4
16	256	34	9	10	4
32	1024	48	9	14	5
64	4096	57	10	18	5
128	16384	67	11	20	5

Acknowledgment. This research was initiated by Professor Tony F. Chan during his visit to the University of Hong Kong in December 1990. We would like to thank him for his very helpful suggestions and guidance in the preparation of this paper.

REFERENCES

[1] G. AMMAR AND W. GRAGG, *Implementation and use of the generalized Schur algorithm*, Computational and Combinatorial Methods in Systems, C. Byrnes and A. Lindquist, eds., North-Holland, Amsterdam, 1986, pp. 265–280.

[2] R. CHAN AND G. STRANG, *Toeplitz equations by conjugate gradients with circulant preconditioner*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 104–119.

[3] R. CHAN, X. JIN, AND M. YEUNG, *The circulant operator in the Banach algebra of matrices*, Linear Algebra Appl., 149 (1991), pp. 41–53.

[4] R. CHAN AND M. YEUNG, *Circulant preconditioners for Toeplitz matrices with positive continuous generating functions*, Math. Comput., to appear.

[5] R. Chan and M. Yeung, *Circulant preconditioners constructed from kernels*, SIAM J. Numer. Anal., 29 (1993), to appear.

[6] R. Chan, *Circulant preconditioners for Hermitian Toeplitz systems*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 542–550.

[7] ———, *Numercal solutions for the inverse heat problems in R^N*, The SEAMS Bull. Math., to appear.

[8] R. Chan and T. Chan, *Circulant preconditioners for elliptic problems*, J. Numer. Linear Algebra Appl., to appear.

[9] T. Chan, *An optimal circulant preconditioner for Toeplitz systems*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 766–771.

[10] T. Chan and J. Olkin, *Preconditioners for Toeplitz-block matrices*, in Second SIAM Conference on Linear Algebra in Signals, Systems, and Control, San Francisco, CA, 1990.

[11] P. Davis, *Circulant Matrices*, John Wiley and Sons, New York, 1979.

[12] G. Golub and C. Van Loan, *Matrix Computations,* Second Edition, The Johns Hopkins University Press, Baltimore, MD, 1989.

[13] T. Huckle, *Circulant/skewcirculant matrices for solving Toeplitz matrix problems*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 767–777.

[14] T. Ku and C. Kuo, *On the Spectrum of a Family of Preconditioned Block Toeplitz Matrices*, USC-SIPI Report #164, Signal and Image Processing Institute, University of Southern California, Los Angeles, CA, 1990.

[15] J. Olkin, *Linear and nonlinear deconvolution problems*, Ph.D. thesis, Department of Mathematics, Rice University, Houston, TX, 1986.

[16] G. Strang, *A proposal for Toeplitz matrix calculations*, Stud. Appl. Math., 74 (1986), pp. 171–176.

[17] R. Thompson, *Principal submatrices* IX: *Interlacing inequalities for singular values of submatrices*, Linear Algebra Appl., 5 (1972), pp. 1–12.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 5, pp. 1236–1264, September 1992

011

EFFICIENT SOLUTION OF PARABOLIC EQUATIONS BY KRYLOV APPROXIMATION METHODS*

E. GALLOPOULOS† AND Y. SAAD‡

Abstract. This paper takes a new look at numerical techniques for solving parabolic equations by the method of lines. The main motivation for the proposed approach is the possibility of exploiting a high degree of parallelism in a simple manner. The basic idea of the method is to approximate the action of the evolution operator on a given state vector by means of a projection process onto a Krylov subspace. Thus the resulting approximation consists of applying an evolution operator of very small dimension to a known vector, which is, in turn, computed accurately by exploiting high-order rational Chebyshev and Padé approximations to the exponential. Because the rational approximation is only applied to a small matrix, the only operations required with the original large matrix are matrix-by-vector multiplications and, as a result, the algorithm can easily be parallelized and vectorized. Further parallelism is introduced by expanding the rational approximations into partial fractions. Some relevant approximation and stability issues are discussed. Some numerical experiments are presented with the method and its performance is compared with a few explicit and implicit algorithms.

Key words. parabolic problems, method of lines, explicit methods, Krylov subspace, parallelism, matrix exponential, polynomial approximation, exponential propagation, rational approximation, partial fractions, stability

AMS(MOS) subject classifications. 65M20, 65F10, 65W05

1. Introduction.

In recent years there has been a resurgence of interest in explicit methods for solving parabolic partial differential equations (PDEs), motivated mainly by the desire to exploit the parallel and vector processing capabilities of new supercomputer architectures. The main attraction of explicit methods is their simplicity, since the basic operations involved in them are matrix-by-vector multiplications, which, in general, are rather easy to parallelize and vectorize. On the other hand, the stringent constraint on the size of the timesteps required to ensure stability reduces efficiency to such an extent that the use of implicit methods becomes almost mandatory when integrating on long time intervals. Implicit methods do not suffer from stability related restrictions but have another disadvantage: they require the solution of linear systems that are often large and sparse. For this reason implicit methods tend to be far more difficult to implement on parallel machines than their explicit counterparts, which only require matrix-by-vector multiplications. Thus the trade-off between the two approaches seems to be a large number of matrix-by-vector multiplications on the one hand, versus linear systems to solve on the other. For two-dimensional and, more important, for three-dimensional problems, methods of an explicit type might be attractive if implemented with care.

* Received by the editors February 26, 1990; accepted for publication (in revised form) August 14, 1991.

† Department of Computer Science and Center for Supercomputing Research and Development, University of Illinois, Urbana, Illinois 61801-2932 (stratis@csrd.uiuc.edu). This research was supported by National Science Foundation grant NSF CCR-8717942. Additional support was provided by AT&T grant AT&T-AFFL-67-Sameh, Air Force Office of Scientific Research grant AFOSR-90-0044, and Department of Energy grant DOE DE-FG02-85ER25001.

‡ Computer Science Department, University of Minnesota, 200 Union St. SE, Minneapolis, Minnesota 55455 (saad@cs.umn.edu). This work was supported by the National Academy of Sciences Systems Division and/or Defense Advanced Research Projects Agency via Cooperative Agreement NCC 2-387 between the National Aeronautics and Space Administration (NASA) and the University Space Research Association (USRA).

We next observe that in spite of the above conventional wisdom, the distinction between the explicit and implicit approaches is not always clear. Consider the simple system of ordinary differential equations $y' = Ay+f$. We first point out that if our only desire is to use matrix-by-vector products exclusively as operations with the matrix A, for example, for the purpose of exploiting modern architectures, then certainly standard explicit methods do not constitute the only possibility. For example, one may use an implicit scheme and solve the linear systems approximately by some iterative method, such as the conjugate gradient (CG) method, with no preconditioning or with diagonal preconditioning. When the accuracy required for each linear system is very low, then the method will be akin to an explicit scheme, although one of rather unusual type since its coefficients will vary at every step. As the accuracy required for the approximations increases, the method will start moving towards the family of purely implicit methods. Therefore, if we were to call any scheme that requires only matrix–vector products "explicit," then the borderline between the two approaches is not so well defined.

We would like to take advantage of this observation to develop schemes that are intermediate between explicit and implicit. In this paper we will use the term *polynomial approximation method* for any scheme for which the only operations required with the matrix A are matrix-by-vector multiplications. The number of such operations may vary from one step to another and may be large.

To derive such intermediate methods, we systematically explore the ways in which an approximation to the local behavior of the ordinary differential equations (ODEs) can be obtained by using polynomials in the operator A. Going back to the comparison sketched above, we note that the process involved in one single step of an implicit method is often simply an attempt to generate some approximation to the operation $\exp(\delta t\ A)v$ via a rational approximation to the evolution operator [52]. If a CG-like method is used to solve the linear systems arising in the implicit procedure, the result will be a polynomial scheme. Thus there are two phases of approximation: the first is obtaining a rational or polynomial approximation to the exponential, and the second is solving the linear systems by some iterative method. We would like to reduce these two phases to only one by attempting to directly approximate $\exp(\delta t\ A)v$. The basic idea is to project the exponential of the large matrix A into a small Krylov subspace.

To make the discussion more specific and introduce some notation, we consider the following linear parabolic PDE:

$$\begin{aligned} \frac{\partial u(x,t)}{\partial t} &= -Lu(x,t) + r(x), \qquad x \in \Omega, \\ u(x,0) &= u_0, \qquad x \in \Omega, \\ u(x,t) &= \sigma(x), \quad x \in \partial\Omega, \quad t > 0, \end{aligned} \tag{1.1}$$

where $-L$ is a second-order partial differential operator of the elliptic type, acting on functions defined on the open, bounded, and connected set Ω. Using a method of lines (MOL) approach, (1.1) is first discretized with respect to space variables, resulting in the system of, say, n ODEs

$$\frac{dw(t)}{dt} = -Aw(t) + r, \qquad w(0) = w_0. \tag{1.2}$$

Here w, r are vectors and A is a matrix of order n. For the remainder of our discussion, we will assume A to be time independent. In this situation the solution is explicitly given by

$$w(t) = A^{-1}r + e^{-tA}(w_0 - A^{-1}r). \tag{1.3}$$

If we let $\hat{w}(t) \equiv w(t) - A^{-1}r$, and accordingly, $\hat{w}_0 \equiv w_0 - A^{-1}r$, then (1.3) is equivalent to the following expression:

$$\hat{w}(t) = e^{-tA}\hat{w}_0.$$

Note that when $r = 0$, $w(t)$ is the same as $\hat{w}(t)$. An ideal one-step method would consist of a scheme of the form

$$\hat{w}(t + \delta) = e^{-\delta A}\hat{w}(t), \tag{1.4}$$

in which δ constitutes the timestep.

The basic operation in the above formula is the computation of the exponential of a given matrix times a vector. If we were able to perform this basic operation with high accuracy, we would have what is sometimes called a nonlinear one-step method [26], because it involves a nonlinear operation with the matrix A. We should stress that there is no need to actually evaluate the matrix exponential $\exp(-\delta A)$, but only its product with a given vector. This brings to mind an analogous situation for linear systems in which it is preferable to solve $Ax = b$ than to compute A^{-1} and then multiply the solution by b.

We point out that we follow an approach common in the literature [3], [42], putting the emphasis on the semidiscrete problem (1.2). As a result, our discussion of stability is purely from an ordinary differential equation point of view and is not concerned with the effect of space discretization errors and convergence. We establish conditions under which our methods, applied to the stiff system of ODEs (1.2), satisfy certain criteria of stability, which, in turn, is an important step toward any investigations of convergence. (See also [5] and [40].)

The Krylov subspace method presented here was introduced in [13] for general nonsymmetric matrices. However, similar ideas have been used previously in various ways in different applications for symmetric or skew-symmetric matrices. For example, recall the use of this basic idea in Park and Light [34] following the work by Nauts and Wyatt [30]. The idea of exploiting the Lanczos algorithm in order to evaluate terms of the exponential of Hamiltonian operators seems to have been first used in chemical physics by Nauts and Wyatt in the context of the recursive-residue-generation method [29]. More recently, Friesner, Tuckerman, Dornblaser, and Russo [9] have demonstrated that these techniques can be extended to solving nonlinear stiff differential equations. The approach developed in this paper is related to that of Gear and Saad [14] and Brown and Hindmarsh [1] on matrix-free methods in the Newton iteration of implicit multistep schemes. Our scheme uses a similar reduction of the underlying matrix but goes further in combining it with high-order approximation schemes for the reduced (exponential) operator. It is also related to the work of Nour-Omid [32], in which systems of ODEs are solved by first projecting into Krylov subspaces with an unsymmetric two-sided Lanczos process and then solving tridiagonal systems of ODEs, the approach of Tal-Ezer and Kosloff [46], and the work of Tal-Ezer [45] and Schaefer [41] on polynomial methods based on Chebyshev expansions. The idea of evaluating arbitrary functions of a Hermitian matrix with the use of the Lanczos algorithm has also been mentioned by van der Vorst [51]. The use of preconditioning for extending the stability interval of explicit methods, thus bringing them closer to fully implicit methods, has been discussed in [37] and [50]. Although our method works from a subspace, it does not suffer from some of the aspects of partitioning methods (see, for example, [55]). Partitioning methods rely on explicitly separating and treating differently the stiff and nonstiff parts. However, it is usually impractical to confine stiffness to a subsystem [4]. The Krylov method, on the other hand, relies on the nice convergence property of Krylov approximations to essentially reach a similar goal in an implicit manner [38]. The outermost eigenvalues, including the largest ones, will be well approximated by the Krylov subspace, so that the Krylov approximation to the matrix exponential will be accurate in those eigenvalues, thus accommodating stiffness. Note that there have been several recent efforts to design algorithms for the solution of time-dependent problems, some of which may be particularly suited to parallel processing; see [19], [21], [22], [43], [48], and [49] for a review. It should also be mentioned that the approach we are using to approximate and evaluate the exponential of the reduced operator has its roots in the work of Varga [52] and in previous work by the present authors and others; see §3 for details.

The structure of our paper is as follows. In §2 we formulate the Krylov subspace approximation algorithm and prove some a priori error bounds. In §3 we present a method for the accurate approximation of the exponential of the Hessenberg matrix produced in the course of the Arnoldi or Lanczos algorithm. In §4 we consider problems with time-dependent forcing and introduce two approaches to handle the integration of the nonhomogeneous term. We then proceed in §5 with a stability analysis of each approach in the context of the quadrature techniques used, leading to Theorem 5.3. In §6 we present numerical experiments for problems of varying difficulty and, finally, in §7, we offer concluding remarks.

2. Polynomial approximation and the use of Krylov subspaces.

In this section we consider using polynomial approximation to (1.4), that is, we seek an approximation of the form

$$e^{-A}v \approx p_{m-1}(A)v, \tag{2.1}$$

where p_{m-1} is a polynomial of degree $m-1$ and A and v are an order-n matrix and vector, respectively. There are several ways in which polynomial approximations can be found. The simplest technique is to attempt to minimize some norm of the error $e^{-z} - p_{m-1}(z)$ on a continuum in the complex plane that encloses the spectrum of A. For example, Chebyshev approximation can be used, but one disadvantage is that it requires some approximation to the spectrum of A. In this paper we consider only approaches that do not require any information on the spectrum of A. This will be considered in §2.1. A theoretical analysis then follows in §2.2.

2.1. The Krylov subspace approximation.

The approximation (2.1) to $e^{-A}v$ is to be taken from the Krylov subspace

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

In order to manipulate vectors in K_m, it is convenient to generate an orthonormal basis $V_m = [v_1, v_2, v_3, \cdots, v_m]$. We take as initial vector $v_1 = v/\|v\|_2$ and generate the basis V_m with the well-known Arnoldi algorithm, described below.

ALGORITHM: ARNOLDI.

```
1. Initialize:
   Compute $v_1 := v/\|v\|_2$.
```

```
2. Iterate: Do $j = 1, 2, \cdots, m$.
    1. Compute $w := Av_j$.
    2. Do $i = 1, 2, \cdots, j$.
        (a) Compute $h_{i,j} := (w, v_i)$.
        (b) Compute $w := w - h_{i,j}v_i$.
    3. Compute $h_{j+1,j} = \|w\|_2$ and $v_{j+1} = w/h_{j+1,j}$.
```

By construction, the above algorithm produces an orthonormal basis V_m of the Krylov subspace K_m, where $V_m = [v_1, v_2, \cdots, v_m]$. If we denote the $m \times m$ upper Hessenberg matrix consisting of the coefficients h_{ij} computed from the algorithm by H_m, we have the relation

$$AV_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^{\mathrm{T}}. \tag{2.2}$$

For the remainder of this discussion, for any given k, e_k will denote the kth unit vector belonging to R^m. From the orthogonality of the columns of V_m we get that $H_m = V_m^{\mathrm{T}} A V_m$. Therefore H_m represents the projection of the linear transformation A to the subspace K_m, with respect to the basis V_m.

Since V_m is orthonormal, the vector $x_{\mathrm{opt}} = V_m V_m^{\mathrm{T}} e^{-A} v$ is the projection of $e^{-A}v$ on K_m, that is, it is the closest approximation to $\exp(-A)v$ from K_m. Since for $\beta \equiv \|v\|_2$, we can write $v = \beta v_1$ and $v_1 = V_m e_1$, it follows that

$$\begin{aligned} V_m V_m^{\mathrm{T}} e^{-A} v &= \beta V_m V_m^{\mathrm{T}} e^{-A} v_1 \\ &= \beta V_m V_m^{\mathrm{T}} e^{-A} V_m e_1. \end{aligned}$$

We can therefore write the optimal solution as $x_{\mathrm{opt}} \equiv V_m y_{\mathrm{opt}}$, where $y_{\mathrm{opt}} \equiv \beta V_m^{\mathrm{T}} e^{-A} V_m e_1$. Unfortunately, y_{opt} is not practically computable since it still involves e^{-A}. We can approximate $V_m^{\mathrm{T}} e^{-A} V_m$ by e^{-H_m}, leading to the approximation $y_{\mathrm{opt}} \approx \beta e^{-H_m} e_1$ and

$$e^{-A} v \approx \beta V_m e^{-H_m} e_1. \tag{2.3}$$

From the practical point of view there remains the issue of efficiently computing the vector $e^{-H_m} e_1$, which we address in §3.

The approximation (2.3) is central to our method, and its effectiveness is discussed throughout the remainder of the paper. The next section is devoted to providing the theoretical justification.

We also note that when A is symmetric, Arnoldi's algorithm simplifies into the Lanczos process, which entails a three-term recurrence. This is a result of the fact that the matrix $H_m = V_m^{\mathrm{T}} A V_m$ must be symmetric and therefore tridiagonal symmetric, and so all $h_{i,j} = 0$ for $i = 1, 2, \cdots, j-2$. However, the resulting vectors, which are in theory orthogonal to each other, tend to lose their orthogonality rapidly.

2.2. A priori error bounds and general theory. The next question that arises concerns the quality of the Krylov subspace approximation defined in §2.1. A first observation is that the above approximation is exact for $m = n$, because in this situation $v_{m+1} = 0$ and (2.2) becomes $AV_m = V_m H_m$, where V_m is an $n \times n$ orthogonal matrix. In fact, as with the conjugate gradient method and the Arnoldi process, the approximation is exact for m whenever m is larger than or equal to the degree of the minimal polynomial of v_1 with respect to A. As for these algorithms, we need to investigate what happens when m is much smaller than this degree.

In the sequel we need to use the concept of the *logarithmic norm* of a matrix. Let B be a given matrix. The logarithmic norm $\mu(.)$ is defined by

$$\mu(B) \equiv \lim_{h \to 0+} \frac{\|I + hB\| - 1}{h}.$$

Note that μ is associated with a particular norm. Unless it is otherwise specified, we assume that the reference norm is the usual two-norm. Then the logarithmic norm $\mu(B)$ is equal to the maximum eigenvalue of the symmetric part of B, that is,

$$\mu(B) = \lambda_{\max}\left(\frac{B + B^{\mathrm{T}}}{2}\right).$$

The function μ satisfies many norm-like properties, but it can also take negative values. We refer to [5] and [6] for a description of its properties. It can be shown in particular that

$$\|e^{Bt}\| \le e^{\mu(B)t}. \tag{2.4}$$

We assume throughout that A is a real matrix. We now state the main theorem of this section.

THEOREM 2.1. *Let A be any matrix and let $\rho \equiv \|A\|_2$, $\beta = \|v\|_2$, and $\eta \equiv \mu(-A)$. Then the error of the approximation* (2.3) *is such that*

$$\|e^{-A}v - \beta V_m e^{-H_m} e_1\|_2 \;\le\; 2\beta\rho^m \phi(\eta) \;\le\; 2\beta\frac{\rho^m}{m!}\max(1, e^{\eta}), \tag{2.5}$$

where

$$\phi(\eta) \equiv \frac{1}{\eta^m}\left(e^{\eta} - \sum_{k=0}^{m-1}\frac{\eta^k}{k!}\right).$$

The proof of the theorem is established in Appendix A.

To see what we can gain in using the logarithmic norm instead of a standard spectral norm, compare Theorem 2.1 with the bound proposed earlier in [13]:

$$\|e^{-A}v - \beta V_m e^{-H_m} e_1\|_2 \le 2\beta\frac{\rho^m e^{\rho}}{m!}. \tag{2.6}$$

For the sake of illustration, let

$$B = \begin{pmatrix} 0.5000 & -0.0938 & 0.0000 \\ -0.4063 & 0.5000 & -0.0938 \\ 0.0000 & -0.4063 & 0.5000 \end{pmatrix}$$

and let $A = I \otimes B + B \otimes I$, where $\otimes$ is the symbol for the Kronecker product, and I the identity matrix. Such an A arises after discretizing $-(u_{xx} + u_{yy}) + \xi(u_x + u_y)$ on the unit square with a grid distance of $\frac{1}{4}$ when $\xi = 5$ and after multiplying all coefficients of the discretization by the scale factor $\frac{1}{64}$ (e.g., representing a timestep). In that case $\mu(-A) = -0.2929$, whereas $\|A\|_2 = 1.7235$. When $m = 7$, the error in the approximation from (2.6) is bounded by 0.1004β, whereas Theorem 2.1 provides the sharper estimate 0.018β.

In general, the advantage of using the logarithmic norm follows from the inequality $\|A\| \geq \mu(-A)$, which is among the properties of $\mu(.)$ (cf. [6]). One can construct examples, however, for which the bounds from using $\mu(-A)$ are as loose as those obtained from using $\|A\|$ (cf. [5]). Also, asymptotically the rates of convergence as estimated by the bounds (2.6) and (2.5) are both of the form $\rho/(m!)^{1/m}$. The following corollary follows trivially from Theorem 2.1.

COROLLARY 2.2. *If the eigenvalues of the symmetric part of the matrix A are nonnegative, then*

$$\|e^{-A}v - \beta V_m e^{-H_m} e_1\|_2 \leq 2\beta \frac{\rho^m}{m!}.$$

Hence the bound of Corollary 2.2 holds for many important classes of matrices, including positive definite matrices and normal matrices with eigenvalues in the positive half-plane.

We note that when A is normal, the bound of Corollary 2.2 can be derived without invoking logarithmic norms because we can write $A = Q\Lambda Q^{\mathrm{H}}$ where Λ is the diagonal matrix of eigenvalues of A and Q is unitary. Let us denote the remainder after m terms of the Taylor series expansion of e^{-z} by $r_m(z)$. Then

$$\|r_m(A)\|_2 = \left\|\sum_{k=m}^{\infty} \frac{1}{k!}(-A)^k\right\|_2 = \left\|\sum_{k=m}^{\infty} \frac{1}{k!}(-\Lambda)^k\right\|_2 .$$

From the assumption on A, $\Re[\Lambda] \geq 0$. Applying componentwise a result of Landau [24] (cf. [36, p. 35, problem 151]), the remainder can be bounded by its first term:

$$\|r_m(A)\|_2 \leq \frac{\|-\Lambda^m\|_2}{m!} = \frac{\|A^m\|_2}{m!}.$$

The bound of Corollary 2.2 follows after using a similar treatment for H_m and combining the results as shown in Lemma A.1 of Appendix A.

When we know that A is symmetric positive definite, an even better bound can be obtained by applying the previous theory to $A-\zeta I$, where $\zeta \geq \lambda_{\min}(A)$ (the minimum eigenvalue of A). We refer to [12] for the proof.

THEOREM 2.3. *Let A be a symmetric positive definite matrix and let $\rho = \|A\|_2$ and $\beta = \|v\|_2$. Then the error of the approximation* (2.3) *is such that*

$$\|e^{-A}v - \beta V_m e^{-H_m} e_1\|_2 \leq \beta \frac{\rho^m}{2^{m-1} m!}. \tag{2.7}$$

These theorems show that in the nonstiff situation, which occurs at the beginning of the integration, the proposed method cannot be much worse than a more conventional explicit method. Experimental results in §6 show that in the stiff situation the actual errors are much smaller than suggested by the theorems. In the nonstiff region the theorems can serve as a guide to choosing the stepsize in a timestepping procedure. Indeed, if we were to replace A by the scaled matrix τA, then the Krylov subspace would remain the same, that is, V_m would not change and H_m would be scaled to τH_m. As a result, for arbitrary τ one can use the approximation

$$e^{-\tau A}v \approx \beta V_m e^{-\tau H_m} e_1, \tag{2.8}$$

and the bound (2.5) becomes

$$\|e^{-\tau A}v - \beta V_m e^{-\tau H_m} e_1\|_2 \leq 2\beta(\tau\rho)^m \phi(\mu(-\tau A)). \tag{2.9}$$

The consequence of (2.9) is that by reducing the stepsize one can always make the scheme accurate enough without changing the dimension m. We note that these bounds are most useful when m is much larger than what is usually used by standard explicit methods. Indeed, in our experiments, we have used large values of m to our advantage. We refer the reader to [39] for additional results on error bounds for this method.

3. Practical computation of $\exp(-H_m)e_1$.

We now address the problem of evaluating $y = e^{-H}e_1$, where H is the Hessenberg or tridiagonal matrix produced by Arnoldi's method or the Lanczos method. We drop the subscript m for convenience. Although H is a small matrix, the cost of computing y can easily become nonnegligible. See [28] for a review of methods for computing the matrix exponential. For example, when H is tridiagonal symmetric, the simplest technique for computing y is based on the QR algorithm. However, this is rather expensive. Therefore, we would like to use approximations that have high accuracy, possess desirable stability properties, and allow fast evaluation. The method we recommend is to use rational approximation to the exponential, evaluated by partial fraction expansion. This technique has been discussed in the context of implicit methods in [10] and [13], and we would like to take advantage of it in the present context. The (serial) complexity of the QR factorization algorithm is $O(m^3)$ for Hessenberg matrices and $O(m^2)$ for tridiagonal matrices, compared with a cost of $O(m^2)$ for Hessenberg and $O(m)$ for tridiagonal matrices when using the rational approximation method. In addition, parallelism can be exploited in this approach.

The rational approximation to the exponential has the form

$$e^{-z} \approx R_{\nu_1,\nu_2}(z) \equiv \frac{p_{\nu_1}(z)}{q_{\nu_2}(z)}, \tag{3.1}$$

where p_{ν_1} and q_{ν_2} are polynomials of degrees ν_1 and ν_2, respectively.

An approximation of this type, referred to as a Padé approximation, is determined by matching the Taylor series expansion of the left-hand side and right-hand side of (3.1) at the origin. Since Padé approximations are local, they are very accurate near the origin but may be inaccurate far away from it. Other schemes have been developed [3], [18], [52] to overcome this difficulty. For typical parabolic problems that involve a second-order partial differential operator $-L$ that is self-adjoint elliptic, the eigenvalues of L are located in the interval $[0, +\infty)$. It is therefore natural to follow the idea introduced by Varga in [52] (see also [2] and [3]) and seek the Chebyshev (uniform) rational approximation to the function e^{-z}, which minimizes the maximum error on the interval $[0, +\infty)$. To unify the Padé and uniform approximation approaches, we restrict ourselves to "diagonal" approximations of the form (ν, ν), that is, in which the numerator has the same degree ν as the denominator. We note, however, that alternative strategies (e.g., $(\nu-1, \nu)$) frequently work better for Padé approximations without altering the principle of the method. Note that the stability properties of the aforementioned rational approximations are discussed extensively in the literature [7], [20], [54].

A comparison between the Padé approximation and the Chebyshev rational approximation reveals the vast superiority of the latter in the context of the Krylov-based methods presented in this paper, at least for symmetric positive definite matrices H and relatively large values of m. To see why this is so, we note that the idea of the method presented in this paper is to allow the use of large timesteps by utilizing Krylov subspaces of relatively high order. However, for our method to be successful, the ability to use a large timestep δ must also carry over to the computation of $\exp(-H\delta)e_1$. We mentioned earlier that the Padé approximations provide good approximation only near the origin. Using the Chebyshev rational approximation to the function e^{-z} over the interval $[0, +\infty)$ [2], [52], it becomes possible to utilize timesteps as large as our Krylov-based method allows.

For example, in the diagonal Chebyshev rational approximation, the infinity norm of the error over the interval $[0, +\infty)$ is of the order of 10^{-10} as soon as ν reaches 10. For each additional degree the improvement is of the order of $9.289025\cdots$ [2]. This means that for all practical purposes e^{-z} can be replaced by a rational function of relatively small degree. When H is nonsymmetric and its eigenvalues are complex, then the rational function is no longer guaranteed to be an accurate approximation to the exponential. Although a rigorous analysis is lacking, we experimentally verified that for the examples we treated, the approximation still remained remarkably accurate when the eigenvalues were near the positive real axis. Although little is known concerning rational uniform approximation in general regions of the complex plane, a promising alternative is to use asymptotically optimal methods based on Faber transformations in the complex plane [8]. We also point out that there exist other techniques for approximating matrix exponentials by rational functions of A; see, for example, [18] and [31]. The restricted Padé approximations of [31] avoid complex arithmetic at the price of a reduced order of approximation and reduced levels of parallelism caused by the occurrence of multiple poles.

For compactness of notation in the diagonal approximations we simply write R_ν from now on for the (ν, ν) rational approximation to e^{-z}. Then, in order to evaluate the corresponding approximation to $e^{-H}e_1$, we need to evaluate the vector $\tilde{y}$, where

$$\tilde{y} = p_\nu(H)q_\nu(H)^{-1}e_1 = q_\nu(H)^{-1}p_\nu(H)e_1. \tag{3.2}$$

It has been proposed in several contexts that an efficient method for computing some rational matrix functions is to resort to their partial fraction expansions [10], [11], [13], [23], [25], [33], [44], [56]. The approach is possible since it can be proved analytically that the diagonal Padé approximation to e^{-z} has distinct poles [57]. Explicit calculations indicate that this seems to be true for the uniform approximation as well. In particular, we write

$$R_\nu(z) = \alpha_0 + \sum_{i=1}^{\nu} \frac{\alpha_i}{z - \lambda_i},$$

where

$$\alpha_0 \equiv \frac{\pi_\nu}{\kappa_\nu} \quad \text{and} \quad \alpha_i \equiv \frac{p_\nu(\lambda_i)}{q'_\nu(\lambda_i)}, \qquad i = 1, 2, \cdots, \nu,$$

in which π_ν, κ_ν are the leading coefficients of the polynomials p_ν and q_ν, respectively.

With this expansion the algorithm for computing (3.2) becomes

```
ALGORITHM
1. For $i = 1, 2, \cdots, \nu$ solve $(H - \lambda_i I)y_i = e_1$.
```

2. Compute $\tilde{y} = \alpha_0 e_1 + \sum_{i=1}^{\nu} \alpha_i y_i$.

The motivation in [10] for using the above scheme was parallelism. The first step in the above algorithm is entirely parallel since the linear systems $(H - \lambda_i I)y_i = e_1$ can be solved independently from one another. The partial solutions are then combined in the second step. The matrices arising in [10] are large and sparse, unlike those here. However, parallel implementation of the above algorithm can be beneficial for small Hessenberg matrices as well. For example, in a parallel implementation of the Krylov scheme, the "Amdahl effect" may cause severe reduction in efficiency unless all stages of the computation are sufficiently parallelized.

We should also point out that even on a scalar machine, the above algorithm represents the best way of computing $\tilde{y}$. It requires fewer operations than a straightforward use of the expression (3.2). It is also far simpler to implement. The poles λ_i and partial fraction coefficients α_i of $R_\nu(z)$ are computed once and for all and coded in a subroutine or tabulated. These are shown in Appendix B for $\nu = 10$ and $\nu = 14$ for the case of Chebyshev rational approximation.

4. The case of a time-dependent forcing term. In the previous sections we made the restrictive assumption that the function r in the right-hand side is constant with respect to time. In this section we address the more general case where r is time dependent. In other words, we now consider the system of ODEs of the form

$$\frac{dw(t)}{dt} = -Aw(t) + r(t). \tag{4.1}$$

As is well known, the solution of this system is

$$w(t) = e^{-tA}w_0 + \int_0^t e^{(s-t)A} r(s)ds.$$

Proceeding as in §1, we now express $w(t+\delta)$ as

$$\begin{aligned} w(t+\delta) &= e^{-\delta A}\left(w(t) + \int_t^{t+\delta} e^{-(t-s)A} r(s)ds\right) \\ &= e^{-\delta A}w(t) + \int_t^{t+\delta} e^{-(\delta+t-s)A} r(s)ds \\ &= e^{-\delta A}w(t) + \int_0^{\delta} e^{-(\delta-\tau)A} r(t+\tau)d\tau. \end{aligned} \tag{4.2}$$

In one way or another, the use of the above expression as the basis for a timestepping procedure requires numerical integration. Note, however, that under the assumption that we can evaluate functions of the form $e^{-As}v$ accurately, we have transformed the initial problem into that of evaluating integrals. Simple though this statement may seem, it means that the concerns about stability disappear as soon as we consider that we are using accurate approximations to the exponential. This is because the variable w does not appear in the integrand. (The issue of stability will be examined in detail in §5.)

We will now address how to evaluate the integral in (4.2); for this we consider two distinct approaches.

4.1. The first approach. Consider a general quadrature formula of the form

$$\int_0^\delta e^{-(\delta-\tau)A} r(t+\tau)d\tau \approx \delta \sum_{j=1}^{p} \mu_j e^{-(\delta-\tau_j)A} r(t+\tau_j), \tag{4.3}$$

where the τ_j's are the quadrature nodes in the interval $[0,\delta]$. One of the simplest rules is the trapezoidal rule on the whole interval $[0,\delta]$, which leads to

$$\begin{aligned} w(t+\delta) &= e^{-\delta A} w(t) + \frac{\delta}{2} e^{-\delta A} r(t) + \frac{\delta}{2} r(t+\delta) \\ &= e^{-\delta A}\left[w(t) + \frac{\delta}{2} r(t)\right] + \frac{\delta}{2} r(t+\delta). \end{aligned}$$

The above formula is attractive because it requires only one evaluation of an exponential times a vector. On the other hand, it may be too inaccurate to be of any practical value since it means that we may have to reduce the stepsize δ drastically in order to get a good approximation to the integrals. The next alternative is to use a higher-order formula, that is, a larger p in (4.3). For example, we tried a Simpson formula instead of trapezoidal rule. The improvements are noticeable, but we have to pay the price of an additional exponential evaluation at the midpoint $t+\delta/2$.

The recommended alternative is again based on a judicious exploitation of Krylov subspaces. In the formula (4.3) we note that each term $e^{-(\delta-\tau_j)A} r(t+\tau_j)$ need not be evaluated exactly. Observe that in the ideal situation where $r(s)$ is constant (equal to r) in the interval $[t, t+\delta]$, then formula (2.8) shows that we can evaluate $e^{-(\delta-\tau)A} r$ for all τ from the Krylov subspace generated for $\tau = 0$ (for example) via

$$e^{-(\delta-\tau)A} r \approx \beta V_m e^{-(\delta-\tau)H_m} e_1,$$

where V_m and H_m correspond to the Krylov subspace $K_m(A, r)$. In the more general case where r varies in the interval $[t, t+\delta]$, we can use a projection formula of the form

$$e^{-(\delta-\tau)A} r(t+\tau) \approx V_m e^{-(\delta-\tau)H_m} V_m^{\mathrm{T}} r(t+\tau). \tag{4.4}$$

The combination of the quadrature formula (4.3) and formula (4.4) has been tested and was found to be remarkably accurate. These tests were conducted in the context of a slightly modified approach, discussed next.

4.2. The second approach. In the above approach we need to compute two Krylov subspaces: one associated with the current iterate $w(t)$ and the other associated with $r(t)$. We show that we can reduce the computation to only one Krylov subspace. The resulting algorithm has different numerical properties from the one presented in §4.1.

The main idea is to use the identity

$$e^{-\delta A} = I - A\int_0^\delta e^{-sA} ds = I - A \int_0^\delta e^{-(\delta-\tau)A} d\tau,$$

which is obtained readily by integration. This is then substituted in the first term of the right-hand side of equation (4.2) to obtain

$$w(t+\delta) = w(t) + \int_0^\delta e^{-(\delta-\tau)A}[r(t+\tau) - Aw(t)]d\tau. \tag{4.5}$$

Note the important fact that the term $e^{-\delta A}w(t)$, which was in the previously used formula (4.2), has been removed at the slight expense of modifying the function $r(t+\tau)$ in the interval $\tau \in (0, \delta)$. The modification consists of subtracting a vector that is *constant* in the interval of integration. In terms of computations, this modification requires one matrix-by-vector multiplication, certainly an inexpensive overhead compared with that of applying the propagation operator to a vector.

With this transformation at hand, we can now proceed in the same manner as for the first approach, combining Krylov projection and numerical quadrature. More precisely, we evaluate the term $v = r(t+\tau) - Aw(t)$ at some point τ in the interval, for example in the middle, i.e., for $\tau = \delta/2$. Then a Krylov subspace is computed with the initial vector $v_1 = v/\|v\|_2$ and a formula similar to (4.4) based on this Krylov subspace is used to approximate the terms inside the integrand of (4.5). The integration is then carried out in a similar way with the appropriate quadrature formulas. The numerical experiments in §6 will illustrate an implementation based on this strategy.

Our experiment in §6.4 shows an example of a rapidly varying forcing term $r(t)$, where the method can perform adequately with only one additional exponential evaluation (at the midpoint). We note, however, that for some highly oscillatory forcing terms, a (preferably adaptive) scheme involving additional exponential evaluations or a reduction of the timestep δ may be needed.

There is one fundamental difference between the scheme (4.2) used in the first approach and the scheme (4.5) of the second approach: the function w now figures in the integrand. This may mean completely different numerical properties and, as is shown in §5, the loss of the unconditional stability.

5. Stability. In this section we investigate the linear stability of the Krylov timestepping methods when used for the solution of the semidiscrete system (1.2). As noted in the introduction, this discussion will not take into account the interaction between space and time discretizations.

We first consider the stability properties of the approximate evolution operator $V_m e^{-H_m} V_m^T$ implicitly involved in (2.3). If A is positive real,[1] i.e., if its symmetric part $S = \frac{1}{2}(A + A^{\mathrm{T}})$ is positive definite, then so is the matrix H_m [35]. Moreover, the eigenvalues of A and H_m have positive real parts and the smallest eigenvalue $\lambda_{\min}(S)$ of the symmetric part of A is a lower bound for the eigenvalues of $S_m = \frac{1}{2}(H_m + H_m^{\mathrm{T}})$, because $S_m = V_m^{\mathrm{T}} S V_m$. In terms of logarithmic norms, $\mu(-H_m) \leq \mu(-A) \leq 0$; see also Lemma A.3 in Appendix A. Since V_m has orthonormal columns, the approximate evolution operator satisfies

$$\|V_m e^{-H_m \delta} V_m^T\|_2 \leq \|e^{-H_m \delta}\|_2.$$

As a result, we can state that in the case where $\exp(-H_m\delta)$ is evaluated exactly, then

$$\begin{aligned} \|V_m e^{-H_m \delta} V_m^T\|_2 &\leq \|e^{-H_m \delta}\|_2 \\ &\leq e^{\mu(-H_m\delta)} \leq e^{\mu(-A\delta)} \\ &\leq 1. \end{aligned}$$

If, on the other hand, $\exp(-H_m\delta)$ is not computed exactly, then stability will depend upon the method of evaluation used. In particular, let $\exp(-H_m\delta)$ be evaluated using a diagonal (ν, ν) Padé approximation R_ν. Diagonal Padé approximations are

[1] A matrix B is positive real if $x^{\mathrm{T}} B x > 0$ for any real vector $x \neq 0$ [53].

A-acceptable, that is, $|R_\nu(z)| < 1$ for all z in the positive half-plane [42]. From above, $\Re x^{\rm H} H_m x \geq 0$ for any x and $\mu(-H_m) \leq 0$. We then obtain

$$\|V_m R_\nu(H_m\delta)V_m^T\|_2 \leq \|R_\nu(H_m\delta)\|_2 \leq 1$$

from a result of von Neumann, which states that, when the field of values of a matrix B is contained in $\mathcal{H}$ (the nonnegative half of the complex plane) and if a rational function f maps $\mathcal{H}$ into the unit disk, then $\|f(B)\|_2 \leq 1$; see also [47] and [15, Thm. 4]. A similar conclusion holds for the subdiagonal $(\nu-1, \nu)$ approximation. When a diagonal Chebyshev approximation is used this no longer holds, as these approximations may amplify small eigenvalues of H_m near zero. We can state only that for symmetric positive definite matrices and large enough values of ν, and $\delta\lambda_{\min}(H_m)$ bounded away from zero, $\|V_m R_\nu(H_m\delta)V_m^T\|_2 \leq 1$.

For the remainder of this section we assume that the exponential terms $\exp(-H_m\delta)$ are computed exactly.

5.1. Stability behavior of the first approach.

Consider a general solution scheme for (4.1) of the form

$$w_{N+1} = e^{-A\delta} w_N + s_N, \tag{5.1}$$

where s_N is some approximation to the integral (4.3). The above scheme is a one-step technique where δ is the timestep. If we assume that the exponential term is evaluated exactly, then the above methods are referred to as the nonlinear multistep methods of Lee [26]. It was remarked in [27] that these methods are stable. More generally, let us assume that the error incurred in the evaluation of the term $e^{-A\delta} w_N$ in (5.1) is $e_{1,N}$, while the error in the evaluation of the integral term s_N is $e_{2,N}$. Then the recurrence (5.1) is replaced by

$$w_{N+1} = e^{-A\delta} w_N + e_{1,N} + s_N^* + e_{2,N}, \tag{5.2}$$

in which s_N^* is the exact integral (4.3). In this situation, the total error at each step is of the form

$$e_{N+1} \equiv w(t_{N+1}) - w_{N+1} = e^{-A\delta} e_N + e_{1,N} + e_{2,N}. \tag{5.3}$$

This shows that if A has no eigenvalues in the negative half-plane of the complex domain, then the above procedure is stable. Note that this is independent of the procedure used to compute the approximation to the matrix exponential by vector product. If one uses the Krylov approximation to the exponential, then we essentially have an explicit procedure that is stable. Although this may seem like a contradiction, note that we have made very special assumptions. The important point is that we are essentially considering accurate one-step methods. In the extreme case where r is constant, the solution can be evaluated *in just one step* at any point in time provided the exponential is accurately approximated.

5.2. Stability behavior of the second approach.

As mentioned earlier, the alternative approach described in §4.2 is attractive from the point of view of efficiency but may have poor numerical properties. We outline in this section a stability analysis of this class of methods in an effort to determine how to select the quadrature formulas that are most likely to lead to robust procedures.

We consider the simple timestepping scheme derived by applying a quadrature formula to (4.5)

$$w_{N+1} = w_N + \delta \sum_{i=1}^{k} \mu_i e^{-(\delta-\tau_i)A}[r(t+\tau_i) - Aw_N], \tag{5.4}$$

where $\tau_i, i = 1, \cdots, k$ are the quadrature nodes in the interval $[0, \delta]$ and μ_i are their corresponding weights. Once more, we assume that the exponential terms in (5.4) are exactly calculated. The above equation can be recast in the form

$$w_{N+1} = \left(I - \delta \sum_{i=1}^{k} \mu_i e^{-(\delta-\tau_i)A} A \right) w_N + g_N,$$

in which g_N is a term that does not contain the variable w_N. We next assume that the matrix is normal. Then the stability of the above recurrence is easily studied by replacing the matrix A by a generic eigenvalue λ. This leads to the scalar recurrence

$$w_{N+1} = \left(1 - \delta \sum_{i=1}^{k} \mu_i e^{-(\delta-\tau_i)\lambda} \lambda \right) w_N + g_N.$$

We need to determine under which conditions the modulus of the evolution operator

$$a(\lambda) = 1 - \delta \sum_{i=1}^{k} \mu_i e^{-(\delta-\tau_i)\lambda} \lambda$$

does not exceed one.

Before proceeding with the more complicated general analysis, we first consider in detail two basic quadrature formulas: the trapezoidal rule and the midpoint rule. For the trapezoidal rule we have

$$a(\lambda) = 1 - \frac{\delta}{2}[\lambda e^{-\lambda\delta} + \lambda].$$

We restrict ourselves to the case where λ is real and positive. We need to have

$$-1 \le a(\lambda) = 1 - \frac{\delta\lambda}{2}[e^{-\lambda\delta} + 1] \le 1$$

or, since the second inequality is trivially satisfied,

$$\delta\lambda[e^{-\lambda\delta} + 1] \le 4. \tag{5.5}$$

Since $e^{-\lambda\delta} \le 1$, a sufficient condition for the above inequality to hold is that $\delta\lambda \le 2$, which is just as restrictive as an ordinary explicit method. Note that a *necessary condition* for (5.5) to be true is that $\delta\lambda \le 4$.

For the midpoint rule we have

$$a(\lambda) = 1 - \delta\lambda e^{-\lambda\delta/2}.$$

Considering again real and positive λ, we seek conditions under which we have

$$-1 \le a(\lambda) = 1 - \delta\lambda e^{-\lambda\delta/2} \le 1. \tag{5.6}$$

The second inequality is always satisfied and from the first we get the condition

$$\lambda\delta e^{-\lambda\delta/2} \le 2.$$

As is easily seen through differentiation, the maximum with respect to $\lambda\delta$ of the left-hand side is reached for $\lambda\delta = 2$, and its value is $2e^{-1}$, which is less than 2. Therefore, inequality (5.6) is *unconditionally satisfied.* This fundamental difference between the trapezoidal rule and the midpoint rule underscores the change of behavior in the second approach depending on the quadrature rule used. We will extend this analysis shortly.

The above development for the midpoint rule was restricted to λ being on the positive real line. Let us consider this case in more detail for λ complex. Setting $u = \lambda\delta = \alpha - i\beta$, we have

$$\begin{aligned} a(\lambda) &= 1 - ue^{-u/2} \\ &= 1 - (\alpha - i\beta)e^{-(\alpha - i\beta)/2} \\ &= 1 - \alpha e^{-\alpha/2}(\cos(\beta/2) + i\sin(\beta/2)) + i\beta e^{-\alpha/2}(\cos(\beta/2) + i\sin(\beta/2)) \\ &= 1 - e^{-\alpha/2}\left((\alpha c + \beta s) + i(\alpha s - \beta c)\right), \end{aligned}$$

where we have set $c = \cos(\beta/2)$ and $s = \sin(\beta/2)$. The modulus of $a(\lambda)$ is easily found to satisfy

$$\begin{aligned} |a(\lambda)|^2 &= 1 + e^{-\alpha}(\alpha^2 + \beta^2) - 2e^{-\alpha/2}(\alpha c + \beta s) \\ &= 1 + e^{-\alpha}\left(|u|^2 - 2e^{\alpha/2}(\alpha c + \beta s)\right). \end{aligned}$$

This leads to the region of stability, symmetric about the positive real axis, defined by

$$(\alpha^2 + \beta^2) \le 2e^{\alpha/2}\left(\alpha\cos(\beta/2) + \beta\sin(\beta/2)\right). \tag{5.7}$$

The shaded regions of Fig. 5.1 show the part of the complex domain $[0, 50] \times [-25, 25]$ that corresponds to values of u satisfying (5.7).

If we concentrate on the shaded region enveloping the positive real axis, we note that for large α, the limits of the curve bounding that section of the stability region are $\beta = \pm\pi$. This is because for $\pi < |\beta| \le 2\pi$, the coefficient $\cos(\beta/2)$ becomes negative, making (5.7) impossible to satisfy *for large* α (β being fixed). On the other hand, for fixed β such that $|\beta| \le \pi$, we have $\cos(\beta/2) \ge 0$ and there is always an α large enough to satisfy (5.7). In addition, all the lines $\beta = \pm 4k\pi, k = 0, 1, 2, \cdots$ belong partly to the region: specifically all those points in these lines with α larger than the (only) positive root of $x(2e^{x/2} - x) = \beta^2$ are acceptable points. As shown in Fig. 5.1, around each of these lines there is a whole subregion of stability, whereas in between, there are regions around the lines $\beta = \pm 2(2k+1)\pi$ that are unstable.

We return now to studying the general scheme and extending the above analysis to the general case. Again we restrict ourselves to the case where λ is real positive. This condition will be replaced by a weaker condition later. However, we do not necessarily assume that the weights are positive. Then we examine the conditions under which

$$-1 \le a(\lambda) = 1 - \delta\sum_{i=1}^{k}\mu_i e^{-(\delta - \tau_i)\lambda}\lambda \le 1$$

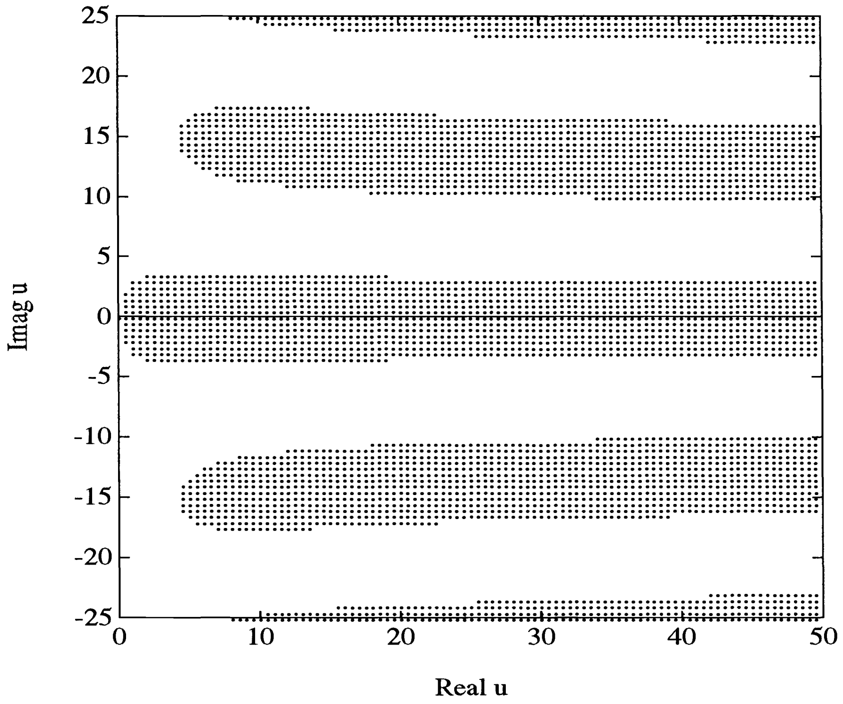


FIG. 5.1. *Stability region for the midpoint rule* (*cf.* (5.7)).

or

$$0 \le \delta\lambda e^{-\delta\lambda}\sum_{i=1}^{k}\mu_i e^{\tau_i\lambda} \le 2. \tag{5.8}$$

Consider now any quadrature rule that satisfies the following two conditions:

1. *Positivity condition.*

$$\sum_{i=1}^{k}\mu_i e^{\lambda\tau_i} \ge 0 \quad \text{for} \quad \lambda \ge 0. \tag{5.9}$$

2. *Undervaluation condition.*

$$E_k = \int_0^\delta e^{\lambda s}ds - \delta\sum_{i=1}^{k}\mu_i e^{\tau_i\lambda} \ge 0 \quad \text{for} \quad \lambda \ge 0. \tag{5.10}$$

The purpose of the positivity condition is to restrict the quadrature rule so that it yields nonnegative approximations to the integral of the function $e^{\lambda s}$ on any positive interval. It is verified whenever the quadrature weights are nonnegative. In particular, we have the following lemma.

LEMMA 5.1. *The positivity condition is satisfied for any Gaussian quadrature formula.*

Proof. The fact that the weights are positive for Gaussian quadrature is well known; see, e.g., [17, p. 328]. □

For the undervaluation condition, we can prove the following lemma.

LEMMA 5.2. (1) *Any composite or simple open Newton–Cotes formula satisfies the undervaluation condition* (5.10).

(2) *Any k-point Gaussian quadrature rule satisfies the undervaluation condition* (5.10).

Proof. This is a consequence of the well-known error formulas for open Newton–Cotes rules [17, pp. 313–314], and for Gaussian quadrature rules [17, p. 330], and the fact that all the derivatives of the function $e^{\lambda s}$ are positive in the interval $[0, \delta]$. □

Going back to condition (5.8), we first observe under the positivity condition (5.9) that the left-hand inequality is trivially satisfied. Moreover, under the undervaluation condition, we have that

$$\delta \sum_{i=1}^{k} \mu_i e^{\tau_i \lambda} \le \int_0^\delta e^{\lambda s} ds = \frac{e^{\lambda\delta} - 1}{\lambda},$$

and as a result,

$$0 \le \delta\lambda e^{-\delta\lambda} \sum_{i=1}^{k} \mu_i e^{\tau_i \lambda} \le \lambda e^{-\delta\lambda} \left(\frac{e^{\lambda\delta} - 1}{\lambda} \right) = 1 - e^{-\lambda\delta} \le 2. \tag{5.11}$$

We have therefore proved the following result.

THEOREM 5.3. *Consider the timestepping procedure* (5.4) *based on the second approach* (§4.2) *using a quadrature formula satisfying the positivity condition* (5.9) *and the undervaluation condition* (5.10). *Then the region of stability of this scheme contains the positive real line.*

Note that schemes with such properties are said to be A_0-stable in the literature [42]. In many of our numerical experiments, we have observed this difference in stability behavior between schemes that satisfy the conditions of the theorem and those that do not. In many instances the composite closed-type Newton–Cotes formulas tended to diverge for a small number of subintervals. On the other hand, we never noticed any stability difficulties with the open Newton–Cotes formulas or with the Gauss–Chebyshev quadrature.

As a general recommendation, it is advisable to use open Newton–Cotes formulas instead of closed formulas. Although these formulas satisfy the undervaluation condition according to the previous lemma, we do not know whether they satisfy the positivity condition (5.9). We know, however, that some of the low-order, closed Newton–Cotes rules (three-point, four-point, and six-point) do satisfy this condition since their weights are positive.

Gaussian rules are extremely attractive not only because of their stability properties but because of their potential to drastically reduce the number of function evaluations needed to produce a certain level of accuracy. There is still much work to be done to determine which of the quadrature formulas will yield the best results.

As is suggested by the analysis of the midpoint rule for complex λ, we expect the full analysis of the stability of the second approach to be very complicated for such cases.

6. Numerical experiments.

6.1. A symmetric model problem.

Our first test problem is issued from the semidiscretization of the heat equation

$$\begin{aligned}\frac{\partial u(x,y,z,t)}{\partial t} &= \Delta u(x,y,z,t)\ x,y,z \in (0,1),\\ u(x,y,z,t) &= 0 \quad (x,y,z) \quad \text{on the boundary,}\end{aligned}$$

where Δ stands for the three-dimensional Laplacian operator using 17 grid points in each direction, yielding a matrix of size $n = 15^3 = 3375$. The initial conditions are chosen after space discretization in such a way that the solution is known for all t. More precisely,

$$u(0, x_i, y_j, z_k) = \sum_{i',j',k'=1}^{n'} \frac{1}{i'+j'+k'} \sin\frac{ii'\pi}{n'+1} \sin\frac{jj'\pi}{n'+1} \sin\frac{kk'\pi}{n'+1},$$

where $n' = 15$. The above expression is simply an explicit linear combination of the eigenvectors of the discretized operator. In order to separate the influence of spatial discretization errors and emphasize the time evolution approximation, for the experiments in this section we consider the solution of the semidiscrete problem $u_t = -Au$ to be the exact solution. The exact solution is readily computed from the explicit knowledge of eigenvalues and eigenvectors of A.

The purpose of the first test is to illustrate one of the main motivations for this paper, namely, the effectiveness of using large-dimensional Krylov subspaces whenever possible. As shown in [10] and [13], similar conclusions also hold for methods based on rational approximations to the exponential.

Assume that we want to integrate the above equation between $t = 0$ and $t = 0.1$, and achieve an error-norm at $t = 0.1$, which is less than $\epsilon = 10^{-10}$. Here by error-norm we mean the two-norm of the absolute error.

We can vary both the degree m and the timestep δ. Normally we would prefer to first choose a degree m and then try to determine the maximum δ allowed to achieve the desirable error level. However, for convenience, we proceed in the opposite way: we first select a stepsize δ and then determine the minimum m that is needed to achieve the desirable error level. This experiment was performed on a Cray Y-MP. What is shown in Table 6.1 are the various timesteps chosen (column 1) and the minimum values of m (column 2) needed to achieve an error norm less than $\epsilon = 10^{-10}$ at $t = 0.1$. We show in the third column the total number of matrix-by-vector multiplications required to complete the integration. The times required to complete the integration on a Cray Y-MP are shown in column 4. We also timed the evaluations of $e^{-\delta H_m} e_1$ separately and found these times to be negligible with respect to the rest of the computation. The last column of the table shows the type of rational approximation used when evaluating $e^{-\delta H_m} e_1$, with $\mathrm{C}(\nu,\nu)$ representing the diagonal (ν,ν) Chebyshev approximation and $\mathrm{P}(\nu,\nu)$ representing the diagonal (ν,ν) Padé approximation.

Another point is that the matrix is symmetric, so we have used a Lanczos algorithm to generate the $v_i^{\prime}s$ instead of the full Arnoldi algorithm. No reorthogonalization of any sort was performed. The matrix consists of 7 diagonals, so the matrix-by-vector products are performed by diagonals resulting in a very effective use of the vector capabilities of the Cray architecture. Based on the time for the last entry of the table,

TABLE 6.1
Performance of the polynomial scheme with varying accuracy on the Cray Y-MP.

δ	m	M-vec's	Time (sec)	$\|Error\|_2$	Method
0.5000E−04	6	12006	0.8173E+01	0.1957E−11	P(2,2)
0.1000E−03	7	7007	0.4793E+01	0.3308E−10	P(2,2)
0.5000E−03	10	2010	0.1342E+01	0.1800E−10	P(4,4)
0.1000E−02	12	1200	0.7983E+00	0.2260E−10	P(4,4)
0.5000E−02	20	400	0.2672E+00	0.5271E−10	P(8,8)
0.1000E−01	26	260	0.1740E+00	0.7247E−10	P(8,8)
0.2000E−01	34	170	0.1080E+00	0.3236E−10	C(14,14)
0.3000E−01	39	156	0.9876E−01	0.6362E−10	C(14,14)
0.4000E−01	44	132	0.8030E−01	0.4122E−10	C(14,14)
0.5000E−01	49	98	0.5932E−01	0.5791E−10	C(14,14)
0.1000E+00	71	71	0.4186E−01	0.9993E−10	C(14,14)

we have estimated that the average Mflops rate reached, excluding the calculation of $e^{-\delta H_m}e_1$, was around 220. This is achieved with little code optimization.

Observe that the total number of matrix-by-vector products decreases rapidly as m increases. The ratio between the lowest degree, $m = 6$, and the highest degree, $m = 71$, is 169. The corresponding ratio between the two times is roughly 200. The case $m = 71$ can achieve the desired accuracy in just one step, that is, with $\delta = 0.1$. On the other hand, for $m = 6$, a timestep of $\delta = 5 \times 10^{-5}$ must be taken, resulting in a total of 2000 steps. We should point out that we are restricting ourselves to a constant timestep, but more efficient variable timestepping procedures are likely to reduce the total number of steps needed. These observations are qualitatively consistent wth Theorems 2.1 and 2.3. In effect, increasing the dimension of the Krylov subspace will increase the accuracy in such a way that a much larger ρ (i.e., a larger δ) can quickly be afforded.

6.2. A nonsymmetric problem with time-varying forcing term. In this section we consider the more difficult problem

$$\frac{\partial u(x,y,z,t)}{\partial t} = \Delta u(x,y,z,t) + \gamma\frac{\partial u(x,y,z,t)}{\partial x} + r(x,y,z,t), \tag{6.1}$$

defined as before on the unit cube, with homogeneous boundary conditions and the following initial conditions:

$$u(x,y,z,0) = x(x-1)y(y-1)z(z-1).$$

The function r is defined in such a way that the exact solution of the above PDE is given by

$$u(x,y,z,t) = \frac{x(x-1)y(y-1)z(z-1)}{1+t}. \tag{6.2}$$

As in the previous example, we took the same number of grid points in each direction, i.e., $n_x = n_y = n_z = 17$, yielding again a matrix of dimension $n = 15^3 = 3375$. This experiment was conducted on a Cray-2. Table 6.2 is the analogue of Table 6.1, except that we only report some representative runs with various values of m and δ. The parameter γ is set equal to 10.0. The integration is carried out from $t = 0.0$ to $t = 1.0$. We note that for the remaining experiments, unless otherwise mentioned, the rational approximation used for $e^{-\delta H_m}$ is the Chebyshev C(10,10).

TABLE 6.2
Performance of the polynomial scheme with varying accuracy on the Cray-2.

δ	m	*npts*	Mvec's	Time (sec)	$\|Error\|_2$
0.2000E+00	40	60	205	0.2402E+01	0.6151E−05
0.1000E+00	40	40	410	0.3690E+01	0.7483E−06
0.1000E+00	40	30	410	0.3114E+01	0.3011E−05
0.1000E+00	35	40	360	0.3177E+01	0.7483E−06
0.1000E+00	30	40	310	0.2617E+01	0.7484E−06
0.1000E+00	25	40	260	0.2110E+01	0.8743E−06
0.1000E+00	25	30	260	0.1721E+01	0.1054E−04
0.5000E−01	25	30	520	0.3413E+01	0.7503E−07
0.5000E−01	25	20	520	0.2726E+01	0.3961E−06
0.5000E−01	20	20	420	0.2124E+01	0.6163E−05
0.5000E−01	15	20	320	0.1550E+01	0.5463E−04
0.2500E−01	20	10	840	0.2992E+01	0.9015E−06
0.2500E−01	15	20	640	0.3086E+01	0.5327E−05
0.2500E−01	15	10	640	0.2109E+01	0.9887E−05
0.1000E−01	10	10	1100	0.3561E+01	0.1743E−05
0.1000E−01	7	10	800	0.2693E+01	0.9483E−05

An important characteristic of this example is that the exact solution of the discrete problem is also an exact solution of the continuous problem. This is because the exact solution is a polynomial of degree not exceeding 2 in each of the space variables. For these functions the centered difference approximation to the partial derivatives in the operator (6.1) entails no errors. This fact has also been verified experimentally. Therefore, all the errors in the computed solution are due to the time integration process and this allows a fair comparison of the various techniques from the standpoint of accuracy achieved in the time integration.

The second approach was used, in which the integrals were calculated with 11-point composite (closed) Newton–Cotes formulas. In most cases we had to take more than 11 points, in which case we simply used a composite rule with a total number of points equal to $1 + k \times 10$. The third column reports the total number of subintervals *npts* used to advance by one timestep of δ. Thus *npts* is a multiple of 10. The time shown in the fifth column is the time in seconds needed to advance the solution from $t = 0.0$ to $t = 1.0$, on a Cray-2. The sixth column shows the two-norm norm of the error with respect to the exact solution of the *continuous* system, that is, with respect to (6.2).

We observe that for larger timesteps a larger number of quadrature points must be used to keep a good level of accuracy. We show the results associated with the smallest number of points for which there are no significant qualitative improvements in the error when we increase *npts*, while keeping m and δ constant. Our tests indicate that the higher the order of the quadrature used, the better. This means that large gains in speed are still likely if we use more optimal Gaussian quadrature formulas. A noticeable difference from the previous simple example is that the increase in δ and the corresponding increase in m in achieving certain error tolerances cause a less significant reduction in the total number of matrix-by-vector multiplications and the total time required for the computation.

6.3. A comparison with other methods. Although an exhaustive comparison with other schemes is beyond the scope of this paper, we would like to give an idea of how the efficiency of the Krylov subspace propagation compares with some immediate contenders. The first of these contenders is simply the forward Euler scheme. This is an explicit scheme and, for not-too-small space mesh sizes, should not be excluded, given that the corresponding process is highly vectorizable. However, it may be far more challenging to use an implicit scheme such as the Crank–Nicolson method

$$\left(I+\frac{\delta}{2}A\right)w_{N+1}=\left(I-\frac{\delta}{2}A\right)w_N+\delta r(t_N+\delta/2), \tag{6.3}$$

combined with an iterative method, for example, the conjugate gradient method, for solving the linear systems. The main attraction here is that we can solve the linear systems inaccurately, making the solution process very inexpensive. From this viewpoint, this "inexact Crank–Nicolson" method shares many of the benefits of the Krylov method, as was already mentioned in the introduction. Finally, a well-known stiff ODE package such as LSODE [16] is also considered.

For this comparison we took the same problem as before, but we needed to take $\gamma = 0.0$ in order to make the matrix A symmetric. This was necessary in order to be able to utilize the usual conjugate gradient algorithm for the linear systems in the Crank–Nicolson scheme. The r function is defined as before, and the number of grid points in each direction is again $n_x = n_y = n_z = 17$, yielding $n = 15^3 = 3375$.

We point out that for the Crank–Nicolson method, we do not use preconditioning, and this is by no means a drawback. Because of timestepping, the matrix is usually very well conditioned and, as a result, the algorithm converges in a rather small number of steps. Moreover, because there is no need to solve the systems with high accuracy, the overhead in setting up the preconditioner would be difficult to amortize. Finally, the good preconditioners, such as the incomplete factorizations, do not generally yield a high performance on vector machines. In our tests, the CG algorithm is stopped as soon as the residual norm is reduced by a factor that does not exceed a tolerance ϵ. We always take the tolerance ϵ that yields the smallest (or close to the smallest) time for the Crank–Nicolson scheme to complete. In addition, note that the timesteps in the Crank–Nicolson scheme are chosen at the outset and, as with the Krylov method, they are not adaptively controlled. We show the performance for two timesteps only, namely, those that deliver close to the desired accuracy at the final point. In this test we used the Chebyshev rational approximation of order (6,6) throughout for the computation of $e^{-\delta H_m}e_1$. A final detail is that both the Crank–Nicolson method, which is able to use the usual CG method, and the Krylov method, in which the Lanczos version of the Arnoldi algorithm is used, take advantage of symmetry.

For LSODE we used the method flag MF = 24, which means that a stiff method is used, and the Jacobian is user-supplied in banded format. The Cray-optimized LINPACK banded solver is called to solve the linear systems. For LSODE we used a relative tolerance of `rtol` $= 10^{-7}$ and an absolute tolerance of `atol` $= 10^{-7}$ at each point is used. These have been chosen so that the level of error produced by LSODE at $t = 1$ is comparable to that of the other methods.

Table 6.3 shows the results. This comparison reveals that the Krylov scheme is superior when one considers the number of matrix-by-vector products to be the primary criterion. There are situations in which these may dominate the cost, in which case the execution time could be proportional to the number of matrix–vector products. When execution time is the primary criterion for comparison, then the Krylov scheme is still faster than Crank–Nicolson but not by as large a margin. The forward Euler scheme was unstable for the timesteps $\delta = 0.001$ and $\delta = 0.00075$. We also performed a set of tests with a larger version of this problem corresponding

TABLE 6.3
Performance comparison of a few methods on problem of §6.3.

Method used	Method parameters	Matrix–vec. products	Total Cray-2 time (sec.)	Final error
Krylov $\delta = 0.2$	$m = 30, npts = 40$	155	0.9374E+00	0.6670E−05
	$m = 40, npts = 40$	205	0.1225E+01	0.6652E−05
	$m = 35, npts = 40$	180	0.1038E+01	0.6672E−05
	$m = 30, npts = 40$	155	0.9355E+00	0.6670E−05
	$m = 20, npts = 40$	105	0.6615E+00	0.7103E−05
	$m = 20, npts = 30$	105	0.5229E+00	0.1764E−04
Krylov $\delta = 0.15$	$m = 25, npts = 30$	182	0.9530E+00	0.9367E−06
	$m = 20, npts = 30$	147	0.7828E+00	0.7185E−05
	$m = 15, npts = 30$	112	0.6151E+00	0.4244E−04
Krylov $\delta = 0.1$	$m = 20, npts = 30$	210	0.1044E+01	0.7956E−06
	$m = 15, npts = 30$	160	0.9086E+00	0.8574E−05
Crank–Nicolson	$\delta = .01, \epsilon = .001$	1053	0.1192E+01	0.1267E−05
	$\delta = .005, \epsilon = .001$	1578	0.1767E+01	0.3329E−06
F-Euler	$\delta = .0005$	2000	0.2779E+01	0.8678E−06
LSODE	MF = 24	400	0.1435E+02	0.2828E−05

to the grid sizes $n_x = n_y = n_z = 22$, leading to a problem of size $n = 8000$. The conclusion is essentially the same in that Crank–Nicolson and the Krylov method are comparable, but the time for the explicit Euler scheme becomes much higher. We should add that we have regarded the problem purely from the perspective of systems of ODEs, although we are aware that in practice a balanced accuracy between space and discretization is generally sought. However, this would lead to comparisons that are too complex.

6.4. A case with highly oscillating forcing term. We consider here an example of the same form as in §6.3; that is, the general equation is of the form (6.1), and the initial and boundary conditions are identical. However, we now consider a forcing term for which the exact solution is given by

$$u(x, y, z, t) = x(x-1)y(y-1)z(z-1)\cos(\alpha\pi t). \tag{6.4}$$

If the coefficient α is chosen to be large, then the problem can be difficult to solve. We take here $\gamma = 0.0$ and $\alpha = 20$. The discretization mesh and interval of time integration are the same as in the previous example. Note that as in the preceding example, the errors in the computed solutions are also entirely related to the time integration.

We compared the same four methods as those of the previous section, the forward Euler scheme, the Crank–Nicolson/CG scheme, LSODE, and the Krylov method using the second approach. In this example LSODE failed to complete in a reasonable amount of time.

One difference with the previous tests is that here we varied the quadrature formulas used. Thus $npts = 4 \times 8$ indicates that we used a composite rule in which the interval of integration is first divided by 4 and then a nine-point formula is used on each subinterval. Apart from this, all of the details concerning implementation are identical to those of §6.3, except that this time we used the Chebyshev rational approximation of order (8,8) instead of (6,6) to compute the vectors $e^{-\delta H_m} e_1$.

The results in Table 6.4 indicate that for this harder problem, the Krylov scheme performs far better than its competitors. The Crank–Nicolson scheme now requires smaller timesteps to achieve acceptable accuracies. The forward Euler scheme would

TABLE 6.4
Performance comparison of a few methods for problem of §6.4.

Method used	Method parameters	Matrix–vec. products	Total Cray-2 time (sec.)	Final error
	error norm			
Krylov $\delta = 0.2$	$m = 40, npts = 3 \times 10$	205	0.1202E+01	0.8051E−04
	$m = 30, npts = 2 \times 10$	155	0.6902E+00	0.2262E−03
	$m = 30, npts = 8 \times 5$	155	0.1196E+01	0.2862E−04
	$m = 25, npts = 20 \times 2$	130	0.1096E+01	0.3188E−04
	$m = 25, npts = 8 \times 5$	130	0.1063E+01	0.2320E−04
Krylov $\delta = 0.1$	$m = 20, npts = 2 \times 10$	210	0.1083E+01	0.7585E−05
	$m = 15, npts = 10 \times 2$	160	0.8995E+00	0.9713E−03
	$m = 15, npts = 2 \times 10$	160	0.8739E+00	0.6988E−04
	$m = 15, npts = 3 \times 8$	160	0.1043E+01	0.1592E−04
	$m = 15, npts = 4 \times 8$	160	0.1298E+01	0.1757E−05
	$m = 10, npts = 4 \times 8$	110	0.1066E+01	0.3504E−04
Crank–Nicolson	$\delta = .001, \epsilon = .001$	4322	0.5723E+01	0.8816E−04
	$\delta = .5\text{E}{-}03$, $\epsilon = .001$	8000	0.1058E+02	0.2203E−04
F-Euler	$\delta = .5\text{E}{-}03$	2000	0.4780E+01	0.2358E−02
	$\delta = .1\text{E}{-}03$	10000	0.2364E+02	0.4712E−03
	$\delta = .5\text{E}{-}05$	20000	0.4861E+02	0.2356E−03
LSODE	MF = 24	—	—	—

require a much smaller timestep that those of the other methods to achieve comparable performance.

7. Summary and conclusion. The goal of this paper was to show how to systematically develop explicit-type schemes or, to use our terminology, polynomial schemes for solving parabolic PDEs by the method of lines. We have proposed one such procedure that has the advantage of being very simple. The method proposed requires no information about the spectrum of the space discretization operator. We have recommended using high-dimension Krylov subspaces whenever possible. By using a Krylov subspace of high dimension to approximate the evolution operator, we are able to use larger timesteps. At each step there is an additional cost due to the increased dimension of the Krylov subspace, which translates into an increase in the number of matrix-by-vector multiplications. On the other hand, because of the larger timestep, the total number of steps required is reduced to such an extent that there is an appreciable net gain in performance. We have also proposed two approaches for handling nonconstant forcing terms, with the view of extending these methods for general ODEs and nonlinear PDEs. The stability analysis of these approaches shows that the first is unconditionally stable and the second is A_0-stable for a large class of integration schemes used. This has been widely confirmed by numerical experiments which indicate that the schemes proposed are competitive with standard methods such as Crank–Nicolson.

Improvements to the approach described in §4.2 are possible by developing quadrature formulas that are more elaborate and specialized than the simple Newton–Cotes formulas used in our numerical experiments. We believe that the method proposed here can be extended to the solution of general time-dependent nonlinear PDEs: the only subtlety is to isolate the action of the evolution operator, which is then well approximated by the schemes proposed here.

Appendix A. Proof of Theorem 2.1. The following lemma provides the basis for establishing error bounds for the error of the approximation (2.3).

LEMMA A.1. *Let A be any matrix, and p be any polynomial of degree smaller than m, approximating e^{-z} with the remainder $r_m(z) = e^{-z} - p(z)$. Then,*

$$\|e^{-A}v - \beta V_m e^{-H_m} e_1\|_2 \leq \beta(\|r_m(A)\|_2 + \|r_m(H_m)\|_2), \tag{A.1}$$

where $\beta = \|v\|_2$.

Proof. As a result of the relation $e^{-z} = p(z) + r_m(z)$, we have

$$e^{-A}v = \beta[p(A)v_1 + r_m(A)v_1]. \tag{A.2}$$

Using induction and the relation (2.2) we can show that $A^j v_1 = V_m H_m^j e_1$ for $j \leq m-1$, and as a consequence we have

$$p(A)v_1 = V_m p(H_m)e_1. \tag{A.3}$$

As a result of the definition of p and r_m, we write

$$p(H_m)e_1 = e^{-H_m}e_1 - r_m(H_m)e_1. \tag{A.4}$$

To complete the proof, we substitute (A.4) in (A.3) and the resulting equation in (A.2) to get

$$e^{-A}v = \beta V_m e^{-H_m} e_1 + \beta[r_m(A)v_1 - V_m r_m(H_m)e_1].$$

The result follows immediately. □

Thus the error can be estimated by bounding each of the two remainder terms. We now use the concept of the *logarithmic norm* of a matrix as defined in §2.2. We will specifically use the inequality $\|e^{Bt}\| \leq e^{\mu(B)t}$.

We next prove the following lemmas.

LEMMA A.2. *Let*

$$s_{m-1}(z) = \sum_{k=0}^{m-1} \frac{(-z)^k}{k!}$$

be the $(m-1)$th partial Taylor sum of e^{-z} and let $r_m(z)$ be the associated remainder $r_m(z) = e^{-z} - s_{m-1}(z)$. Define

$$\phi(\eta) \equiv \frac{1}{\eta^m}\left(e^{\eta} - \sum_{k=0}^{m-1} \frac{\eta^k}{k!}\right).$$

Then

$$\|r_m(A)\| \;\leq\; \|A^m\|\phi(\eta) \;\leq\; \|A^m\|\frac{\max(1, e^{\eta})}{m!}, \tag{A.5}$$

where $\eta \equiv \mu(-A)$.

Proof. The remainder after m terms of the Taylor series expansion in integral form applied to $\exp(-A)$ is given by

$$r_m(A) = \frac{(-A)^m}{(m-1)!} \int_0^1 e^{-A(1-\tau)}\tau^{m-1} d\tau \tag{A.6}$$

and therefore,

$$\|r_m(A)\| \le \frac{\|A^m\|}{(m-1)!}\int_0^1 \|e^{-A(1-\tau)}\|\tau^{m-1}d\tau.$$

Denoting $\eta = \mu(-A)$ for convenience, since $0 < \tau < 1$, we have from (2.4)

$$\|e^{-A(1-\tau)}\| \le e^{\mu(-A)(1-\tau)} \equiv e^{\eta(1-\tau)},$$

from which we get

$$\|r_m(A)\| \le \frac{\|A^m\|}{(m-1)!}\int_0^1 e^{\eta(1-\tau)}\tau^{m-1}d\tau. \tag{A.7}$$

The value of the integral in the above expression is determined by noting that the remainder of the $(m-1)$st Taylor expansion of e^η satisfies

$$e^\eta - \sum_{k=0}^{m-1}\frac{\eta^k}{k!} = \frac{\eta^m}{(m-1)!}\int_0^1 e^{\eta(1-\tau)}\tau^{m-1}d\tau,$$

which gives

$$\phi(\eta) = \frac{1}{(m-1)!}\int_0^1 e^{\eta(1-\tau)}\tau^{m-1}d\tau.$$

Incidentally, this expression shows that $\phi(\eta)$ is nonnegative. Substituting this in (A.7) proves the first inequality in (A.5).

To prove the second part of the inequality, we observe that

$$\phi(\eta) = \frac{1}{(m-1)!}\int_0^1 e^{\eta(1-\tau)}\tau^{m-1}d\tau \tag{A.8}$$

$$\le \frac{1}{(m-1)!}\int_0^1 \max(1, e^\eta)\tau^{m-1}d\tau = \frac{\max(1, e^\eta)}{m!}. \tag{A.9}$$ □

We remark that the upper bound for $\phi(\eta)$ used in the above lemma can be somewhat refined. More specifically, it can be shown that.

$$\phi(\eta) \le \begin{cases} \frac{1}{m!} & \text{if } \eta < 0, \\ \frac{e^\eta}{m!} & \text{if } 0 < \eta \le \sqrt[m-1]{(m-2)!m}, \\ \frac{e^\eta}{(m-1)\eta^{m-1}} & \text{if } \sqrt[m-1]{(m-2)!m} \le \eta. \end{cases}$$

Finally, we need the following lemma.

LEMMA A.3. *If A is any real matrix and H_m is the associated $m \times m$ upper Hessenberg matrix generated by m steps of the Arnoldi algorithm, then*

$$\mu(-H_m) \le \mu(-A).$$

Proof. By construction, V_m consists of m orthonormal vectors and H_m satisfies $H_m = V_m^{\mathrm{T}} A V_m$. Since the maximum eigenvalues of the symmetric parts of A and H_m can be characterized as the maximum values taken by their Rayleigh quotients, it easily follows that

$$\mu(-H_m) = \max_i \lambda_i\left(-\frac{V_m^{\mathrm{T}}AV_m + V_m^{\mathrm{T}}A^{\mathrm{T}}V_m}{2}\right) \le \max_i \lambda_i\left(-\frac{A + A^{\mathrm{T}}}{2}\right) = \mu(-A).$$ □

TABLE B.1
Coefficients of the partial fraction expansion for degrees 10 *and* 14.

Degree	Coef/root	Real part	Imaginary part
10	α_0	0.136112052334544905E−09	
	α_1	0.963676398167865499E+01	−0.421091944767815675E+02
	α_2	−0.142343302081794718E+02	0.176390663157379776E+02
	α_3	0.513116990967461106E+01	−0.243277141223876469E+01
	α_4	−0.545173960592769901E+00	0.284234540632477550E−01
	α_5	0.115698077160221179E−01	0.137170141788336280E−02
	λ_1	−0.402773246751880265E+01	0.119385606645509767E+01
	λ_2	−0.328375288323169911E+01	0.359438677235566217E+01
	λ_3	−0.171540601576881357E+01	0.603893492548519361E+01
	λ_4	0.894404701609481378E+00	0.858275689861307000E+01
	λ_5	0.516119127202031791E+01	0.113751562519165076E+02
14	α_0	0.183216998528140087E−11	
	α_1	0.557503973136501826E+02	−0.204295038779771857E+03
	α_2	−0.938666838877006739E+02	0.912874896775456363E+02
	α_3	0.469965415550370835E+02	−0.116167609985818103E+02
	α_4	−0.961424200626061065E+01	−0.264195613880262669E+01
	α_5	0.752722063978321642E+00	0.670367365566377770E+00
	α_6	−0.188781253158648576E−01	−0.343696176445802414E−01
	α_7	0.143086431411801849E−03	0.287221133228814096E−03
	λ_1	−0.562314417475317895E+01	0.119406921611247440E+01
	λ_2	−0.508934679728216110E+01	0.358882439228376881E+01
	λ_3	−0.399337136365302569E+01	0.600483209099604664E+01
	λ_4	−0.226978543095856366E+01	0.846173881758693369E+01
	λ_5	0.208756929753827868E+00	0.109912615662209418E+02
	λ_6	0.370327340957595652E+01	0.136563731924991884E+02
	λ_7	0.889777151877331107E+01	0.166309842834712071E+02

Proof of Theorem 2.1. First note that as in Lemma A.2 we can show that

$$\|r_m(H_m)\|_2 \leq \rho_m^m \phi(\mu(-H_m)),$$

where $\rho_m = \|H_m\|_2 = \|V_m^T A V_m\|_2$. The right-hand side of the above inequality is an increasing function of ρ_m and $\rho_m \leq \rho$. From Lemma A.3, $\mu(-H_m) \leq \mu(-A) = \eta$, and therefore,

$$\|r_m(H_m)\|_2 \leq \rho^m \phi(\eta). \tag{A.10}$$

Using Lemma A.1 the proof follows. □

Appendix B. Partial fraction coefficients. In Table B.1 we list some of the coefficients of the partial fraction expansion for the Chebyshev rational approximation to the exponential. These are the (ν, ν) approximations for $\nu = 10$ and $\nu = 14$. Note that because the roots go in complex conjugate pairs, we only need to show those with nonnegative imaginary parts. In fact there are exactly $\lceil \frac{\nu}{2} \rceil$ such roots for the (ν, ν) approximation. Moreover, in the case of a complex pair the corresponding coefficient α_i in the partial fraction expansion is doubled. The roots are also distinct and we can therefore write, for a real x,

$$e^{-x} \approx \alpha_0 + 2\Re \left[\sum_{i=1}^{\nu/2} \frac{\alpha_i}{x - \lambda_i} \right]. \tag{B.1}$$

Acknowledgments. We thank Professor Richard Varga for reading the paper and for providing many enlightening suggestions, Dr. Roland Freund for many helpful discussions, and Dr. Randall Bramley for his comments on an early version of the paper. We also thank the referees for their recommendations and for bringing to our attention reference [8].

REFERENCES

[1] P. N. Brown and A. C. Hindmarsh, *Matrix-free methods for stiff systems of* ODEs, SIAM J. Numer. Anal., 23 (1986), pp. 610–638.

[2] A. J. Carpenter, A. Ruttan, and R. S. Varga, *Extended numerical computations on the 1/9 conjecture in rational approximation theory*, in Rational Approximation and Interpolation, P. R. Graves-Morris, E. B. Saff, and R. S. Varga, eds., Lecture Notes in Mathematics 1105, Springer-Verlag, Berlin, 1984, pp. 383–411.

[3] J. C. Cavendish, W. E. Culham, and R. S. Varga, *A comparison of Crank–Nicolson and Chebyshev rational methods for numerically solving linear parabolic equations*, J. Comput. Phys., 10 (1972), pp. 354–368.

[4] A. R. Curtis, *Jacobian matrix properties and their impact on the choice of software for stiff* ODE *systems*, IMA J. Numer. Anal., 3 (1983), pp. 397–415.

[5] K. Dekker and J. G. Verwer, *Stability of Runge-Kutta Methods for Stiff Nonlinear Differential Equations*, North–Holland, Amsterdam, 1984.

[6] C. Desoer and H. Haneda, *The measure of a matrix as a tool to analyze computer algorithms for circuit analysis*, IEEE Trans. Circuit Theory, 19 (1972), pp. 480–486.

[7] B. L. Ehle, *A-stable methods and Padé approximations to the exponential*, SIAM. J. Numer. Anal., 4 (1973), pp. 671–680.

[8] S. W. Ellacott, *On the Faber transformation and efficient numerical rational approximation*, SIAM J. Numer. Anal., 20 (1983), pp. 989–1000.

[9] R. A. Friesner, L. S. Tuckerman, B. C. Dornblaser, and T. V. Russo, *A method for exponential propagation of large systems of stiff nonlinear differential equations*, J. Sci. Comput., 4 (1989), pp. 327–354.

[10] E. Gallopoulos and Y. Saad, *Efficient parallel solution of parabolic equations: Implicit methods on the Cedar multicluster*, in Proc. Fourth SIAM Conf. Parallel Processing for Scientific Computing, Chicago, Dec. 1989, J. Dongarra, P. Messina, D. C. Sorensen, and R. G. Voigt, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1990, pp. 251–256.

[11] ———, *Parallel block cyclic reduction algorithm for the fast solution of elliptic equations*, Parallel Comput., 10 (1989), pp. 143–160.

[12] ———, *Efficient solution of parabolic equations by polynomial approximation methods*, Tech. Report 969, Center for Supercomputing Research and Development, University of Illinois, Urbana, IL, February 1990.

[13] ———, *On the parallel solution of parabolic equations*, in Proc. 1989 ACM Internat. Conference on Supercomputing, Herakleion, Greece, June 1989, pp. 17–28; also in Tech. Report 854, Center for Supercomputing Research and Development, University of Illinois, Urbana, IL, June 1989.

[14] C. W. Gear and Y. Saad, *Iterative solution of linear equations in* ODE *codes*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 583–601.

[15] E. Hairer, G. Bader, and C. Lubich, *On the stability of semi-implicit methods for ordinary differential equations*, BIT, 22 (1982), pp. 211–232.

[16] A. C. Hindmarsh, ODEPACK, *A systematized collection of* ODE *solvers*, in Scientific Computing, R. S. Stepleman, ed., North–Holland, Amsterdam, 1983, pp. 55–64.

[17] E. Isaacson and H. B. Keller, *Analysis of Numerical Methods*, John Wiley, New York, 1966.

[18] A. Iserles, *Rational interpolation to* $\exp(-x)$ *with application to certain stiff systems*, SIAM J. Numer. Anal., 18 (1981), pp. 1–12.

[19] A. Iserles and S. P. Nørsett, *On the theory of parallel Runge–Kutta methods*, IMA J. Numer. Anal., 10 (1990), pp. 463–488.

[20] A. Iserles and M. J. D. Powell, *On the A-acceptability of rational approximations that interpolate the exponential function*, IMA J. Numer. Anal., 1 (1981), pp. 241–251.

[21] O. A. Karakashian and W. Rust, *On the parallel implementation of implicit Runge–Kutta methods*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 1085–1090.

[22] S. Keeling, *Galerkin Runge–Kutta discretizations for parabolic equations with time-dependent coefficients*, Math. Comp., 52 (1989), pp. 561–586.

[23] H. T. Kung, *New algorithms and lower bounds for the parallel evaluation of certain rational expressions and recurrences*, J. Assoc. Comput. Mach., 23 (1976), pp. 252–261.

[24] E. Landau, *Über einen Mellinshen Satz*, Arch. Math. Phys. Ser. 3, 24 (1915), pp. 97–107.

[25] J. D. Lawson and D. A. Swayne, *High-order near best uniform approximations to the solution of heat conduction problems*, in Proc. IFIPS Congress—80, New York, 1980, North–Holland, Amsterdam, 1980, pp. 741–746.

[26] D. Lee, *Nonlinear multistep methods for solving initial value problems in ordinary differential equations*, Ph.D. thesis, Polytechnic Institute of New York, Brooklyn, NY, 1974.

[27] D. Lee and J. S. Papadakis, *Numerical solutions of underwater acoustic wave propagation problems*, Tech. Report NUSC TR. 5929, Naval Underwater Systems Center, New London, CT, 1979.

[28] C. Moler and C. V. Loan, *Nineteen dubious ways to compute the exponential of a matrix*, SIAM Rev., 20 (1978), pp. 801–836.

[29] A. Nauts and R. E. Wyatt, *New approach to many-state quantum dynamics: The recursive-residue-generation method*, Phys. Rev. Lett., 51 (1983), pp. 2238–2241.

[30] ———, *Theory of laser-module interaction: The recursive-residue-generation method*, Phys. Rev., 30 (1984), pp. 872–883.

[31] S. P. Nørsett, *Restricted Padé approximations to the exponential function*, SIAM J. Numer. Anal., 15 (1978), pp. 1008–1029.

[32] B. Nour-Omid, *Applications of the Lanczos algorithm*, Comput. Phys. Comm., 53 (1989), pp. 153–168.

[33] P. Pandey, C. Kenney, and A. J. Laub, *A parallel algorithm for the matrix sign function*, Internat. J. High Speed Comput., 2 (1990), pp. 181–191.

[34] T. J. Park and J. C. Light, *Unitary quantum time evolution by iterative Lanczos reduction*, J. Chem. Phys., 85 (1986), pp. 5870–5876.

[35] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice–Hall, Englewood Cliffs, NJ, 1980.

[36] G. Pólya and G. Szegö, *Problems and Theorems in Analysis* I, Springer–Verlag, New York, 1972.

[37] G. Rodrigue and D. Wolitzer, *Preconditioned time-differencing for the parallel solution of the heat equation*, in Proc. Fourth SIAM Conf. Parallel Processing for Scientific Computing, Chicago, Dec. 1989, J. Dongarra, P. Messina, D. C. Sorensen, and R. G. Voigt, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1990, pp. 268–272.

[38] Y. Saad, *On the rates of convergence of the Lanczos and the block-Lanczos methods*, SIAM J. Numer. Anal., 17 (1980), pp. 687–706.

[39] ———, *Analysis of some Krylov subspace approximations to the matrix exponential operator*, SIAM J. Numer. Anal., 29 (1992), pp. 209–228.

[40] J. M. Sanz-Serna and J. G. Verwer, *Stability and convergence at the* PDE/*stiff* ODE *interface*, Appl. Numer. Math., 5 (1989), pp. 117–132.

[41] M. J. Schaefer, *A polynomial based iterative method for linear parabolic equations*, Tech. Report 661, Center for Supercomputing Research and Development, University of Illinois, Urbana, IL, May 1987.

[42] W. L. Seward, G. Fairweather, and R. L. Johnston, *A survey of high-order methods for the numerical integration of semidiscrete parabolic problems*, IMA J. Numer. Anal., 4 (1984), pp. 375–425.

[43] Q. Sheng, *Solving linear partial differential equations by exponential splitting*, IMA J. Numer. Anal., 9 (1989), pp. 199–212.

[44] R. A. Sweet, *A parallel and vector cyclic reduction algorithm*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 761–765.

[45] H. Tal-Ezer, *Spectral methods in time for parabolic problems*, SIAM J. Numer. Anal., 26 (1989), pp. 1–11.

[46] H. Tal-Ezer and R. Kosloff, *An accurate and efficient scheme for propagating the time dependent Schrödinger equation*, J. Chem. Phys., 81 (1984), pp. 3967–3971.

[47] J. V. Neumann, *Eine Spektraletheorie für allgemeine Operatoren eines unitären Raumes*, Math. Nachr., 4 (1950–51), pp. 258–281.

[48] P. J. van der Houwen and B. P. Sommeijer, *Parallel iteration of high-order Runge–Kutta methods with stepsize control*, J. Comput. Appl. Math., 29 (1990), pp. 111–127.

[49] ———, *Parallel* ODE *solvers*, in Proc. 1990 ACM Internat. Conf. Supercomputing, Amsterdam, June 1990, Association for Computing Machinery, New York, 1990, pp. 71–81.

[50] P. J. van der Houwen, B. P. Sommeijer, and F. W. Wubs, *Analysis of smoothing oper-*

ators in the solution of partial differential equations by explicit difference schemes, Appl. Numer. Math., 6 (1989/90), pp. 501–521.

[51] H. van der Vorst, *An iterative solution method for solving $f(A)x = b$ using Krylov subspace information obtained for the symmetric positive definite matrix A*, J. Comput. Appl. Math., 18 (1987), pp. 249–263.

[52] R. S. Varga, *On higher order stable implicit methods for solving parabolic partial differential equations*, J. Math. Phys., 40 (1961), pp. 220–231.

[53] E. L. Wachspress, *Iterative Solution of Elliptic Systems*, Prentice–Hall, Englewood Cliffs, NJ, 1966.

[54] G. Wanner, *Order stars and stability*, in The State of the Art in Numerical Analysis, A. Iserles and M. J. D. Powell, eds., Clarendon Press, Oxford, 1987, pp. 451–472.

[55] D. S. Watkins and R. W. Hansonsmith, *The numerical solution of separably stiff systems by precise partitioning*, ACM Trans. Math. Software, 9 (1983), pp. 293–301.

[56] D. M. Young, *The search for "high-order" parallelism for iterative sparse linear system solvers*, in Parallel Supercomputing: Methods, Algorithms and Applications, G. F. Carey, ed., John Wiley, Chichester, U.K., 1989, pp. 89–106.

[57] V. Zakian, *Properties of I_{MN} and J_{MN} approximants and applications to numerical inversion of Laplace transforms and initial value problems*, J. Math. Anal. Appl., 50 (1975), pp. 191–222.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 6, pp. 1265-1286, November 1992

MOVING MESH TECHNIQUES BASED UPON EQUIDISTRIBUTION, AND THEIR STABILITY*

YUHE REN† AND ROBERT D. RUSSELL†

Abstract. Various aspects of the moving mesh problem are investigated for the solution of partial differential equations (PDEs) in one space dimension. In particular, methods based (explicitly or implicitly) upon an equidistribution principle are studied. It is shown that equidistribution implicitly corresponds to finding a solution to a PDE involving a new set of computational coordinates. Implementation of a discrete version of equidistribution to compute a moving mesh corresponds to solving a weak form of the PDE. The stability of equidistribution is discussed, and it is argued that stability can be significantly affected by the way in which this solution process is carried out. Simple moving mesh methods are constructed using this framework, and numerical examples are given to illustrate their robustness.

Key words. moving mesh, monitor function, equidistribution, conservative methods for PDEs

AMS(MOS) subject classifications. 65M50, 65L50, 65N50

1. Introduction. One of the most important computational considerations when solving partial differential equations (PDEs) having nontrivial solutions is the decision of how to automatically and stably choose a nonuniform mesh that suitably adapts to the solution behaviour. For initial value PDEs, constructing a moving mesh in time can be essential if the problem is to be solved efficiently, and often if it is to be solved at all. The resolution of this issue has proven surprisingly difficult, and theoretical results have been particularly slow in forthcoming. Considerable controversy surrounds the questions of which overall strategy to use and how best to choose a moving mesh for a given strategy [14], even though few basic mesh selection principles are available. Here, we investigate one of the key mesh selection strategies: that in which equidistribution is explicitly done with respect to some nonnegative measure of the error. We also discuss the implications of these results with regard to some other strategies, and focus on PDEs in one space dimension.

First, consider the case of solving an ordinary differential equation (ODE), e.g.,

$$u_{xx}=f(x, u, u_x) \tag{1}$$

with boundary conditions $u(a)=\beta_1$, $u(b)=\beta_2$. The equidistribution idea, introduced by de Boor [7] and Dodson [10], is based upon the simple idea that if some measure of the error $M(x)$ is available, then a good choice for a mesh π: $a=x_0<x_1<\cdots<x_N=b$ would be one in which the contributions to the error over the subintervals are equalized (or "distributed equally"). In practice, most strategies find π by only approximately equidistributing with respect to the so-called monitor function $M(x)$, although White [41] provides a framework for doing this distribution exactly. He defines a change of variables

$$s=\frac{1}{\theta}\int_a^x M(\xi)\,d\xi, \qquad \theta=\int_a^b M(\xi)\,d\xi,$$

* Received by the editors October 3, 1990; accepted for publication (in revised form) September 9, 1991. This research was supported by Natural Sciences and Engineering Research Council of Canada grant A8781.

† Department of Mathematics and Statistics, Simon Fraser University, Burnaby, British Columbia, V5A 1S6, Canada.

and then forms a new system of ODEs consisting of the original ODE (say, (1)) rewritten in terms of this computational variable s, and the ODE

$$\frac{dx}{ds}=\frac{\theta}{M(x)}. \tag{2}$$

Equidistribution then corresponds to choosing π with $s(x_{i+1})-s(x_i)=1/N, i=0,1,\cdots,N-1$.

This continuous form has been a useful theoretical tool for interpreting schemes, but it is generally unreliable computationally because the new ODE system can be extremely sensitive to solve. While all of the reasons for numerical difficulties are not well understood, a major one is that the *transformed* ODE (in s) can be extremely nonlinear and its solution badly behaved due to the introduction of interior layers [34], [36]. Still, equidistribution strategies are widely used in conjunction with the original ODE (such as (1)), and in this way they have enjoyed general success.

Now consider an initial boundary value PDE

$$u_t=f(u, u_x, u_{xx}), \tag{3}$$

with $u(x,0), a\leqq x\leqq b$, and $u(a,t), u(b,t), t>0$ given. Our concern is to investigate properties of an equidistribution procedure, and in many respects this does not depend upon the form of the PDE itself. For example, the PDE could be a system of equations in $\mathbf{u}=(u_1,\cdots,u_n)^T$ such as

$$\mathbf{F}(\mathbf{u}_t, \mathbf{u}_x, \mathbf{u}_{xx})=0. \tag{4}$$

Many variations of equidistribution strategies have been investigated in practice. The first ones generally did a *static regridding* [19], where equidistribution to determine a new mesh is done after the solution to the PDE is computed at the new time level (see, e.g., [2]). Later, the PDE and mesh solution processes were combined to do *dynamic regridding* [19]. Several moving mesh methods based upon equidistribution were investigated by Coyle, Flaherty, and Ludwig [8]. Hyman [19] studied a moving mesh strategy for PDEs of the form (3), and later Petzold [32] did so for the implicit PDE (4).

Mathematically, the goal of finding mesh *functions* $\{x_i(t)\}_{i=1}^{N-1}$, or moving meshes

$$\pi\colon \{a=x_0<x_1(t)<\cdots<x_{N-1}(t)<x_N=b\}, \tag{5}$$

which are equidistributing for all values of t means that we want

$$\int_{x_{i-1}(t)}^{x_i(t)} M(x,t)\,dx=\frac{1}{N}\int_a^b M(x,t)\,dx=:\frac{1}{N}\theta(t), \qquad i=1,\cdots,N. \tag{6}$$

This equidistribution equation can be written equivalently as

$$\int_a^{x_i(t)} M(x,t)\,dx=\frac{i}{N}\int_a^b M(x,t)\,dx=\frac{i}{N}\theta(t), \qquad i=0,1,\cdots,N. \tag{7}$$

With static regridding, (6) or (7) is approximately satisfied at every new time level, where the monitor function $M(x,t)$ depends upon the just computed solution of the PDE at this time level. (For notational convenience, the explicit dependence of M upon u is not specified.) With dynamic regridding, the PDE is solved together with (6) or (7), and the rate at which the mesh moves is a function of how θ changes with time.[1]

[1] In actual fact, the regridding strategies only solve (7) approximately, producing so-called *asymptotically equidistributing* meshes, but this distinction will not be a focus of our presentation.

White [42] also studies the PDE case. As for the ODE, he replaces the physical variables x, t with a new set of computational coordinates s, T defined via the exact equidistribution process, viz.,

$$s = \frac{1}{\theta} \int_a^x M(\xi, t)\, d\xi, \qquad T = t. \tag{8}$$

He obtains a new PDE system consisting of the original PDE for u rewritten in terms of s and T, and

$$\frac{\partial x}{\partial s} = \frac{\theta}{M(x, T)}. \tag{9}$$

Several attempts to solve this transformed PDE for u (as a function of these new computational variables) have been made (see, e.g., [42] and [12]). This involves forming a discretization of the transformed PDE and solving a coupled system for the numerical solution and the equidistributing mesh which (approximately) satisfies (6). The resulting system is, however, generally sensitive to solve numerically [37]. The transformed PDE is nonlinear even if the original one is linear, and sometimes physically meaningless solutions are obtained. Still, as we shall see, it provides a useful model with which to interpret particular numerical schemes.

In [8], the stability of the equidistribution process is studied. In particular, differentiating (7) with respect to t, they study the equations

$$M(x_i, t)\dot{x}_i + \int_a^{x_i} M_t(x, t)\, dx = \frac{i}{N}\frac{d\theta}{dt}, \qquad i = 1, \cdots, N-1. \tag{10}$$

Using linear perturbation techniques for the mesh points, they perturb x_i by δx_i and take

$$M(x_i + \delta x_i, t)(\dot{x}_i + \delta\dot{x}_i) + \int_a^{x_i+\delta x_i} M_t(x, t)\, dx = \frac{i}{N}\frac{d\theta}{dt}, \tag{11}$$

and linearize (11) to get the first-order terms

$$M(x_i(t), t)\delta\dot{x}_i(t) + \frac{\partial M}{\partial x}(x_i(t), t)\dot{x}_i(t)\delta x_i(t) + \frac{\partial M}{\partial t}(x_i(t), t)\delta x_i(t) = 0,$$

that is

$$\frac{d}{dt}[M(x_i(t), t)\delta x_i(t)] = 0. \tag{12}$$

Integrating from $t = 0$ to t,

$$\delta x_i(t) = \frac{M(x_i(0), 0)}{M(x_i(t), t)}\delta x_i(0). \tag{13}$$

Doing this analysis and an accompanying numerical study, they conclude that mesh equidistribution, while unquestionably a desirable property for the mesh points, requires extreme care for its implementation because of potential instabilities for dissipative PDEs, where the perturbation terms in (13) can grow rapidly.

A number of attempts have been made to eliminate this potential numerical instability and to prevent mesh points from crossing by solving various forms of differential equations for the moving meshes. Generally, these attempts have involved some form of regularization [21], [32]. In the next section, we discuss the equidistribution problem within another framework, showing why the stability analysis needs to be used cautiously.

It is interesting to conjecture about the stability properties of equidistribution as they relate to what is known about moving finite element (MFE) methods [30]. MFE methods have proven very effective for parabolic PDEs [15], especially convection-dominated problems [3]. The methods have been shown to be related to a weak form of equidistribution [13], [17], [38]. Also, the MFE matrix can only become singular if an equidistribution relationship is violated [17], and for hyperbolics, they run into difficulty at the very point where the equidistribution property is lost [14]. In order to avoid difficulties, Miller [28] introduces regularization terms, and the result is that the penalty function in this regularization plays the key role of preserving equidistribution [17]. Nevertheless, the practical implications of these often tenuous theoretical connections are difficult to interpret, leaving many stability issues open to question.

2. Equidistribution PDE. In order to analyze further the stability of moving meshes satisfying an equidistribution principle, we derive a PDE that provides a new interpretation of exact equidistribution. From (8), $s\theta(t)=\int_a^x M(\xi,t)\,d\xi$, so assuming that M is a smooth function, along lines where $s(t)$ is *constant* with respect to time t,

$$s_t\theta+s\dot{\theta}=s\dot{\theta}=\int_a^x \frac{\partial M}{\partial t}(\xi,t)\,d\xi+M\dot{x}, \tag{14}$$

implying

$$s_x\dot{\theta}+s\dot{\theta}_x=s_x\dot{\theta}=\frac{\partial M}{\partial t}(x,t)+\frac{\partial}{\partial x}(M\dot{x}). \tag{15}$$

Thus we have the differential form

$$\frac{\partial}{\partial t}M(x,t)+\frac{\partial}{\partial x}(M(x,t)\dot{x})=\frac{\dot{\theta}}{\theta}M(x,t), \tag{16}$$

or

$$\frac{\partial}{\partial t}M+\operatorname{div}(M\dot{x})=\frac{\dot{\theta}}{\theta}M. \tag{17}$$

Consequently, doing equidistribution implicitly corresponds to finding a solution to (16). Although this is technically an integro-differential equation, we refer to it as a hyperbolic conservation-type PDE. (In the next section, the integral term is eliminated through a change of variables.) Differentiating (6) with respect to t, it is not difficult to show that the discrete equidistribution process for the mesh (5) corresponds to finding a solution to

$$\int_{x_{i-1}}^{x_i}\left[\frac{\partial M}{\partial t}(x,t)+\frac{\partial}{\partial x}(M(x,t)\dot{x})\right]dx=\frac{1}{N}\dot{\theta}=\int_{x_{i-1}}^{x_i}\frac{\dot{\theta}}{\theta}M(x,t)\,dx. \tag{18}$$

This viewpoint is valuable in several respects. Practically, the effects of discretizing (6) and (16) are similar. However, considerable experience has been gained from solving PDEs in conservation-type form like (16), so it is natural to try to develop new methods and interpret previous ones using this formulation. It also provides a physical interpretation of these moving-mesh methods in terminology common in fluid dynamics. The mesh points serve a similar function as particles of flow. In particular, the equidistribution coordinates are chosen using a quasi-Lagrangian approach: The moving mesh is along lines of constant

$$s(t)=\int_a^{x(t)}\frac{M(\xi,t)}{\theta(t)}\,d\xi.$$

Here, (6) satisfies a finite version of the integral form (8), or a *weak form.* The weak form (6) of the PDE shows that the "flux" of the error density function M is equivalent across the subintervals, or across each cell $[x_{i-1}, x_i]$, for each fixed time level. If the total measure of error in the interval $[a, b]$ is constant, then $\dot{\theta}=0$, and the moving mesh equation (16) becomes the Euler equation for the "fluid" with density function $M(x, t)$. Finally, the case where meshes are calculated using static regridding can be viewed as corresponding to the steady flow case in fluid dynamics, where the error density function $M(x, t)$ is independent of time. It is interesting to observe how the choice $M(x, t)=u_x$ and $\dot{\theta}\equiv 0$ corresponds to a Lagrangian coordinate system. In particular, for this case the integration of (16) leads to the well-known conservation law [24]

$$u_t + u_x \dot{x} = 0, \tag{19}$$

and $x(t)$ is simply a characteristic. For the arclength monitor function

$$M(x, t)=\sqrt{1+u_x^2},$$

if $u_x \gg 0$ then $M(x, t)\approx u_x$, and we see how the moving mesh equation reflects the shock behaviour where characteristics cross. For hyperbolic PDEs, the MFE method with no regularization has also been shown to produce moving meshes along characteristics [4], [16].

While (16) has not, to our knowledge, been used previously to interpret mesh selection schemes in a general setting, similar approaches have been investigated in special contexts. Larrouturou [23] develops an inexpensive moving mesh method for which the mesh points move with a time-dependent velocity $\dot{x}(t)$, the monitor function is chosen as a physical quantity (temperature), and the total energy $\theta(t)$ is constant. This gives a PDE for solving $\dot{x}(t)$ which is similar to (16), but with $\dot{\theta}(t)=0$. In Larrouturou's actual implementation, the new mesh is computed using static regridding.

Theoretically, White's approach [42], being based upon equidistribution, involves satisfying (16) exactly. He writes the original PDE in terms of the computational quasi-Lagrangian coordinates (s, T), and, since the solution has no steep gradients in these coordinates, a uniform mesh with $s(x_{i+1})-s(x_i)=1/N$ is used for the transformed PDE. He works with arclength as the monitor function, i.e.,

$$M(x, t)=\sqrt{1+u_x^2}.$$

From the relation between the moving mesh problem with equidistribution and (16), it is easy to see how problems can arise computationally. Approximation to the left-hand side should generally be done with a conservative scheme, or one could expect difficulties to arise. While many excellent methods of such type are available, when $\dot{\theta}\neq 0$ this term can cause considerable numerical difficulty, and finding suitable numerical methods just to solve a PDE of this type is not well understood [25]. The situation here is, of course, further complicated because the moving mesh PDE is coupled to the *original* PDE.

To see how difficulties can generally arise for the moving mesh equation, suppose we assume that $\dot{\theta}=0$. Then (16) becomes

$$\frac{\partial M}{\partial t}+\frac{\partial}{\partial x}(M\dot{x})=0. \tag{20}$$

If we use the nonconservative form

$$\frac{\partial M}{\partial t}+M\frac{\partial}{\partial x}(\dot{x})+\frac{\partial M}{\partial x}\dot{x}(t)=0, \tag{21}$$

and discretize using a standard method of lines procedure, we obtain

$$\frac{\partial M}{\partial t}(x_i(t), t)+M(x_i(t), t)\frac{\dot{x}_{i+1}(t)-\dot{x}_i(t)}{x_{i+1}(t)-x_i(t)}+\frac{\partial M(x_i(t), t)}{\partial x}\dot{x}_i(t)=0,$$

or

$$\frac{\partial M}{\partial t}(x_i, t)(x_{i+1}-x_i)+M(x_i, t)(\dot{x}_{i+1}-\dot{x}_i)+\frac{\partial M}{\partial x}(x_i, t)\dot{x}_i(x_{i+1}-x_i)=0. \tag{22}$$

Thus

$$\frac{d}{dt}[M(x_i(t), t)(x_{i+1}(t)-x_i(t))]=0, \tag{23}$$

and upon integrating, we get

$$x_{i+1}(t)-x_i(t)=\frac{M(x_i(0), 0)}{M(x_i(t), t)}(x_{i+1}(0)-x_i(0)). \tag{24}$$

Unless

$$\frac{M(x_i(0), 0)}{M(x_i(t), t)}$$

remains small, which it generally would *not* do for dissipative PDEs, the moving mesh points can easily leave the domain $[a, b]$.

These observations apply as well to the differential equation (13) developed in [8] using linear perturbation techniques. It is useful to investigate this linear perturbation analysis further. Expanding (12) and dividing by δx_i, we have

$$M(x_i(t), t)\frac{\delta\dot{x}_i(t)}{\delta x_i(t)}+\frac{\partial M}{\partial x}\dot{x}_i(t)+\frac{\partial M}{\partial t}(x_i(t), t)=0. \tag{25}$$

Letting $\delta x_i \to 0$, we obtain

$$M(x_i(t), t)\frac{\partial}{\partial x}(\dot{x}_i(t))+\frac{\partial M}{\partial x}\dot{x}_i+\frac{\partial M}{\partial t}(x_i(t), t)=0, \tag{26}$$

or

$$\left[\frac{\partial M}{\partial t}+\frac{\partial}{\partial x}(M\dot{x})\right]_{x=x_i}=0. \tag{27}$$

The steps from (25) to (27) can be retraced.

Thus the perturbation equation (13), used in [8] to study stability of the equidistribution process, can be obtained from setting the right-hand side in (16) to zero and writing the resulting equation at $x=x_i$ in nonconservative form. In retrospect, we see that (13) resembles a conservation of mass equation, where θ corresponds to total mass, which is unchanging with time. It is obtained from perturbing only the left-hand side of (7) or (10), since there is no perturbation expansion for the term i/N. We conclude that, while (13) is extremely useful for interpreting the stability of many *implementations* of equidistribution procedures, the stability properties of the equidistribution principle itself are more complicated.

Through a simple change of variables, (16) can be converted from a differential-integral equation in $M(x, t)$ to a differential equation for which the stability analysis of [8] is more generally applicable. Introducing the transformation

$$W(x, t) := \frac{M(x, t)}{\theta(t)} = \frac{M(x, t)}{\int_a^b M(x, t)\, dx}, \tag{28}$$

it is easy to see that (16) takes the equivalent form

$$\frac{\partial W(x, t)}{\partial t} + \frac{\partial (W(x, t)\dot{x})}{\partial x} = 0. \tag{29}$$

Thus, the "average energy" function $W(x, t)$, for which

$$\int_a^b W(x, t)\, dx \equiv 1,$$

satisfies the conservation equation (29). The transformation (28) is similar to the Cole-Hopf transformation [40], although the context and purpose are quite different. From (6), the weak form of the PDE (29) is

$$\int_{x_i}^{x_{i+1}} W(x, t)\, dx = \frac{1}{N}, \qquad i = 1, \cdots, N,$$

i.e., the total "average energy" between any two mesh lines remains constant. In principle, there is no reason why the moving mesh approaches constructed in terms of $M(x, t)$ cannot use $W(x, t)$ instead. The derivation from (20) through (24) can be repeated with W replacing M and, under the appropriate corresponding conditions (except with no right-hand side that needs to be ignored), we see that the potential for mesh crossings now occurs if

$$\frac{W(x_i(0), 0)}{W(x_i(t), t)} = \frac{M(x_i(0), 0)}{M(x_i(t), t)} \frac{\theta(t)}{\theta(0)},$$

a measure of the *average* change in $M(x_i(t), t)$, grows. In comparison with (24), we hope that the moving mesh equations derived using this new variable would be more robust, if not necessarily more efficient. Finally, the analysis [8] *is* more directly applicable to (29), so stability for the discretization of a nonconservative form of an equidistribution process is given by (13), with W replacing M.

3. Implementations of equidistribution. In this section, we consider ways in which the equidistribution process can be implemented. First, we consider how to choose the monitor or density function that controls the movement of the mesh points. This is more difficult than for ODEs due to the additional variable t. There are three basic choices of $M(x, t)$, which have been widely used in practice: (i) an arclength monitor function [42], [11]; (ii) a combination of gradient and curvature [27], [12], [20], [11], [31]; and (iii) truncation error or solution residual—used directly for ODEs [34], and either explicitly [2], [5] or implicitly [30], [17] for moving finite element methods for PDEs.

Stability properties of the moving mesh equations, while dependent upon the choice of monitor function, are to some extent arbitrary, since they usually behave asymptotically much like some fractional power of a solution derivative (see, e.g., [35]). Here, we use the arclength monitor function

$$M(x, t) = \sqrt{1 + u_x^2(x, t)}. \tag{30}$$

Our first implementation of a moving mesh method involves using an approximation for (16) of the form

$$\frac{\partial}{\partial t}M(x_i(t),t)+\frac{M_{i+1}\dot{x}_{i+1}(t)-M_i\dot{x}_i(t)}{x_{i+1}(t)-x_i(t)}=\frac{\dot{\theta}}{\theta}M(x_i(t),t), \qquad 1\leqq i\leqq N-1. \tag{31}$$

For the numerical examples presented in the next section, $\dot{x}>0$, so this simple upwind approximation to $(M\dot{x})_x$ is sufficient. On the interval $[x_i, x_{i+1}]$, we use the monitor function discretization

$$M_i := M(x_i,t)=\sqrt{1+\left(\frac{u_{i+1}-u_i}{x_{i+1}-x_i}\right)^2}. \tag{32a}$$

To maintain discrete conservative laws,

$$\theta(t)=\int_a^b M(x,t)\,dx=\sum_{i=0}^{N-1}\int_{x_i}^{x_{i+1}} M(x,t)\,dx$$

and

$$\dot{\theta}(t)=\int_a^b M_t(x,t)\,dx=\sum_{i=0}^{N-1}\int_{x_i}^{x_{i+1}} M_t(x,t)\,dx$$

are approximated using left rectangle rules

$$\int_{x_i}^{x_{i+1}} M(x,t)\,dx\approx(x_{i+1}-x_i)M(x_i(t),t), \tag{32b}$$

$$\int_{x_i}^{x_{i+1}} M_t(x,t)\,dx\approx(x_{i+1}-x_i)M_t(x_i(t),t). \tag{32c}$$

We test this moving mesh strategy, hereafter called Method I, both with and without the right-hand side in (31). For both, only the fixed boundary case $\dot{x}_0(t)=\dot{x}_N(t)=0$ is considered.

We also consider a moving mesh method developed using (29). Approximating this PDE over $[x_i, x_{i+1}]$ at $t=t_{n+1}=(n+1)\Delta t$ by

$$\frac{W_i^{n+1}-W_i^n}{\Delta t}+\frac{W_i^{n+1}\dot{x}_{i+1}-W_{i-1}^{n+1}\dot{x}_i}{x_{i+1}-x_i}=0, \tag{33}$$

upon rearrangement, we obtain

$$W_i^{n+1}(x_{i+1}-x_i)+\Delta t(W_i^{n+1}\dot{x}_{i+1}-W_{i-1}^{n+1}\dot{x}_i)=(x_{i+1}-x_i)W_i^n. \tag{34a}$$

As it stands, (34a) would be awkward to implement because it depends upon $\theta(t_n)$ and $\theta(t_{n+1})$, and therefore on *all* of the solution approximations at time levels t_n and t_{n+1}. Fortunately, it is possible to simplify by working with two adjacent interval discretizations. A similar approximation to (34a) on $[x_{i-1}, x_i]$ gives

$$W_{i-1}^{n+1}(x_i-x_{i-1})+\Delta t(W_{i-1}^{n+1}\dot{x}_i-W_{i-2}^{n+1}\dot{x}_{i-1})=(x_i-x_{i-1})W_{i-1}^n. \tag{34b}$$

Equidistribution implies

$$\int_{x_i}^{x_{i+1}} W(x,t)\,dx=\int_{x_{i-1}}^{x_i} W(x,t)\,dx. \tag{35}$$

Equating the right-hand sides of (34a) and (34b), which are approximations to (35) at $t = t_n$, we obtain

$$W_i^{n+1}(x_{i+1} - x_i) + \Delta t(W_i^{n+1}\dot{x}_{i+1} - W_{i-1}^{n+1}\dot{x}_i) = W_{i-1}^{n+1}(x_i - x_{i-1}) + \Delta t(W_{i-1}^{n+1}\dot{x}_i - W_{i-2}^{n+1}\dot{x}_{i-1}).$$

Since each term involves

$$W(x, t_{n+1}) = \frac{M(x, t_{n+1})}{\theta(t_{n+1})},$$

$\theta(t)$ can be eliminated, leaving the discrete approximation

$$\tau(M_i\dot{x}_{i+1} - 2M_{i-1}\dot{x}_i + M_{i-2}\dot{x}_{i-1}) = M_{i-1}(x_i - x_{i-1}) - M_i(x_{i+1} - x_i) \tag{36}$$

at $t = t_{n+1}$. Here, to avoid confusion with the time integration steps later on (when the moving mesh equations are integrated with a method of lines approach), we write $\tau := \Delta t$. This moving mesh strategy, which we refer to as Method II (also using (32a) and with $\dot{x}_0(t) = \dot{x}_N(t) = 0$), is considered in the next section.

A great variety of moving mesh equations have been obtained by others, using the various choices of monitor functions and approximation schemes. In the remainder of this section, we show how some of these equations are related to the equidistribution relationships derived in § 2, either in the differential form (16) or the weak form (6).

For the moving finite element methods of Miller and Miller [30] and Herbst, Schoombie, and Mitchell [17], the moving mesh equations are derived from the weak form of the PDEs written in Lagrangian form. In particular, a given PDE $u_t = L(u)$ is converted to its Lagrangian form $\dot{u} - u_x\dot{x} = L(u)$. Suitable weight functions $\phi_i(x)$ and $\psi_i(x)$ are chosen, and the residual

$$R(u) = \dot{u} - u_x\dot{x} - L(u)$$

is required to satisfy the orthogonality relations

$$\int_a^b \phi_i(x)R(u)\,dx = 0, \tag{37}$$

$$\int_a^b \psi_i(x)R(u)\,dx = 0. \tag{38}$$

The choice

$$\phi_i(x) = \alpha_i(x) \tag{39}$$

and

$$\psi_i(x) = \beta_i(x) = -u_x\alpha_i(x), \tag{40}$$

where $\alpha_i(x)$ is the hat function and $\beta_i(x)$ is a discontinuous piecewise linear function on $[x_{i-1}, x_{i+1}]$, is made in [30], and the choice of the piecewise cubic Hermite polynomials

$$\phi_i(x) = [\alpha_i(x)]^2[3 - 2\alpha_i(x)], \tag{41}$$

$$\psi_i(x) = [\alpha_i(x)]^2[\alpha_i(x) - 1]\left[\frac{d\alpha_i(x)}{dx}\right]^{-1}, \tag{42}$$

is used in [17]. Both of these can be shown to be implicitly based upon a weak form of (16). In particular, requiring that

$$\int_{x_{i-1}}^{x_i} M(x, t)\,dx = \int_{x_i}^{x_{i+1}} M(x, t)\,dx \tag{43}$$

is satisfied, for the monitor function

$$M_i(x, u) = (x_i - x_{i-1})R(u_i),$$

we obtain the moving mesh equation corresponding to (41) and (42), and for

$$M(x, t) = \begin{cases} -u_x R(u), & x \in [x_{i-1}, x_i], \\ u_x R(u), & x \in [x_i, x_{i+1}], \end{cases}$$

we obtain the moving mesh equation corresponding to (39) and (40) [13].

Aside from stability, one of the most troublesome problems for a moving mesh method is the tendency for mesh points to cross. For equidistribution (6), this happens easily if M changes sign, so to avoid this, the early papers on equidistribution define a monitor function to be nonnegative. For MFE methods, the associated equidistribution property above holds for a monitor function that changes sign, and consistent with this is the fact that regularization terms generally must be added to prevent mesh crossings. In contrast, we find that for discretizations formed directly from (16), for positive monitor functions the problem of mesh crossing itself can be minimal (see §§ 4 and 5).

If instead of (16) we take

$$\frac{\partial}{\partial x}(M\dot{x}) = 0,$$

and write it in the nonconservative form

$$\frac{\partial M}{\partial x}\dot{x} + M\frac{\partial \dot{x}}{\partial x} = 0,$$

then the discretization

$$\frac{M_i - M_{i-1}}{x_i - x_{i-1}}\dot{x}_i + M_i\frac{\dot{x}_i - \dot{x}_{i-1}}{x_i - x_{i-1}} = 0$$

gives

$$\dot{x}_i - \dot{x}_{i-1} = -\frac{\dot{x}_i}{M_i}(M_i - M_{i-1}).$$

This is similar to the moving mesh equation of [1], except that they attempt to optimize a parameter value that is used in place of $\dot{x}_i/M_i$.

If the monitor function is simply the solution to the PDE, i.e., $M(x, t) = u$, then (16) becomes

$$\frac{\partial u}{\partial t} + \frac{\partial(u\dot{x})}{\partial x} = u\frac{\dot{\theta}}{\theta},$$

or, in nonconservative form,

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x}\dot{x} + u\frac{\partial \dot{x}}{\partial x} = u\frac{\dot{\theta}}{\theta}. \tag{44}$$

In developing a moving mesh strategy, Petzold [32] attempts to minimize, for a suitable parameter α, the objective function

$$\phi(\dot{x}, \dot{u}) = \left(\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x}\dot{x}\right)^2 + \alpha\dot{x}^2,$$

which is a measure of the change in the solution u and mesh x with respect to time t.

Since meshpoints can easily cross with this choice, she introduces a penalty function

$$\lambda\left[\left(\frac{\dot{x}_i - \dot{x}_{i-1}}{x_i - x_{i-1}}\right)^2 + \left(\frac{\dot{x}_{i+1} - \dot{x}_i}{x_{i+1} - x_i}\right)^2\right].$$

This can be viewed as "compensation" for the extra term $u(\partial\dot{x}/\partial x)$ in (44), which gives a scheme that in some sense minimizes the source error energy $u(\dot{\theta}/\theta)$ for the PDE when moving mesh points in time. The usefulness of this interpretation of Petzold's scheme to develop other practical moving mesh strategies remains to be investigated.

One of the most reliable moving mesh discretizations is due to Dorfi and Drury [11] and analyzed in [39]. It is similar to (36), where the general relationship between them involves using an artificial dissipation term in conjunction with (36) [33].

4. Numerical results. Here we give some numerical examples to examine the moving mesh strategy from Method I, with and without the right-hand side of (31), and the strategy from Method II. We choose three examples, consisting of one hyperbolic and two parabolic problems.

To discretize the PDE

$$\frac{\partial u}{\partial t} = f(u, u_x, u_{xx}), \tag{45}$$

we first write it in the Lagrangian form

$$\dot{u} - u_x\dot{x} = f(u, u_x, u_{xx}). \tag{46}$$

Using a central difference scheme for the spatial derivatives, we obtain

$$\dot{u}_i - \frac{u_{i+1} - u_{i-1}}{x_{i+1} - x_{i-1}}\dot{x}_i = f_i, \qquad i = 2, \cdots, N. \tag{47}$$

Thus we solve the coupled system of equations (47) and (31), with and without $\dot{\theta} = 0$, and the coupled system (47) and (36). This ODE system is solved using the code LSODI of Hindmarsh [18]. An approximate Jacobian is computed by LSODI internally using difference quotients. For simplicity, an initial uniform mesh is used in each case. In the tables of numerical results reported, *nst* and *nje* are, respectively, the number of steps and number of Jacobian evaluations taken by LSODI up to the time given; and *nqn* and *tstep* are, respectively, the order of the last successful method and the last successful stepsize. All runs were made on Sparcstations in a distributed computing environment, and computer times are not given. Method I is more expensive using the right-hand side in (31) than not using it, but the difference is not very significant (always less than 20 percent for these problems).

Problem I. This problem, a scalar reaction diffusion problem from combustion theory, has been used by several authors to test their moving mesh strategies [2], [32], [14]. It is a model of a single-step reaction with diffusion,

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + D(1 + a - u)\, e^{-\delta/u}, \qquad t > 0, 0 < x < 1,$$

$$u_x(0, t) = 0, u(1, t) = 1, \qquad t > 0,$$

$$u(x, 0) = 1, \qquad 0 \leqq x \leqq 1,$$

where the constant heat release is a, reaction rate is R, activation energy is δ, and Damkohler number is $D = Re^{\delta}/(a\delta)$. The solution represents the temperature of a reactant in a combustion. For a short time, the temperature gradually increases from unity with a "hot spot" forming at $x=0$. At a finite time ignition occurs and the temperature at $x=0$ jumps rapidly from unity to $1+a$. A flame front then forms and propagates towards $x=1$ with speed proportional to $e^{a\delta}/2(1+a)$. Here, a is about unity and δ is large, so that the flame front moves exponentially fast after ignition. The solution reaches a steady state once the flame propagates to $x=1$. This problem serves as a good test of moving mesh methods because of the sensitivity of tracking the flame front [1], [2].

The derivative boundary condition $(\partial u/\partial x)(0, t) = 0$ is approximated by

$$\frac{u_1 - u_2}{x_1 - x_2} = 0 \quad \text{or} \quad u_1 - u_2 = 0.$$

The problem is solved for $a=1$, $\delta = 20$, and $R=5$, using a moving mesh with $N=20$ and with $N=40$. The results are compared with a reference solution (solid lines in the figures) obtained by LSODI, using the method of lines with standard central differences on (45) and $N=500$ equally spaced mesh points, with absolute tolerance $\text{atol} = 10^{-8}$ and relative tolerance $\text{rtol} = 10^{-6}$. The problem is quite sensitive to the tolerances for LSODI. For example, for $\text{atol} = \text{rtol} = 10^{-3}$, the numerical solution (not given here) moves too quickly and is very inaccurate.

Figure 1 shows the numerical solution computed using Method I with $\dot{\theta} \neq 0$, for $N=20$, $\text{atol} = 10^{-5}$, and $\text{rtol} = 10^{-4}$. The solution is fairly accurate except for an error caused by the solution moving too quickly, so that it gives a slight shift for $t=0.27$ and 0.28. This error is caused largely by the time integration, as the results change qualitatively when smaller tolerances are used in LSODI (see below). The corresponding results for $\dot{\theta} = 0$ are shown in Fig. 2. Note that the solution is inaccurate at the left boundary when $t=0.26$, and the solution is not very well equidistributed with respect to arclength, especially near the left boundary. The sensitivity of the problem with respect to integrator tolerances is severe, as performing the same runs with larger tolerances can easily give poorer results, but even using $\text{atol} = 10^{-6}$ and $\text{rtol} = 10^{-5}$ gives lower accuracy (cf. Figs. 3 and 4).

For Method II with $\text{atol} = 10^{-5}$, $\text{rtol} = 10^{-5}$, $\tau = 10^{-5}$, and $N=20$, the numerical solution moves slightly slower than the reference solution before reaching steady state (see Fig. 5). Reducing the spatial mesh to $N=40$, the solution has fairly high accuracy throughout, as shown in Fig. 6. Reducing τ or the integrator tolerances does not qualitatively affect the numerical solution, although from our experience τ should be kept smaller than the time integration stepsize used in LSODI. Note that the arclength is considerably better distributed between mesh points than for the other moving mesh equation.

The timestepping information for the runs summarized in the figures is given in Table 1. In particular, the number of steps and Jacobian evaluations, order of the integration method, and final stepsize used by LSODI are listed.

Problem II. Our next example is Burgers' equation

$$\frac{\partial u}{\partial t} = -\frac{\partial f(u)}{\partial x} + \varepsilon \frac{\partial^2 u}{\partial x^2}, \qquad t>0, 0<x<1,$$

$$u(0,t)=0, u(1,t)=0, \qquad t>0,$$

$$u(x,0)=u^0(x),$$

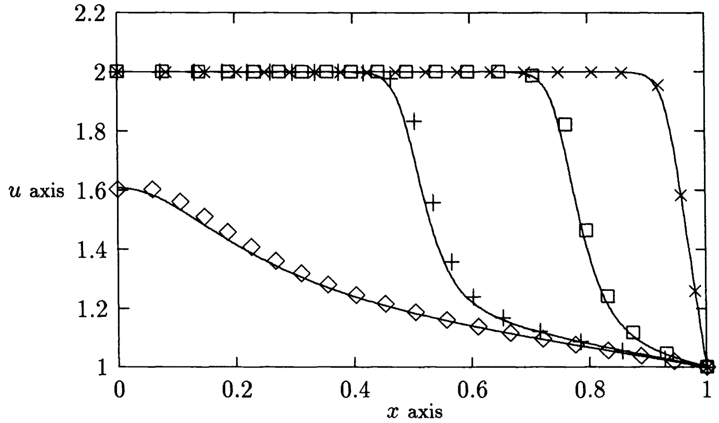


FIG. 1. *First example, using Method* I *with* $\dot{\theta} \neq 0$, $t = 0.26, 0.27, 0.28, 0.29$; atol $= 10^{-5}$, rtol $= 10^{-4}$, *mesh points* $N = 20$.

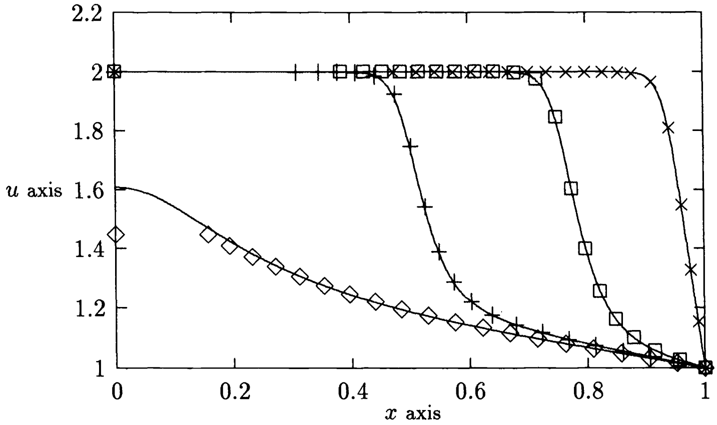


FIG. 2. *First example, using Method* I *with* $\dot{\theta} = 0$, $t = 0.26, 0.27, 0.28, 0.29$; atol $= 10^{-5}$, rtol $= 10^{-4}$, *mesh points* $N = 20$.

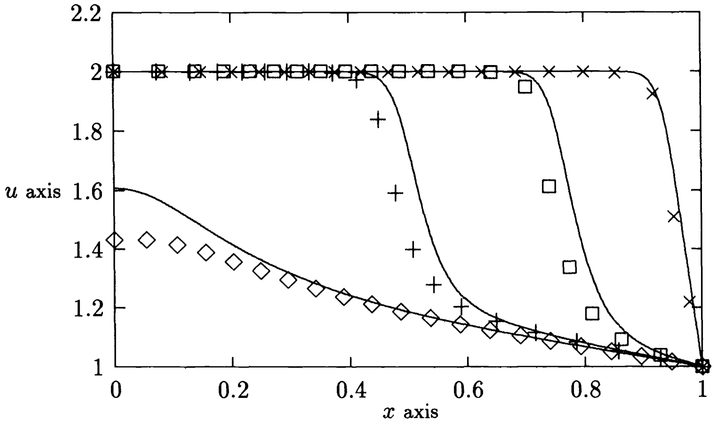


FIG. 3. *First example, using Method* I *with* $\dot{\theta} \neq 0$, $t = 0.26, 0.27, 0.28, 0.29$; atol $= 10^{-6}$, rtol $= 10^{-5}$, *mesh points* $N = 20$.

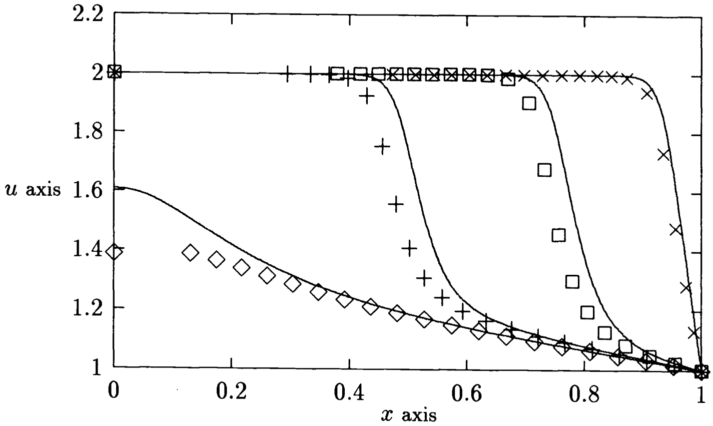


FIG. 4. *First example, using Method* I *with* $\dot{\theta}=0$, $t=0.26, 0.27, 0.28, 0.29$; $\text{atol}=10^{-6}$, $\text{rtol}=10^{-5}$, *mesh points* $N=20$.

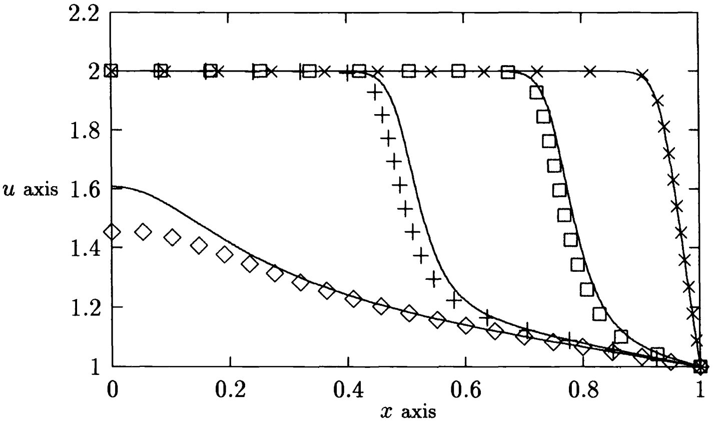


FIG. 5. *First example, using Method* II, $t=0.26, 0.27, 0.28, 0.29$; $\text{atol}=10^{-5}$, $\text{rtol}=10^{-5}$, *mesh points* $N=20$.

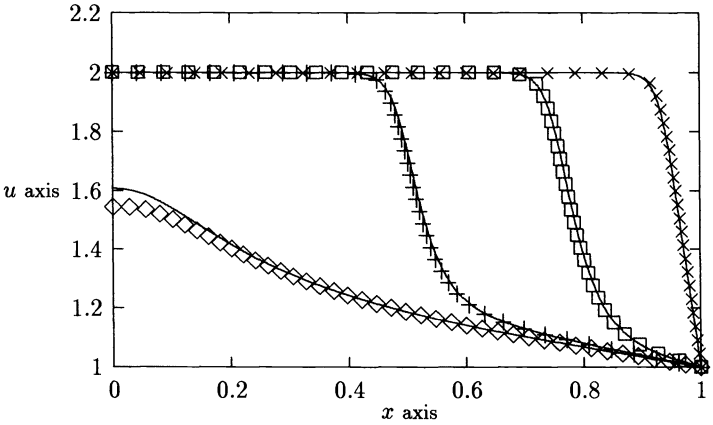


FIG. 6. *First example, using Method* II, $t=0.26, 0.27, 0.28, 0.29$; $\text{atol}=10^{-5}$, $\text{rtol}=10^{-5}$, *mesh points* $N=40$.

TABLE 1

			Fig. 1				Fig. 2	
t	nst	nje	nqn	tstep	nst	nje	nqn	tstep
0.26	33	14	3	.000286	30	11	3	.000728
0.27	80	21	3	.000374	80	25	3	.000390
0.28	108	25	3	.000369	114	33	2	.000368
0.29	138	28	3	.000395	145	40	3	.000466
t			Fig. 3				Fig. 4	
0.26	53	13	3	.000280	54	12	3	.000514
0.27	129	23	4	.000175	121	24	3	.000216
0.28	187	31	3	.000161	168	31	3	.000274
0.29	246	39	3	.000187	212	38	3	.000263
t			Fig. 5				Fig. 6	
0.26	46	12	3	.000302	57	16	3	.000121
0.27	146	34	3	.000183	143	34	1	.000109
0.28	193	43	3	.000259	197	44	2	.000265
0.29	233	52	2	.000348	236	56	2	.000393

where $f(u)=u^2/2$. This problem is also often used as a test (occasionally the only test) of mesh selection strategies.

We use $\varepsilon=10^{-2}$ and $\varepsilon=10^{-4}$ and the smooth initial solution $u^0(x)=\sin(2\pi x)+\frac{1}{2}\sin(\pi x)$. For small times and ε, the exact solution is a pulse that moves in the positive x direction while steepening. The reference solution (solid lines) is computed as in Problem I except with $N=1000$, rtol $=10^{-6}$, and atol $=10^{-8}$. The solution is shown for time $=0.2, 0.4, 0.8, 1.0, 1.4$, and 2.0. For Method II, $\tau=10^{-5}$.

Using Method I with $\dot{\theta}=0$, the method easily breaks down due to mesh crossing for $\varepsilon=10^{-2}$. For example, for $N=20$, atol $=10^{-3}$, and rtol $=10^{-3}$, breakdown occurs because the second mesh point crosses the left boundary and becomes negative at $t=0.35$. For $\varepsilon=10^{-4}$, atol $=10^{-4}$, and rtol $=10^{-5}$ several mesh points cross each other on the interval $[0.519, 0.597]$ at $t=0.2$. This is consistent with the theoretical and numerical findings of [8] regarding potential instability of (20).

For $\varepsilon=10^{-2}$, the presence of the right-hand side term $\dot{\theta}\neq 0$ now stabilizes the results. Figures 7 and 8 show the solutions and mesh points for $N=20$ with atol $=10^{-3}$, rtol $=10^{-3}$ and atol $=10^{-4}$, rtol $=10^{-5}$, respectively. The corresponding timestepping information is given in Table 2. The solutions are quite accurate except at the points of zero gradient ($u_x=0$), where the graph is somewhat higher than that for the reference solution. This same problem occurs using Method II: results for the same parameter values are given in Fig. 9 and Table 2. Note, too, that the degree of equidistribution is rather poor in this region. Using $N=40$, these inaccuracies are remedied, and the problem resolution is generally quite satisfactory (see Fig. 9(a)).

For $\varepsilon=10^{-4}$, the problem causes considerable difficulty. At about $t=0.2$, a shock layer forms near $x=0.6$. Setting $N=20$, using Method I with $\dot{\theta}\neq 0$, LSODI stops at the very steep layer at $t=0.218224$ due to a small stepsize (tstep $=10^{-6}$ and nst $=955$ for atol $=10^{-4}$, rtol $=10^{-5}$). With (36) and corresponding parameter values, LSODI also stops, now at $t=0.341905$ with tstep $=0$ and nst $=949$. Using $N=40$, LSODI is able to progress further, but still soon fails. The same difficulty of breakdown when the shock develops can occur for this problem with a high-order MFE method using Hermite cubic test functions [16], although other methods are successful [14], [28].

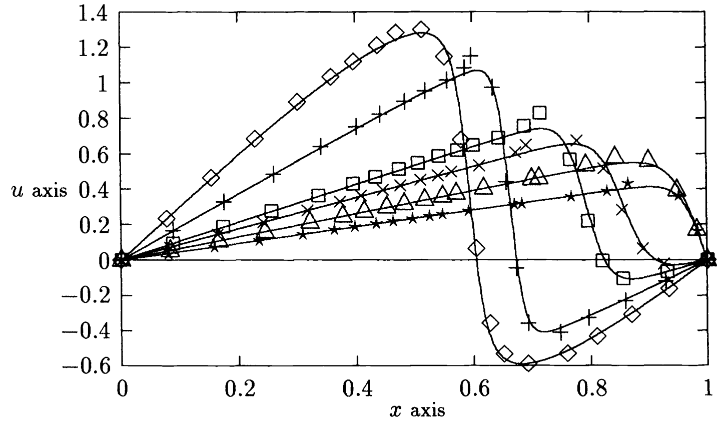


FIG. 7. *Burgers' problem, using Method* I *with* $\dot{\theta} \neq 0$, $\varepsilon = 10^{-2}$, $t = 0.2, 0.4, 0.8, 1.0, 1.4, 2.0$; atol $= 10^{-3}$, rtol $= 10^{-3}$, *mesh points* $N = 20$.

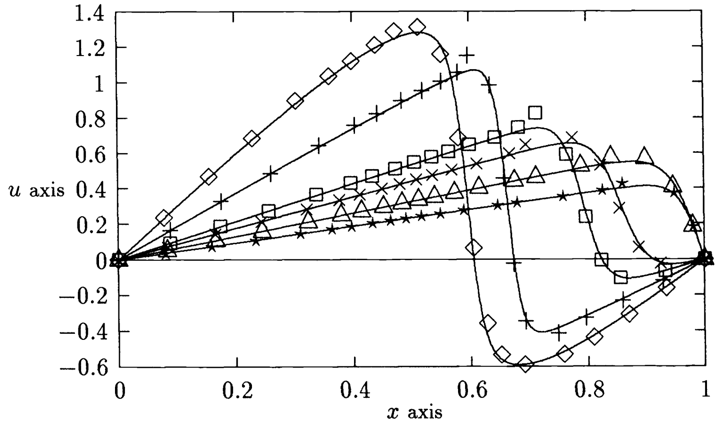
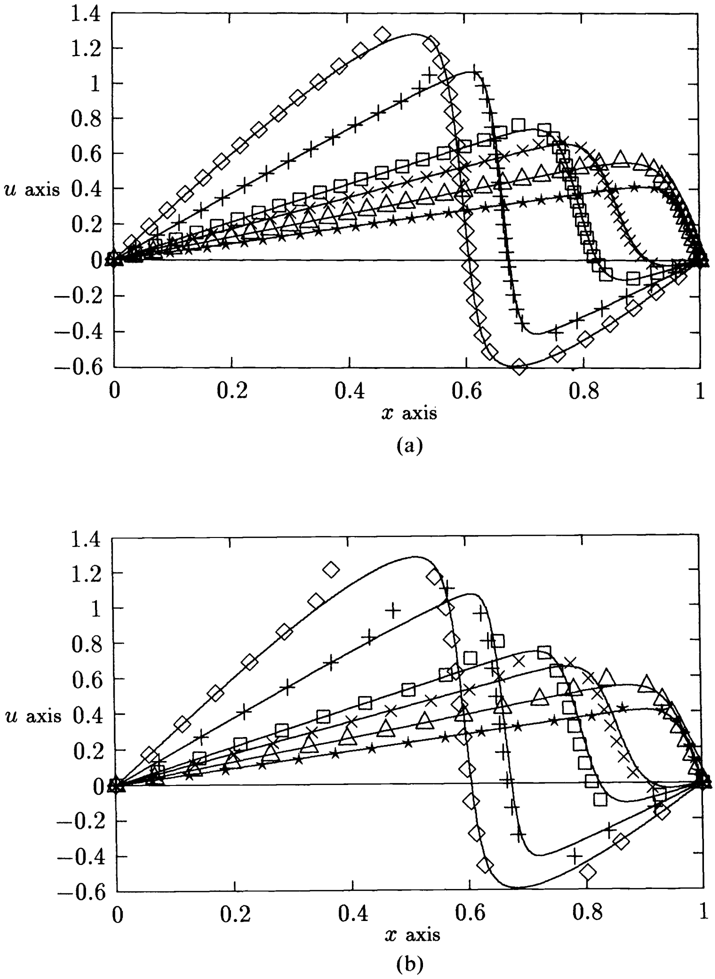


FIG. 8. *Burgers' problem, using Method* I *with* $\dot{\theta} \neq 0$, $\varepsilon = 10^{-2}$, $t = 0.2, 0.4, 0.8, 1.0, 1.4, 2.0$; atol $= 10^{-4}$, rtol $= 10^{-5}$, *mesh points* $N = 20$.

TABLE 2

	Fig. 8				Fig. 9			
t	nst	nje	nqn	tstep	nst	nje	nqn	tstep
0.2	39	10	3	.005603	88	24	3	.004330
0.4	83	21	3	.009411	103	29	2	.002011
0.8	165	43	3	.012868	171	57	2	.008990
1.0	186	49	4	.014660	197	64	3	.012833
1.4	218	57	3	.022651	215	69	3	.033861
2.0	234	59	3	.050789	242	79	2	.055271



(a)

(b)

FIG. 9. (a) *Burgers' problem, using Method* II, $\varepsilon = 10^{-2}$, $t = 0.2, 0.4, 0.8, 1.0, 1.4, 2.0$; atol $= 10^{-4}$, rtol $= 10^{-5}$, *mesh points* $N = 40$. (b) *Burgers' problem, using Method* II, $\varepsilon = 10^{-2}$, $t = 0.2, 0.4, 0.8, 1.0, 1.4, 2.0$; atol $= 10^{-4}$, rtol $= 10^{-5}$, *mesh points* $N = 20$.

Problem III. The last example is the hyperbolic conservative Buckley-Leverett equation

$$u_t + f(u)_x = 0$$

with the nonconvex flux function

$$f(u) = \frac{u^2}{u^2 + \frac{1}{2}(1-u)^2},$$

as in, e.g., [9]. The moving mesh methods of [1] and [15] test the problem with an artificial viscosity term εu_{xx} added. (See also [22].)

We consider the continuous initial data condition

$$u(x,0) = \frac{0.1}{0.1+x}, \qquad 0 \leqq x \leqq 1,$$

and boundary conditions

$$u(0,t)=1, \qquad u(1,t)=\frac{1}{11},$$

where we express the right boundary condition for LSODI in the form $\dot{u}_N(t)=0$. The reference solution is determined as in the other two problems, with $N=500$, atol $=10^{-8}$, and rtol $=10^{-6}$, and the solution profile shown for $t=0.1, 0.2, 0.3, 0.4$. With $N=20$, results with and without the right-hand side term in Method I are given in Figs. 10 and 11, respectively. These numerical solutions are virtually identical and move faster than the reference solution. For Method II with atol $=10^{-4}$, rtol $=10^{-5}$, and $\tau=10^{-5}$, LSODI stops due to the steep layer for $t=0.303228$ with tstep $=0$ and nst $=575$. (Again, mesh crossing is not a problem.) Adding the artificial viscosity term mentioned above, here with $\varepsilon=10^{-4}$, the problem is solved more satisfactorily than before, using atol $=10^{-4}$ and rtol $=10^{-5}$. The results, given in Fig. 12, are qualitatively unchanged for smaller tolerances, for example, atol $=10^{-5}$ and rtol $=10^{-6}$ (see Table 3). The scheme developed in [1] has no difficulty for this problem when solved as a parabolic PDE

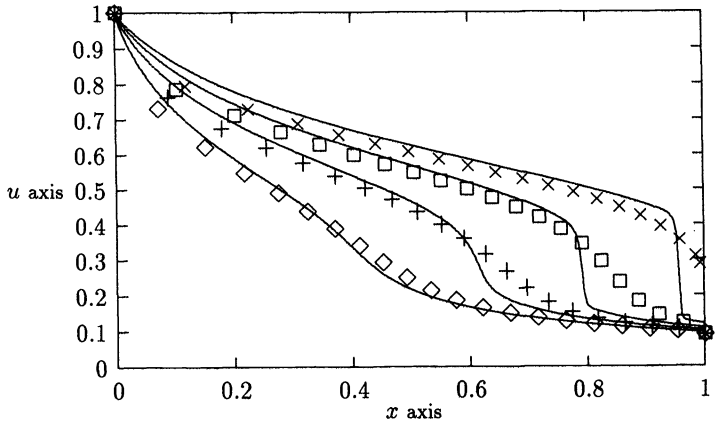


FIG. 10. *Buckley-Leverett problem, using Method* I *with* $\dot{\theta}\neq 0$, $t=0.1, 0.2, 0.3, 0.4$; atol $=10^{-5}$, rtol $=10^{-6}$, *mesh points* $N=20$.

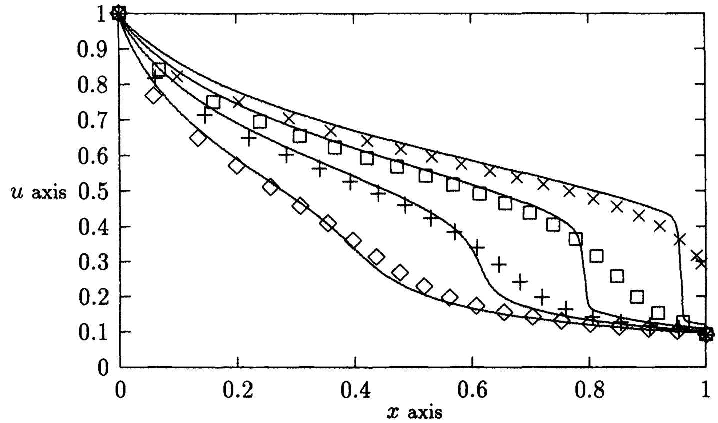


FIG. 11. *Buckley-Leverett problem, using Method* I *with* $\dot{\theta}=0$, $t=0.1, 0.2, 0.3, 0.4$; atol $=10^{-5}$, rtol $=10^{-6}$, *mesh points* $N=20$.

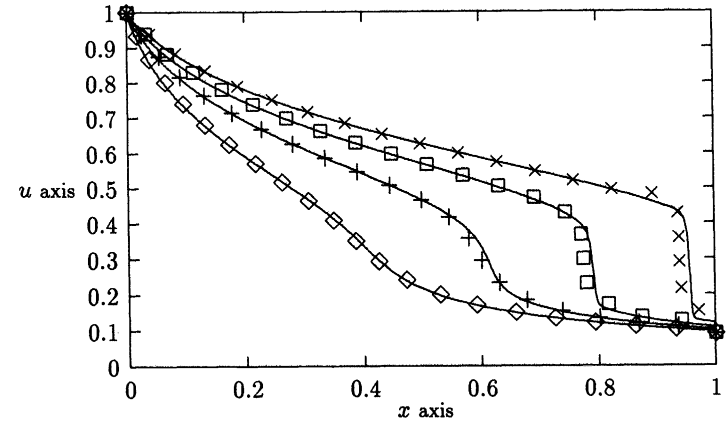


FIG. 12. *Buckley-Leverett problem, using Method* II, $t = 0.1, 0.2, 0.3, 0.4$; *mesh points* $N = 20$, (a) atol $= 10^{-4}$, rtol $= 10^{-5}$. (*Results for* (b) atol $= 10^{-5}$, rtol $= 10^{-6}$ *indistinguishable.*)

TABLE 3

	Fig. 10				Fig. 11			
t	nst	nje	nqn	tstep	nst	nje	nqn	tstep
0.1	21	4	4	.009126	22	4	4	.008524
0.2	32	6	4	.009126	33	5	4	.008524
0.3	43	8	4	.009126	48	7	4	.006048
0.4	57	13	4	.005544	74	13	3	.003204
t	Fig. 12(a)				Fig. 12(b)			
0.1	60	14	3	.014108	75	17	3	.012381
0.2	67	15	3	.014108	85	18	3	.009835
0.3	75	19	3	.013226	95	24	3	.007773
0.4	606	307	3	.000147	883	372	2	.000415

using real artificial viscosity, $\varepsilon = 10^{-3}$. However, it is interesting to wonder when a difficulty arises solving hyperbolic PDEs because the scheme is nonconservative when viewed as a scheme for solving the moving mesh PDE.

5. Conclusions. We have presented a new formulation of the equidistribution strategy in terms of a PDE. Previously, authors who have explicitly used equidistribution have generally developed moving mesh procedures that use (6), the integrated or weak form of the conservative integral. We intend to further develop robust moving mesh strategies based directly upon the differential form (16) or (29). Here, our intention has been to present some simple strategies. The purpose has not been to give extensive numerical results or a detailed comparison with other methods; this is done elsewhere [33]. Nevertheless, the results indicate that the schemes given here, with simple improvements such as smoothing of the mesh (for Problem II) when necessary, should prove competitive with those that have been recommended by others [14]. Use of conservative-type schemes to approximate the PDEs is natural and probably essential in many contexts. The importance of the right-hand side term of (16) is unclear, and we have included numerical results for $\dot{\theta} = 0$ partly to determine the effect and partly because this corresponds to what many previous implementations have used.

The numerical methods used here are quite simple and are presented mainly for illustrative purposes. Constructing more robust moving mesh methods could well require the incorporation of regularization terms as in [11], [14], [26], [29], [39], and possibly a more complicated monitor function, an obvious choice being some combination of arclength and curvature. However, while using the arclength monitor function can limit the number of mesh points that are placed in the transition region, strong nonlinearities that arise using a curvature monitor function can also cause computational difficulties [6]. Efficient ways to produce the moving mesh equations using this approach, particularly for higher-order systems (4), and for the higher-dimensional form of (17) or (29), remain to be investigated. Still, it is important to realize that the scheme is not plagued with mesh crossings the way most other simple moving mesh schemes are. When the PDE (16) (including the right-hand side) is approximated, we find very little difficulty of this type. In one case (Burgers' equation with different initial conditions than given here) (31) gave mesh crossing with a large tolerance, but this was fixed when the tolerance was reduced. While there is no need to add penalty functions for this reason, it may still be necessary to perform a mesh smoothing to prevent problem stiffness when steep solution layers occur (as was the difficulty in Problem II in the previous section). Obviously, a desirable ultimate goal is the development of a robust scheme with minimal requirements for a user to select contentious problem-dependent parameters.

This moving mesh PDE interpretation can be used to understand stability properties for moving mesh strategies, and extends the understanding of the stability properties as given in [8]. While the stability issue for methods based upon equidistribution is a very complicated one, and there is no doubt that a complicated interaction takes place between the PDE (3) and the mesh PDE (16) or (29), we expect that this viewpoint will be used to develop a deeper understanding of stability properties for currently used methods which have proven reliable. It is important to realize how many moving mesh methods are based upon equidistribution, making it possibly the single most important concept in the development of moving mesh methods. Many of these methods use equidistribution explicitly (e.g., [2], [20], [32], and those in [8]), and many of these often have stability difficulties [8]. There are also those, like the moving finite element methods [30], [17] and the elliptic grid generation methods [27], that have been developed from another viewpoint, but for which equidistribution has played a role—just how fundamental is unclear at this stage. The considerable success of some of these methods may be due in part to the fact that the moving mesh PDE (16) is solved implicitly, so that inadequate approximations from using nonconservative schemes, or from ignoring the important right-hand side term, have been circumvented. Of course, another underlying issue of critical importance is that of deciding what monitor function to use, and it is unrealistic to expect that a single choice for M would serve as a panacea for most problems.

Acknowledgments. The authors are very grateful to the referees, who suggested a number of significant improvements to the paper.

REFERENCES

[1] S. Adjerid and J. E. Flaherty, *A moving-mesh finite element method with local refinement for parabolic partial differential equations*, Comput. Methods Appl. Mech. Engrg., 55 (1986), pp. 3-26.

[2] ———, *A moving finite element method with error estimation and refinement for one-dimensional time dependent partial differential equations*, SIAM J. Numer. Anal., 23 (1986), pp. 778-795.

[3] M. J. BAINES, *Moving finite elements and approximate Legendre transformations*, Numerical Analysis Report 5/89, Dept. of Mathematics, University of Reading, Reading, UK.

[4] ———, *Moving finite elements and envelopes*, manuscript.

[5] M. BIETERMAN AND I. BABUŠKA, *An adaptive method of lines with error control for parabolic equations of the reaction-diffusion type*, J. Comput. Phys., 63 (1986), pp. 33-66.

[6] J. G. BLOM AND J. G. VERWER, *On the use of the arclength and curvature monitor in a moving-grid method which is based on the method of lines*, Report NM-N8902, Centrum voor Wiskunde en Informatica, Amsterdam, the Netherlands, 1989.

[7] C. DE BOOR, *Good approximation by splines with variable knots*. II, in Springer Lecture Notes Series 363, Springer-Verlag, Berlin, 1973.

[8] J. M. COYLE, J. E. FLAHERTY, AND R. LUDWIG, *On the stability of mesh equidistribution strategies for time-dependent partial differential equations*, J. Comput. Phys., 62 (1986), pp. 26-39.

[9] P. CONCUS AND W. PROSKUROWSKI, *Numerical solution of a nonlinear hyperbolic equation by the Random Choice Method*, J. Comput. Phys., 30 (1979), pp. 153-166.

[10] D. S. DODSON, *Optimal order approximation by polynomial spline functions*, Ph.D. thesis, Purdue University, West Layfette, IN, 1972.

[11] E. A. DORFI AND L. O'C. DRURY, *Simple adaptive grids for 1-D initial value problems*, J. Comput. Phys., 69 (1987), pp. 175-195.

[12] H. A. DWYER, R. J. KEE, AND B. R. SANDERS, *Adaptive grid method for problems in fluid mechanics and heat transfer*, AIAA J., 18 (1980), pp. 1205-1212.

[13] R. M. FURZELAND, *The construction of adaptive space meshes for the discretization of ordinary and partial differential equations*, TNER. 85.022, Thornton Research Centre, the Netherlands, 1985.

[14] R. M. FURZELAND, J. G. VERWER, AND P. A. ZEGELING, *A numerical study of three moving grid methods for one-dimensional partial differential equations which are based on the method of lines*, J. Comput. Phys., 89 (1990), pp. 349-388.

[15] R. J. GELINAS, S. K. DOSS, AND K. MILLER, *The moving finite element method: Application to general partial differential equations with multiple large gradients*, J. Comput. Phys., 40 (1981), pp. 202-249.

[16] B. M. HERBST, A. R. MITCHELL, AND S. W. SCHOOMBIE, *A moving Petrov-Galerkin method for transport equations*, Internat. J. Numer. Methods Engrg., 18 (1982), pp. 1321-1336.

[17] B. M. HERBST, S. W. SCHOOMBIE, AND A. R. MITCHELL, *Equidistributing principles in moving finite element methods*, J. Comput. Appl. Math., 9 (1983), pp. 377-389.

[18] A. C. HINDMARSH, LSODE *and* LSODI, *two new initial value ordinary differential equation solvers*, ACM SIGNUM Newsletter, 15 (1980), pp. 10-11.

[19] J. M. HYMAN, *Adaptive moving mesh methods for partial differential equations*, in Advances in Reactor Computations, American Nuclear Society Press, La Grange Park, IL, 1983, pp. 24-43.

[20] J. M. HYMAN AND B. LARROUTUROU, *Dynamic rezone methods for partial differential equations in one space dimension*, Los Alamos Report LA-UR-86-1678, Los Alamos National Laboratory, Los Alamos, NM, 1986.

[21] J. M. HYMAN AND M. J. NAUGHTON, *Static rezone methods for tensor-product grids*, in Proc. SIAM-AMS Conference on Large Scale Computations in Fluid Mechanics, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1984.

[22] I. W. JOHNSON, A. J. WATHEN, AND M. J. BAINES, *Moving finite element methods for evolutionary problems*, II. *Applications*, J. Comput. Phys., 79 (1988), pp. 270-297.

[23] B. LARROUTUROU, *A conservative adaptive method for flame propagation*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 742-755.

[24] P. D. LAX, *Hyperbolic Systems of Conservation Laws and Mathematical Theory of Shock Waves*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1973.

[25] R. J. LEVEQUE AND H. C. YEE, *A study of numerical methods for hyperbolic conservation laws with stiff source terms*, Report 100075, NASA Ames Research Center, Moffett Field, CA, 1988.

[26] N. K. MADSEN, MOLAG: *A method of lines adaptive grid interface for nonlinear partial differential equations*, in PDE Software: Modules, Interfaces and Systems, B. Engquist and T. Smedsaas, eds., North Holland, Amsterdam, 1984.

[27] K. MATSUNO AND H. A. DWYER, *Adaptive methods for elliptic grid generation*, J. Comput. Phys., 77 (1988), pp. 40-52.

[28] K. MILLER, *Moving finite elements* II, SIAM J. Numer. Anal., 18 (1981), pp. 1033-1057.

[29] ———, *Alternate modes to control the nodes in the moving finite element method*, in Adaptive Computational Methods for Partial Differential Equations, I. Babuška, J. Chandra, and J. E. Flaherty, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.

[30] K. MILLER AND R. N. MILLER, *Moving finite elements* I, SIAM J. Numer. Anal., 18 (1981), pp. 1019-1032.

[31] M. C. Mosher, *A variable node finite element method*, J. Comput. Phys., 57 (1985), pp. 157-187.

[32] L. R. Petzold, *Observations on an adaptive moving grid method for one-dimensional systems of partial differential equations*, Appl. Numer. Math., 3 (1987), pp. 347-360.

[33] Y. Ren and R. D. Russell, *A study of moving mesh methods*, in preparation.

[34] R. D. Russell, *Mesh selection methods*, in Codes for Boundary Value Problems, Lecture Notes in Computer Science 74, B. Childs et al., eds., Springer-Verlag, Berlin, 1979.

[35] R. D. Russell and J. Christiansen, *Adaptive mesh selection strategies for solving boundary value problems*, SIAM J. Numer. Anal., 15 (1978), pp. 59-80.

[36] M. D. Smooke, *Solution of burner-stabilized pre-mixed laminar flames by boundary value methods*, J. Comput. Phys., 48 (1982), pp. 72-105.

[37] M. D. Smooke and M. L. Koszykowski, *Fully adaptive solutions of one-dimensional mixed initial-boundary value problems with applications to unstable problems in combustion*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 301-329.

[38] R. Thrasher and K. Sepehrnoori, *On equidistributing principles in moving finite element methods*, J. Comp. Appl. Math., 16 (1986), pp. 309-318.

[39] J. G. Verwer, J. G. Blom, R. M. Furzeland, and P. A. Zegeling, *A moving-grid method for one-dimensional* PDEs *based on the method of lines*, in Adaptive Methods for Partial Differential Equations, J. E. Flaherty, P. J. Paslow, M. S. Shephard, and J. D. Vasilakis, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989.

[40] G. B. Whitham, *Linear and Nonlinear Waves*, John Wiley, New York, 1974.

[41] A. B. White, Jr., *On selection of equidistributing meshes for two-point boundary-value problems*, SIAM J. Numer. Anal., 16 (1979), pp. 472-502.

[42] ———, *On the numerical solution of initial/boundary-value problems in one space dimension*, SIAM J. Numer. Anal., 19 (1982), pp. 683-697.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 6, pp. 1287-1313, November 1992

002

FINITE DIFFERENCE SCHEMES ON TRIANGULAR CELL-CENTERED GRIDS WITH LOCAL REFINEMENT*

P. S. VASSILEVSKI†, S. I. PETROVA‡, AND R. D. LAZAROV‡

Abstract. Based on approximation of the balance equation, finite difference schemes on triangular cell-centered grids are derived. A priori estimates and error analyses are provided. For certain regular triangulations, $\mathbb{O}(h^2)$ convergence in the discrete $\mathbb{H}^1$-norm is established. Also, finite difference schemes on triangular cell-centered grids with local refinement are derived with accuracy $\mathbb{O}(h^{1/2+\alpha})$, where $\alpha=0$ for a simple symmetric scheme, $\alpha=1$ for a nonsymmetric and a more accurate symmetric scheme, and $\alpha=\frac{3}{2}$ for a more accurate nonsymmetric scheme.

Certain algebraic properties of the corresponding matrices of the derived finite difference schemes are verified, thus allowing the recently proposed algebraic theory for the Bramble-Ewing-Pasciak-Schatz (BEPS) and Fact Adaptive Composite (FAC) two-grid preconditioners to apply.

Numerical experiments that demonstrate the accuracy of the difference schemes and the fast convergence of the two-grid BEPS and FAC preconditioners in conjugate gradient-type iterative methods are presented.

Key words. triangular grids, cell-centered grids, local refinement, two-grid preconditioner, conjugate gradients, finite volume method, elliptic problems

AMS(MOS) subject classifications. 65N05, 65N15, 65F10, 65N20, 65N30

1. Introduction. The finite volume method (also called the control volume method or the balance method) has been used in many applications as a systematic approach to effective discretization of fluid flow equations (cf., e.g., Patankar and Spalding [18]). Pioneering work in this area for one-dimensional elliptic and parabolic equations with piecewise smooth coefficients has been done by Tikhonov and Samarskii [23] and Samarskii [20], [21]. Recently this approach was augmented by new techniques and results by Kreiss, Manteuffel, and White [11]; Manteuffel and White [13]; and Weiser and Wheeler [25]. An important feature of this approach is that the corresponding discretizations are based on approximation of the balance equation and therefore they satisfy exactly the discrete conservation law (of mass, heat, momentum, etc.). A key aspect here is the choice of finite volumes (control volumes). In this context, we distinguish two principal ways of choosing these volumes: by vertex-centered and cell-centered grids.

In the context of vertex-centered grids (which are closely related to the finite element discretization), finite difference schemes are studied by Hackbusch [9]; Bank and Rose [3]; Samarskii, Lazarov, and Makarov [22]; and Heinrich [10], where the basic theory of the stability and convergence analysis is developed. Such schemes on vertex-centered grids with local refinement were proposed by McCormick [14] and Cai and McCormick [5], [6] for the diffusion equation; see also the recent paper by Cai, Mandel, and McCormick [6].

Convergence analysis of the difference schemes for elliptic equations on rectangular cell-centered grids has been presented by Weiser and Wheeler in [25], where the relation of the constructed schemes with mixed finite element discretization is used.

* Received by the editors May 14, 1990; accepted for publication (in revised form) October 22, 1991. This research was partially supported by Bulgarian Committee for Science grant 55-26-3-87 and by National Science Foundation grant INT-8914472.

† Center for Computer Science and Technology, Bulgarian Academy of Sciences, "Acad. G. Bontchev" Str., Block 25 A, 1113 Sofia, Bulgaria. Present address, Department of Mathematics, University of California, 405 Hilgard Ave., Los Angeles, California, 90024-1555.

‡ Institute of Mathematics, Bulgarian Academy of Sciences, "Acad. G. Bontchev" Str., Block 8, 1113 Sofia, Bulgaria.

The finite volume method based on cell-centered grids has been used by Pedrosa [19] for efficient computation of fluid flow in porous media, and by Ewing, Lazarov, and Vassilevski [7], [8] for elliptic equations on grids with local refinement.

The approximations on cell-centered and vertex-centered grids actually coincide on uniform rectangular partitionings. Essential differences in these two approximations appear when using partitioning of the domain into triangles. Then the finite volumes of the cell-centered grid are, by definition, the triangles of the partitioning, while those of the vertex-centered grid are the Voronoi volumes [26].

In this paper, we investigate approximations of second-order elliptic equations on model cell-centered grids with local refinement introduced by uniform triangulation of the domain. The uniformity of the triangulation allows us to derive error estimates of superconvergence type (see below). The results can be considered as an extension of the results of [7] and [8], where similar problems were considered on rectangular grids with local refinement.

Combining both discretizations, on triangular and rectangular finite volumes, we can cover a wide range of applications, including elliptic problems on general polygonal domains. The results presented here can also be extended with minor modifications to curvilinear triangles near the boundary of the domain.

The difference schemes we develop are constructed by appealing to the balance equation, which is valid for a general class of finite volumes. The first step in this construction is to partition the region into a number of triangles. We assume that they have angles less than $\pi/2$. (Such grids can be used in combination with rectangular grids, for example, when the original region can be partitioned into a number of rectangles and triangles.)

The grid points of the cell-centered grid are the centers of the circumscribed circle of every triangle. Note that they are internal nodes with respect to the triangles and any pair of two neighboring nodes is on the line perpendicular to the common edge of the corresponding triangles. Thus the derivative of a function in a direction normal to such an edge can be approximated using the difference of the values of the function at these nodes. Such approximations are required when we use the balance equations. The approximations on grids with local refinement are derived similarly. In this case, however, we must use certain interpolations of values of the grid functions near the interface boundary between refined and nonrefined triangles. Depending on this approximation, we derive simple symmetric and nonsymmetric schemes, as well as more accurate ones. These give rise to symmetric or nonsymmetric linear algebraic problems, correspondingly.

Requiring certain regularity of the triangulation, a priori estimates are established and corresponding error analysis is provided. In particular, we show $\mathbb{O}(h^2)$ convergence of the schemes in the discrete $\mathbb{H}^1$-norm for schemes without local refinement. For grids with local refinement, we prove a convergence rate in the discrete $\mathbb{H}^1$-norm of $\mathbb{O}(h^{1/2})$, $\mathbb{O}(h^{3/2})$, *or* $\mathbb{O}(h^2)$, depending upon the interpolation used (piecewise constant, linear, or quadratic).

The second question studied in the paper is related to the construction of optimal preconditioners for solving the resulting linear algebraic systems of equations (which can be symmetric or nonsymmetric) on the composite grid, that is, on the grid with local refinement. We verify the necessary algebraic properties of the composite grid matrix and apply the algebraic theory for the BEPS and FAC preconditioners proposed in Ewing, Lazarov, and Vassilevski [7].

Originally, the FAC method was proposed by McCormick [14], [15] and McCormick and Thomas [16], and the BEPS technique by Bramble, Ewing, Pasciak, and Schatz [4]. In Mandel and McCormick [12], these two methods were compared in the case of finite element discretization with local refinement, noting that the BEPS method can be viewed as a symmetrization of the FAC preconditioner. The theory in [7] is based on the strengthened Cauchy inequality, which, in our case of finite difference schemes, need not necessarily exist. None of these theories applies to the case of nonsymmetric matrices.

The main result from Ewing, Lazarov, and Vassilevski [7] is valid here also, namely, that the two-grid BEPS and FAC preconditioners B are spectrally equivalent to the composite grid matrix A. In the nonsymmetric case, this is understood as follows: the spectrum of $B^{-1}A$ is contained in a fixed subset of a disc in the right-half complex plane independent of the grid parameter h. This implies that the generalized conjugant gradient (GCG)-type methods, such as generalized minimum residual (GMRES) from Saad and Schultz [27] or the GCG least squares (LS) method from Axelsson [1], [2], have a rate of convergence independent of the grid parameter h. We point out that throughout this paper the so-called aspect ratio, h_c/h_f, i.e., the coarse-grid parameter h_c over the fine-grid parameter h_f, is assumed to be bounded.

The remainder of the paper is organized as follows. In § 2, we formulate the problem and derive approximations of the balance equation on regular grids and on grids with local refinement. In § 3, we state the discrete problem and verify certain algebraic properties of the finite difference matrices on the composite grid. In § 4, we provide the error analysis for the derived finite difference schemes on uniform grids and on grids with local refinement. Finally, in § 5, we formulate the BEPS and the FAC preconditioners and develop their spectral properties, while in § 6, a number of test examples are presented.

2. Approximations on regular cell-centered grids and on cell-centered grids with local refinement.

We consider the following differential equation in a polygonal region Ω. Find u such that

$$-\operatorname{div}(a\nabla u)=f(x), \qquad x\in\Omega, \tag{2.1}$$

with the following mixed boundary conditions

$$u(x)=g(x) \quad \text{on } \Gamma_D, \tag{2.2}$$

$$W_\nu \equiv -\nu\cdot a\nabla u=0, \qquad x\in\Gamma_N=\partial\Omega\backslash\Gamma_D, \tag{2.3}$$

where ν is the unit outward normal vector on $\partial\Omega$. The coefficient function $a(x)$ should be piecewise smooth (or piecewise continuous) and bounded away from the origin and from above; that is, for some constants a_1, a_2, we have that

$$0<a_1\leqq a(x)\leqq a_2, \qquad x\in\Omega. \tag{2.4}$$

We cover the region Ω by a set of triangles $\tilde{\tau}=\{T\}$ such that if two triangles are interesecting, they have either only a common edge or only a common vertex. For every finite volume $T\subset\tilde{\tau}$, the following balance equation is valid:

$$\int_{\partial T} W_\nu\, ds=\int_T f(\xi)\, d\xi. \tag{2.5}$$

Let T be the triangle shown in Fig. 2.1 with edges s_1, s_2, and s_3. Then (2.5) reads

$$\int_{s_1} W_{\nu_1}\, ds_1+\int_{s_2} W_{\nu_2}\, ds_2+\int_{s_3} W_{\nu_3}\, ds_3=\int_T f(\xi)\, d\xi, \tag{2.6}$$

where ν_l are outward unit normal vectors on s_l, $l=1, 2, 3$.

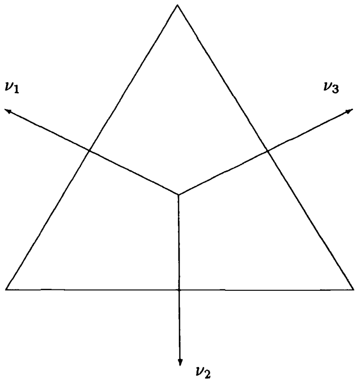


FIG. 2.1. *Triangle* $T \in \tau$.

Consider now the following model situation. We assume that all triangles in τ have no angles greater than or equal to $\pi/2$. Let the grid $\tilde{\omega}$ consist of the centers of the circumscribed circles of all the triangles $T \in \tilde{\tau}$. Here we consider only uniform partitionings of Ω, by which we mean that every two neighboring triangles form a parallelogram. We asume further that the triangles $T \in \tilde{\tau}$ have sides parallel to three fixed directions, or equivalently, the normal vectors to their edges are three fixed vectors ν_1, ν_2, and ν_3. Then we can introduce a coordinate system using ν_1 and ν_3 as coordinate axes. See Fig. 2.2 as an example of a region Ω covered by geometrically equal triangles.

The finite difference schemes are obtained by approximations of the fluxes W_{ν_l} across the edges s_l, $l = 1$, 2, 3, for every triangle $T \in \tilde{\tau}$. For every center x of the circumscribed circle of a triangle $T = T(x) \in \tau$, we denote by $w_l(x)$ some approximation to

$$\int_{s_l} W_{\nu_l}\, ds_l, \quad \text{i.e.,} \quad w_l(x) \approx \int_{s_l} W_{\nu_l}\, ds_l, \qquad l = 1, 2, 3.$$

These approximations can be derived in the following way. For definiteness we consider the four neighboring nodes $x_{i,j}$, $x_{i+1,j}$, $x_{i-1,j-1}$, and $x_{i,j+1}$ as shown in Fig. 2.2. Since $a(x) > 0$, then

$$\frac{\partial u}{\partial \nu_l} = -\frac{W_{\nu_l}(x)}{a(x)}, \qquad l = 1, 2, 3.$$

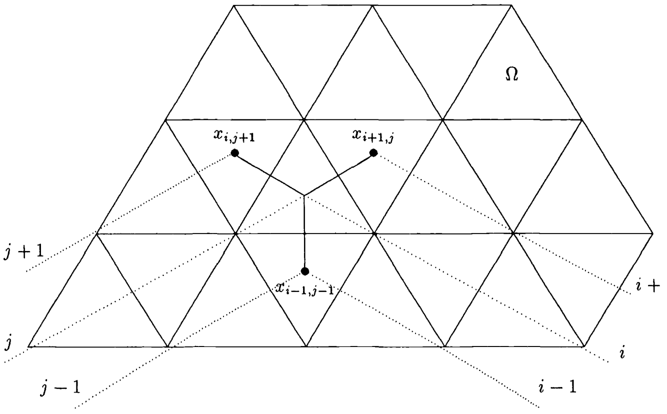


FIG. 2.2. *Region* Ω.

Then integrating for $l=1$ over the segment $(x_{i,j}, x_{i,j+1})$ along the vector ν_1 we get

$$u(x_{i,j+1})-u(x_{i,j})=-\int_{x_{i,j}}^{x_{i,j+1}} \frac{W_{\nu_1}(s)}{a(s)}\,ds \approx -W_{\nu_1}(x_{i,j+1/2})\int_{x_{i,j}}^{x_{i,j+1}} \frac{ds}{a(s)},$$

where $x_{i,j+1/2}$ is the point of intersection of s_1 and the segment $(x_{i,j}x_{i,j+1})$. For uniform partitioning this is the midpoint of the segment $(x_{i,j}x_{i,j+1})$. Thus we get the following approximation

$$W_{\nu_1}(x_{i,j+1/2}) \approx -(u(x_{i,j+1})-u(x_{i,j}))\bigg/\int_{x_{i,j}}^{x_{i,j+1}} \frac{ds}{a(s)}. \tag{2.7}$$

Then the integral $\int_{s_1} W_{\nu_1}\,ds$ can be approximated as follows:

$$\begin{aligned}\int_{s_1} W_{\nu_1}(s)\,ds &\approx W_{\nu_1}(x_{i,j+1/2}) \text{ meas } (s_1)\\ &\approx -\frac{\text{meas } (s_1)}{h_1}(u(x_{i,j+1})-u(x_{i,j}))\bigg/\frac{1}{h_1}\int_{x_{i,j}}^{x_{i,+1}} \frac{ds}{a(s)}\end{aligned} \tag{2.8}$$

where

$$\text{meas } (\Delta)=\int_{\Delta} ds.$$

We can approximate the other integrals similarly.

Let $y(x)=y_{i,j}$ be a grid function that approximates $u=u(x)$ for $x\in\tilde{\omega}$. According to (2.8), we define

$$\begin{aligned} w_1(x)&=\frac{\text{meas } (s_1)}{h_1}(y_{i,j}-y_{i,j+1})\bigg/\frac{1}{h_1}\int_{x_{i,j}}^{x_{i,j+1}}\frac{ds}{a(s)}\equiv k_1(x)\Delta_1 y(x),\\ w_2(x)&=\frac{\text{meas } (s_2)}{h_2}(y_{i,j}-y_{i-1,j-1})\bigg/\frac{1}{h_2}\int_{x_{i,j}}^{x_{i-1,j-1}}\frac{ds}{a(s)}\equiv k_2(x)\Delta_2 y(x),\\ w_3(x)&=\frac{\text{meas } (s_3)}{h_3}(y_{i,j}-y_{i+1,j})\bigg/\frac{1}{h_3}\int_{x_{i,j}}^{x_{i+1,j}}\frac{ds}{a(s)}\equiv k_3(x)\Delta_3 y(x), \end{aligned} \tag{2.9}$$

where

$$\begin{aligned} k_1(x)&=\frac{\text{meas } (s_1)}{h_1}\left(\frac{1}{h_1}\int_{x_{i,j}}^{x_{i,j+1}}\frac{ds}{a(s)}\right)^{-1},\\ k_2(x)&=\frac{\text{meas } (s_2)}{h_2}\left[\frac{1}{h_2}\int_{x_{i,j}}^{x_{i-1,j-1}}\frac{ds}{a(s)}\right]^{-1},\\ k_3(x)&=\frac{\text{meas } (s_3)}{h_3}\left[\frac{1}{h_3}\int_{x_{i,j}}^{x_{i+1,j}}\frac{ds}{a(s)}\right]^{-1}. \end{aligned} \tag{2.10}$$

For $l=1, 2, 3$, we define

$$\Delta_l y(x)=y(x)-y(x+\varepsilon h_l\nu_l)=-\Delta_l y(x+\varepsilon h_l\nu_l), \tag{2.11}$$

where h_l is the mesh size in the ν_l direction so that $x+\varepsilon h_l\nu_l$, $\varepsilon=\pm 1$ are the neighboring grid points in the same direction ($l=1, 2, 3$).

In this way, we get the following cell-centered finite difference scheme

$$w_1(x)+w_2(x)+w_3(x)=\phi(x)\equiv\int_{T(x)} f(\xi)\,d\xi \tag{2.12}$$

for all $T = T(x) \in \tau$ and the boundary conditions (2.3) for $w_2(x)$, $w_3(x)$ on $\partial\Omega\backslash\Gamma_D$ (see Fig. 2.3) and $y(x) = g(x)$, $x \in \Gamma_D$.

Consider approximation (2.12) at the grid points near the boundary Γ_D. Note that the triangles along Γ_D are aligned so that the centers of their circumscribed circles lie on Γ_D. For triangles about interior points that intersect Γ_D, we can apply the balance equation by supposing that the solution $u(x)$ is extended continuously in the whole triangle T (see Fig. 2.3).

Now consider the case with local refinement, where some of the triangles of $\tilde{\tau}$ are refined into a number of congruent triangles. This new triangulation we denote by τ. The circumcenters of all triangles form the new grid ω, called the composite grid.

The approximation of the balance equation written for every triangle $T \in \tau$ is derived as above if T has four neighbors. For an interface triangle T, as shown in Fig. 2.4, we introduce the slave nodes (denoted by the superscript*), which are centers of the circumscribed circles of the corresponding triangles obtained if T were to be refined into congruent triangles. Now the difference scheme is derived as in the regular case. However, the slave nodes are not grid points, so the values of the grid function at these nodes must be obtained by interpolation. In order to simplify the exposition, we suppose that the refinement process is organized so that any nonrefined cell has at most one refined adjacent cell.

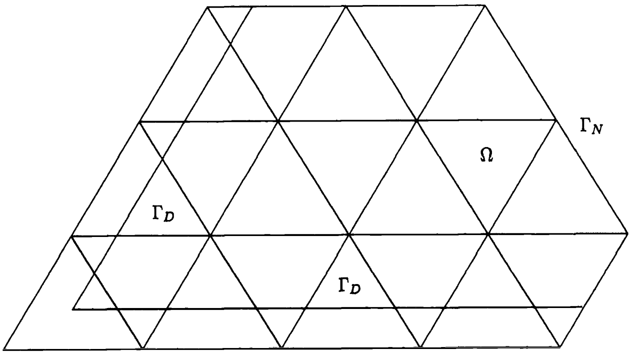


FIG. 2.3

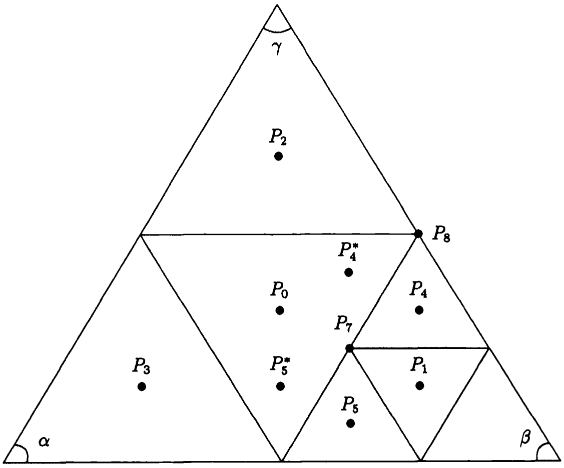


FIG. 2.4. *Interface triangle $T = T(x)$ with angles α, β, and γ.*

For definiteness, consider the case shown in Fig. 2.4. Let $y = y(x)$ be a grid function defined at the composite grid nodes $P_0, P_1, \cdots, P_5$. In order to derive the finite difference scheme on the composite grid, we need values of the grid function y at the points P_4^* and P_5^*. Let $p = p(x)$ be a polynomial that interpolates $y = y(x)$ at some of the nodes $P_0, P_1, \cdots, P_5$. For $x = P_4^*$ and P_5^* we set $y(x) = p(x)$. Then the relation

$$\int_s W_{\nu_1}\,ds = \int_{s_1} W_{\nu_1}\,ds + \int_{s_2} W_{\nu_1}\,ds \tag{2.13}$$

is approximated by

$$\begin{aligned} w_1(P_0) &= -k_1(P_4)(y(P_4) - y(P_4^*)) - k_1(P_5)(y(P_5) - y(P_5^*)) \\ &= -\frac{\text{meas}\,(s_1)}{\bar{h}}(y(P_4) - y(P_4^*)) \bigg/ \frac{1}{\bar{h}} \int_{P_4}^{P_4^*} \frac{ds}{a(s)} \\ &\quad -\frac{\text{meas}\,(s_2)}{\bar{h}}(y(P_5) - y(P_5^*)) \bigg/ \frac{1}{\bar{h}} \int_{P_5}^{P_5^*} \frac{ds}{a(s)}, \end{aligned} \tag{2.14}$$

where $\bar{h} = \text{dist}\,(P_4, P_4^*) = \text{dist}\,(P_5, P_5^*)$, i.e., $\bar{h}$ is the distance between the corresponding points.

We distinguish the following four cases for defining the polynomial $p = p(x)$.

2.1. Simple symmetric approximation. This corresponds to the choice of piecewise constant interpolation, i.e., $p = p(x) = y(P_0)$. Then

$$y(P_4^*) = y(P_5^*) = y(P_0),$$

and we get (for the particular case of Fig. 2.4)

$$w_1(P_4) = k_1(P_4)[y(P_4) - y(P_0)] = k_1(P_4)\Delta_1 y(P_4). \tag{2.15}$$

From this definition of the flux $w_1(P_4)$, we can see that the matrix corresponding to the finite difference scheme (2.12) is symmetric (the coefficient in front of $y(P_0)$ in the equation corresponding to the point P_4 equals the coefficient in front of $y(P_4)$ in the equation for P_0).

2.2. Simple nonsymmetric approximation. This scheme corresponds to piecewise linear interpolation, i.e., $p = p(x) = c_0 + c_1x_1 + c_2x_2$, $p(P_0) = y(P_0)$, $p(P_2) = y(P_2)$, and $p(P_3) = y(P_3)$. Then

$$y(P_4^*) = p(P_4^*), \qquad y(P_5^*) = p(P_5^*),$$

and

$$\begin{aligned} w_1(P_4) &= k_1(P_4)[y(P_4) - \tfrac{1}{4}(1 - \tan\gamma \text{ cotan } \beta)y(P_2) + \tfrac{1}{4}(1 + \tan\alpha \text{ cotan } \beta)y(P_3) \\ &\qquad - \tfrac{1}{4}((\tan\alpha + \tan\gamma) \text{ cotan } \beta + 4)y(P_0)] \\ &= k_1(P_4)[\Delta_1 y(P_4) + \tfrac{1}{4}(1 - \tan\gamma \text{ cotan } \beta)\Delta_2 y(P_0) \\ &\qquad - \tfrac{1}{4}(1 + \tan\alpha \text{ cotan } \beta)\Delta_3 y(P_0)]. \end{aligned} \tag{2.16}$$

For the case of equilateral triangles we have

$$\begin{aligned} w_1(P_4) &= k_1(P_4)[y(P_4) + \tfrac{1}{2}y(P_3) - \tfrac{3}{2}y(P_0)] \\ &= k_1(P_4)[\Delta_1 y(P_4) - \tfrac{1}{2}\Delta_3 y(P_0)]. \end{aligned} \tag{2.17}$$

Obviously, in the finite difference equation for the point P_4, the unknown $y(P_2)$, for example, enters with nonzero coefficient, while $y(P_4)$ is not included in the equation corresponding to the point P_2. This means that the finite difference matrix is not symmetric.

2.3. More accurate symmetric approximation. In this case, we again use the linear polynomial $p=p(x)=c_0+c_1x_1+c_2x_2$, but p now interpolates y using the nodes P_0, P_4, and P_5, that is, $p(P_0)=y(P_0)$, $p(P_4)=y(P_4)$, and $p(P_5)=y(P_5)$. Then

$$y(P_4^*)=p(P_4^*), \qquad y(P_5^*)=p(P_5^*),$$

and

$$\begin{aligned} w_1(P_4) &= \tfrac{1}{3}k_1(P_4)[-2y(P_0)+y(P_4)+y(P_5)] \\ &= \tfrac{1}{3}k_1(P_4)[\Delta_1 y(P_4)+\Delta_1 y(P_5)]. \end{aligned} \tag{2.18}$$

Here symmetry of the matrix can be established by inspection.

2.4. More accurate nonsymmetric approximation. Here we use the quadratic polynomial

$$p=p(x)=c_0+c_1x_1+c_2x_2+c_3x_1^2+c_4x_1x_2+c_5x_2^2$$

and interpolate $y=y(x)$ using the nodes P_0, P_1, P_2, P_3, P_4, and P_5. Then

$$y(P_4^*)=p(P_4^*), \qquad y(P_5^*)=p(P_5^*),$$

and when $\alpha=\beta=\gamma$, we get

$$\begin{aligned} w_1(P_4) &= \tfrac{1}{30}k_1(P_4)[-15y(P_0)+15y(P_1)-3y(P_2)+3y(P_3)+6y(P_4)-6y(P_5)] \\ &= \tfrac{1}{60}k_1(P_4)[6\Delta_2 y(P_0)-6\Delta_3 y(P_0)-15\Delta_2 y(P_4)-15\Delta_3 y(P_5) \\ &\qquad +27\Delta_1 y(P_4)+3\Delta_1 y(P_5)]. \end{aligned} \tag{2.19}$$

3. Formulation of the discrete problems and properties of the corresponding cell-centered composite grid matrices. All finite difference schemes derived in § 2 can be written in the general form

$$w_1(x)+w_2(x)+w_3(x)=\phi(x)\equiv\int_T f(\xi)\,d\xi, \quad T=T(x), \quad x\in\omega, \tag{3.1}$$

and

$$y=g(x) \quad \text{for } x\in\gamma_D, \quad w_\nu \text{ given for } x \text{ near } \partial\Omega\backslash\Gamma_D,$$

where $y=y(x)$ is the approximate solution at the grid nodes $x\in\omega$ and $w_l(x)$, $l=1, 2, 3$, are approximate fluxes defined accordingly by (2.9).

Since for every two regular neighboring triangles (see, e.g., Fig. 3.1.), we have $k_1(x)=k_1(x_+)$, then

$$w_1(x)=-w_1(x_+). \tag{3.2}$$

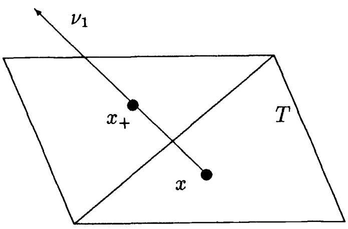


FIG. 3.1

For a regular grid without local refinement, we have the following identity, which is valid for every grid function $z(x)$ such that $z=0$ on γ_D.

$$\begin{aligned}&\sum_{x\in\omega}\{w_1(x)+w_2(x)+w_3(x)\}z(x)\\ &\quad=\sum_{x\in\omega}\{\alpha_1(x)w_1(x)\Delta_1 z(x)+\alpha_2(x)w_2(x)\Delta_2 z(x)+\alpha_3(x)w_3(x)\Delta_3 z(x)\},\end{aligned}\tag{3.3}$$

where, for example,

$$\Delta_1 z(x)=z(x)-z(x_+),$$

$$\alpha_1(x)=\begin{cases}\frac{1}{2} & \text{for } x\text{-internal node,}\\ 1 & \text{for } x \text{ near } \gamma_D,\end{cases}\qquad l=1,2,3.$$

The equation (3.3) is in accordance with (3.2) since, for example, in the ν_1-direction for the first two terms of the sum in the left-hand side of (3.3), we have

$$\begin{aligned}&w_1(x)z(x)+w_1(x_+)z(x_+)\\ &\quad=\tfrac{1}{2}w_1(x)z(x)-\tfrac{1}{2}w_1(x_+)z(x)+\tfrac{1}{2}w_1(x_+)z(x_+)-\tfrac{1}{2}w_1(x)z(x_+)\\ &\quad=\tfrac{1}{2}w_1(x)z(x)-\tfrac{1}{2}w_1(x_+)z(x)-\tfrac{1}{2}w_1(x)z(x_+)+\tfrac{1}{2}w_1(x_+)z(x_+)\\ &\quad=\tfrac{1}{2}w_1(x)\Delta_1 z(x)+\tfrac{1}{2}w_1(x_+)\Delta_1 z(x_+).\end{aligned}$$

In the case of a regular grid, since $w_1(x)=k_1(x)\Delta_1 y(x)$, $l=1$, 2, 3, we have

$$\begin{aligned}&\sum_{x\in\omega}\{w_1(x)+w_2(x)+w_3(x)\}z(x)\\ &\quad=\sum_{x\in\omega}\{\tilde{k}_1(x)\Delta_1 y(x)\Delta_1 z(x)+\tilde{k}_2(x)\Delta_2 y(x)\Delta_2 z(x)\\ &\qquad+\tilde{k}_3(x)\Delta_3 y(x)\Delta_3 z(x)\},\end{aligned}\tag{3.4}$$

where $\tilde{k}_l(x)=\alpha_l(x)k_l(x)$ and $k_l(x)$ are defined by (2.10).

Writing the difference scheme (3.1) as a system of linear algebraic equations

$$Ay=\phi,\tag{3.5}$$

then

$$z^tAy=\sum_{x\in\omega}\sum_{l=1}^{3}\tilde{k}_l(x)\Delta_l y(x)\Delta_l z(x).\tag{3.6}$$

Obviously, (3.6) represents a symmetric and positive definite form, hence the matrix A is invertible. In fact, A is an M-matrix (see, e.g., Varga [24]): it is weakly diagonally dominant with nonpositive off-diagonal entries.

Consider the case of the composite grid, where we prove that a similar formula is valid. Denote

$$z^tAy=\sum_{l=1}^{3}\sum_{x\in\omega}w_l(x)z(x)=\sum_{l=1}^{3}I_l.\tag{3.7}$$

Using (3.4) for every subregion of Ω with regular grid points (see Fig. 3.2, where Ω_2 is a subregion of Ω covered by a refined grid ω_2), we present the term I_1 in the following

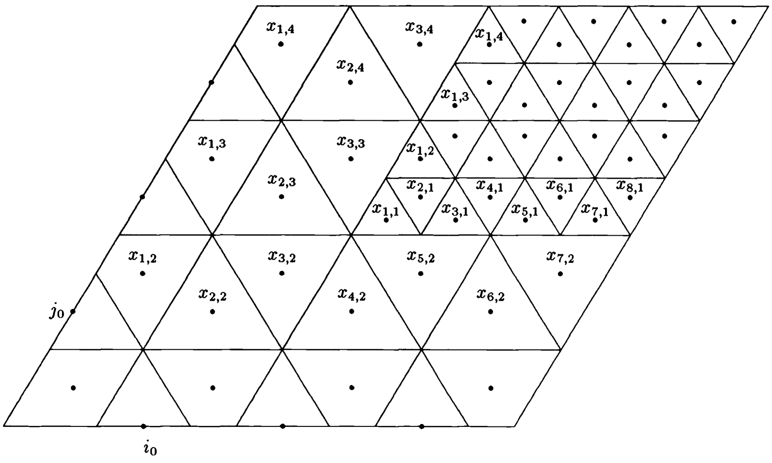


FIG. 3.2. *Composite grid* $(i_0=2, j_0=2)$.

form:

$$I_1=\sum_{\substack{x\in\tilde\omega\\ j\leq j_0}}\alpha_1(x_{i,j})w_1(x_{i,j})\Delta_1 z(x_{i,j})+\sum_{\substack{x\in\tilde\omega\\ j>j_0,i<2i_0-1}}\alpha_1(x_{i,j})w_1(x_{i,j})\Delta_1 z(x_{i,j})$$
$$+\sum_{\substack{x\in\tilde\omega\\ j>j_0}}w_1(x_{2i_0-1,j})z(x_{2i_0-1,j})$$
$$+\sum_{\substack{x\in\omega_2\\ i>1}}\alpha_1(x_{i,j})w_1(x_{i,j})\Delta_1 z(x_{i,j})+\sum_{\substack{x\in\omega_2\\ j\geq 1}}w_1(x_{1,j})z(x_{1,j}),$$

where, in accordance with (3.2) and the requirement for mass conservation, we have

$$w_1(x_{2i_0-1,j_0+k})=-(w_1(x_{1,2k-1})+w_1(x_{1,2k}))\quad\text{for } k=1,2,\cdots.$$

Define

$$\Delta_1 z(x_{1,2k-1})=z(x_{1,2k-1})-z(x_{2i_0-1,j_0+k}),$$
$$\Delta_1 z(x_{1,2k})=z(x_{1,2k})-z(x_{2i_0-1,j_0+k}).$$

Using these in the expression for I_1 above, we get

$$I_1=\sum_{x\in\omega\backslash S_1}\alpha_1(x)w_1(x)\Delta_1 z(x),$$

where $\alpha_1(x)=1$ for the irregular grid points in Ω_2 and $S_1=\{x=x_{i,j}, j>j_0,\ i=2i_0-1\}$ is the set of irregular coarse grid points for which $\Delta_1 z(x)$ is not defined.

Similarly, in the ν_2-direction we have

$$I_2=\sum_{\substack{x\in\tilde\omega\\ 1\leq i\leq 2i_0-1\\ j\geq j_0}}\alpha_2(x_{i,j})w_2(x_{i,j})\Delta_2 z(x_{i,j})+\sum_{\substack{x\in\tilde\omega\\ j<j_0}}\alpha_2(x_{i,j})w_2(x_{i,j})\Delta_2 z(x_{i,j})$$
$$+\sum_{\substack{x\in\tilde\omega\\ k\geq 1}}w_2(x_{2(i_0+k)-1,j_0})z(x_{2(i_0+k)-1,j_0})+\sum_{\substack{x\in\omega_2\\ j\geq 1}}\alpha_2(x_{i,j})w_2(x_{i,j})\Delta_2 z(x_{i,j})$$
$$+\sum_{\substack{x\in\omega_2\\ k\geq 1}}\alpha_2(x_{2k,1})w_2(x_{2k,1})\Delta_2 z(x_{2k,1})+\sum_{\substack{x\in\omega_2\\ k\geq 1\\ i=2k-1}}w_2(x_{i,1})z(x_{i,1}),$$

where

$$w_2(x_{2(i_0+k)-1,j_0}) = -(w_2(x_{4k-3,1}) + w_2(x_{4k-1,1})) \quad \text{for } k = 1, 2, \cdots .$$

Define

$$\Delta_2 z(x_{4k-3,1}) = z(x_{4k-3,1}) - z(x_{2(i_0+k)-1,j_0}),$$
$$\Delta_2 z(x_{4k-1,1}) = z(x_{4k-1,1}) - z(x_{2(i_0+k)-1,j_0}).$$

Then

$$I_2 = \sum_{x \in \omega \backslash S_2} \alpha_2(x) w_2(x) \Delta_2 z(x),$$

where $\alpha_2(x) = 1$ for the irregular grid points in Ω_2 and $S_2 = \{x = x_{i,j},\ i = 2(i_0 + k) - 1,\ j = j_0,\ k = 1, 2, \cdots\}$ is the set of irregular coarse-grid points for which $\Delta_2 z(x)$ is not defined.

Since in the direction of the normal vector ν_3 there are no irregular grid points (see Fig. 3.2), we get $I_3 = \sum_{x \in \omega} \alpha_3(x) w_3(x) \Delta_3 z(x)$ directly from (3.4). In general, inserting these expressions for I_1, I_2, and I_3 in (3.7), we get the basic representation

$$z^t A y = \sum (\alpha_1(x) w_1(x) \Delta_1 z(x) + \alpha_2(x) w_2(x) \Delta_2 z(x) + \alpha_3(x) w_3(x) \Delta_3 z(x)). \tag{3.8}$$

In (3.8) and in all expressions for $z^t Ay$ below, the summation $\sum$ is taken to mean over those grid points $x \in \omega$, for which $\Delta_l z(x)$ is defined $(l = 1, 2, 3)$.

In order to simplify the considerations below we suppose that $a(x) \equiv 1$. First, we consider the *simple symmetric scheme* for which the fluxes are defined in (2.15). In this case we have

$$z^t A_0 y = \sum_{x \in \omega} \sum_{l=1}^{3} \alpha_l(x) w_l(x) \Delta_l z(x) = \sum_{x \in \omega} \sum_{l=1}^{3} \tilde{k}_l(x) \Delta_l y(x) \Delta_l z(x). \tag{3.9}$$

Obviously, $z^t A_0 y$ is a symmetric bilinear form and

$$y^t A_0 y = \sum_{x \in \omega} \sum_{l=1}^{3} \tilde{k}_l(x) (\Delta_l y(x))^2.$$

Here A_0 is a positive definite matrix, since $y^t A_0 y = 0$ if and only if $y(x) \equiv 0$ (we have $y(x) = 0$, $x \in \gamma_D$).

Next we consider the *simple nonsymmetric approximation* of the fluxes at the irregular grid points defined by (2.16). In the particular case of equilateral triangles, by (2.17) we have

$$\begin{aligned} z^t A y = {} & \sum_{\omega} (\tilde{k}_1 \Delta_1 y \Delta_1 z + \tilde{k}_2 \Delta_2 y \Delta_2 z + \tilde{k}_3 \Delta_3 y \Delta_3 z) \\ & - \frac{1}{2} \sum_{k \geqq 1} \Big\{ \tilde{k}_1(x_{1,2k-1}) \Delta_2 y(x_{2i_0-1,j_0+k}) \Delta_1 z(x_{1,2k-1}) \\ & \qquad + \tilde{k}_1(x_{1,2k}) \Delta_3 y(x_{2i_0-1,j_0+k}) \Delta_1 z(x_{1,2k}) \Big\} \\ & - \frac{1}{2} \sum_{k \geqq 1} \Big\{ \tilde{k}_2(x_{4k-3,1}) \Delta_1 y(x_{2(i_0+k)-1,j_0}) \Delta_2 z(x_{4k-3,1}) \\ & \qquad + \tilde{k}_2(x_{4k-1,1}) \Delta_3 y(x_{2(i_0+k)-1,j_0}) \Delta_2 z(x_{4k-1,1}) \Big\}. \end{aligned} \tag{3.10}$$

It is a nonsymmetric bilinear form and, applying the Cauchy inequality to the right-hand side of (3.10), we get

$$|z^tAy| \leqq \tfrac{3}{2}(z^tA_0z)^{1/2}(y^tA_0y)^{1/2}.$$

For $z=y$ the estimate from below is obtained in the following way: the terms of the nonsymmetric part of the corresponding bilinear form are estimated from below using the inequality $-|ab| \geqq -(a^2+b^2)/2$ and compensated with the terms from the symmetric part. Thus

$$\tfrac{3}{4}y^tA_0y \leqq y^tAy \leqq \tfrac{3}{2}y^tA_0y.$$

In the case of the *more accurate symmetric scheme*, using (2.18), the following expression for the inner product is valid

$$\begin{aligned} z^tAy &= \sum_{\omega} (\tilde{k}_1\Delta_1 y\Delta_1 z + \tilde{k}_2\Delta_2 y\Delta_2 z + \tilde{k}_3\Delta_3 y\Delta_3 z) \\ &\quad + \frac{1}{3}\sum_{k\geqq 1}\Big\{\tilde{k}_1(x_{1,2k-1})[-2\Delta_1 y(x_{1,2k-1}) + \Delta_1 y(x_{1,2k})]\Delta_1 z(x_{1,2k-1}) \\ &\qquad + \tilde{k}_1(x_{1,2k})[-2\Delta_1 y(x_{1,2k}) + \Delta_1 y(x_{1,2k-1})]\Delta_1 z(x_{1,2k})\Big\} \\ &\quad + \frac{1}{3}\sum_{k\geqq 1}\Big\{\tilde{k}_2(x_{4k-3,1})[-2\Delta_2 y(x_{4k-3,1}) + \Delta_2 y(x_{4k-1,1})]\Delta_2 z(x_{4k-3,1}) \\ &\qquad + \tilde{k}_2(x_{4k-1,1})[-2\Delta_2 y(x_{4k-1,1}) + \Delta_2 y(x_{4k-3,1})]\Delta_2 z(x_{4k-1,1})\Big\}. \end{aligned} \tag{3.11}$$

Obviously, z^tAy is a symmetric bilinear form and, by the Cauchy inequality, we easily get the upper bound

$$|z^tAy| \leqq \tfrac{5}{3}(z^tA_0z)^{1/2}(y^tA_0y)^{1/2}.$$

When $z=y$, we obtain the lower bound

$$\tfrac{1}{3}y^TA_0y \leqq y^tAy \leqq \tfrac{5}{3}y^tA_0y.$$

Hence, in both cases of linear interpolation, we get that A is invertible and that the corresponding finite difference scheme has a unique solution.

Finally, we consider the case of the *more accurate nonsymmetric approximation* of the fluxes defined by (2.18) at the irregular grid points when $\alpha=\beta=\gamma$. Then

$$\begin{aligned} z^tAy &= \sum_{\omega} (\tilde{k}_1\Delta_1 y\Delta_1 z + \tilde{k}_2\Delta_2 y\Delta_2 z + \tilde{k}_3\Delta_3 y\Delta_3 z) \\ &\quad + \frac{1}{60}\sum_{k\geqq 1}\{\tilde{k}_1(x_{1,2k-1})[-33\Delta_1 y(x_{1,2k-1}) - 15\Delta_2 y(x_{1,2k-1}) - 15\Delta_3 y(x_{1,2k-1}) \\ &\qquad + 6\Delta_3 y(x_{2i_0-1,j_0+k}) - 6\Delta_2 y(x_{2i_0-1,j_0+k}) + 3\Delta_1 y(x_{1,2k})]\Delta_1 z(x_{1,2k-1})\} \\ &\quad + \frac{1}{60}\sum_{k\geqq 1}\{\tilde{k}_1(x_{1,2k})[-33\Delta_1 y(x_{1,2k}) - 15\Delta_3 y(x_{1,2k-1}) - 15\Delta_2 y(x_{1,2k}) \\ &\qquad - 6\Delta_3 y(x_{2i_0-1,j_0+k}) + 6\Delta_2 y(x_{2i_0-1,j_0+k}) + 3\Delta_1 y(x_{1,2k-1})]\Delta_1 z(x_{1,2k})\} \\ &\quad + \frac{1}{60}\sum_{k\geqq 1}\{\tilde{k}_2(x_{4k-3,1})[-33\Delta_2 y(x_{4k-3,1}) - 15\Delta_1 y(x_{4k-1,1}) - 15\Delta_3 y(x_{4k-3,1}) \\ &\qquad + 6\Delta_3 y(x_{2(i_0+k)-1,j_0}) - 6\Delta_1 y(x_{2(i_0+k)-1,j_0}) + 3\Delta_2 y(x_{4k-1,1})]\Delta_2 z(x_{4k-3,1})\} \\ &\quad + \frac{1}{60}\sum_{k\geqq 1}\{\tilde{k}_2(x_{4k-1,1})[-33\Delta_2 y(x_{4k-1,1}) - 15\Delta_3 y(x_{4k-3,1}) - 15\Delta_1 y(x_{4k-1,1}) \\ &\qquad - 6\Delta_3 y(x_{2(i_0+k)-1,j_0}) + 6\Delta_1 y(x_{2(i_0+k)-1,j_0}) + 3\Delta_2 y(x_{4k-3,1})]\Delta_2 z(x_{4k-1,1})\}. \end{aligned} \tag{3.12}$$

Applying the Cauchy inequality to the right-hand side of (3.12), we get

$$|z^tAy| \leqq \tfrac{31}{20}(z^tA_0z)^{1/2}(y^tA_0y)^{1/2}.$$

In order to obtain a lower bound for $z=y$, we combine this inequality with a similar one and get

$$\tfrac{1}{20}y^tA_0y \leqq y^tAy \leqq \tfrac{31}{20}y^tA_0y.$$

The following theorem summarizes the results for all four approximations.

THEOREM 3.1. *The following basic inequalities are valid*:

$$|z^tAy| \leqq \gamma_2(z^tA_0z)^{1/2}(y^tA_0y)^{1/2}, \tag{3.13}$$

$$\gamma_1 y^tA_0y \leqq y^tAy \leqq \gamma_2 y^tA_0y, \tag{3.14}$$

where A_0 is the matrix of the simple symmetric approximation and A is the matrix of any of the four approximations. For the case of the simple nonsymmetric or more accurate nonsymmetric scheme, γ_1 is a positive constant only under additional assumptions for the angles α, β, and *γ (see Remark* 3.1). *If $a(x)\equiv 1$ and the partition is with equilateral triangles $T\in\tau$, the constants γ_1 and γ_2 are*: $\gamma_1=\frac{3}{4}$, $\gamma_2=\frac{3}{2}$ *for the simple nonsymmetric scheme*; $\gamma_1=\frac{1}{3}$, $\gamma_2=\frac{5}{3}$ *for the more accurate symmetric scheme*; *and* $\gamma_1=\frac{1}{20}$, $\gamma_2=\frac{31}{20}$ *for the more accurate nonsymmetric scheme.*

Remark 3.1. In the general case (see Fig. 2.4) we have

(i) for the case of the simple nonsymmetric scheme,

$$\{1-\max(|c_1|,|c_2|)\}y^tA_0y \leqq y^tAy \leqq \{1+2\max(|c_1|,|c_2|)\}y^tA_0y,$$

where

$$c_1=\frac{1}{8}\left(1\pm\frac{\tan\gamma}{\tan\beta}\right) \quad\text{and}\quad c_2=\frac{1}{8}\left(1\pm\frac{\tan\alpha}{\tan\beta}\right).$$

Obviously, $\gamma_1=1-\max(|c_1|,|c_2|)$ and $\gamma_2=1+2\max(|c_1|,|c_2|)$ depend only on the angles α, β, and γ, but γ_1 is not always positive. Therefore, the matrix A in this case is invertible only for triangles for which $\tan\gamma \operatorname{cotan}\beta<7$ or $\tan\alpha\operatorname{cotan}\beta<7$.

(ii) For the case of the more accurate symmetric scheme,

$$\tfrac{1}{3}y^tA_0y \leqq y^tAy \leqq \tfrac{5}{3}y^tA_0y.$$

We get the same constants $\gamma_1=\frac{1}{3}$ and $\gamma_2=\frac{5}{3}$ as in our particular case. Hence the matrix A is always invertible, so that the more accurate symmetric scheme, in general, has a unique solution.

4. Error estimates.

4.1. An a priori estimate for the error. If $\varepsilon(x)=y(x)-u(x)$, where $x\in\omega$ is the error of the finite difference method, then $y(x)=\varepsilon(x)+u(x)$, $x\in\omega$, or, in vector form, $y=\varepsilon+u$. Substituting y into (3.5), we get

$$A\varepsilon=\phi-Au\equiv\psi, \tag{4.1}$$

where $Au(x)=w_1(x)+w_2(x)+w_3(x)$ and the approximate fluxes are defined by the values of $u(x)$ at the grid points, i.e., $w_l(x)=k_l(x)\Delta_lu(x)$, $x\in\omega$, $l=1,2,3$. The vector ϕ is defined by the right-hand side of (3.1). We represent the right-hand side of (4.1) in the form

$$\psi(x)=\sum_{l=1}^{3}\left\{\int_{s_l} W_{\nu_l}\,ds-w_l(x)\right\}\equiv\sum_{l=1}^{3}\eta_l(x), \tag{4.2}$$

where the approximate fluxes w_l, $l=1, 2, 3$, are defined by the values of $u(x)$ at the grid points. Here we have implicitly defined

$$\eta_l(x)=\int_{s_l} W_{\nu_l}\,ds - w_l(x), \qquad x\in\omega. \tag{4.3}$$

Obviously, according to (3.2) at the regular grid points (see Fig. 3.1), $\eta_1(x)=-\eta_1(x_+)$ holds.

On Γ_D we have Dirichlet boundary conditions, so that $y(x)=u(x)$ and $\varepsilon(x)=0$ for $x\in\gamma_D$. In order to prove convergence of the finite difference schemes in the energy norm, we are interested in a priori estimates for the error $\varepsilon(x)$. From (4.1) we get

$$\varepsilon^t A\varepsilon=\varepsilon^t\psi. \tag{4.4}$$

According to (3.8),

$$\begin{aligned}\varepsilon^t A\varepsilon&=\sum_{x\in\omega}\{w_1(x)+w_2(x)+w_3(x)\}\varepsilon(x)\\&=\sum_{x\in\omega}\sum_{l=1}^{3}\alpha_l(x)w_l(x)\Delta_l\varepsilon(x),\end{aligned} \tag{4.5}$$

where $w_l(x)=k_l(x)\Delta_l\varepsilon(x)$, $l=1, 2, 3$. On the other hand, from (4.2) and (3.8) we have

$$\varepsilon^t\psi=\sum_{x\in\omega}\psi(x)\varepsilon(x)=\sum_{x\in\omega}\sum_{l=1}^{3}\eta_l(x)\varepsilon(x)=\sum_{x\in\omega}\sum_{l=1}^{3}\alpha_l(x)\eta_l(x)\Delta_l\varepsilon(x). \tag{4.6}$$

Hence, by (4.4), we have

$$\varepsilon^t A\varepsilon\equiv\left|\sum_{\omega}\sum_{l=1}^{3}\alpha_l\omega_l\Delta_l\varepsilon\right|=\left|\sum_{\omega}\sum_{l=1}^{3}\alpha_l\eta_l\Delta_l\varepsilon\right|. \tag{4.7}$$

From (3.14) and the Cauchy inequality applied to the right-hand side of (4.7), we get

$$\gamma_1\varepsilon^t A_0\varepsilon\leqq(\varepsilon^t A_0\varepsilon)^{1/2}\left(\sum_{\omega}\sum_{l=1}^{3}\eta_l^2(x)\right)^{1/2}. \tag{4.8}$$

Since A_0 is symmetric and positive definite and $\varepsilon(x)=0$, $x\in\gamma_D$, we can define the energy norm

$$\|\varepsilon\|_{1,\omega}\equiv\|\varepsilon\|_{A_0}=(\varepsilon^t A_0\varepsilon)^{1/2}. \tag{4.9}$$

THEOREM 4.1. *The error* $\varepsilon(x)=y(x)-u(x)$, $x\in\omega$, *of the finite difference scheme* (3.1) *satisfies the a priori estimate*

$$\|\varepsilon\|_{1,\omega}\leqq C(\|\eta_1\|_{0,\omega}+\|\eta_2\|_{0,\omega}+\|\eta_3\|_{0,\omega}), \tag{4.10}$$

where the discrete $\mathbb{L}^2$*-norm is defined by*

$$\|y\|_{0,\omega}=\left(\sum_{x\in\omega}y^2(x)\right)^{1/2},$$

where η_l, $l=1, 2, 3$, *are the components of the local truncation error defined by* (4.3) *and* w_l, $l=1, 2, 3$, *are the approximate fluxes for our four difference schemes* (2.15)-(2.19).

4.2. Error estimates. If the diffusion coefficient satisfies $a(x)\equiv1$, from (4.3) we have that the local truncation error is

$$\eta_l(x)=-\int_{s_l}\frac{\partial u}{\partial\nu_l}\,ds-w_l(x), \qquad x\in\omega, \tag{4.11}$$

where the fluxes $w_l(x)$ are defined by the values $u(x)$ at the grid points $x \in \omega$ $(l=1, 2, 3)$. Now we estimate the terms $\eta_l(x)$, $l=1, 2, 3$.

In the case of regular grid points for $l=1$ (see Fig. 4.1(a)), using (2.15) we have

$$\eta_1(P_0) = -\int_{P_2}^{P_3} \frac{\partial u}{\partial \nu_1}(s)\, ds - k_1(P_0)[u(P_0)-u(P_1)],$$

where

$$k_1(P_0) = \frac{\text{dist}\,(P_2, P_3)}{\text{dist}\,(P_0, P_1)} = \tan \beta.$$

The expression on the right-hand side is a linear functional of $u(x)$, bounded in $\mathbb{H}^m(\bar{T})$, $m \geqq \frac{3}{2}$, $\bar{T} = \bar{T}(x) = \bar{T}(P_0) = T(P_0) \cup T(P_1)$. This functional vanishes for all polynomials of second degree. Therefore, by the Bramble–Hilbert lemma, we get

$$(4.12) \qquad |\eta_1(x)| \leqq Ch^m |u|_{m+1, \bar{T}(x)}, \qquad \tfrac{1}{2} < m \leqq 2,$$

where $h = \text{diam } T(P_0)$.

In the case of collection of uniform triangulation on the interface between these two uniform triangulations, we have the situation shown in Fig. 4.1(b). Since $P_0M \neq P_1M$, the linear functional $\eta_1(P_0)$ vanishes for polynomials of first degree and therefore

$$|\eta_1(x)| \leqq Ch|u|_{2, \bar{T}(x)} \quad \text{for } u \in \mathbb{H}^m(\bar{T}), \quad m \geqq 2.$$

Now consider the different cases of the flux approximation at the irregular grid points of ω.

First, in the case of the simple symmetric approximation, for the point P_4 from Fig. 2.4, the following expression is valid:

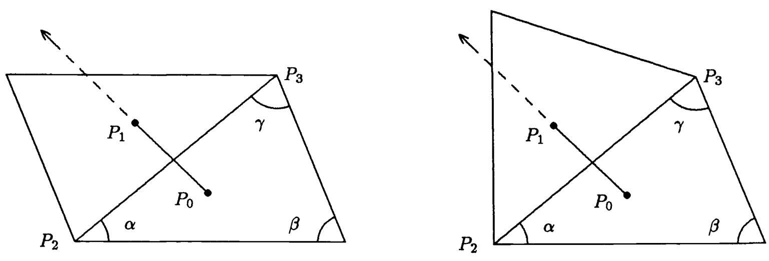
$$\eta_1(P_4) = -\int_{P_7}^{P_8} \frac{\partial u}{\partial \nu_1}(s)\, ds - k_1(P_4)[u(P_4)-u(P_0)],$$

where

$$k_1(P_4) = \frac{\text{dist}\,(P_7, P_8)}{\text{dist}\,(P_4, P_4^*)} = \tan \beta.$$

This functional vanishes only for $u \equiv \text{const}$. Then

$$(4.13) \qquad |\eta_1(x)| \leqq C(|u|_{1, \bar{T}(x)} + h^m |u|_{m+1, \bar{T}(x)})$$



(a) uniform

(b) piecewise uniform

FIG. 4.1. *Triangulation of* Ω.

and, with the help of (4.12) and (4.13), we get

$$\begin{aligned}\sum_{x\in\omega}\eta_1^2(x)&\leqq C\Bigg(\sum_{\substack{x\in\omega\\ x\text{-irregular}}}|u|^2_{1,\bar T(x)}+\sum_{x\in\omega}h^{2m}|u|^2_{m+1,\bar T(x)}\Bigg)\\ &\leqq C(|u|^2_{1,\Omega_h}+h^{2m}|u|^2_{m+1,\Omega}),\qquad m\geqq\frac{1}{2},\end{aligned}\tag{4.14}$$

where Ω_h is a strip of width $2h$ around the interface between the coarse- and fine-grid regions and h is the coarse-grid size.

The first term on the right-hand side of (4.14) can be estimated using the so-called Il'in's inequality (see [17])

$$\|\phi\|_{0,\Omega\delta}\leqq c\delta^{1/2}\|\phi\|_{m,\Omega},\qquad m>\tfrac{1}{2},\tag{4.15}$$

where Ω_δ is a strip in Ω with width δ. Therefore we have

$$\|\eta_1\|_{0,\omega}=\Bigg(\sum_{x\in\omega}\eta_1^2(x)\Bigg)^{1/2}\leqq Ch^{1/2}\|u\|_{m+1,\Omega},\qquad m>\tfrac{1}{2}.\tag{4.16}$$

In the same way, we can estimate $\|\eta_2\|_{0,\omega}$ and $\|\eta_3\|_{0,\omega}$, yielding in the case of the simple symmetric difference scheme an $\mathbb{O}(h^{1/2})$ rate of convergence in the energy norm:

$$\|y-u\|_{1,\omega}\leqq Ch^{1/2}\|u\|_{m+1,\Omega}\qquad m>\tfrac{1}{2}.\tag{4.17}$$

Second, for the case of the simple nonsymmetric difference scheme, using (2.16) we have (see Fig. 2.4)

$$\begin{aligned}\eta_1(P_4)=-\int_{P_7}^{P_8}\frac{\partial u}{\partial\nu_1}(s)\,ds-k_1(P_4)\Bigg[&u(P_4)-\frac{1}{4}(1-\tan\gamma\,\text{cotan}\,\beta)u(P_2)\\ &+\frac{1}{4}(1+\tan\alpha\,\text{cotan}\,\beta)u(P_3)\\ &-\frac{1}{4}((\tan\alpha+\tan\gamma)\,\text{cotan}\,\beta+4)u(P_0)\Bigg].\end{aligned}\tag{4.18}$$

For the case of the more accurate symmetric scheme, by (2.17) we have

$$\eta_1(P_4)=-\int_{P_7}^{P_8}\frac{\partial u}{\partial\nu_1}(s)\,ds-\frac{1}{3}k_1(P_4)[-2u(P_0)+u(P_4)+u(P_5)].\tag{4.19}$$

The functionals from (4.18) and (4.19) vanish for all linear polynomials, so the Bramble-Hilbert lemma yields

$$|\eta_1(x)|\leqq Ch^m|u|_{m+1,\bar T(x)},\qquad \tfrac{1}{2}<m\leqq 1.$$

In the same way as in the cases of linear interpolation, we get

$$\|\eta_1\|_{0,\omega}=\Bigg(\sum_{x\in\omega}\eta_1^2(x)\Bigg)^{1/2}\leqq Ch^{m+1/2}\|u\|_{m+3/2,\Omega},\qquad \tfrac{1}{2}<m\leqq 1.\tag{4.20}$$

For the case of the simple nonsymmetric or more accurate symmetric scheme, the rate of convergence in energy norm is $\mathbb{O}(h^{3/2})$:

$$\|y-u\|_{1,\omega}\leqq Ch^{m+1/2}\|u\|_{m+3/2,\Omega},\qquad \tfrac{1}{2}<m\leqq 1.\tag{4.21}$$

Third, in the case of the more accurate nonsymmetric scheme, when we consider equilateral triangles, using (2.18) for the approximate fluxes, the functional has the form

$$\eta_1(P_4) = -\int_{P_7}^{P_8} \frac{\partial u}{\partial \nu_1}(s)\,ds - \frac{1}{30}k_1(P_4)[-15u(P_0)+15u(P_1)-3u(P_2) \\ +3u(P_3)+6u(P_4)-6u(P_5)], \tag{4.22}$$

where $k_1(P_4)=\tan\beta=\sqrt{3}$. This functional vanishes for polynomials of second degree. Hence, for the irregular grid points, we have

$$|\eta_1(x)| \leqq Ch^m|u|_{m+1,\bar{T}(x)}, \qquad \tfrac{1}{2}<m\leqq 2.$$

Then, in a similar way as in (4.14) and (4.15), the following inequalities are valid:

$$\begin{aligned} \sum_{x\in\omega}\eta_1^2(x) &\leqq C\Bigg(\sum_{\substack{x\in\omega\\ x\text{-irregular}}} h^{2m}|u|^2_{m+1,\bar{T}(x)} + \sum_{x\in\omega} h^{2m}|u|^2_{m+1,\bar{T}(x)}\Bigg) \\ &\leqq C(h^{2m}|u|^2_{m+1,\Omega_h}+h^{2m}|u|^2_{m+1,\Omega}) \\ &\leqq Ch^{2m}\|u\|^2_{m+1,\Omega}, \qquad \tfrac{1}{2}<m\leqq 2. \end{aligned} \tag{4.23}$$

Thus the case of the more accurate nonsymmetric scheme yields the best convergence rate of $\mathbb{O}(h^2)$:

$$\|y-u\|_{1,\omega} \leqq Ch^m\|u\|_{m+1,\Omega}, \qquad \tfrac{1}{2}<m\leqq 2.$$

The following theorem summarizes the results.

THEOREM 4.2. *If the solution $u(x)$ of the problem* (2.1)-(2.3) *with constant coefficient $a(x)$ is $\mathbb{H}^m$-regular, $m\geqq\frac{3}{2}$, then the rate of convergence in the energy norm* (4.9) *is $\mathbb{O}(h^{1/2})$ for the simple symmetric scheme, $\mathbb{O}(h^{3/2})$ for the simple nonsymmetric and more accurate symmetric schemes, and $\mathbb{O}(h^2)$, $m\geqq 2$, for the more accurate nonsymmetric scheme.*

Remark 4.1. Let h_c be the coarse-grid cell size and h_f be the fine-grid cell size. Our analysis is done for $m=h_c/h_f=2$. It is easy to see that we can construct and study the corresponding schemes for any integer ratio m. For instance, in § 6 we have performed a series of computational experiments for $m=2$, 4.

From a more careful look at the proofs of the estimates (4.16), (4.20), and (4.23), we can see that the constant C in (4.16) does not depend on m, and estimates in (4.20) and (4.23) depend on m linearly.

Remark 4.2. In the case of collection of uniform triangulations, applying the estimate of $\eta_1(x)$ on the interface cells and Il'in's inequality (4.15), we can show that the rate of convergence in the energy norm is $\mathbb{O}(h^{1/2})$ for the simple symmetric scheme and $\mathbb{O}(h^{3/2})$ for all remaining approximations.

5. Efficient iterative methods.

5.1. Algebraic formulation of the BEPS preconditioner. We partition the nodes in the composite grid ω into two groups: ω_2, the nodes in the refined subregion Ω_2, and ω_1, the nodes in Ω_1. Then the matrix A admits the two-by-two block form

$$A=\begin{array}{cc}\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22}\end{bmatrix} & \begin{array}{l}\}\omega_2\\ \}\omega_1\end{array}\end{array}.$$

The coarse-grid matrix $\tilde{A}$ is partitioned in the same manner as the composite grid matrix A into a two-by-two block structure, where the first block represents the coarse-grid points in Ω_2

$$\tilde{A}=\begin{array}{cc}\begin{bmatrix} \tilde{A}_{11} & \tilde{A}_{12} \\ \tilde{A}_{21} & \tilde{A}_{22}\end{bmatrix} & \begin{array}{l}\}\tilde{\omega}_2\\ \}\omega_1\end{array}\end{array}.$$

The preconditioner B of Bramble, Ewing, Pasciak and Schatz [4] can be constructed as follows.

For a given vector

$$b=\begin{bmatrix} b_1 \\ b_2 \end{bmatrix},$$

the solution of $B\,v=b$ is computed in the following steps: (i) solve in Ω_2:

$$A_{11}v_1^F=b_1;$$

(ii) compute the defect

$$d=b-A\begin{bmatrix} v_1^F \\ 0 \end{bmatrix}=\begin{bmatrix} 0 \\ b_2-A_{21}v_1^F \end{bmatrix}\}\omega_2;$$

(iii) form the coarse-grid correction (solve the coarse-grid problem)

$$\tilde{A}\tilde{v}=\begin{bmatrix} 0 \\ b_2-A_{21}v_1^F \end{bmatrix}\}\tilde{\omega}_2;$$

(iv) find v_1^H in ω_2 such that

$$A_{11}v_1^H+A_{12}\tilde{v}_2=0$$

(i.e., compute the harmonic component). Then set

$$v=B^{-1}b=\begin{bmatrix} v_1^F+v_1^H \\ \tilde{v}_2 \end{bmatrix}.$$

In matrix notation, we have the following factored form of the BEPS preconditioner [8]:

$$B=\begin{bmatrix} A_{11} & 0 \\ A_{21} & \tilde{S} \end{bmatrix}\begin{bmatrix} I & A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix}, \tag{5.1}$$

where $\tilde{S}=\tilde{A}_{22}-\tilde{A}_{21}\tilde{A}_{11}^{-1}\tilde{A}_{12}$ is the Schur complement for the coarse-grid matrix. The corresponding block Choleski factorization of A reads

$$A=\begin{bmatrix} A_{11} & 0 \\ A_{21} & S_2 \end{bmatrix}\begin{bmatrix} I & A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix}, \tag{5.2}$$

where $S_2=A_{22}-A_{21}A_{11}^{-1}A_{12}$ is the Schur complement for the composite-grid matrix. Note that factorizations of A and B agree except for these Schur complements.

5.2. Algebraic formulation of the two-grid FAC preconditioner of McCormick [14], [16]. Consider the composite-grid matrix A (symmetric or nonsymmetric) and the coarse-grid matrix $\tilde{A}$ partitioned into the two-by-two block structures according to the partitioning of Ω into regions Ω_1 and Ω_2. Then we formulate the FAC preconditioner B for solving the system

$$Av=b, \qquad v=\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}\begin{matrix}\}\omega_2 \\ \}\omega_1\end{matrix}$$

as follows:

(i) solve the fine-grid problem in Ω_2:

$$A_{11}v_1^F=b_1;$$

(ii) restrict the defect to the coarse grid

$$\begin{aligned}\tilde{d} &= P^t\left[b - A\begin{bmatrix} v_1^F \\ 0 \end{bmatrix}\right] \\ &= P^t\left[b - \begin{bmatrix} b_1 \\ A_{21}v_1^F \end{bmatrix}\right] \\ &= P^t\begin{bmatrix} 0 \\ b_2 - A_{21}A_{11}^{-1}b_1 \end{bmatrix};\end{aligned}$$

(iii) solve for coarse-grid correction

$$\tilde{A}\tilde{c} = \tilde{d};$$

(iv) interpolate the correction

$$c = P\tilde{A}^{-1}P^t\begin{bmatrix} 0 \\ b_2 - A_{21}A_{11}^{-1}b_1 \end{bmatrix};$$

(v) then set

$$B^{-1}b = \begin{bmatrix} v_1^F \\ 0 \end{bmatrix} + c.$$

In matrix notation, we have [7]

$$B_{\mathrm{FAC}}^{-1} = \begin{bmatrix} A_{11}^{-1} & 0 \\ 0 & 0 \end{bmatrix} + (P\tilde{A}^{-1}P^t)\begin{bmatrix} 0 & 0 \\ -A_{21}A_{11}^{-1} & I \end{bmatrix}. \tag{5.3}$$

5.3. Optimal order two-grid preconditioners. As shown in [8], the following representation,

$$B^{-1}A = \begin{bmatrix} I & * \\ 0 & \tilde{S}^{-1}S_2 \end{bmatrix},$$

is valid for both two-grid preconditioners considered here. Using the local analysis technique from [8], we can show the following main result concerning the spectrum of $\tilde{S}^{-1}S_2$ and, hence, that of $B^{-1}A$.

THEOREM 5.1. *The Schur complements S_2 and $\tilde{S}$ are spectrally equivalent with constants independent of h. These constants are also independent of jumps in the coefficient $a = a(x)$ if it is continuous within each coarse-grid cell. In the nonsymmetric case the spectral equivalence reads as follows. The spectrum of $\tilde{S}^{-1}S_2$ is contained in a segment of the disc in the right-half complex plane*

$$\{z \in \mathbb{C}, \operatorname{Re} z \geqq \gamma_1, |z| \leqq \gamma_2\}$$

with γ_1 and γ_2 constants independent of h and of jumps in the coefficient $a = a(x)$.

Proof. From Theorem 3.1, we have a spectral equivalence between the composite-grid matrix A for any of our four approximations and the matrix of the simple symmetric approximation A_0. Hence it suffices to prove that A_0 and the coarse-grid matrix $\tilde{A}$ are spectrally equivalent, i.e., that there exist two positive constants γ_1 and γ_2, such that

$$\gamma_1\tilde{y}^t\tilde{A}\tilde{y} \leqq \inf_{\bar{y}} y^tA_0y \leqq \gamma_2\tilde{y}^t\tilde{A}\tilde{y} \tag{5.4}$$

and

$$y = \begin{bmatrix} \bar{y} \\ \tilde{y} \end{bmatrix}\begin{matrix} \}\omega\backslash\tilde{\omega} \\ \}\tilde{\omega} \end{matrix}.$$

Then by taking inf over all the nodes from ω_2, we obtain

$$\gamma_1 y_2^t \tilde{S} y_2 \leqq y_2^t S_2 y \leqq \gamma_2 y_2^t \tilde{S} y_2 .$$

The constants γ_1 and γ_2 in the above inequalities, which imply the desired result, are bounded uniformly with respect to h_c, as well as with respect to jumps of the coefficient $a(x)$, as long as it is continuous within each coarse-grid cell.

The estimate (5.4) from above holds immediately from (3.9) and the flow definition (2.15). We get

$$y^t A_0 y \leqq \tilde{y}^t \tilde{A} \tilde{y},$$

where $y(x)$ is a piecewise constant interpolant of $\tilde{y}$ over each coarse-grid cell. Hence $\gamma_2 = 1$ when $a(x)$ is constant over each coarse-grid cell.

The estimate below is based on local analysis. For simplicity, we consider the case when $a(x) \equiv 1$ in Ω.

We cover the region Ω with cells $E(x)$ obtained by connecting the centers of every six neighboring cells (see Fig. 5.1). Then the following representation of the quadratic form $y^t A_0 y$ is valid:

$$y^t A_0 y = \sum_{\substack{E(x) \\ x \in \tilde{\omega}}} \sum_{\xi \in E(x) \cap \omega} \sum_{l=1}^{3} \{\alpha_l(\xi) k_l(\xi) (\Delta_l y(\xi))^2\} = \sum_{x \in \tilde{\omega}} J(x; y),$$

where the cells $E(x)$ are counted only once.

Here

$$\alpha_l(\xi) = \begin{cases} \frac{1}{2} & \text{for the internal nodes,} \\ 1 & \text{for } \xi \text{ near } \gamma_D. \end{cases}$$

In a similar way, we have

$$\tilde{y}^t \tilde{A} \tilde{y} = \sum_{\substack{E(x) \\ x \in \tilde{\omega}}} \sum_{\xi \in E(x) \cap \tilde{\omega}} \sum_{l=1}^{3} \{\alpha_l(\xi) k_l(\xi) (\Delta_l \tilde{y}(\xi))^2\} = \sum_{x \in \tilde{\omega}} \tilde{J}(x; \tilde{y}).$$

Note that $J(x; y)$ and $\tilde{J}(x; \tilde{y})$ are nonnegative quadratic forms for each $x \in \tilde{\omega}$. They are positive definite if $E(x)$ has at least one point on $\gamma_D(u_{|_{\gamma_D}} = 0)$ or when $v|_{E(x)} \neq$ const and $\tilde{v}|_{E(x)} \neq$ const.

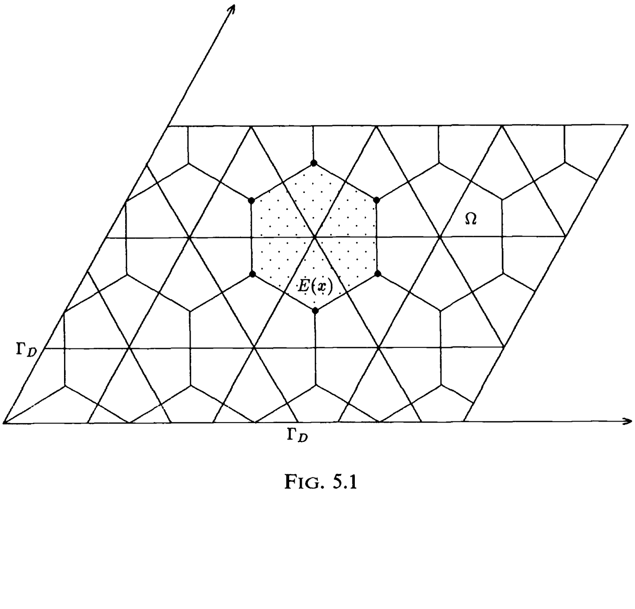


FIG. 5.1

We introduce

$$J(x; \bar{y}, \tilde{y}) = J(x; y), \qquad y = \begin{bmatrix} \bar{y} \\ \tilde{y} \end{bmatrix} \begin{matrix} \}\omega\backslash\tilde{\omega} \\ \}\tilde{\omega} \end{matrix}.$$

The local quadratic forms $\tilde{J}(x; \tilde{y})$ and $\inf_{\bar{y}} J(x; \bar{y}, \tilde{y})$ are defined in the finite-dimensional space associated with the coarse-grid unknowns in $E(x)$. Therefore, there exists a positive constant $\gamma_{1,E(x)}$, such that

$$\min_{\bar{y}} J(x; \bar{y}, \tilde{y}) \geqq \gamma_{1,E(x)} \tilde{J}(x; \tilde{y}).$$

The constant $\gamma_{1 \cdot E(x)}$, $x \in \tilde{\omega}$, do not depend on h_c but they are not, in general, bounded with respect to h_c/h_f. Hence

$$\begin{aligned} \min_{\bar{y}} y^t A_0 y &\geqq \sum_{x \in \tilde{\omega}} \min_{\bar{y}|_{E(x)}} J(x; \bar{y}, \tilde{y}) \geqq \sum_{x \in \tilde{\omega}} \gamma_{1,E(x)} \tilde{J}(x; \tilde{y}) \\ &\geqq \min_{x \in \tilde{\omega}} \gamma_{1,E(x)} \sum_{x \in \tilde{\omega}} \tilde{J}(x; \tilde{y}) = \gamma_1 \tilde{y}^t \tilde{A} \tilde{y}, \end{aligned}$$

where

$$\gamma_1 = \min_{x \in \tilde{\omega}} \gamma_{1,E(x)}.$$

We note that, since the coarse-grid cells are geometrically similar to a fixed number of triangles, the constant γ_1 is independent of the number of the coarse-grid cells, but our analysis allows some dependence on the aspect ratio h_c/h_f, which we have assumed to be bounded.

Actually, as demonstrated in Ewing, Lazarov, and Vassilevski [8], we can show that in the case of $a(x) \neq 1$, there exist constants γ_1 and γ_2 which are bounded independently of possible jumps of $a(x)$ as long as it is continuous within each coarse-grid cell. This has also been confirmed by our numerical experiments. □

Theorem 5.1 implies the following corollary.

COROLLARY 5.1. *The* GCG-LS *method from Axelsson* [1], [2] *for solving the composite grid system with any of the preconditioners B will have a rate of convergence independent of h and it will be insensitive to jumps in the coefficient* $a = a(x)$.

6. Numerical experiments. In this section, we present numerical results for solving model problems of type (2.1)-(2.3). We investigate experimentally the accuracy of the finite difference schemes and how rapidly the proposed preconditioned methods converge.

Our test problem is the diffusion equation on the region Ω, the unit parallelogram with acute angles equal to $\pi/3$:

$$\Omega = \{(x, y): 0 \leqq y \leqq \sqrt{\tfrac{3}{2}},\ y/\sqrt{3} \leqq x \leqq y\sqrt{3} + 1\}.$$

We assume the Dirichlet boundary

$$\Gamma_D = \{(x, 0): 0 \leqq x \leqq 1\} \cup \{(x, \sqrt{3}\,x): 0 \leqq x \leqq \tfrac{1}{2}\}.$$

The discretizations are done on a cell-centered grid, as shown in Fig. 6.1, with $h = 1/(n - \frac{1}{3})$ for some given integer n and ratio between coarse- and fine-grid cell sizes equal to m, $m \geqq 1$, integer.

We partition Ω into $\Omega_1 \cup \Omega_2$, where

$$\Omega_2 = \{(x, y): y_0 < y < \sqrt{\tfrac{3}{2}},\ x_0 + (y - y_0)/\sqrt{3} < x < y/\sqrt{3} + 1\}$$

and

$$x_0 = (i_0 + (j_0 - 1)/2)h, \qquad y_0 = (j_0 - \tfrac{1}{3})h\sqrt{\tfrac{3}{2}},$$

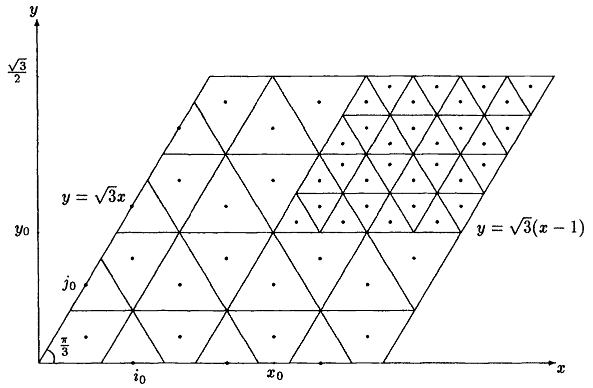


FIG. 6.1. *The composite grid* $(m = 2)$.

for some integers $0 < i_0 < n$, $0 < j_0 < n$, and $\bar{\Omega}_1 = \bar{\Omega} \backslash \Omega_2$. All numerical experiments were performed for $i_0 = j_0 = n/2$, i.e., the refined region is one-quarter of the whole domain.

We investigate the following cell-centered finite difference schemes: Scheme 0 (simple symmetric) and Scheme 1 (simple nonsymmetric). Our test problems are as follows.

PROBLEM 1 (a smooth solution).

$$u = y\left(y - \frac{\sqrt{3}}{2}\right)(y - \sqrt{3}x)(y - \sqrt{3}(x-1))$$

with a diffusion coefficient

$$a(x, y) = 1/(1 + 10(x^2 + y^2)).$$

PROBLEM 2 (a smooth solution with a discontinuous diffusion coefficient $a(x, y)$).

$$u = (\sqrt{3}(x - \hat{x}) - (y - \hat{y}))(y - \hat{y})\phi(x, y)/a(x, y),$$

where

$$\phi(x, y) = \sin\left(\frac{\pi}{2}(y - \sqrt{3}(x-1))\right) \sin\left(\frac{\pi}{2}\left(y - \frac{\sqrt{3}}{2}\right)\right) \sin\left(\frac{\pi}{2}y\right) \sin\left(\frac{\pi}{2}(y - \sqrt{3}x)\right),$$

$$a(x, y) = \begin{cases} 100 & \text{when } R < 0 \text{ or } y < \hat{y}, \\ 1 & \text{otherwise,} \end{cases}$$

$$R = \sqrt{3}(x - \hat{x}) - (y - \hat{y}),$$

and $\hat{x} = (\hat{i} + (\hat{j} - 1)/2)h$, $\hat{y} = (\hat{j} - \frac{1}{3})h\frac{\sqrt{3}}{2}$. For our experiments we choose $\hat{i} = i_0 + 2$ and $\hat{j} = j_0 + 2$.

PROBLEM 3 (a smooth solution with a support in Ω_2).

$$u = p(x, y)\phi(x, y),$$

where

$$p(x, y) = \begin{cases} (y - \sqrt{3}(x - \tilde{x}))(y - \tilde{y}), & \frac{y}{\sqrt{3}} + \tilde{x} < x < \frac{y}{\sqrt{3}} + 1, \quad \tilde{y} < y < \frac{\sqrt{3}}{2}, \\ 0, & \text{otherwise;} \end{cases}$$

$$\phi(x, y) = \begin{cases} \sin\left(\frac{\pi}{2}(y - \sqrt{3}(x - \tilde{x}))\right) \sin\left(\frac{\pi}{2}(y - \tilde{y})\right), & \frac{y}{\sqrt{3}} + \tilde{x} < x < \frac{y}{\sqrt{3}} + 1, \quad \tilde{y} < y < \frac{\sqrt{3}}{2}, \\ 0, & \text{otherwise;} \end{cases}$$

and $\tilde{x} = \tilde{y} = 0.75$.

We consider the following error estimators:

(i) the discrete $\mathbb{L}^2$-error, defined by

$$\varepsilon_0=\left(\sum_{x\in\omega} h^2(y(x)-u(x))^2\right)^{1/2};$$

(ii) the discrete $\mathbb{H}^1$-error, defined by

$$\varepsilon_1=|(y-u)^tA(y-u)|^{1/2}.$$

We also consider the following preconditioners: Preconditioner 1 (BEPS preconditioner) and Preconditioner 2 (FAC preconditioner).

We report on the number of iterations required of our preconditioned iterative methods: PCG in the symmetric case and GCG-LS in the nonsymmetric case. The stopping criterion is

$$r^tr<\varepsilon, \qquad \varepsilon=10^{-18},$$

where $r=b-Ay$ is the residual vector and y is the current iterate. As an initial guess, we always choose $P\tilde{A}^{-1}\tilde{b}$, i.e., a piecewise constant interpolant of the coarse-grid approximation. Also, we report on the average reduction factor

$$q=(\Delta/\Delta_0)^{1/\text{iter}},$$

where Δ_0 is the corresponding norm of the initial residual, Δ is the norm of the last residual, and iter is the number of iterations required to achieve the desired accuracy ε. The numerical results are collected on Tables 6.1-6.7. N is the number of unknowns and $\tilde{\varepsilon}_0$ and $\tilde{\varepsilon}_1$ are the corresponding discrete $\mathbb{L}^2$- and $\mathbb{H}^1$-errors of the coarse-grid approximations, which we interpolate piecewise constantly in order to get initial guesses in our iterative methods.

TABLE 6.1
Accuracy results for problems without local refinement.

	Problem 1		
n	$\tilde{\varepsilon}_0$	$\tilde{\varepsilon}_1$	N
12	0.14−3	0.26−3	265
24	0.36−4	0.62−4	1105
48	0.89−5	0.15−4	4513
	Problem 2		
n	$\tilde{\varepsilon}_0$	$\tilde{\varepsilon}_1$	N
12	0.18−4	0.57−3	265
24	0.75−5	0.29−3	1105
48	0.23−5	0.11−3	4513
	Problem 3		
n	$\tilde{\varepsilon}_0$	$\tilde{\varepsilon}_1$	N
12	0.32−5	0.87−4	265
24	0.65−6	0.23−4	1105
48	0.14−6	0.65−5	4513

TABLE 6.2
Accuracy results for problems with local refinement ($m = 2$).

Problem 1

n	iter	Scheme	ε_0	ε_1	N
12	5	0	0.98-3	0.82-2	481
	6	1	0.54-3	0.53-2	
24	5	0	0.47-3	0.55-2	1969
	6	1	0.13-3	0.26-2	
48	5	0	0.23-3	0.38-2	7969
	6	1	0.34-4	0.12-2	

Problem 2

n	iter	Scheme	ε_0	ε_1	N
12	5	0	0.51-5	0.74-3	481
	7	1	0.54-5	0.43-3	
24	6	0	0.24-5	0.38-3	1969
	6	1	0.20-5	0.16-3	
48	6	0	0.10-5	0.26-3	7969
	6	1	0.61-6	0.54-4	

Problem 3

n	iter	Scheme	ε_0	ε_1	N
12	4	0	0.94-6	0.32-4	481
	4	1	0.94-6	0.32-4	
24	4	0	0.15-6	0.56-5	1969
	4	1	0.15-6	0.56-5	
48	3	0	0.32-7	0.18-5	7969
	3	1	0.32-7	0.18-5	

From Tables 6.2 and 6.3, which use two different refinement factors 2^m, we observe a monotonic improvement of the accuracy in both norms. Since the solution of Problem 3 has support entirely in the refined region, predictably both schemes give consistently the same results, since the truncation error is actually localized in the refined region. For Problem 2, where the solution is piecewise smooth and almost local, we obtain good convergence results. Both schemes behave in almost the same way, with the simple nonsymmetric scheme giving slightly better results.

Although symmetric and nonsymmetric schemes show almost the same accuracy and number of iterations, we recommend the use of the symmetric scheme since the PCG method is more effective in both storage and operational count (hence, more stable). Both preconditioners show fast convergence of the iterations which do not depend on the discretization parameters h and m (see Tables 6.4–6.7).

In the nonsymmetric case, we recommend the use of the FAC preconditioner because it is less expensive in operation counts (one fine-grid solution is replaced by interpolation, which is much faster).

Acknowledgments. The authors would like to express their gratitude to Professor Richard Ewing and Professor Steve McCormick for their useful comments and remarks which improved substantially the presentation. Careful comments by the referee are also gratefully acknowledged.

TABLE 6.3
Accuracy results for problems with local refinement ($m = 4$).

Problem 1

n	iter	Scheme	ε_0	ε_1	N
12	6	0	0.20 − 2	0.15 − 1	1345
	8	1	0.85 − 3	0.10 − 1	
24	6	0	0.10 − 2	0.10 − 1	5425
	8	1	0.21 − 3	0.51 − 2	
48	6	0	0.54 − 3	0.72 − 2	21793
	8	1	0.53 − 4	0.25 − 2	

Problem 2

n	iter	Scheme	ε_0	ε_1	N
12	6	0	0.33 − 5	0.13 − 2	1345
	8	1	0.44 − 5	0.74 − 3	
24	7	0	0.21 − 5	0.73 − 3	5425
	9	1	0.10 − 5	0.23 − 3	
48	7	0	0.12 − 5	0.54 − 3	21793
	8	1	0.26 − 6	0.71 − 4	

Problem 3

n	iter	Scheme	ε_0	ε_1	N
12	5	0	0.13 − 6	0.52 − 5	1345
	6	1	0.13 − 6	0.52 − 5	
24	5	0	0.36 − 7	0.13 − 5	5425
	5	1	0.36 − 7	0.13 − 5	
48	4	0	0.74 − 8	0.82 − 6	21793
	4	1	0.74 − 8	0.82 − 6	

TABLE 6.4
Iterative convergence results for Problem 1 *with local refinement.*

Preconditioner 1

$m = 2$					$m = 4$				
n	iter	Scheme	q	N	n	iter	Scheme	q	N
12	5	0	0.32 − 1	481	12	6	0	0.46 − 1	1345
	6	1	0.55 − 1			8	1	0.73 − 1	
24	5	0	0.32 − 1	1969	24	6	0	0.57 − 1	5425
	6	1	0.53 − 1			8	1	0.118	
48	5	0	0.34 − 1	7969	48	6	0	0.60 − 1	21793
	6	1	0.49 − 1			8	1	0.109	

TABLE 6.5
Iterative convergence results for Problem 1 with local refinement.

Preconditioner 2									
$m = 2$					$m = 4$				
n	iter	Scheme	q	N	n	iter	Scheme	q	N
12	6	0	0.37 − 1	481	12	8	0	0.63 − 1	1345
	7	1	0.80 − 1			9	1	0.108	
24	6	0	0.45 − 1	1969	24	7	0	0.68 − 1	5425
	7	1	0.81 − 1			10	1	0.155	
48	6	0	0.49 − 1	7969	48	7	0	0.80 − 1	21793
	7	1	0.74 − 1			9	1	0.151	

TABLE 6.6
Iterative convergence results for Problem 2 with local refinement.

Preconditioner 1									
$m = 2$					$m = 4$				
n	iter	Scheme	q	N	n	iter	Scheme	q	N
12	5	0	0.28 − 1	481	12	6	0	0.46 − 1	1345
	7	1	0.64 − 1			8	1	0.83 − 1	
24	6	0	0.45 − 1	1969	24	7	0	0.64 − 1	5425
	6	1	0.51 − 1			9	1	0.112	
48	6	0	0.46 − 1	7969	48	7	0	0.72 − 1	21793
	6	1	0.51 − 1			8	1	0.103	

TABLE 6.7
Iterative convergence results for Problem 2 with local refinement.

Preconditioner 2									
$m = 2$					$m = 4$				
n	iter	Scheme	q	N	n	iter	Scheme	q	N
12	6	0	0.44 − 1	481	12	7	0	0.61 − 1	1345
	7	1	0.78 − 1			9	1	0.119	
24	6	0	0.40 − 1	1969	24	7	0	0.68 − 1	5425
	7	1	0.79 − 1			10	1	0.149	
48	7	0	0.64 − 1	7969	48	8	0	0.90 − 1	21793
	7	1	0.81 − 1			10	1	0.146	

REFERENCES

[1] O. AXELSSON, *A generalized conjugate gradient, least squares method*, Numer. Math., 51 (1987), pp. 209-227.

[2] ———, *A restarted version of a generalized preconditioned conjugate gradient method*, Comm. Appl. Numer. Methods, 4 (1988), pp. 521-530.

[3] R. BANK AND D. ROSE, *Some error estimates for the box method*, SIAM J. Numer. Anal., 24 (1987), pp. 777-787.

[4] J. H. BRAMBLE, R. E. EWING, J. E. PASCIAK, AND A. H. SCHATZ, *A preconditioning technique for the efficient solution of problems with local grid refinement*, Comput. Meth. Appl. Mech. Engrg., 67 (1988), pp. 149-159.

[5] Z. CAI AND S. F. MCCORMICK, *On the accuracy of the finite volume method for diffusion equations on composite grids*, SIAM J. Numer. Anal., 27 (1990), pp. 636-655.

[6] Z. CAI, J. MANDEL, AND S. F. MCCORMICK, *The finite element method for diffusion equations on general triangulations*, SIAM J. Numer. Anal., 28 (1991), pp. 392-403.

[7] R. E. EWING, R. D. LAZAROV, AND P. S. VASSILEVSKI, *Local refinement technique for elliptic problems on cell-centered grids*, I: *Error analysis*, Math. Comp. 56 (1991), pp. 437-462.

[8] ———, *Local refinement techniques for elliptic problems on cell-centered grids*, II: *Two-grid iterative methods*, Report No. 1989-47, Enhanced Oil Recovery Institute, University of Wyoming, Laramie, WY, 1989; J. Numer. Linear Algebra Appl., to appear.

[9] W. HACKBUSCH, *On first and second order box schemes*, Computing, 41 (1989), pp. 277-296.

[10] B. HEINRICH, *Finite Difference Methods on Irregular Networks*, Akademie-Verlag, Berlin, 1987.

[11] H. O. KREISS, T. A. MANTEUFFEL, B. SWARTZ, B. WENDROFF, AND A. B. WHITE, JR., *Superconvergent schemes on irregular grids*, Math. Comp., 47 (1986), pp. 537-554.

[12] J. MANDEL AND S. MCCORMICK, *Iterative solution of elliptic equations with refinement*: *The two-level case*, in Proc. Second Internat. Sympos. Domain Decompositions Methods, January 14-16, 1988, University of California, Los Angeles, CA, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989, pp. 81-92.

[13] T. A. MANTEUFFEL AND A. WHITE, JR., *The numerical solution of second order boundary value problems on nonuniform meshes*, Math. Comp., 47 (1986), pp. 511-535.

[14] S. MCCORMICK, *Fast adaptive composite grid* (FAC) *methods*: *Theory for the variational case*, Computing, Suppl., 5 (1984), pp. 115-121.

[15] S. MCCORMICK, *Multilevel Adaptive Methods for Partial Differential Equations*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989.

[16] S. MCCORMICK AND J. THOMAS, *The fast adaptive composite grid* (FAC) *method for elliptic equations*, Math. Comp., 46 (1986), pp. 439-456.

[17] L. A. OGANESJAN AND L. A. RUHOVEC, *Variational difference methods for the solution of elliptic problems*, Izd. Acad. Nauk Armjanskoi SSR, Jerevan, 1979. (In Russian.)

[18] S. V. PATANKAR AND D. B. SPALDING, *Heat and mass transfer in boundary layers*, Morgan-Grampian, London, 1967.

[19] O. A. PEDROSA, JR., *Use of hybrid grid in reservoir simulation*, Ph.D. thesis, Stanford University, CA, 1984.

[20] A. A. SAMARSKII, *Introduction to Theory of Difference Schemes*, Nauka, Moscow, 1971. (In Russian.)

[21] ———, *Local one dimensional difference schemes on nonuniform nets*, USSR Comput. Math. and Math. Phys., 3 (1963), pp. 572-619.

[22] A. A. SAMARSKII, R. D. LAZAROV, AND V. L. MAKAROV, *Difference schemes for differential equations having generalized solutions*, Vysshaya Shkola Publishers, Moscow, USSR, 1987. (In Russian.)

[23] A. N. TIKHONOV AND A. A. SAMARSKII, *Homogeneous difference schemes on nonuniform nets*, USSR Comput. Math. and Math. Phys., 2 (1962), pp. 927-953.

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

[25] A. WEISER AND M. F. WHEELER, *On convergence of block-centered finite-differences for elliptic problems*, SIAM J. Numer. Anal., 25 (1988), pp. 351-375.

[26] G. VORONOI, *Nouvelles applications des paramètres continus à la théorie des forms quadratures*, J. Reine Angew Math., 134 (1908), pp. 198-287.

[27] Y. SAAD AND M. H. SCHULTZ, GMRES: *A generalized conjugate residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856-869.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 6, pp. 1314-1329, November 1992

SEMICOARSENING MULTIGRID ON A HYPERCUBE*

RICHARD A. SMITH† AND ALAN WEISER‡

Abstract. A semicoarsening multigrid algorithm suitable for the kinds of problems arising in reservoir simulation has been implemented on the Intel iPSC/2 hypercube. The method is an extension to nonsymmetric problems of a method in [Dendy et al., *Paper SPE* 18409, presented at Society of Petroleum Engineers Symposium on Reservoir Simulation, Houston, TX, 1989]. It performs well for strongly anisotropic problems and problems with strongly discontinuous coefficients. For a test set of reservoir simulation problems, residual reduction factors for a full-multigrid V-cycle range from 0.0022 to 0.19. The current codes achieve about 50 percent parallel efficiency in two dimensions and about 30 percent parallel efficiency in three dimensions with about $\sqrt{N}/8$ processors for a grid with N unknowns.

Key words. multigrid, hypercubes, reservoir simulation

AMS(MOS) subject classifications. 15, 65, 68

1. Introduction.

Multigrid was first applied to reservoir simulation in the early 1980s [3]. For three-dimensional (3d) problems, multigrid was not found to be competitive with other solution methods because of the large expense per cycle of performing preconditioned conjugate gradient smoothing in the *xy*, *xz*, and *yz* planes. Subsequently, Dendy [6] reduced the cost of the method by using more efficient two-dimensional (2d) multigrid solvers for the smoothing subproblems. More recently, Dendy et al. [8] presented a semicoarsening version of multigrid for symmetric problems which performs well for reservoir simulation problems, with about the same (sequential) CPU cost per cycle but much simpler coding requirements than the previous multigrid methods.

We present an extension of this semicoarsening method to nonsymmetric problems. Among the features of the method are promising intergrid transfer operators due to Schaffer [7]. We investigate ways to implement the method efficiently on hypercube-type parallel processors, both to take advantage of current cost/benefit efficiencies of such machines and to predict how method performance scales with future massively parallel distributed-memory computers.

In the following five sections, respectively, we describe our method, describe several ways of implementing it on hypercube machines, present convergence results, present parallel timing results, and indicate future plans for this project.

2. The sequential method.

Consider a one-dimensional (1d) problem resulting from the one-dimensional pressure equation in reservoir simulation or any similar scalar second-order elliptic partial differential equation. The nonzero structure of the resulting tridiagonal matrix is depicted in Fig. 1. Let P be the permutation matrix ordering the unknowns red-black. The resulting red-black matrix P^TAP is depicted in Fig. 2.

In block form, the red-black linear system is

$$Ax = b \tag{1}$$

* Received by the editors February 19, 1991; accepted for publication October 18, 1991.

† 9550 Ella Lee Lane, Houston, Texas 77063. This work was performed while the author was at the Exxon Production Research Company, 3120 Buffalo Speedway, Houston, Texas 77252.

‡ Center for Research on Parallel Computation, Rice University, Houston, Texas 77251 (deceased).

$$\begin{pmatrix} a & b & & & & & \\ c & d & e & & & & \\ & f & a & b & & & \\ & & c & d & e & & \\ & & & f & a & b & \\ & & & & c & d & e \\ & & & & & f & a \end{pmatrix}$$

FIG. 1. *One-dimensional nonzero structure* $(n1 = 7)$.

$$\begin{pmatrix} a & & & & b & & \\ & a & & & f & b & \\ & & a & & & f & b \\ & & & a & & & f \\ c & e & & & d & & \\ & c & e & & & d & \\ & & c & e & & & d \end{pmatrix}$$

FIG. 2. *One-dimensional red-black matrix nonzero structure.*

or

$$\begin{pmatrix} A_{rr} & A_{rb} \\ A_{br} & A_{bb} \end{pmatrix} \begin{pmatrix} x_r \\ x_b \end{pmatrix} = \begin{pmatrix} b_r \\ b_b \end{pmatrix}. \tag{2}$$

The nonzeros in Figs. 1 and 2 are labeled so that each nonzero diagonal of A_{rr}, A_{rb}, A_{br}, or A_{bb} is labeled with a different letter. One way to solve (2) is to form and solve the smaller Schur complement linear system

$$A_s x_b = b_s, \tag{3}$$

where

$$A_s = A_{bb} - A_{br} A_{rr}^{-1} A_{rb} \tag{4}$$

and

$$b_s = b_b - A_{br} A_{rr}^{-1} b_r, \tag{5}$$

and then backsolve

$$x_r = A_{rr}^{-1}(b_r - A_{rb} x_b). \tag{6}$$

Note that in this 1d case A_s is tridiagonal like A.

Now consider a two-level multigrid method for the original system, where the black unknowns are the coarse-grid unknowns. The main steps in such a method are:

(i) smooth on the fine grid,

(ii) transfer the fine-grid residual to the coarse grid and solve for the coarse-grid correction,

(iii) transfer the correction back to the fine grid and add it in to the current solution,

(iv) smooth again on the fine grid.

In the black-box multigrid framework [5] it is customary to construct the coarse-grid system $A_c x_c = b_c$ as follows:

$$(T_{br} \quad I_{bb})\begin{pmatrix} A_{rr} & A_{rb} \\ A_{br} & A_{bb} \end{pmatrix}\begin{pmatrix} T_{rb} \\ I_{bb} \end{pmatrix}(x_c) = (T_{br} \quad I_{bb})\begin{pmatrix} b_r \\ b_b \end{pmatrix}. \tag{7}$$

Here b_r and b_b denote the current residuals, and are only equal to the original right-hand side if the current iterate is zero. I_{bb} denotes the coarse-grid identity matrix: Pure injection is used for interpolating coarse-grid unknowns to black unknowns on the fine grid.

A key observation is that if

$$T_{br} = -A_{br}A_{rr}^{-1} \tag{8}$$

and

$$T_{rb} = -A_{rr}^{-1}A_{rb}, \tag{9}$$

then

$$A_c = T_{br}A_{rr}T_{rb} + T_{br}A_{rb} + A_{br}T_{rb} + A_{bb} \tag{10}$$

$$= A_{br}A_{rr}^{-1}A_{rb} - A_{br}A_{rr}^{-1}A_{rb} - A_{br}A_{rr}^{-1}A_{rb} + A_{bb} \tag{11}$$

$$= -A_{br}A_{rr}^{-1}A_{rb} + A_{bb} \tag{12}$$

$$= A_s \tag{13}$$

and

$$b_c = -A_{br}A_{rr}^{-1}b_r + b_b = b_s, \tag{14}$$

so that the coarse-grid multigrid system is identical to the Schur complement system. If the smoother on the fine grid solves exactly for the red unknowns in terms of the black unknowns, e.g., the smoother is red-black Gauss-Seidel, then the two-level multigrid solver is an exact solver. Furthermore, if the coarse-grid system itself is solved by a two-level multigrid solver, and so on recursively until the coarsest-grid system with one unknown is solved directly, the resulting multigrid V-cycle is a direct solver.

Now consider a two-dimensional problem with a nine-point operator. The resulting nine-diagonal matrix is depicted in Fig. 3. Let P be the permutation matrix ordering lines of unknowns red-black. The resulting red-black matrix P^TAP is depicted in Fig. 4.

As in the 1d case, if $T_{br} = -A_{br}A_{rr}^{-1}$ and $T_{rb} = -A_{rr}^{-1}A_{rb}$, then the coarse-grid system for two-level multigrid is exactly the Schur complement system $A_s x_s = b_s$. However, the Schur complement matrix is dense. Fill occurs for two reasons. First, A_{rr}^{-1} has dense diagonal blocks, so T_{br} and T_{rb} have dense diagonal blocks. Second, extra fill occurs in direction 1. For instance, the black unknowns corresponding to $(i1, i2) = (2, 1)$ and $(4, 3)$ are both connected to the red unknown corresponding to $(i1, i2) = (3, 2)$ via T_{rb} and T_{br} terms. Elimination of red terms results in direct connections between $(2, 1)$ and $(4, 3)$ in A_s even though their indices differ by more than 1 in direction 1. The only way to avoid this extra fill is to allow connections in T_{br} and T_{rb} in direction 2 only. The locations in Fig. 4 corresponding to connections in direction 2 only are the locations for diagonals c, g, k, and p.

Some new notation must be introduced to deal with connections in direction 2 only. Let an "l" (respectively, "r") superscript appended to A_{rb} or A_{br} denote the result of zeroing out all entries except those corresponding to connection of a black unknown to a red unknown with a smaller (respectively, larger) index in direction 2.

$$\begin{pmatrix}
a & b & & c & d & & & & & & & & & & \\
e & a & b & f & c & d & & & & & & & & & \\
& e & a & & f & c & & & & & & & & & \\
g & h & & i & j & & k & l & & & & & & & \\
m & g & h & n & i & j & o & k & l & & & & & & \\
& m & g & & n & i & & o & k & & & & & & \\
& & & p & q & & a & b & & c & d & & & & \\
& & & r & p & q & e & a & b & f & c & d & & & \\
& & & & r & p & & e & a & & f & c & & & \\
& & & & & & g & h & & i & j & & k & l & \\
& & & & & & m & g & h & n & i & j & o & k & l \\
& & & & & & & m & g & & n & i & & o & k \\
& & & & & & & & & p & q & & a & b & \\
& & & & & & & & & r & p & q & e & a & b \\
& & & & & & & & & & r & p & & e & a
\end{pmatrix}$$

FIG. 3. *Two-dimensional matrix nonzero structure* $(n1=3, n2=5)$.

$$\begin{pmatrix}
a & b & & & & & & & & c & d & & & & \\
e & a & b & & & & & & & f & c & d & & & \\
& e & a & & & & & & & & f & c & & & \\
& & & a & b & & & & & p & q & & c & d & \\
& & & e & a & b & & & & r & p & q & f & c & d \\
& & & & e & a & & & & & r & p & & f & c \\
& & & & & & a & b & & & & & p & q & \\
& & & & & & e & a & b & & & & r & p & q \\
& & & & & & & e & a & & & & & r & p \\
g & h & & k & l & & & & & i & j & & & & \\
m & g & h & o & k & l & & & & n & i & j & & & \\
& m & g & & o & k & & & & & n & i & & & \\
& & & g & h & & k & l & & & & & i & j & \\
& & & m & g & h & o & k & l & & & & n & i & j \\
& & & & m & g & & o & k & & & & & n & i
\end{pmatrix}$$

FIG. 4. *Two-dimensional red-black matrix nonzero structure.*

Then $A_{rb}=A^l_{rb}+A^r_{rb}$ and $A_{br}=A^l_{br}+A^r_{br}$. For example, in Fig. 4, A^l_{rb} may contain nonzeros on diagonals c, d, and f; A^r_{rb} may contain nonzeros on diagonals p, q, and r; A^l_{br} may contain nonzeros on diagonals g, h, and m; and A^r_{br} may contain nonzeros on diagonals k, l, and o. Similarly, let $\text{diag}^l_{rb}(v)$ (respectively, $\text{diag}^r_{rb}(v)$, $\text{diag}^l_{br}(v)$, $\text{diag}^r_{br}(v)$) denote the matrix with the same nonzero structure as A_{rb} (respectively, A_{rb}, A_{br}, A_{br}) with the only nonzero entries obtained from vector v and put on the diagonal corresponding to connections of black unknowns to red unknowns with lower (respectively, higher, lower, higher) index in direction 2 and with the same index in direction 1. For example, in Fig. 4, $\text{diag}^l_{rb}(v)$ (respectively, $\text{diag}^r_{rb}(v)$, $\text{diag}^l_{br}(v)$, $\text{diag}^r_{br}(v)$) puts nonzeros in the locations of diagonal c (respectively, p, g, k).

With this notation, we define the following transfer operators due to Steve Schaffer at the New Mexico Institute of Mining and Technology [7]:

$$T_{rb} = -(\text{diag}^{l}_{rb}\,(A_{rr}^{-1}A^{l}_{rb}e) + \text{diag}^{r}_{rb}\,(A_{rr}^{-1}A^{r}_{rb}e)), \tag{15}$$

$$T_{br} = -(\text{diag}^{l}_{br}\,(e^{T}A^{l}_{br}A_{rr}^{-1}) + \text{diag}^{r}_{br}\,(e^{T}A^{r}_{br}A_{rr}^{-1})), \tag{16}$$

where e is the vector of all ones.

We use (15) and (16) for transfer operators. We take the initial guess for the solution to (1) on a given grid to be the best constant solution

$$x_0 = \frac{b^T e}{e^T A e}\, e, \tag{17}$$

rather than zero. Here $(e^T A e)^{-1}$ is precomputed and calculation of $b^T e$ is very cheap, involving only adds. For our smoother we use red-black line Gauss-Seidel with lines oriented in direction 1. This completes the specification of our basic two-level 2d semicoarsening multigrid method. The adjective semicoarsening is used because the coarse grid is only coarsened in direction 2, while the usual "full coarsening" multigrid involves coarsening in both directions 1 and 2 simultaneously.

The method works particularly well for strongly anisotropic problems. Suppose connections are much stronger in direction 1 than direction 2. Then line Gauss-Seidel is a good iterative method in direction 1. Also, entries in A_{rb} and A_{br} are small, so entries in T_{rb} and T_{br} are small, and by (7), A_c is a good approximation to A_s (and A_{bb}). Conversely, suppose connections are much stronger in direction 2 than direction 1. Then A_{rr} is approximately diagonal, and A^{l}_{rb}, A^{r}_{rb}, A^{l}_{br}, and A^{r}_{br} are well approximated by $\text{diag}^{l}_{rb}\,(A^{l}_{rb})$, $\text{diag}^{r}_{rb}(A^{r}_{rb})$, $\text{diag}^{l}_{br}\,(A^{l}_{br})$, and $\text{diag}^{r}_{br}\,(A^{r}_{br})$, respectively. Thus T_{rb} and T_{br} are very good approximations to $-A_{rr}^{-1}A_{rb}$ and $-A_{br}A_{rr}^{-1}$ and hence the coarse-grid system is very close to the Schur complement.

This good convergence for strongly anisotropic problems is borne out in Tables 2, 3 and 4. (Here *ratio* is the anisotropy ratio in (18).)

Now consider a 3d model problem with a 15-point operator (a three-point operator in direction 3 tensored with a five-point operator in directions 1 and 2). The resulting 15-diagonal matrix is depicted in Fig. 5. Let P be the permutation matrix ordering planes of unknowns red-black. The resulting red-black matrix P^TAP is depicted in Fig. 6.

Again, the same two considerations cause fill in the Schur complement matrix. Our notation to deal with connections in direction 3 is similar to the 2d case. For Fig. 6, A^{l}_{rb} has nonzeros only in diagonals a and b, A^{r}_{rb} has nonzeros only in diagonals c and d, A^{l}_{br} has nonzeros only in diagonals a and b, A^{r}_{rb} has nonzeros only in diagonals g and h. Also, $\text{diag}^{l}_{rb}\,(v)$ (respectively, $\text{diag}^{r}_{rb}\,(v)$, $\text{diag}^{l}_{br}\,(v)$, $\text{diag}^{r}_{br}\,(v)$) has nonzeros only in diagonal a (respectively, c, e, g).

We would again like to use transfer operators (15) and (16). However, now A_{rr} is a five-point operator which is expensive to invert. Hence, following Schaffer [7], we use one cycle of our 2d semicoarsening multigrid method to approximately solve the A_{rr} systems needed in constructing T_{rb} and T_{br}.

We use (17) for initial guesses and approximate red-black Gauss-Seidel plane relaxation for our fine-grid smoothing step, where the 2d semicoarsening multigrid method approximately solves the plane Gauss-Seidel equations. This completes the specification of our basic two-level 3d semicoarsening multigrid method.

Note that a fine-grid 3d seven-point operator forms 15-point operators on the coarser 3d grids, and a fine-grid 2d five-point operator forms nine-point operators on the coarser 2d grids.

$$\left(\begin{array}{ccccccccccccccccccccccccccc}
x&x&&x&&&&& & a&b&&b&&&&& & &&&&&&&& \\
x&x&x&&x&&&& & b&a&b&&b&&&& & &&&&&&&& \\
&x&x&&&x&&& & &b&a&&&b&&& & &&&&&&&& \\
x&&&x&x&&x&& & b&&&a&b&&b&& & &&&&&&&& \\
&x&&x&x&x&&x& & &b&&b&a&b&&b& & &&&&&&&& \\
&&x&&x&x&&&x & &&b&&b&a&&&b & &&&&&&&& \\
&&&x&&&x&x& & &&&b&&&a&b& & &&&&&&&& \\
&&&&x&&x&x&x & &&&&b&&b&a&b & &&&&&&&& \\
&&&&&x&&x&x & &&&&&b&&b&a & &&&&&&&& \\
e&f&&f&&&&& & x&x&&x&&&&& & g&h&&h&&&&& \\
f&e&f&&f&&&& & x&x&x&&x&&&& & h&g&h&&h&&&& \\
&f&e&&&f&&& & &x&x&&&x&&& & &h&g&&&h&&& \\
f&&&e&f&&f&& & x&&&x&x&&x&& & h&&&g&h&&h&& \\
&f&&f&e&f&&f& & &x&&x&x&x&&x& & &h&&h&g&h&&h& \\
&&f&&f&e&&&f & &&x&&x&x&&&x & &&h&&h&g&&&h \\
&&&f&&&e&f& & &&&x&&&x&x& & &&&h&&&g&h& \\
&&&&f&&f&e&f & &&&&x&&x&x&x & &&&&h&&h&g&h \\
&&&&&f&&f&e & &&&&&x&&x&x & &&&&&h&&h&g \\
&&&&&&&& & c&d&&d&&&&& & x&x&&x&&&&& \\
&&&&&&&& & d&c&d&&d&&&& & x&x&x&&x&&&& \\
&&&&&&&& & &d&c&&&d&&& & &x&x&&&x&&& \\
&&&&&&&& & d&&&c&d&&d&& & x&&&x&x&&x&& \\
&&&&&&&& & &d&&d&c&d&&d& & &x&&x&x&x&&x& \\
&&&&&&&& & &&d&&d&c&&&d & &&x&&x&x&&&x \\
&&&&&&&& & &&&d&&&c&d& & &&&x&&&x&x& \\
&&&&&&&& & &&&&d&&d&c&d & &&&&x&&x&x&x \\
&&&&&&&& & &&&&&d&&d&c & &&&&&x&&x&x
\end{array}\right)$$

FIG. 5. *Three-dimensional matrix nonzero structure* $(n1 = n2 = n3 = 3)$.

$$\left(\begin{array}{ccccccccccccccccccccccccccc}
x&x&&x&&&&& & &&&&&&&& & a&b&&b&&&&& \\
x&x&x&&x&&&& & &&&&&&&& & b&a&b&&b&&&& \\
&x&x&&&x&&& & &&&&&&&& & &b&a&&&b&&& \\
x&&&x&x&&x&& & &&&&&&&& & b&&&a&b&&b&& \\
&x&&x&x&x&&x& & &&&&&&&& & &b&&b&a&b&&b& \\
&&x&&x&x&&&x & &&&&&&&& & &&b&&b&a&&&b \\
&&&x&&&x&x& & &&&&&&&& & &&&b&&&a&b& \\
&&&&x&&x&x&x & &&&&&&&& & &&&&b&&b&a&b \\
&&&&&x&&x&x & &&&&&&&& & &&&&&b&&b&a \\
&&&&&&&& & x&x&&x&&&&& & c&d&&d&&&&& \\
&&&&&&&& & x&x&x&&x&&&& & d&c&d&&d&&&& \\
&&&&&&&& & &x&x&&&x&&& & &d&c&&&d&&& \\
&&&&&&&& & x&&&x&x&&x&& & d&&&c&d&&d&& \\
&&&&&&&& & &x&&x&x&x&&x& & &d&&d&c&d&&d& \\
&&&&&&&& & &&x&&x&x&&&x & &&d&&d&c&&&d \\
&&&&&&&& & &&&x&&&x&x& & &&&d&&&c&d& \\
&&&&&&&& & &&&&x&&x&x&x & &&&&d&&d&c&d \\
&&&&&&&& & &&&&&x&&x&x & &&&&&d&&d&c \\
e&f&&f&&&&& & g&h&&h&&&&& & x&x&&x&&&&& \\
f&e&f&&f&&&& & h&g&h&&h&&&& & x&x&x&&x&&&& \\
&f&e&&&f&&& & &h&g&&&h&&& & &x&x&&&x&&& \\
f&&&e&f&&f&& & h&&&g&h&&h&& & x&&&x&x&&x&& \\
&f&&f&e&f&&f& & &h&&h&g&h&&h& & &x&&x&x&x&&x& \\
&&f&&f&e&&&f & &&h&&h&g&&&h & &&x&&x&x&&&x \\
&&&f&&&e&f& & &&&h&&&g&h& & &&&x&&&x&x& \\
&&&&f&&f&e&f & &&&&h&&h&g&h & &&&&x&&x&x&x \\
&&&&&f&&f&e & &&&&&h&&h&g & &&&&&x&&x&x
\end{array}\right)$$

FIG. 6. *Three-dimensional red-black matrix nonzero structure.*

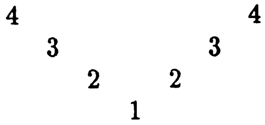


FIG. 7. V-*cycle.*

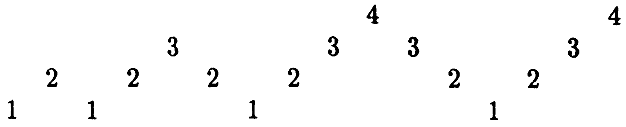


FIG. 8. FMV-*cycle.*

Again this method works particularly well for strongly anisotropic problems. This good convergence for strongly anisotropic problems is borne out in Table 5.

We have used several standard multigrid cycling approaches. The "V-cycle" (Fig. 7) starts on the finest grid with a two-level method and solves the resulting coarser-grid system recursively with another two-level method, and so on until the coarsest-grid system with one unknown is solved directly. The full multigrid V-cycle, or FMV-cycle (Fig. 8), generates an initial guess for the fine-grid system by preceding the fine-grid V-cycle recursively with another V-cycle on the next-finest grid, and so on so that the very first calculation is a direct solve on the coarsest grid. The initial FMV-cycle, or IFMV-cycle, takes the first cycle to be an FMV-cycle and the remaining cycles to be V-cycles.

Our 2d sequential code requires about $30N$ words of storage, where N is the number of unknowns on the fine grid. Our 3d sequential code requires about $148N$ words of storage. This is a large storage requirement for an iterative solver. It can be reduced if the matrix is known to be symmetric, if single precision is acceptable, or if the fine grids are known to be seven-point in three dimensions and five-point in two dimensions, rather than 15-point in three dimensions and nine-point in two dimensions.

3. The parallel method. We have implemented our method on distributed-memory parallel computers of hypercube type by partitioning the problem domain into subdomains and assigning a rectangular subdomain to each node (processor). We have left the number of subdomains in each direction variable, so that we can experiment with all possible rectangular mappings of our line and plane algorithms to our hypercube architecture. Our results in Tables 7–12 indicate experimentation with various node configurations. For each node, we pad local arrays by one at the lower and higher bounds of each array in each direction. The padding areas are used as buffers to exchange boundary information with neighbor nodes. Because the method mainly consists of subtasks of the form

```
for all unknowns in direction $i$
perform task $T$ in directions $j$ and $k$
```

or

```
for all unknowns in direction $i$
for all unknowns in direction $j$
perform task $T$ in direction $k$
```

where (i, j, k) = some permutation of $(1, 2, 3)$, the parallelization tasks required in directions 1, 2, and 3 are essentially independent. Boundary data are exchanged as needed before loops. In the following example, the call to *pad* exchanges boundary data for array *b* in direction 2, as needed by loop 10.

```
call pad(2,b)
do 10 i3 = i3l,i3h
do 10 i2 = i2l,i2h
do 10 i1 = i1l,i1h
   a(i1,i2,i3) = b(i1,i2-1,i3) + b(i1,i2+1,i3)
10 continue
```

A major aspect of parallel implementation is treatment of coarse grids with no unknowns for some nodes (such grids are designated "below C-level" in Briggs, Hart, McCormick, and Quinlan [4]). FMV-cycles tend to be relatively more expensive than V-cycles in parallel, because a greater portion of their computations are carried on below C-level. Both Hempel and Schuller [10] and Briggs et al. [4] use the "sleeping nodes" approach to C-level, in which nodes which are allocated no unknowns are set idle and then re-awakened when they rise above C-level again. We have taken two different approaches to C-level in our current 2d and 3d codes.

2d. Our current 2d code uses a 1d global approach. When the grid goes below C-level in direction *i*, global copies of the current problem are distributed to all nodes in direction *i*. The distribution is performed using a 1d version of a global concatenation operation (GCOL in [11]). Each node then proceeds to handle the global problem below C-level. No more internode communication is needed in this direction until computations resume above C-level. The penalty, of course, is that sub-C computations are duplicated many times, and extra communication is required for the global broadcasts.

This 1d global approach does not scale well. As the number of nodes grows large, so does the size of the sub-C level, and hence the parallel CPU time. However, for a moderate number of nodes good parallel efficiency is achieved, balancing the size of the sub-C level with the communication performed above the sub-C level. This approach can efficiently accommodate up to the order of $\sqrt{N}$ nodes, where $N = n1 \cdot n2$ is the number of fine-grid points.

Solution of the tridiagonal 1d linear systems in our 2d code is consistent with this approach. We solve the tridiagonal systems with line Gauss–Seidel using a version of two-level cyclic reduction (Johnsson [12]). The steps in two-level cyclic reduction are:

1. Forward solve local interior unknowns in each subgrid;
2. Broadcast and solve a small global system for boundary unknowns in each subgrid (1d global concatenate in direction 1);
3. Backsolve local interior unknowns in each subgrid (Fig. 9).

The small global system is distributed exactly at C-level. We combine the two-level cyclic reduction approach with the burn-at-both-ends (BABE) idea, assigning an upper and a lower node to each subgrid and eliminating unknowns for the two nodes for each subgrid in parallel (Fig. 10).

3d. Our current 3d code handles C-level using a 1d local duplication approach. As a node goes below C-level, it copies the local problem from a neighboring node. In this way all nodes stay busy working on systems of small size. This approach is closely related to the superconvergent parallel multigrid approach [9]. However, identical rather than different small systems are solved, so that the numerical answer is independent of the number of nodes.

$$\begin{pmatrix}
n & o & & & f & & & & & & & \\
o & n & o & & f & & & & & & & \\
& o & n & o & f & & & & & & & \\
& & o & N & N & & & & & & & \\
& & & N & N & o & & & F & & & \\
& & & f & o & n & o & & f & & & \\
& & & f & & o & n & o & f & & & \\
& & & F & & & o & N & N & & & \\
& & & & & & & N & N & o & & \\
& & & & & & & f & o & n & o & \\
& & & & & & & f & & o & n & o \\
& & & & & & & f & & & o & n
\end{pmatrix}$$

n : nonzero

o : created zero

f : fill-in

N, F : part of small global system for boundary unknowns

FIG. 9. *Two-level cyclic reduction.*

$$\begin{pmatrix}
n & c & & & & & & \\
a & n & c & & & & & \\
& a & n & c & & & & \\
& & a & N & N & & & \\
& & & N & N & b & & \\
& & & & d & n & b & \\
& & & & & d & n & b \\
& & & & & & d & n
\end{pmatrix}$$

n : nonzero

a : created zero in initial stage for upper node

b : created zero in initial stage for lower node

N : part of small shared system for center unknowns

c : created zero in final stage for upper node

d : created zero in final stage for lower node

FIG. 10. *Burn-at-both-ends elimination.*

As the grid gets coarser, the increment between neighboring nodes grows. Thus on the next-to-coarsest grid there are $n$node/2 copies of small system 1, $n$node/2 copies of small system 2, and a nodal increment to neighbors of $n$node/2. This situation is illustrated in Fig. 11, where there are 8 nodes and 16 fine-grid unknowns, and ninc = nodal increment to neighbors.

Solution of the tridiagonal 1d linear systems in our 3d code is consistent with our local duplication approach: We use the 1d semicoarsening multigrid direct solve outlined in § 2, which is equivalent to fully recursive cyclic reduction.

The local duplication approach scales fairly well. As a number of nodes grows large, the amount of work done by each node is proportional to log (N), the number of levels. However, in practice there is a penalty for using this approach, as well as the "sleeping nodes" approach: internode communication must continue at all sub-C levels.

	node 1	node 2	node 3	node 4	node 5	node 6	node 7	node 8
ninc = 1	*ab*	*cd*	*ef*	*gh*	*ij*	*kl*	*mn*	*op*
coarsen	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*
copy								
ninc = 1								
coarsen		*a*		*b*		*c*		*d*
copy	*a*		*b*		*c*		*d*	
ninc = 2								
coarsen			*a*	*a*			*b*	*b*
copy	*a*	*a*			*b*	*b*		
ninc = 4								
coarsen					*a*	*a*	*a*	*a*
copy	*a*	*a*	*a*	*a*				
ninc = 8								

FIG. 11. *Duplication of neighboring systems.*

Our 2d hypercube code requires about $56N$ words of storage, where N is the number of unknowns on the fine-grid local subdomain. Our 3d sequential code requires about $136N$ words of storage. So far, storage has been the bottleneck in determining the size of problems we can run on hypercubes.

4. Convergence results (2d). Our 2d test problems are Neumann model problems

$$-(u_1)_1-(\text{ratio}\cdot u_2)_2=1 \tag{18}$$

in the unit square,

$$\frac{\partial u}{\partial n}=0 \tag{19}$$

on the boundary, with the matrix made nonsingular by doubling the first main diagonal entry.

The next few tables depict residual reduction factors for various runs. The initial guess x_0 for x is taken to be 0. The residual reduction factor is averaged over five iterations as

$$(\|b-Ax_5\|/\|b\|_2)^{1/5}.$$

Factors less than .005 or so are affected by roundoff and may actually represent even faster convergence.

The effect of (17) is seen for ratio = 1, FMV-cycles in Table 1. Based on these results, (17) is used in all other runs reported.

V-cycle, IFMV-cycle (one FMV-cycle followed by four V-cycles), and FMV-cycle residual reduction factors for different values of ratio are presented in Tables 2, 3, and 4, respectively. Residual reduction factors are generally largest for ratio near one and smaller for ratio either very large or very small. Since an FMV-cycle costs roughly twice as much as a V-cycle, the IFMV-cycle generally seems most efficient, with little more cost than a V-cycle and with residual reduction factors intermediate between those of V and FMV.

TABLE 1
Effect of (17).

$n1 = n2$	$x_0 = 0$	(17)
10	.054	.016
20	.070	.020
40	.14	.024
80	.24	.027

TABLE 2
Two-dimensional V-*cycle residual reduction factors.*

Ratio	$n1 = n2 = 10$	$n1 = n2 = 20$	$n1 = n2 = 40$	$n1 = n2 = 80$
1000.	.033	.079	.097	.11
100.	.078	.090	.11	.12
64.	.075	.091	.11	.12
32.	.072	.090	.10	.12
16.	.069	.086	.10	.11
8.	.061	.076	.090	.10
4.	.055	.068	.085	.099
2.	.055	.068	.084	.099
1.	.054	.067	.083	.098
.5	.053	.068	.082	.096
.25	.056	.065	.079	.093
.125	.050	.061	.077	.090
.0625	.035	.065	.073	.087
.03125	.017	.056	.071	.085
.015625	.0055	.039	.073	.080
.01	.0036	.025	.067	.078
.001	.0052	.0068	.0092	.047

TABLE 3
Two-dimensional IFMV-*cycle residual reduction factors.*

Ratio	$n1 = n2 = 10$	$n1 = n2 = 20$	$n1 = n2 = 40$	$n1 = n2 = 80$
1000.	.0078	.017	.020	.023
100.	.028	.030	.037	.043
64.	.029	.034	.042	.048
32.	.033	.041	.048	.055
16.	.040	.048	.056	.064
8.	.045	.053	.061	.070
4.	.046	.053	.061	.071
2.	.044	.051	.059	.068
1.	.042	.050	.057	.066
.5	.042	.048	.056	.064
.25	.040	.046	.054	.062
.125	.032	.045	.052	.060
.0625	.020	.044	.049	.058
.03125	.0083	.036	.048	.056
.015625	.0030	.024	.047	.050
.01	.0035	.016	.043	.030
.001	.0051	.0069	.0092	.030

TABLE 4
Two-dimensional FMV-*cycle residual reduction factors.*

Ratio	$n1 = n2 = 10$	$n1 = n2 = 20$	$n1 = n2 = 40$	$n1 = n2 = 80$
1000.	.0055	.0080	.010	.014
100.	.0037	.0051	.0068	.0091
64.	.0034	.0044	.0059	.0085
32.	.0029	.0041	.0057	.0078
16.	.0048	.0056	.0065	.0077
8.	.0090	.010	.012	.014
4.	.014	.016	.019	.021
2.	.017	.020	.023	.026
1.	.017	.019	.022	.025
.5	.014	.017	.019	.022
.25	.012	.015	.017	.019
.125	.011	.013	.015	.017
.0625	.010	.011	.013	.015
.03125	.0072	.011	.013	.015
.015625	.0037	.012	.014	.015
.01	.0034	.011	.012	.015
.001	.0052	.0068	.0092	.015

3d. We use the test set of reservoir simulation problems from Dendy et al. [8]. The problems are symmetric except for Problem 7, which was symmetrized before solution in [8]. Several of the problems have large coefficient discontinuities. Average residual reduction factors are given in Table 5 for the results from [8] and for our code running with V-cycles, IFMV-cycles, and FMV-cycles. In these runs our code used two-dimensional FMV-cycles to do the red-black smoothing by planes.

Our results for Problems 6 and 7 depend strongly on ordering of axes. Other solvers that do particularly well on anisotropic problems share this property, e.g., nested factorization [2]. Table 6 presents average residual reduction factors for an

TABLE 5
Three-dimensional residual reduction factors.

Problem	Dendy V	V	IFMV	FMV
1	.27	.044	.027	.012
2	.27	.059	.025	.0022
3	.27	.11	.011	.0070
4	.25	.058	.028	.0083
5a	.09	.21	.16	.054
5b	.02	.27	.22	.091
6	.29	.24	.20	.10
7	.27	.37	.29	.19

TABLE 6
Three-dimensional residual reduction factors.

Ordering:	123	132	213	231	312	321
Problem 6	.43	.13	.43	.31	.10	.31
Problem 7	.19	>1	>1	>1	>1	.16

FMV-cycle with our code for the various axis orderings. The results in Table 5 are given for ordering 312 for Problem 6.

5. Parallel timing results (2d). Table 7 (respectively, 8) presents timing results for a five-iteration V-cycle (respectively, FMV-cycle) of the 2d code on the iPSC/2 hypercube at Oak Ridge with 64 Intel scalar 386 nodes. CPU is seconds of dedicated wall clock hypercube CPU time, and efficiency is

$$\frac{\text{CPU}_1}{n\text{nodes} \cdot \text{CPU}_{n\text{nodes}}}.$$

The OLM optimizing compiler was used. An S indicates that the run ran out of storage and the CPU time was estimated based on other runs. The $n$node1 and $n$node2 columns show the numbers of nodes in directions 1 and 2, respectively. The shown values resulted in the smallest CPU time for that value of $n$nodes = $n$node1 · $n$node2. The fact that this usually occurred with an approximately square configuration of nodes indicates that the additional arithmetic incurred by using cyclic reduction in direction 1 is not a significant factor, and that communication costs comprise the bulk of the parallel inefficiency. The * values achieve about 50 percent efficiency. This is obtained with up to about $n1/8$ nodes, scaling as expected. The ·'s represent runs that were not made because of configuration or storage constraints.

3d. Table 9 (respectively, 11) presents timing results for a five-iteration V-cycle (respectively, FMV-cycle) of the 3d code on the iPSC/2 64-node hypercube at Oak Ridge. Table 10 (respectively, 12) presents the resulting efficiencies. For V-cycles, scaling behavior is more uniform than in the 2d case, in that CPU time for a given grid consistently decreases as more nodes are used. Unfortunately, efficiencies are not as high as in the 2d case. About 30 percent efficiency for V-cycles is achieved with

TABLE 7
Two-dimensional V-cycle hypercube timing results.

$n1 = n2$	$n$nodes	$n$node1	$n$node2	CPU	Efficiency
32	1	1	1	3.8	1.0
	2	2	1	2.8	.7
	4	2	2	2.1	.5*
	8	4	2	1.9	.3
	16	4	4	1.6	.1
	32	8	4	1.9	.1
	64	8	8	2.0	·
64	1	1	1	14.9	1.0
	2	2	1	9.7	.8
	4	2	2	5.9	.6
	8	4	2	4.8	.4*
	16	4	4	3.8	.2
	32	8	4	3.8	.1
	64	8	8	4.8	·
128	1	·	·	59.6S	·
	2	·	·	·	·
	4	2	2	20.7	.7
	8	4	2	15.0	.5*
	16	4	4	10.8	.3
	32	8	4	9.7	.2
	64	8	8	10.3	.1

TABLE 8
Two-dimensional FMV-*cycle hypercube timing results.*

$n1 = n2$	$n$nodes	$n$node1	$n$node2	CPU	Efficiency
32	1	1	1	7.9	1.0
	2	2	1	5.4	.7
	4	2	2	4.7	.4*
	8	4	2	4.4	.2
	16	4	4	4.0	.1
	32	8	4	4.0	·
	64	8	8	4.2	·
64	1	1	1	30.9	1.0
	2	2	1	18.1	.9
	4	2	2	12.5	.6
	8	4	2	9.9	.4*
	16	8	2	7.7	.3
	32	8	4	6.8	.1
	64	16	4	7.3	·
128	1	·	·	123.6*S*	·
	2	·	·	·	·
	4	2	2	41.2	.8
	8	4	2	25.9	.6
	16	8	2	17.8	.4*
	32	8	4	14.6	.3
	64	16	4	13.4	.1

TABLE 9
Three-dimensional hypercube V-*cycle* CPU *seconds for given node configuration.*

$n1 = n2$	$n3$	8×8×1	4×4×2	4×4×1	2×2×2	2×2×1	1×1×1
32	1	4.6	·	4.9	·	5.0	5.3
32	2	15.8	16.6	17.1	18.4	18.9	29.6*S*
32	4	27.8	28.4	31.3	32.4	38.7	70.0*S*
64	1	6.5	·	7.5	·	9.6	21.2*S*
64	2	22.6	26.5	27.6	·	·	118.4*S*
64	4	41.0	46.3	54.8	·	·	280.0*S*
128	1	9.3	·	13.2	·	·	84.8*S*

TABLE 10
Three-dimensional hypercube V-*cycle parallel efficiencies.*

$n1 = n2$	$n3$	8×8×1	4×4×2	4×4×1	2×2×2	2×2×1	1×1×1
32	1	<.1	·	.1	·	.3	1.0
32	2	<.1	.1	.1	.2	.4	·
32	4	<.1	.1	.1	.3	.5	·
64	1	.1	·	.2	·	.6	·
64	2	.1	.1	.3	·	·	·
64	4	.1	.2	.3	·	·	·
128	1	.1	·	.4	·	·	·

TABLE 11
Three-dimensional hypercube FMV-*cycle* CPU *seconds for given node configuration.*

$n1 = n2$	$n3$	$8 \times 8 \times 1$	$4 \times 4 \times 2$	$4 \times 4 \times 1$	$2 \times 2 \times 2$	$2 \times 2 \times 1$	$1 \times 1 \times 1$
32	1	14.3	·	14.2	·	13.6	11.5
32	2	63.7	64.1	66.5	64.0	65.7	74.4*S*
32	4	149.0	148.1	150.4	150.4	161.7	211.6*S*
64	1	22.9	·	23.8	·	27.3	46.0*S*
64	2	102.2	110.4	114.5	·	·	297.6*S*
64	4	242.3	257.5	284.3	·	·	846.4*S*
128	1	35.6	·	45.0	·	·	184.0*S*

TABLE 12
Three-dimensional hypercube FMV-*cycle parallel efficiencies.*

$n1 = n2$	$n3$	$8 \times 8 \times 1$	$4 \times 4 \times 2$	$4 \times 4 \times 1$	$2 \times 2 \times 2$	$2 \times 2 \times 1$	$1 \times 1 \times 1$
32	1	<.1	·	.1	·	.2	1.0
32	2	<.1	<.1	.1	.1	.3	·
32	4	<.1	<.1	.1	.2	.3	·
64	1	<.1	·	.1	·	.4	·
64	2	<.1	.1	.2	·	·	·
64	4	.1	.1	.2	·	·	·
128	1	.1	·	.3	·	·	·

$\sqrt{n1 \cdot n2 \cdot n3}/8$ nodes, with relatively higher efficiencies for finer grids. The lower efficiency in three dimensions than in two dimensions may be partly due to the treatment of coarse grids, and partly due to the larger surface-to-volume ratio for 3d grids than 2d grids, with finer grids needed to get the same ratio of interior work to boundary work.

6. Plans. Plans include trying different coarse-grid approaches in three dimensions, timing our codes on the new generation of Intel RX hypercubes with faster communication networks, and writing a version of the codes for the Connection Machine.

We are grateful to Joel Dendy at Los Alamos for access to his sequential semi-coarsening multigrid codes and for many helpful conversations; and to Mary Wheeler, Lawrence Cowsar, and Ashok Chilakapati at Rice University for helpful conversations.

REFERENCES

[1] R. E. ALCOUFFE, A. BRANDT, J. E. DENDY, JR., AND J. W. PAINTER, *The multi-grid method for the diffusion equation with strongly discontinuous coefficients*, SIAM J. Sci. Statist. Comput., 2 (1981), pp. 430-454.

[2] J. R. APPLEYARD AND I. M. CHESHIRE, *Nested factorization*, Paper SPE 12264, presented at the Seventh Society of Petroleum Engineers Symposium on Reservoir Simulation, Nov. 16-18, 1983.

[3] A. BEHIE AND P. A. FORSYTH, JR., *Multi-grid solution of the pressure equation in reservoir simulation*, Soc. Pet. Engrg. J., 23 (1983), pp. 623-632.

[4] B. BRIGGS, L. HART, S. MCCORMICK, AND D. QUINLAN, *Multigrid methods on a hypercube*, in Multigrid Methods: Theory, Applications, and Supercomputing, S. F. McCormick, ed., Marcel Dekker, Inc., 1988.

[5] J. E. DENDY, JR., *Black box multigrid*, J. Comp. Phys., 48 (1982), pp. 366-386.

[6] ———, JR., *Two multigrid methods for three-dimensional problems with discontinuous and anisotropic coefficients*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 673-685.

[7] J. E. DENDY, JR., private communication.

[8] J. E. DENDY, JR., S. F. MCCORMICK, J. W. RUGE, T. F. RUSSELL, AND S. SCHAFFER, *Multigrid methods for three-dimensional petroleum reservoir simulation*, Paper SPE 18409, presented at the Society of Petroleum Engineers Symposium on Reservoir Simulation, Houston, TX, Feb. 6-8, 1989.

[9] P. O. FREDERICKSON AND O. A. MCBRYAN, *Parallel superconvergent multigrid*, in Multigrid Methods: Theory, Applications, and Supercomputing, S. F. McCormick, ed., Marcel Dekker, Inc., 1988.

[10] R. HEMPEL AND A. SCHULLER, *Experiments with parallel multigrid algorithms using the SUPRENUM communications subroutine library*, GMD Studie 141, St. Augustin, April 1988.

[11] iPSC/2 Programmer's Reference Manual, Intel Corporation, 1988.

[12] S. L. JOHNSSON, *Solving tridiagonal systems on ensemble architectures*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 354-392.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 6, pp. 1330–1346, November 1992

004

EFFICIENT PARALLEL ALGORITHMS FOR SOLVING INITIAL-BOUNDARY VALUE AND TIME-PERIODIC PARABOLIC PARTIAL DIFFERENTIAL EQUATIONS*

STEFAN VANDEWALLE† AND ROBERT PIESSENS†

Abstract. The numerical solution of a parabolic partial differential equation is usually calculated by a timestepping method. This precludes the efficient use of vectorization and parallelism if the problem to be solved on each time level is not very large. In this paper an algorithm that overcomes the limitations of the standard marching schemes by solving the problem at all the time levels simultaneously is discussed. The method is applicable to linear and nonlinear problems on arbitrary domains. It can be used to solve initial-boundary value problems as well as time-periodic equations. We have implemented the method on an Intel iPSC/2-VX hypercube. The numerical properties of the method are illustrated by two numerical examples and its performance is compared to that of the best standard solvers.

Key words. parallel processing, waveform relaxation, multigrid, parabolic partial differential equation

AMS(MOS) subject classifications. 65M20, 65N20, 65W05

1. Introduction. The class of parabolic partial differential equations plays a very important role in many branches of science and engineering. As a result, considerable effort has been expended in formulating numerical solution methods that are both accurate and efficient. Due to the advent of parallel computers these well-known algorithms must be reconsidered, judged, and eventually restructured, to take advantage of the facilities offered by the new hardware.

In a previous study, we analyzed and compared the parallel characteristics of several classical techniques (see [16]). We illustrated that standard parabolic marching schemes can only be parallelized efficiently for problems that are *large* enough, i.e., the number of unknowns (and, consequently, the arithmetic complexity) per processor at each time level is large enough to outweigh communication. In that case, the best sequential algorithm will run efficiently and new parallel algorithms are not really needed. For relatively *small* problems, i.e., problems with few unknowns per time level or problems solved on large-scale parallel machines, only explicit methods retain some parallel efficiency. They suffer, however, from a severe stability constraint, which necessitates the use of very small timesteps and makes them numerically unattractive for solving problems on fine meshes. The best standard methods, e.g., implicit discretization with multigrid solution of the system of unknowns at each time level, perform totally unsatisfactorily.

New algorithms are therefore needed for solving parabolic problems on large-scale parallel machines. These algorithms should either improve the numerical quality of the explicit methods, or increase the parallel efficiency of the implicit methods. The former is explored in a series of papers by Evans (see, e.g., [3] and [4]) and in a paper by Rodrigue [12]. Both approaches essentially lead to new explicit methods with extended stability regions. An improvement in the implicit methods may be achieved by operating on several, or on all, time levels at once. With the *windowed relaxation* methods proposed by Saltz and Naik, the standard Jacobi and successive overrelaxation

* Received by the editors November 14, 1990; accepted for publication (in revised form) December 2, 1991.

† Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Heverlee, Belgium. The second author is a senior research assistant of the National Science Foundation, Belgium.

(SOR) methods are extended to operate on a window of time levels [13]. In the *parallel timestepping* method of Womble, one or more processors are assigned to each time level [21]. While the solution is being calculated on one time level, other processors update the approximation on the subsequent time levels. In 1984, the multigrid technique was extended by Hackbusch to solve a parabolic equation on several time levels simultaneously (see [6]). A parallel implementation of this *parabolic multigrid* method was presented by Bastian, Burmeister, and Horton in [2].

In this paper, we discuss a technique that belongs to the latter class of methods. It is different from the previous approaches in that it can be defined without explicit reference to any time discretization technique or to any time levels. The method is based on *waveform relaxation*, which is briefly discussed in § 2. We discuss the application of waveform relaxation for solving initial-boundary value problems in § 3. In § 4, the solution of time-periodic parabolic equations is considered. We have implemented the method on an Intel hypercube. Some implementation aspects are discussed in § 5. The communication and arithmetic complexities are calculated in § 6. In § 7, we illustrate the method by two examples, and we compare its performance to that of a parallel implementation of the best standard methods. The results are summarized in the final section.

2. The waveform relaxation method. Waveform relaxation, also called dynamic iteration or Picard–Lindelöf iteration, is an iterative solution technique for systems of ordinary differential equations [10], [11], [20]. It differs from any other method in that the iteration in the algorithm is defined on functions. The successive iterates are defined as the solutions of a sequence of systems of differential equations. These systems may be derived similarly to the way in which the systems of equations are derived for the classical relaxation methods.

Consider the following linear, constant-coefficient system of d first-order differential equations:

$$\frac{d}{dt}U+LU=F, \quad \text{with } U(t_0)=U_0 \quad (L\in\mathbb{R}^{dxd};\ U(t), F(t)\in\mathbb{R}^d). \tag{2.1}$$

The application of a waveform relaxation step to $U^{(n)}$, an approximation of the solution U, corresponds to calculating the solution $U^{(n+1)}$ of the following equation:

$$\frac{d}{dt}U^{(n+1)}+NU^{(n+1)}=MU^{(n)}+F, \quad \text{with } U^{(n+1)}(t_0)=U_0 \quad (N, M\in\mathbb{R}^{dxd}). \tag{2.2}$$

The matrices N and M are the so-called splitting matrices and satisfy $L=N-M$. In the case of Jacobi, Gauss-Seidel, or SOR waveform relaxation, these matrices are identical to the splitting matrices used for solving the stationary problem $LU=F$ by the corresponding standard relaxation methods. With the above splittings, each of the d differential equations can be solved separately, one after the other. As such, the method is very similar to the iterative techniques for solving algebraic systems, except that each variable is a function of time rather than a scalar unknown. The idea immediately extends to variable-coefficient and nonlinear differential equations. It is straightforward to determine waveform equivalents to the familiar nonlinear relaxation methods. Furthermore, it may be advantageous to solve for the variables in groups of unknowns, which leads to a waveform extension of the block-relaxation procedures.

The waveform idea has proven very effective in solving the systems of differential equations that describe the behavior of large, complex, very large scale integration (VLSI) devices (see [20]). Its success is to a large extent due to the fact that it is a multirate integration method. Each equation or each block of equations may be solved independently with a timestep that reflects the behavior of the corresponding variables. A second reason for its success in the above application has to do with the loosely coupled nature of the governing equations. The speed of the Gauss-Seidel relaxation depends on the order in which the functions are updated. Because of the loose coupling, efficient orderings may be derived from the system dependency graph (see [8]). Finally, we should also mention the advantages of conceptual simplicity and ease of parallelization.

A theoretical analysis of the linear iteration (2.2) is presented in the papers of Miekkala and Nevanlinna [10], [11]. In [20], one of the basic papers on waveform relaxation in the electrical engineering literature, uniform convergence is proven for nonlinear systems. Further convergence results are given in [8], in which the relation is established between the number of iterations and the accuracy order, i.e., the number of correct terms in the Taylor expansion of a partially converged solution.

3. Waveform relaxation applied to initial-boundary value problems.

We consider the following linear parabolic initial-boundary value problem

$$\frac{\partial u}{\partial t}+Lu=f_1, \qquad (x,t)\in\Omega\times[t_0,t_f], \tag{3.1a}$$

$$Bu=f_2, \qquad (x,t)\in\partial\Omega\times[t_0,t_f], \tag{3.1b}$$

$$u(x,t_0)=u_0, \qquad x\in\Omega, \tag{3.1c}$$

where $\Omega\subset\mathbb{R}^n$, L is a uniformly elliptic linear operator, and B is the boundary operator. After spatial discretization and incorporation of the boundary conditions, the parabolic problem is transformed into a system of ordinary differential equations (ODEs) with one equation defined at each grid point.

3.1. Standard waveform relaxation.

The waveform relaxation algorithm may be applied to solving the semidiscretized parabolic equation. For instance, in the case of a five-point finite-difference discretization of the heat equation on a regular mesh with mesh spacing h, one iteration of the Jacobi algorithm would read as follows:

for all grid points (x_i, y_j):

$$\text{solve } \frac{d}{dt}u_{ij}^{(n+1)}=\frac{1}{h^2}\left(u_{i+1,j}^{(n)}+u_{i-1,j}^{(n)}-4u_{ij}^{(n+1)}+u_{i,j+1}^{(n)}+u_{i,j-1}^{(n)}\right) \quad \text{with } u_{ij}^{(n+1)}(t_0)=u_{ij,0}.$$

At each grid point a simple first-order differential equation is solved. This can be done by using any standard, stiff ODE integrator, possibly combined with variable-timestep variable-order techniques.

Attempts to use waveform relaxation in the way described above to solve parabolic problems have not led to a satisfactory algorithm. This is due to the slow convergence of the method, which, in turn, is due to the strong coupling between the grid-point variables. For the semidiscretized heat equation, it was shown in [10] that the convergence rates of the Jacobi, Gauss-Seidel, and SOR schemes are of order $1-O(h^2)$. (Strictly speaking, this result is only true in the case of infinite time intervals, as the *asymptotic* convergence rate equals zero on any finite time interval. However, for sufficiently long time intervals, the actual convergence behavior is adequately described by the above formula.) The convergence is thus rapidly deteriorating with an increasing number of grid lines and, unfortunately, successive overrelaxation does not significantly improve convergence characteristics.

3.2. Multigrid waveform relaxation. The convergence can be accelerated if the waveform algorithm is combined with the multigrid idea. The resulting algorithm was published by Lubich and Ostermann in [9] and rediscovered by the current authors [14], [15]. The multigrid method is extended to time-dependent problems in essentially the same way as the classical relaxation methods were extended. Each of the standard multigrid operations is replaced by a similar operation defined to operate on functions.

- *Smoothing* is performed by applying one or more damped Jacobi or Gauss-Seidel waveform relaxations of the form (2.2). In the latter case a red-black or any multicolor ordering can be used. Smoothing rates for these relaxations are defined in [9].
- The *defect* of an approximation $U^{(n+1)}$ is defined and calculated as follows:

$$D := \frac{d}{dt} U^{(n+1)} + LU^{(n+1)} - F = M(U^{(n)} - U^{(n+1)}), \tag{3.2}$$

where $U^{(n)}$ and $U^{(n+1)}$ are two successive iterates obtained with the smoothing procedure.

- *Restriction, prolongation, and correction* are calculated with identical formulae as in the elliptic case. However, these formulae now operate on functions. By way of illustration, we formulate the one-dimensional waveform full-weighting restriction operator

$$u_I(t) = (u_{i+1}(t) + 2u_i(t) + u_{i-1}(t))/4, \tag{3.3}$$

where $u_I(t)$ and $u_i(t)$ are corresponding grid-point functions on the coarse and the fine grid.

With the above changes to the elliptic multigrid algorithm, we may immediately state the waveform equivalent of the *multigrid correction scheme*, which is the usual variant for solving linear problems. Let G_i, $i = 0, 1, \cdots, k$, be the hierarchy of grids required in the multigrid procedure, with G_k the finest grid and G_0 the coarsest grid. Equation (2.1) or, equivalently, $dU_k/dt + L_k U_k = F_k$, with $U_k(t_0) = U_{k,0}$, is solved by iteratively calling the following procedure with the parameters k, F_k and an approximation of U_k.

```
PROCEDURE mgm ($k$, $F_k$, $U_k$)
    if $k = 0$ solve the coarse-grid problem $(dU_0/dt) + L_0 U_0 = F_0$ exactly
    else
        —perform $\nu_1$ smoothing operations (e.g., by red-black waveform
          relaxation)
        —compute the defect: $D_k := (dU_k/dt) + L_k U_k - F_k$
        —project the defect on $G_{k-1}$: $F_{k-1} := I_k^{k-1} D_k$
        —solve on $G_{k-1}$: $(dU_{k-1}/dt) + L_{k-1} U_{k-1} = F_{k-1}$ with $U_{k-1}(t_0) = 0$
            repeat $\gamma_k$ times mgm ($k-1$, $F_{k-1}$, $U_{k-1}$), starting with $U_{k-1} := 0$.
        —interpolate the correction to $G_k$ and correct $U_k$: $U_k := U_k - I_{k-1}^k U_{k-1}$
        —perform $\nu_2$ smoothing operations (e.g., by red-black waveform
          relaxation)
    endif
```

The algorithm is completely defined by specifying the grid sequence G_i, $i = 0, \cdots, k$, the discretized operators L_i, the intergrid transfer operators I_{i-1}^i (prolongation) and I_i^{i-1} (restriction), and the nature of the smoothing relaxations, and by assigning a value to ν_1, ν_2, and γ_i. Additionally, the algorithm can be combined with the idea of nested iteration, which leads to the waveform equivalent of the *full multigrid* *method.* The starting approximation on G_i is then derived from the solution obtained on G_{i-1} in the following way:

$$U_i(t)=\bar{I}^i_{i-1}U_{i-1}(t)+(U_{i,0}-\bar{I}^i_{i-1}U_{i-1}(t_0)). \tag{3.4}$$

The interpolation operator $\bar{I}^i_{i-1}$ may be different from the corresponding operator I^i_{i-1} in the multigrid procedure. The former is often biquadratic or bicubic while bilinear interpolation mostly satisfies for the latter. The second term in (3.4) is needed to satisfy the initial condition.

The linear multigrid waveform algorithm is theoretically analyzed in [9]. For a model problem, the authors explicitly calculate convergence rates. They show that convergence characteristics are not quite as good as those for the corresponding elliptic multigrid method, yet are sufficient to obtain typical multigrid convergence speeds. With some obvious modifications, the algorithm may be used for solving systems of parabolic partial differential equations. Furthermore, the multigrid waveform relaxation algorithm can be extended to nonlinear parabolic problems (see [14] and [17]). The nonlinear algorithm is easily derived from the well-known multigrid full approximation scheme.

4. Waveform relaxation applied to time-periodic parabolic problems.

We consider the nonautonomous parabolic problem (3.1a)-(3.1c) where the initial condition (3.1c) is replaced by the periodicity condition

$$u(x,t_0)=u(x,t_f). \tag{4.1}$$

This problem is of considerable importance in various areas of practical interest. A possible solution strategy is based on a dynamic simulation of the studied system, starting from an arbitrary initial condition and lasting until a stable periodic orbit is reached. A second approach uses difference methods where a large system of equations is obtained after discretization. A third approach is based on the shooting method. Finally, a very fast algorithm was presented by Hackbusch, in which the periodic problem is reformulated as an integral equation and solved by the multigrid method of the second kind (see [5]). We briefly review this algorithm here, as it will be compared with a new waveform relaxation-based algorithm.

4.1. Multigrid method of the second kind.

The solution of the linear initial-boundary value problem (3.1a)-(3.1c), restricted to t_f, can be written as the outcome of an affine mapping applied to the initial condition u_0, that is, $u_f=Ku_0+f$. The operator K is a linear integral operator, such that Ku_0 equals $u(x,t_f)$, the solution to (3.1a)-(3.1c) with homogeneous right-hand sides ($f_1=0$ and $f_2=0$), while $f(x)$ equals $u(x,t_f)$, the solution to (3.1a)-(3.1c) with zero initial condition ($u_0=0$). With this notation, the periodicity condition (4.1) becomes a Fredholm integral equation

$$y=Ky+f. \tag{4.2}$$

The unknown function $y(x)$, defined on Ω, is such that the solution of the initial-boundary value problem (3.1a)-(3.1c) with $u_0=y$, is the solution of the time-periodic problem. Equation (4.2) can be solved by the *multigrid method of the second kind.* We refer to [7] for an analysis of this technique and for a discussion of various applications. In a similar way as in the multigrid method for elliptic equations, (4.2) is discretized on a set of grids, G_i, $i=0,\cdots,k$, which results in a set of discrete equations

$$Y_i=K_iY_i+F_i, \quad \text{on } G_i. \tag{4.3}$$

The problem on the fine grid is solved by repeated application of the following procedure to an approximation of Y_k.

```
PROCEDURE mgm_2nd ($k$, $F_k$, $Y_k$)
    if $k=0$ solve $Y_0 = K_0 Y_0 + F_0$ exactly
    else
        —smoothing: $Y_k := K_k Y_k + F_k$
        —compute the defect: $D_k := Y_k - K_k Y_k - F_k$
        —project the defect on $G_{k-1}$: $F_{k-1} := I_k^{k-1} D_k$
        —solve on $G_{k-1}$: $Y_{k-1} = K_{k-1} Y_{k-1} + F_{k-1}$
            repeat 2 times mgm_2nd ($k-1, F_{k-1}, Y_{k-1}$), starting with $Y_{k-1} := 0$.
        —interpolate the correction to $G_k$ and correct $Y_k$: $Y_k := Y_k - I_{k-1}^{k} Y_{k-1}$
    endif
```

No explicit representation of the discretized integral operator K_i is required. Indeed, application of K_i to a function Y_i is equivalent to calculating the solution of one discrete initial-boundary value problem defined on G_i. $K_i Y_i$ may thus be computed by using standard parabolic solvers, such as a timestepping method, or by using the waveform relaxation algorithm of § 3. In [5], the convergence factor of the algorithm with proper choice of the time-integrator is shown to be of the order $O(h^2)$, where h is the fine-grid mesh size. As such, one iteration step is usually sufficient to solve (4.2) to discretization accuracy.

4.2. A waveform relaxation algorithm. Spatial discretization of the partial differential equation (3.1a)-(3.1b) with the periodicity condition (4.1) leads to the following system of ordinary differential equations:

$$\frac{d}{dt} U + LU = F \quad \text{with } U(t_0) = U(t_f). \tag{4.4}$$

This two-point boundary value problem can be solved with a waveform relaxation algorithm that is only slightly different from the algorithm discussed in § 2. The basic iteration step closely resembles (2.2) and is given by

$$\frac{d}{dt} U^{(n+1)} + NU^{(n+1)} = MU^{(n)} + F, \quad \text{with } U^{(n+1)}(t_0) = U^{(n+1)}(t_f). \tag{4.5}$$

Instead of repeatedly solving an ODE of initial value type at each grid point, one repeatedly solves a *periodic differential equation* at each grid point. This boundary value problem may be solved by a discretization method, which results in a sparse linear system. Application of an implicit one-step discretization method, such as the trapezoidal rule or the backward Euler method, leads to an easily solvable, almost bidiagonal system.

The modified waveform relaxation method can be used as such, or it can be integrated as a smoother into the multigrid waveform scheme of § 3. The only algorithmic change is then in the core of the ODE solver. The other multigrid operations remain totally unchanged. Numerical results show that the new method, *periodic multigrid waveform relaxation*, leads to a rapidly converging iteration, with typical multigrid convergence rates. The algorithm was theoretically analyzed in [18], where we proved that the convergence characteristics of the time-periodic iteration are closely related to those of initial-value iteration. There it is shown that convergence of the initial-value algorithm is a sufficient condition to guarantee the convergence of the time-periodic iteration.

5. Implementation aspects.

5.1. Time discretization. Up to this point, the waveform relaxation method has been defined without explicit reference to time discretization. We have not considered data structures or memory requirements either. We briefly address these issues now.

Any stiff integrator can be used to solve the differential equations that arise in the smoothing step of the algorithm. A sophisticated variable-timestep variable-order integrator from the ODEPACK library was used in our early experiments [14]. However, the overheads of stepsize and order selection, reoccurring at each grid point, badly deteriorated the execution times. Consequently, our experience now advocates the use of a simple but fast ODE integrator. Only then can the waveform method remain competitive with the standard techniques. For the examples in the next section, we used the second-order accurate trapezoidal rule (or Crank–Nicolson method) and the backward differentiation method, both with fixed timestep Δt. The same timestep was used for the integrations on the fine grid and on the coarse grids. This choice of time integration precludes taking advantage of waveform relaxation as a multirate integration method. However, it simplifies implementation and, as is explained in § 5.3, it allows for vectorization.

In the waveform method, several functions are associated with each grid point. For these functions, an effective and compact computer representation should be chosen. It should allow for efficient evaluation of the functions and, more importantly, for efficient algebraic manipulation (mainly the summation of functions and multiplication with a scalar). For the latter reason, we have represented each function as a *vector* of function values evaluated at equidistant time levels. We denote the vector length by n_t (number of time levels). The time increment is chosen equal to the time increment used by the integration formula. With the above implementation choices, a correspondence arises between the waveform method and standard timestepping techniques. A careful analysis shows that the set of equations that determines the unknowns in the waveform method is exactly the same as the set of equations that determines the solution when a timestepping method is used with the same discretization formula and the same time increment. Thus the solution of both techniques is identical.

In Fig. 5.1 we depict the set of unknowns for a problem on a one-dimensional domain. In the case of a timestepping procedure the unknowns must be stored in the computer memory on a few time levels only. In the case of waveform relaxation the values at every time level must be available. Storage requirements are therefore very high. However, if storage is a problem, the time interval of interest can be partitioned into several smaller *windows*, which can be solved in sequence (in the case of an initial-boundary value problem).

If we are willing to pay the memory price involved in waveform relaxation, we could wonder whether it is possible to apply multigrid on the whole of the *space-time* grid, i.e., with coarsening, restriction, and interpolation also in the time direction.

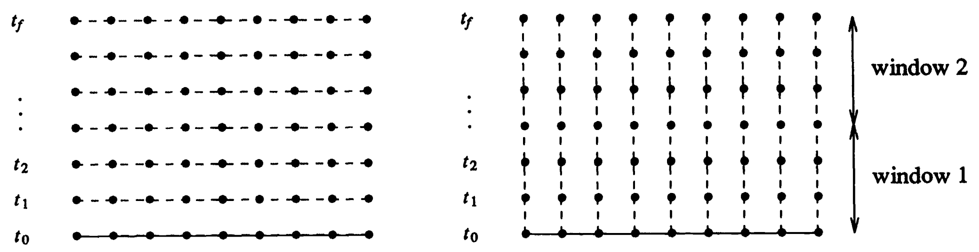


FIG. 5.1. *Distribution of grid points: timestepping method (left) and waveform relaxation (right).*

Although we certainly do not want to claim that such an approach is not possible, we may argue that it is not natural. Consider the following elliptic partial differential equation with an aligned anisotropy in the y-direction:

$$-\frac{\partial^2 u}{\partial x^2}-\varepsilon\frac{\partial^2 u}{\partial y^2}+\frac{\partial u}{\partial y}=f \qquad (\varepsilon \ll 1). \tag{5.1}$$

In the limiting case $\varepsilon=0$, this becomes the one-dimensional heat equation. Problems like (5.1) are often solved with the following multigrid parameter choices. For handling the ε-term, the use of either x-line smoothing or semicoarsening, i.e., coarsening only in the x-direction, is advocated [7, Chap. 10]. The first-order derivative term is handled by backward differencing, and either y-line smoothing or pointwise smoothing from low to high y-values. Thus two "natural" parameter choices arise: standard coarsening combined with x-line smoothing from low to high y-values, and semicoarsening combined with y-line smoothing or pointwise smoothing in the direction of increasing y. In the limiting case, when the elliptic problem becomes a parabolic problem, the first approach is equivalent to standard timestepping (the smoother is an exact solver). The second approach leads to a discrete multigrid waveform relaxation algorithm as defined above, i.e., without coarsening in the time direction.

5.2. Parallelization. We have implemented the algorithms on an Intel iPSC/2-VX hypercube. For a description of this multiprocessor, its hardware characteristics, and various performance benchmarks, see [1]. The implementation of the waveform algorithm and the parallelization of the standard timestepping methods are discussed in detail in our studies [15], [16]. In this section, we only review some of the main issues.

A classical data decomposition is used to evenly distribute the computational workload. The processors are arranged in a rectangular array and are mapped onto the domain of the partial differential equation, which, in our implementation, is restricted to be a rectangle. Each processor is responsible for doing all computations on the grid points in its part of the physical domain. This straightforward partitioning scheme ensures a fairly good load balance, especially when the number of grid points is large. However, some load imbalance cannot be avoided whenever the number of processors in a coordinate direction does not divide the number of grid lines in that direction. During the computation, communication with neighboring processors is needed mainly to update local boundary values. Some other communication strategies are further used to improve the parallel performance. We mention, in particular, the use of an *agglomeration* strategy to reduce the communication complexity of the coarse-grid operations [16]. Agglomeration is especially important for use with multigrid cycles that frequently visit the coarse grids, such as the W-cycle. Although this cycle has superior convergence properties, in a parallel environment without the use of agglomeration, this cycle is easily outdone by the less robust but more efficiently parallelizable V- and F-cycles.

We may now qualitatively analyze the parallel characteristics of the waveform methods. A more quantitative approach is presented in § 6. The arithmetic complexity of the waveform operators increases linearly with the value of n_t. In the same way, the total length of the messages exchanged during the computation is proportional to n_t. The number of messages, however, and the sequential overhead due to program control are independent of n_t. From the high message startup time on most parallel machines (and in particular on the iPSC/2), it is clear that the communication time-to-calculation time ratio will decrease with increasing function length. The larger the number of time levels calculated simultaneously, the better the parallel efficiency will be.

5.3. Vectorization. The use of vector processors for executing the multigrid waveform algorithm may result in a substantial reduction of computing time. Indeed, most of the operators can be expressed as simple arithmetic operations on functions, i.e., on *vectors* (see, e.g., the restriction operator (3.3)). In contrast to the standard approach *vectorization is in the time direction* and not in the spatial direction. The vector speedup of the arithmetic part of the computation mainly depends on the value of n_t. It is virtually independent of the size of the spatial grid, the number of multigrid levels, the multigrid cycle used, and the number of processors. This contrasts sharply with standard multigrid vectorization results. Standard vectorization does not lead to a performance improvement unless the number of grid points per processor is very large. Its application is therefore of very limited use on large-scale parallel processors.

The only operation that is not perfectly vectorizable is the core of the ODE integrator, which is used in the smoothing step. It consists of one or two first-order recurrence formulae of length n_t (one in the case of an initial value problem, and two in the case of a time-periodic problem solved by discretization with a one-step formula). This reduces the possible gain through vectorization. It can be shown, however, that the cost of the recurrence relation is only a small fraction of the total computation cost, typically 5 to 10 percent [19]. This leads to a possible vector speedup of 10 or more. As an additional advantage of vectorization in the time direction, we mention the ease of implementation. As the vector operations at each grid point involve the vectors at neighboring grid points only, no complex grid restructuring is needed.

6. A comparison of arithmetic and communication complexities. It is instructive to compare the complexities of the multigrid waveform algorithms with the complexities of the corresponding standard sequential methods. In this section, we assume that the discretization and integration formula used in the waveform algorithms is chosen as explained in § 5.1.

6.1. Initial-boundary value problems. We first introduce some notation. The grids are again denoted by $G_0, G_1, \cdots, G_k$, where G_k identifies the fine grid. Let N_i be the number of messages sent by each processor in one multigrid cycle, when an elliptic problem is solved on G_i. If N_i differs from processor to processor, we take the maximum value. Let L_i be the corresponding total message length. α_E denotes the number of multigrid cycles needed to solve the elliptic problem. α_{WR} is the number of multigrid waveform cycles needed for convergence of the waveform relaxation algorithm. The values of α_E and α_{WR} depend on the particular equation solved and the multigrid cycle used. Finally, $n_{t,i}$ is the number of time levels used for the time discretization of an initial-boundary value problem on G_i. If we further assume that the multigrid cycle type used in the timestepping algorithm corresponds to the one in the waveform method, and that α_E is the same on each time level, then we can immediately derive the communication complexity values presented in Table 6.1. TM_k and TL_k denote the total number of messages and the total message length to solve the initial-boundary value problem (IBVP) on G_k. In the case of the full multigrid approach, a value of 1 for α_E and α_{WR} frequently suffices to obtain a solution that is accurate to discretization

TABLE 6.1
Communication complexity of the initial-boundary value solvers.

	Timestepping algorithm	Multigrid waveform relaxation
TM_k(IBVP)	$n_{t,k}N_k\alpha_E$	$N_k\alpha_{WR}$
TL_k(IBVP)	$n_{t,k}L_k\alpha_E$	$n_{t,k}L_k\alpha_{WR}$

accuracy. The total message length in both solvers is then identical, whereas the number of messages differs by a factor of n_t.

In addition, a count of the number of floating point operations reveals that the arithmetic complexity of both methods is very similar (at least if the same cycle types and a same number of cycles are used). When using the Crank-Nicolson discretization, differences are typically in the range of 10 to 20 percent, with timestepping being more expensive. This higher cost is due to the cost of setting up the system of equations in each timestep, more precisely the cost of constructing the right-hand side (see [19]).

6.2. Time-periodic problems. To derive the complexities of the time-periodic solvers, some further assumptions are helpful. We assume, in particular, that the length of the messages in each of the multigrid operators is inversely proportional to the mesh size, and that the number of message exchanges is a constant independent of the discretization. For instance, in a Jacobi smoothing step, precisely four messages are needed on any grid, namely, one to each neighboring processor. The length of each message, however, is proportional to the number of grid lines. For simplicity, we only consider the use of multigrid V-cycles and standard coarsening, i.e., the mesh size is halved in going from a coarse to a fine grid. This immediately leads to the following formulae:

$$N_i = \frac{i}{k} N_k \quad \text{and} \quad L_i = \frac{2^i - 1}{2^k - 1} L_k \qquad (\approx 2^{i-k} L_k \text{ if } i, k \text{ sufficiently large}). \tag{6.1}$$

In the multigrid method of the second kind, several IBVPs are to be solved on every grid level. In each iteration, two such problems are to be solved on the fine grid. Each coarser grid, G_i, is visited 2^{k-i} times. In half of the visits, two IBVPs are to be solved (one in smoothing and one in defect calculation). In the other half, the problem in the smoothing step can be skipped. Indeed, when the initial condition is zero, the smoothing step simplifies to $Y_k := F_k$. Consequently, the number of IBVPs on G_i, with $i < k$, denoted by n_i, satisfies the formula

$$n_i = \frac{3}{2} 2^{k-i}. \tag{6.2}$$

If a time discretization method is used with sufficient smoothing behavior, e.g., a backward differentiation formula, the number of timesteps may be divided by 2 in going from one grid level to a coarser one. We assume that $n_{t,k}$ is large enough so that this process of halving the number of timesteps can be continued down to the coarsest grid. In that case, the number of timesteps used to solve an IBVP on G_i, denoted by $n_{t,i}$, equals

$$n_{t,i} = 2^{i-k} n_{t,k}. \tag{6.3}$$

The total number of messages and the total message length for solving a time-periodic problem (TPP) with α_M iterations of the multigrid method of the second kind can now be calculated as follows:

$$TM_k(\text{TPP}) = \left(2TM_k(\text{IBVP}) + \sum_{i=0}^{k-1} n_i TM_i(\text{IBVP})\right)\alpha_M = \frac{(5+3k)}{4} n_{t,k} N_k \alpha_E \alpha_M, \tag{6.4}$$

$$\begin{aligned} TL_k(\text{TPP}) &= \left(2TL_k(\text{IBVP}) + \sum_{i=0}^{k-1} n_i TL_i(\text{IBVP})\right)\alpha_M \\ &= \left(2 + \frac{3}{2}\left(1 - \frac{k}{2^k - 1}\right)\right) n_{t,k} L_k \alpha_E \alpha_M \end{aligned} \tag{6.5}$$

$$\approx \frac{7}{2} n_{t,k} L_k \alpha_E \alpha_M. \tag{6.6}$$

The communication complexity of one time-periodic multigrid waveform cycle is identical to that of one initial-value multigrid waveform cycle. However, the number of cycles needed to obtain convergence, α_{WRP}, may be different from α_{WR}.

A fair comparison of the time-periodic problem solvers should take the convergence characteristics into account. From several experiments, it is shown that an algorithm based on one iteration of the multigrid method of the second kind ($\alpha_M = 1$), or on one periodic full multigrid waveform step, is sufficient to obtain a solution accurate to discretization accuracy. Consequently, we should compare these two approaches. We have added a column to Table 6.2 for the full multigrid waveform step with one V-cycle on each grid level. Its communication complexity is derived from the formulae

$$TM_k(\text{TPP}) = \sum_{i=0}^{k} N_i = \frac{k+1}{2} N_k, \tag{6.7}$$

$$TL_k(\text{TPP}) = n_{t,k} \sum_{i=0}^{k} L_i = \left(2 - \frac{k}{2^k - 1}\right) n_{t,k} L_k \approx 2 n_{t,k} L_k. \tag{6.8}$$

The arithmetic complexity remains to be analyzed. By $W_k(\text{IBVP})$, we denote the complexity of solving one IBVP on G_k. The cost of solving a similar problem on G_i is then equal to

$$W_i(\text{IBVP}) = 8^{i-k} W_k(\text{IBVP}), \tag{6.9}$$

when we assume that an algorithm is used the complexity of which is proportional to the number of grid points. The cost of the multigrid method of the second kind equals

$$W_k(\text{TPP}) = \left(2 W_k(\text{IBVP}) + \sum_{i=0}^{k-1} n_i W_i(\text{IBVP})\right) \alpha_M \approx 2.5\, W_k(\text{IBVP}) \alpha_M. \tag{6.10}$$

The arithmetic complexity of a time-periodic multigrid waveform cycle with one-step time discretization hardly differs from that of the corresponding initial-value cycle [19]. Furthermore, since the convergence rates are very similar, we may safely state that the cost of solving a time-periodic problem by a waveform relaxation method is similar to the cost of solving the corresponding initial-boundary value problem with the waveform method.

The communication complexity of the periodic waveform relaxation algorithm is substantially smaller than that of the multigrid method of the second kind (see Table 6.2). Consider, e.g., the following typical set of parameters, taken from the example in § 7.2: $k = 4$, $n_{t,4} = 32$, $\alpha_E = 2$, $\alpha_M = 1$. With these parameters the total number of messages differs by a factor of more than 100, while the total length differs by a factor of 3.5. In addition, the arithmetic complexity of the time-periodic multigrid waveform

TABLE 6.2
Communication complexity of the time-periodic problem solvers.

	Multigrid second kind	Periodic MWR (cycling)	Periodic MWR (full mgrid)
$TM_k(\text{TPP})$	$(5+3k)/4 n_{t,k} N_k \alpha_E \alpha_M$	$N_k \alpha_{WRP}$	$(k+1)/2 N_k$
$TL_k(\text{TPP})$	$7/2 n_{t,k} L_k \alpha_E \alpha_M$	$n_{t,k} L_k \alpha_{WRP}$	$2 n_{t,k} L_k$

relaxation is smaller with a factor of 2.5. The latter holds, whenever the sequential complexity of the time integrator in the multigrid method of the second kind is similar to that of the initial-value multigrid waveform relaxation (MWR) method. (Numerical experiments have shown that this is very often the case.)

7. Numerical examples.

7.1. An initial-boundary value problem. We consider the following IBVP:

$$\frac{\partial u}{\partial t}=\frac{\partial^2 u}{\partial x^2}+xy\frac{\partial^2 u}{\partial x\,\partial y}+\frac{\partial^2 u}{\partial y^2}+f \quad \text{on } \Omega=[0,1]\times[0,1] \quad \text{for } t\in[0,0.5], \tag{7.1}$$

with Dirichlet conditions on the northern, eastern, and southern boundary, and a Neumann condition on the boundary to the west. The right-hand side function f is chosen in such a way that the solution is given by $u(t,x,y)=\sin(5x+y+10t)\,e^{-4t}$. For this problem, we compare the performance of MWR with a parallel implementation of the Crank-Nicolson method. In both cases, multigrid is used with a four-color nine-point Gauss-Seidel smoother, standard coarsening to a 3×3 coarse grid, full weighting restriction, bilinear interpolation, and a coarse-grid solver that performs two Gauss-Seidel iterations. The timestep Δt was set to 0.01, which leads to a vector length of 50. In the waveform method, we apply the trapezoidal rule to guarantee solutions identical to those obtained with Crank-Nicolson timestepping. The initial waveform solution is a function constant in time and equal to the value of the initial condition. The problem is solved on a 16-processor Intel iPSC/2 hypercube. Load distribution is performed by means of a two-dimensional partitioning with four processors in each coordinate direction.

The timing results are depicted in Fig. 7.1. The graphs show the accuracy of the solution (largest error at the grid points) versus execution time. The figures show *smooth curves* for the waveform method. The error of the initial waveform approximation decreases as more and more multigrid cycles are applied. (The results for the successive iterates are indicated by "∘" symbols.) The Crank-Nicolson results show up as *discrete points.* The Crank-Nicolson solution process is advanced timestep per timestep in a total of t seconds. The accuracy of the result is represented by a "+" sign at position (t, error) in the figure. Depending on the multigrid cycle type used, different execution times are needed. As such several results are presented for each technique. They are annotated in Fig. 7.1 in the following way: with "WR $V(1,1)$ with FMG" we mean "waveform relaxation using V-cycles with one pre- and one post-smoothing step and use of the full multigrid technique with one cycle at each grid level to determine the initial approximation on the fine grid." In the Crank-Nicolson method the initial approximation at a time level is either calculated by a linear extrapolation of the solutions at two previous time levels, or is determined by using full multigrid. It is then corrected by means of a fixed number of V- or F-cycles. This is denoted as "C-N, 1 $V(1,1)$ with FMG," which indicates the use of full multigrid with one $V(1,1)$ cycle on each grid level. Two sets of curves are given for the waveform method. The dashed lines represent the results obtained with vectorization, while the solid lines represent the results obtained in scalar execution mode.

On 16 processors, MWR is faster than the Crank-Nicolson method with a factor of 7 (for the 65×65 problem) up to a factor of 10 (for the 17×17 problem). This is due to the *smaller arithmetic complexity* of the waveform method, its *superior parallel characteristics,* and the use of *vectorization.* In Table 7.1, we have tabulated the execution time of the full multigrid solver with one $V(1,1)$ cycle on each grid level, on 1 and on 16 processors. We have also added the parallel speedup Sp. In this particular

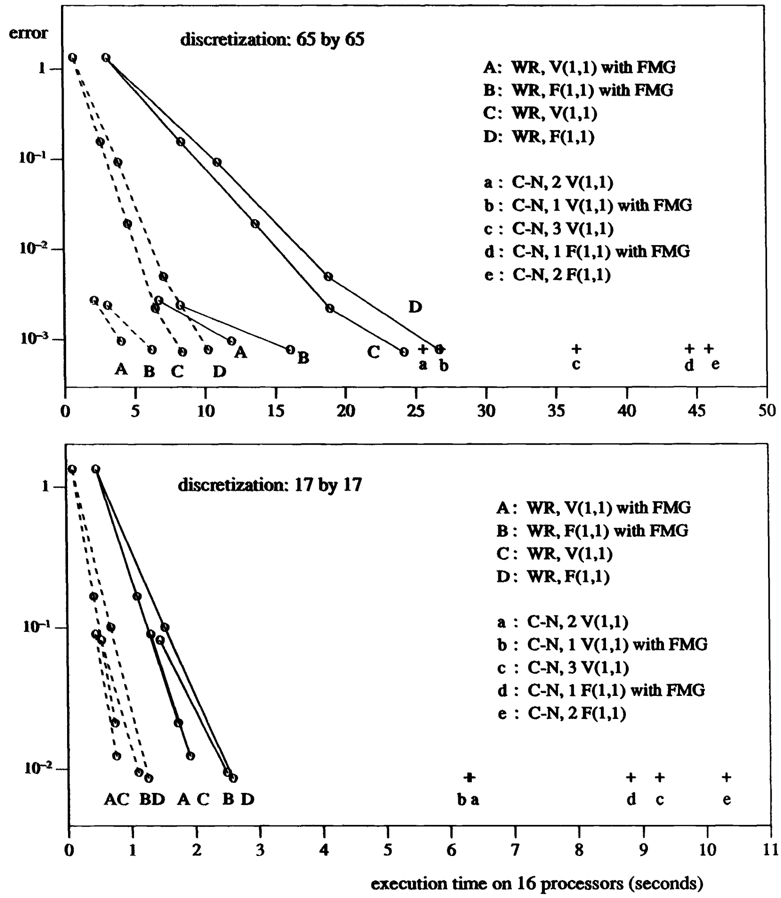


FIG. 7.1. *Comparison of Crank-Nicolson and multigrid waveform relaxation.*

TABLE 7.1
Execution time, in seconds, of the full multigrid solver on 1 *and* 16 *processors.*

	$h = 1/16, n_t = 50$			$h = 1/32, n_t = 50$		
Method	1 proc.	16 proc.	Sp	1 proc.	16 proc.	Sp
Waveform relaxation	9.35	1.91	5.0	36.00	4.30	8.4
Crank-Nicolson	12.90	6.32	2.0	48.29	12.52	3.9

example, waveform relaxation outperforms the standard method by a factor of approximately 1.3 on a single processor. This is due to a somewhat smaller arithmetic complexity, a reduced initialization cost (some intermediate results may be retained when setting up system (2.1), especially when evaluating the right-hand side), and the much lower computational overheads associated with program control (loop overhead, indexing overhead, procedure call overhead, etc.). An additional factor of 2 to 2.5 results from the better parallel characteristics of the MWR method. This immediately shows from the speedup figures.

The remaining performance difference is due to vectorization. In Table 7.2, we give the execution times of the full multigrid solver, together with the values of the

TABLE 7.2
Execution time, in seconds, of the WR FMG $V(1,1)$ *solver* (*on* 16 *processors*).

	$h=1/16$			$h=1/32$			$h=1/64$		
n_t	Scalar	Vector	Sp	Scalar	Vector	Sp	Scalar	Vector	Sp
100	3.54	1.12	3.16	8.20	2.42	3.39	(na)	(na)	(na)
50	1.91	0.76	2.51	4.30	1.59	2.70	11.88	4.02	2.96
25	1.10	0.59	1.86	2.43	1.20	2.03	6.53	2.91	2.24
10	0.62	0.49	1.27	1.29	0.97	1.33	3.21	2.25	1.43

speedup obtained by vectorization. (The largest problem, 65×65 grid lines and 100 timesteps, could not run on the machine due to lack of memory on the vector board.) The low values of the vector speedup are due to the high startup time of the vector operations on the iPSC/2. The speedup values should be substantially higher on a processor with more specialized vector hardware. The dependence of the speedup on the vector length is obvious. It should be noted that for problems of this size on a 16-processor machine, vectorization in the Crank-Nicolson method would not lead to any speedup.

7.2. A time-periodic problem. We consider the time-periodic parabolic partial differential equation,

$$\frac{\partial u}{\partial t}=\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+f, \quad \text{with } u(0,x,y)=u(1,x,y), \tag{7.2}$$

defined on the unit square with four Dirichlet boundary conditions. The function f is chosen in such a way that the solution of the equation becomes $u(t,x,y)=1+100(x-x^2)^2(y-y^2)^2\sin(2\pi t)$. In Fig. 7.2, we represent the timing results which were again obtained on a 16-processor Intel iPSC/2 hypercube. However, no vectorization was used.

Three methods are compared. The first method is a parallel implementation of the multigrid method of the second kind. The second-order backward differentiation method (BDF(2)) is used for time integration. It has excellent smoothing properties and is of high accuracy. The spatial mesh size of each grid G_l is determined by standard coarsening from a fine grid, G_k, with 65 or 17 grid lines in the x- and y-direction. The time increment Δt_l is chosen equal to the mesh size. The linear systems obtained by the BDF(2) scheme on each time level are solved by using the standard multigrid method, with two $V(1,1)$ cycles or by full multigrid with one $V(1,1)$ cycle on each level. Due to its $O(h^2)$ convergence rate, one iteration of the method is sufficient to solve the problem to discretization accuracy. Various programming techniques are applied to optimize the parallel performance of the implementation. In particular, the use of an agglomeration technique is crucial to reducing the parallel overhead of the coarse-grid operations.

A related method is obtained if the multigrid waveform algorithm is used as the IBVP solver inside the multigrid method of the second kind. The resulting algorithm is about 1.5 to 2 times as fast as the method with timestepping, as shown in Fig. 7.2. This was to be expected, from the speedup values given in Table 7.1. The periodic multigrid waveform method is faster than the best standard algorithm, by a factor of 7 to 10. This is in part due to its *lower arithmetic complexity*, as explained in § 6. To verify the complexity estimates of both methods we report the one-processor execution times in Table 7.3.

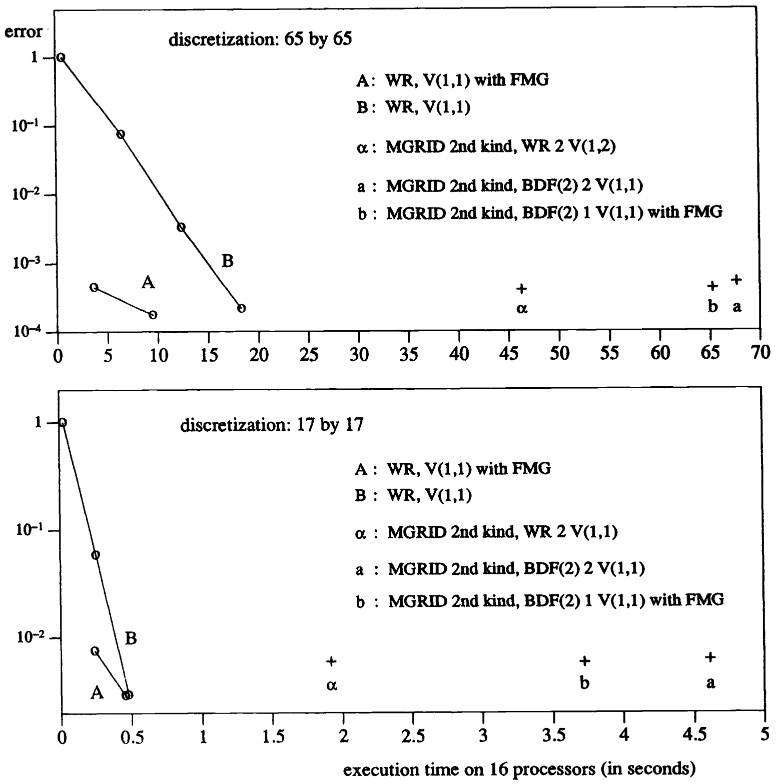


FIG. 7.2. *Comparison of multigrid waveform relaxation and multigrid of the second kind.*

TABLE 7.3

Execution time, in seconds, of two time-periodic problem solvers: time-periodic full multigrid waveform relaxation and one cycle of multigrid of the second kind.

	$h=1/16, n_t=16$			$h=1/32, n_t=32$		
Method	1 proc.	16 proc.	Sp	1 proc.	16 proc.	Sp
Waveform relaxation	1.77	0.46	3.9	14.33	1.78	8.1
Multigrid sec. kind	4.83	3.74	1.3	35.11	15.74	2.2

The remaining performance difference results from the *better parallel characteristics* of waveform relaxation. This is illustrated in Table 7.3, which shows that hardly any speedup is obtained with the multigrid method of the second kind. As noted above, it is difficult to parallelize coarse-grid operations efficiently. The multigrid method of the second kind visits the coarse grids very frequently because of its "double multigrid" nature. It is basically a multigrid *W*-cycle where, in each smoothing step, a timestepping method is used which applies standard multigrid for solving the elliptic problems on each time level. Consequently, the algorithm is not well suited for parallel implementation.

We should also note that vectorization leads to an additional speedup only in the case of the waveform algorithm. The performance difference on the 16-processor machine is then in the range of 20 to 40, depending on the problem size.

8. Summary and concluding remarks. The transformation of the parabolic problem into the sequential process of solving small problems on successive time levels seriously degrades the parallel efficiency of the standard marching schemes. While these methods can be used efficiently for problems with a very large number of grid points per processor, they perform totally unsatisfactorily for small problems and/or large numbers of processors.

To tackle these problems, we have presented a family of methods based on waveform relaxation. They show multigrid convergence speeds and are competitive with, or even surpass, the standard solvers on sequential processors. They can be implemented efficiently on parallel machines, as they have a very low communication complexity. As an added advantage they can be vectorized straightforwardly, with no substantial changes in the program code. This vectorization leads to an important reduction in execution time, even if the number of grid points per processor is too small to justify vectorization of the timestepping approach. In addition, the algorithms are not restricted to linear problems or rectangular domains; nonlinear problems and systems of equations on arbitrary domains can be treated in a similar way with qualitatively the same parallelization and vectorization characteristics. The main price to be paid is memory cost, as the iterates generated by the algorithm are to be stored along the entire time interval of interest.

REFERENCES

[1] L. Bomans and D. Roose, *Benchmarking the iPSC/2 hypercube multiprocessor*, Concurrency: Practice and Experience, 1 (1989), pp. 3-18.

[2] P. Bastian, J. Burmeister, and G. Horton, *Implementation of a parallel multigrid method for parabolic partial differential equations*, in Parallel Algorithms for Parabolic PDEs, W. Hackbusch, ed., Vieweg-Verlag, Wiesbaden, Germany, 1990, pp. 18-27.

[3] D. Evans, *New parallel algorithms for partial differential equations*, in Parallel Computing 83, M. Feilmeier, J. Joubert, and U. Schendel, eds., North-Holland, Amsterdam, 1984, pp. 3-56.

[4] ———, *Alternating group explicit method for the diffusion equation*, Appl. Math. Modelling, 9 (1985), pp. 201-206.

[5] W. Hackbusch, *Fast numerical solution of time-periodic parabolic problems by a multigrid method*, SIAM J. Sci. Statist. Comput., 2 (1981), pp. 198-206.

[6] ———, *Parabolic multi-grid methods*, in Computing Methods in Applied Sciences and Engineering, VI, R. Glowinski and J.-L. Lions, eds., North-Holland, Amsterdam, 1984, pp. 189-197.

[7] ———, *Multi-grid methods and Applications*, Springer Series in Comp. Math. 4, Springer-Verlag, Berlin, 1985.

[8] F. Juang, *Accuracy increase in waveform relaxation*, Report No. UIUCDCS-R-1466, Department of Computer Science, University of Illinois, Urbana, IL, October 1988.

[9] Ch. Lubich and A. Ostermann, *Multi-Grid Dynamic Iteration for Parabolic Equations*, BIT, 27 (1987), pp. 216-234.

[10] U. Miekkala and O. Nevanlinna, *Convergence of dynamic iteration methods for initial value problems*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 459-482.

[11] ———, *Sets of convergence and stability regions*, BIT, 27 (1987), pp. 554-584.

[12] G. Rodrigue and D. Wolitzer, *Preconditioned time-differencing for the parallel solution of the heat equation*, in Proc. Fourth SIAM Conf. on Parallel Processing for Scientific Computing, J. Dongarra, P. Messina, D. Sorensen,and R. Voigt, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990, pp. 268-272.

[13] J. Saltz and V. Naik, *Towards developing robust algorithms for solving partial differential equations on* MIMD *machines*, Parallel Comput., 6 (1988), pp. 19-44.

[14] S. Vandewalle and D. Roose, *The Parallel waveform relaxation multigrid method*, in Parallel Processing for Scientific Computing, G. Rodrigue, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989, pp. 152-156.

[15] S. Vandewalle and R. Piessens, *A comparison of the Crank-Nicolson and waveform relaxation multigrid methods on the Intel hypercube*, in Proc. of the Fourth Copper Mountain Conference on Multigrid Methods, J. Mandel, S. McCormick, J. Dendy, C. Farhat, G. Lonsdale, S. Parter, J. Ruge, and K. Stüben, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990, pp. 417-434.

[16] S. Vandewalle, R. Van Driessche and R. Piessens, *The parallel performance of standard parabolic marching schemes*, Internat. J. High Speed Comput., 3 (1991), pp. 1-29.

[17] S. Vandewalle and R. Piessens, *Numerical experiments with nonlinear multigrid waveform relaxation on a parallel processor*, Appl. Numer. Math., 8 (1991), pp. 149-161.

[18] S. Vandewalle, *On dynamic iteration methods for solving time-periodic differential equations*, Report TW 148, Department of Computer Science, Katholieke Universiteit Leuven, Heverlee, Belgium, March 1991; SIAM J. Numer. Anal., to appear.

[19] ———, *The parallel solution of parabolic partial differential equations by multigrid waveform relaxation methods*, Ph.D. thesis, Katholieke Universiteit Leuven, Heverlee, Belgium, 1992.

[20] J. White, F. Odeh, A. S. Sangiovanni-Vincentelli, and A. Ruehli, *Waveform relaxation: Theory and practice*, Memorandum No. UCB/ERL M85/65, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA, 1985.

[21] D. Womble, *A timestepping algorithm for parallel computers*, SIAM J. Sci. Statist. Comput., 11 (1990) pp. 824-837.

EVALUATING BEST-CASE AND WORST-CASE VARIANCES WHEN BOUNDS ARE AVAILABLE*

GEORGE S. FISHMAN†, BORIS L. GRANOVSKY‡, AND DAVID S. RUBIN†§

Abstract. This paper describes procedures for computing the tightest possible best-case and worst-case bounds on the variance of a discrete, bounded, random variable when lower and upper bounds are available for its unknown probability mass function. An example from the application of the Monte Carlo method to the estimation of network reliability illustrates the procedures and, in particular, reveals considerable tightening in the worst-case bound when compared to the trivial worst-case bound based exclusively on range.

Key words. best-case variance, computational complexity, Monte Carlo, parametric linear programming, sample size, worst-case variance

AMS(MOS) subject classification. 62

Introduction. Let X denote a discrete random variable with unknown probability mass function (pmf) $\{\mu_i = \text{pr}(X=i);\ \mu_i > 0,\ 1 \leqq i \leqq r < \infty;\ \mu_1 + \cdots + \mu_r = 1\}$ and suppose one wishes to estimate

$$g(\boldsymbol{\mu}, \mathbf{w}) = \mu_1 w_1 + \cdots + \mu_r w_r, \tag{1}$$

where

$$\boldsymbol{\mu} = (\mu_1, \cdots, \mu_r),$$

and where $\mathbf{w} = (w_1, \cdots, w_r)$ is known, with $w_i \neq w_j$ for $i \neq j$. Let

$$\phi_i(j) = \begin{cases} 1 & \text{if } j = 1, \\ 0 & \text{otherwise} \quad 1 \leqq i, j \leqq r. \end{cases} \tag{2}$$

Then the random variable $Z = \phi_1(X) w_1 + \cdots + \phi_r(X) w_r$ has expectation (1) and variance

$$\text{var}\, Z = h(\boldsymbol{\mu}, \mathbf{w}) = \sum_{i=1}^{r} \mu_i w_i^2 - \left(\sum_{i=1}^{r} \mu_i w_i \right)^2. \tag{3}$$

This paper describes algorithms for evaluating the tightest possible worst-case lower bound and best-case upper bound on $h(\boldsymbol{\mu}, \mathbf{w})$ when nontrivial lower and upper bounds are available for $\boldsymbol{\mu}$. Formally, we solve the worst-case nonlinear program

$$h^*(w) = \max_{\boldsymbol{\mu}} h(\boldsymbol{\mu}, \mathbf{w}),$$

subject to

$$\text{(NLPW)} \qquad \begin{aligned} &\mu_i \geqq e_i \geqq 0, \\ &\mu_i \leqq f_i \leqq 1, \\ &e_i \leqq f_i, \qquad 1 \leqq i \leqq r \\ &\mu_1 + \cdots + \mu_r = 1, \end{aligned}$$

* Received by the editors October 17, 1989; accepted for publication (in revised form) September 9, 1991. This research was partially supported by Air Force Office of Scientific Research grant AFOSR-84-0140 and by a grant from the Business Foundation of North Carolina.

† Department of Operations Research, University of North Carolina, Chapel Hill, North Carolina 27599.

‡ Mathematics Department, Technion-Israel Institute of Technology, Haifa, Israel. The research of this author was partially supported by a grant from the Technion V.P.R. Fund-J. and S. Frankel Research Fund.

§ Kenan-Flagler Business School, University of North Carolina, Chapel Hill, North Carolina 27599.

and we also solve the best-case nonlinear program (NLPB), the analogous problem of finding $h_*(\mathbf{w}) = \min_{\boldsymbol{\mu}} h(\boldsymbol{\mu}, \mathbf{w})$, subject to the same constraints. Knowing $h_*(\mathbf{w})$ and $h^*(\mathbf{w})$ is of considerable value when designing a sampling experiment to achieve a variance no greater than, say, a specified quantity $v(\mathbf{w})$. In particular, a sample size of no less than $n_*(\mathbf{w}) = \lceil h_*(\mathbf{w})/v(\mathbf{w}) \rceil$, but no larger than $n^*(\mathbf{w}) = \lceil h^*(\mathbf{w})/v(\mathbf{w}) \rceil$ is required to achieve this accuracy. Moreover, if we wish to estimate the function $\{g(\boldsymbol{\mu}, \mathbf{w}), \mathbf{w} \in \mathscr{W}\}$, where $\mathscr{W}$ denotes a set of alternative weight vectors, to achieve variance accuracies $\{v(\mathbf{w}), \mathbf{w} \in \mathscr{W}\}$, then

$$n = \left\lceil \max_{\mathbf{w} \in \mathscr{W}} [h^*(\mathbf{w})/v(\mathbf{w})] \right\rceil$$

gives the worst-case sample size.

For the trivial $\{0, 1\}$ bounds on each μ_i, the best-case bound for $h(\boldsymbol{\mu}, \mathbf{w})$ is zero, and we immediately derive the worst-case bound

$$h^{**}(\mathbf{w}) = \left(\max_{1 \leqq i \leqq r} w_i - \min_{1 \leqq i \leqq r} w_i \right)^2 /4. \tag{4}$$

As the example in § 2 makes clear, even relatively modest improvements in these bounds can lead to substantially tighter $h_*(\mathbf{w})$ and $h^*(\mathbf{w})$, thus significantly reducing their difference as well as the corresponding worst-case sample size. This improved bound is the reward for going to the effort of solving the nonlinear programs in (NLPW) and (NLPB).

Problems of this type arise in many areas where the setting of the problem makes a set of bounds $\mathbf{e} = (e_1, \cdots, e_r)$ and $\mathbf{f} = (f_1, \cdots, f_r)$ available to the analyst at low or no cost before data gathering begins. This is especially true when applying the Monte Carlo method where the mathematical structure of the problem often allows one to deduce the bounds. Section 1 describes the algorithms for solving (NLPW) and (NLPB) and § 2 illustrates their use in a Monte Carlo example designed to estimate system reliability in a stochastic network.

1. Method of solution.

1.1. Determining $h^*(\mathbf{w})$. We begin by characterizing the solution to (NLPW). Since (NLPW) has a continuous objective function, and since the set of points that satisfy its constraints is compact, the problem has an optimal solution.

THEOREM 1. *Let $w_i \neq w_j$ for $i \neq j$, and let $\boldsymbol{\mu}^0 = (\mu_1^0, \cdots, \mu_r^0)$ denote the solution to* (NLPW). *There exists a partition $\mathscr{S}_e$, $\mathscr{S}$, $\mathscr{S}_f$ of the integers $\{1, \cdots, r\}$ such that*

$$\mu_i^0 = \begin{cases} e_i & \text{for } i \in \mathscr{S}_e, \\ \theta_i, \quad e_i < \theta_i < f_i, & \text{for } i \in \mathscr{S}, \\ f_i & \text{for } i \in \mathscr{S}_f, \end{cases}$$

and $|\mathscr{S}| \leqq 2$.

Proof. The only result requiring proof is $|\mathscr{S}| \leqq 2$. Let $q = q(\boldsymbol{\mu}, \mathbf{w}) = h(\boldsymbol{\mu}, \mathbf{w}) + L(1 - \mu_1 - \cdots - \mu_r)$, with L a Lagrange multiplier, be the usual Lagrangian function for (NLPW). Let

$$\lambda^0 = \sum_{i=1}^{r} \mu_i^0 w_i,$$

and let L^0 be an optimal value for L. If $e_i < \mu_i^0 < f_i$, then at $\boldsymbol{\mu} = \boldsymbol{\mu}^0$ and $L = L^0$, $\partial q/\partial \mu_i = w_i^2 - 2\lambda^0 w_i - L^0 = 0$. Since λ^0 and L^0 are fixed, this can happen for at most two $i \in \{1, \cdots, r\}$, because $\partial q/\partial \mu_i$ is a quadratic polynomial in w_i, and the w_i are distinct.

Therefore, $|\mathcal{S}| \leqq 2$, implying that the remaining μ_j^0 must be either at their lower or upper bounds. □

The proposed method of computing $h^*(\mathbf{w})$ is based on consideration of a parametric family of linear programs related to (NLPW). Let

$$\Phi(\lambda) = \max \sum_{j=1}^{r} \mu_j w_j^2,$$

subject to

$$\text{(LPW}(\lambda)) \qquad \begin{aligned} &\sum_{j=1}^{r} \mu_j = 1, \\ &\sum_{j=1}^{r} \mu_j w_j = \lambda, \\ &e_j \leqq \mu_j \leqq f_j, \qquad 1 \leqq j \leqq r, \end{aligned}$$

where λ and $\mathbf{w}$ are fixed and the indices are assigned so that $w_1 > w_2 > \cdots > w_r$. Let $\Psi(\lambda) = \Phi(\lambda) - \lambda^2$,

$$\underline{\lambda} = \min \left(\sum_{i=1}^{r} \mu_i w_i : \sum_{i=1}^{r} \mu_i = 1;\ e_i \leqq \mu_i \leqq f_i,\ 1 \leqq i \leqq r \right),$$

and

$$\bar{\lambda} = \max \left(\sum_{i=1}^{r} \mu_i w_i : \sum_{i=1}^{r} \mu_i = 1;\ e_i \leqq \mu_i \leqq f_i,\ 1 \leqq i \leqq r \right).$$

Then (NLPW) has the equivalent representation

$$\text{(PW)} \qquad h^*(\mathbf{w}) = \max_{\underline{\lambda} \leqq \lambda \leqq \bar{\lambda}} \Psi(\lambda).$$

The proposed method solves a sequence of linear programs (LPW(λ)) for changing λ until the optimal solution to (PW) is found. Since (LPW(λ)) has two structural equations, there are exactly two variables that are basic in any basic feasible solution to (LPW(λ)), an observation consistent with the result in Theorem 1.

We assume that the w_i are distinct: if, in reality, $w_i = w_j$ for some $i \neq j$, we can replace μ_i and μ_j by $\hat{\mu}_i = \mu_i + \mu_j$, with bounds $\hat{e}_i = e_i + e_j$ and $\hat{f}_i = f_i + f_j$, and reduce the dimension of the problem by one.

For feasibility we must have $\sum_{i=1}^{r} e_i \leqq 1$ and $\sum_{i=1}^{r} f_i \geqq 1$, and these inequalities must be strict or else the problem is uninteresting, since $\sum_{i=1}^{r} e_i = 1$ implies that $\mu_i = e_i$, $1 \leqq i \leqq r$, is the only feasible solution, and $\sum_{i=1}^{r} f_i = 1$ implies that $\mu_i = f_i$, $1 \leqq i \leqq r$, is the only feasible solution.

We assume that $e_i < f_i$ for all $1 \leqq i \leqq r$: if for some $r' < r$, $e_i = f_i$ for $r' < i \leqq r$, then we prefer to work with the random variable

$$Z' = \sum_{i=r'+1}^{r} e_i w_i + \sum_{i=1}^{r'} \phi_i(X) w_i$$

and then maximize

$$\sum_{i=1}^{r'} \mu_i w_i^2 - \left(\sum_{i=1}^{r'} \mu_i w_i \right)^2$$

subject to $\mu_1 + \cdots + \mu_{r'} = 1 - e_{r'+1} - \cdots - e_r$, and $e_i \leqq \mu_i \leqq f_i$, $1 \leqq i \leqq r'$. A suitable reassignment of indices allows this reformulation to apply when $e_i = f_i$ for any i.

LEMMA 2. *$\Psi(\lambda)$ is continuous, piecewise differentiable, and strictly concave for λ in the interval $(\underline{\lambda}, \bar{\lambda})$.*

Proof. That $\Phi(\lambda)$ is a piecewise-linear, continuous, concave function of λ is a standard result in the theory of linear programming [4, p. 288]. The lemma now follows because $-\lambda^2$ is continuous, differentiable, and strictly concave. □

The next result is an immediate corollary of Lemma 2.

COROLLARY 2.1. *The quantity $\hat{\lambda} \in (\underline{\lambda}, \bar{\lambda})$ is optimal for* (PW) *if and only if*

$$\lim_{\lambda \uparrow \hat{\lambda}} \Psi'(\lambda) \geqq 0 \quad \text{and} \quad \lim_{\lambda \downarrow \hat{\lambda}} \Psi'(\lambda) \leqq 0;$$

$\underline{\lambda}$ is optimal for (PW) *if and only if* $\lim_{\lambda \downarrow \underline{\lambda}} \Psi'(\lambda) \leqq 0$; *$\bar{\lambda}$ is optimal for* (PW) *if and only if* $\lim_{\lambda \uparrow \bar{\lambda}} \Psi'(\lambda) \geqq 0$; *furthermore,* (PW) *has a unique optimal solution.*

Assume that λ initially takes the value λ^0 and suppose we have found (by the simplex algorithm or any other method) $\boldsymbol{\mu}^0 = (\mu_1^0, \cdots, \mu_r^0)$, an optimal basic solution to (LPW(λ^0)). Appendix A discusses the choice of a starting value for λ. Suppose the basic variables are μ_i and μ_j with $i < j$. Then the basis matrix is

$$\mathbf{B} = \begin{pmatrix} 1 & 1 \\ w_i & w_j \end{pmatrix}, \quad \text{with inverse } \mathbf{B}^{-1} = \frac{1}{w_i - w_j} \begin{pmatrix} -w_j & 1 \\ w_i & -1 \end{pmatrix}.$$

The usual linear programming optimality conditions now reduce to

$$-(w_k - w_i)(w_k - w_j) \begin{cases} >0 & \text{i.e., } i < k < j \Rightarrow \mu_k^0 = e_k, \\ <0 & \text{i.e., } k < i \quad \text{or} \quad k > j \Rightarrow \mu_k^0 = f_k. \end{cases}$$

In addition, we have

$$\begin{pmatrix} e_i \\ e_j \end{pmatrix} \leqq \begin{pmatrix} \mu_i^0 \\ \mu_j^0 \end{pmatrix} \leqq \begin{pmatrix} f_i \\ f_j \end{pmatrix}.$$

Let $\theta = w_i - w_j > 0$. Then

$$\frac{d\mu_i}{d\lambda} = \frac{1}{\theta}, \quad \frac{d\mu_j}{d\lambda} = -\frac{1}{\theta}, \quad \frac{d\Phi(\lambda)}{d\lambda} = w_i + w_j, \quad \text{and} \quad \frac{d\Psi(\lambda)}{d\lambda} = w_i + w_j - 2\lambda,$$

for all $\lambda \in [\lambda_L, \lambda_U]$, where

$$(5) \qquad \lambda_L = \lambda^0 - \theta \min(f_j - \mu_j^0, \mu_i^0 - e_i) \quad \text{and} \quad \lambda_U = \lambda^0 + \theta \min(f_i - \mu_i^0, \mu_j^0 - e_j).$$

The derivatives given above are to be interpreted as two-sided for $\lambda \in (\lambda_L, \lambda_U)$, but as one-sided derivatives to the right at $\lambda = \lambda_L$ and to the left at $\lambda = \lambda_U$.

The following algorithm solves (PW):

ALGORITHM 1.

1. If $((w_i + w_j)/2 \in [\lambda_L, \lambda_U]$, then the optimal solution to (LPW($(w_i + w_j)/2$)) is optimal in (PW). If $((w_i + w_j)/2) - \lambda_L < 0$, go to Step 2. If $((w_i + w_j)/2) - \lambda_U > 0$, go to Step 3.
2. If $\lambda_L = \underline{\lambda}$, then the optimal solution to (LPW($\underline{\lambda}$)) is optimal in (PW). If $\lambda_L > \underline{\lambda}$, then let $\lambda^0 = \lambda_L - \varepsilon$ (where ε is a positive infinitesimal). Use standard linear programming sensitivity analysis techniques to solve the new (LPW(λ^0)), to determine the new indices i and j, and to determine the new $[\lambda_L, \lambda_U]$. (Note that the new λ_U equals the old λ_L.) If $((w_i + w_j)/2) - \lambda_U \geqq 0$, then the optimal solution to (LPW(λ_U)) is optimal in (PW). Otherwise, go to Step 1.
3. If $\lambda_U = \bar{\lambda}$, then the optimal solution to (LPW($\bar{\lambda}$)) is optimal in (PW). If $\lambda_U < \bar{\lambda}$, then let $\lambda^0 = \lambda_U + \varepsilon$ (where ε is a positive infinitesimal). Use standard linear

programming sensitivity analysis techniques to solve the new (LPW(λ^0)), to determine the new indices i and j, and to determine the new $[\lambda_L, \lambda_U]$. (Note that the new λ_L equals the old λ_U.) If $((w_i+w_j)/2)-\lambda_L \leqq 0$, then the optimal solution to (LPW(λ_L)) is optimal in (PW). Otherwise, go to Step 1.

Finiteness of the algorithm follows from the fact that the sequence of the λ^0 values is monotone increasing if the initial λ^0 is too small and monotone decreasing if the initial λ^0 is too large. Hence, no pair (i, j) ever repeats. Optimality of the claimed solution to (PW) follows from Corollary 2.1 to Lemma 2. Given the structure of (LPW(λ)), the linear programming sensitivity analysis techniques can be specialized so that this algorithm can be implemented with $O(r)$ time complexity. Algorithmic details and the proof of this complexity result are given in Appendix A.

1.2. Determining $h_*(\mathbf{w})$. Let (LPB(λ)) be the linear program to determine

$$\phi(\lambda) = \min \sum_{j=1}^{r} \mu_j w_j^2$$

subject to the same constraints as (LPW(λ)), and let $\varphi(\lambda) = \phi(\lambda) - \lambda^2$. Then (NLPB) has the equivalent representation

$$\text{(PB)} \qquad h_*(\mathbf{w}) = \min_{\underline{\lambda} \leqq \lambda \leqq \bar{\lambda}} \varphi(\lambda).$$

Analogous to the results of § 1.1, if $\boldsymbol{\mu}^0$ is an optimal basic solution to (LPB(λ^0)), with basic variables μ_i and μ_j with $i<j$, it follows that

$$i<k<j \Rightarrow \mu_k^0 = f_k, \quad k<i \quad \text{or} \quad k>j \Rightarrow \mu_k^0 = e_k,$$

and

$$\begin{pmatrix} e_i \\ e_j \end{pmatrix} \leqq \begin{pmatrix} \mu_i^0 \\ \mu_j^0 \end{pmatrix} \leqq \begin{pmatrix} f_i \\ f_j \end{pmatrix}.$$

Again, let $\theta = w_i - w_j$ and determine λ_L and λ_U by (5). As before, $d\phi(\lambda)/d\lambda = w_i + w_j$ for $\lambda \in (\lambda_L, \lambda_U)$.

LEMMA 3. *Let* **B** *be an optimal basis matrix for* (LPB(λ^0)). *Then* $\varphi(\lambda)$ *is continuous and strictly concave for* λ *in the interval* $[\lambda_L, \lambda_U]$.

Proof. $\phi(\lambda)$ is linear on $[\lambda_L, \lambda_U]$ (see [4, p. 288]) and $-\lambda^2$ is continuous and strictly concave. □

COROLLARY 3.1. *If* $\hat{\lambda}$ *is optimal in* (PB), *then either* $\hat{\lambda} = \bar{\lambda}$, *or else* $\hat{\lambda} = \lambda_L$ *for some basis matrix* **B** *which is optimal in* (LPB($\hat{\lambda}$)).

Proof. This is immediate from Lemma 3 because of the concavity of $\varphi(\lambda)$. □

COROLLARY 3.2. (NLPB) *has an optimal solution in which at most one of the components of* $\boldsymbol{\mu}$ *lies strictly between its lower and upper bounds.*

Proof. This follows immediately from Corollary 3.1 and the theory of linear programming, since one of the basic variables is zero at the endpoint λ_L. □

The following algorithm solves (PB).

ALGORITHM 2.

1. Set $\lambda^0 = \bar{\lambda}$. Let $\boldsymbol{\mu}_B$ solve (LPB($\bar{\lambda}$)). Set $\lambda_B = \bar{\lambda}$ and $\varphi_B = \phi(\bar{\lambda}) - \bar{\lambda}^2$.
2. If $\phi(\lambda_U) - (w_i + w_j)(\lambda_U - \lambda_L) - \lambda_L^2 \geqq \varphi_B$, go to Step 3.
   Otherwise, set $\lambda_B = \lambda_L$, $\varphi_B = \phi(\lambda_U) - (w_i + w_j)(\lambda_U - \lambda_L) - \lambda_L^2$, let $\boldsymbol{\mu}_B$ solve (LPB(λ_L)), and go to Step 3.
3. If $\lambda_L = \underline{\lambda}$, stop: $\boldsymbol{\mu}_B$ is the optimal solution to (PB).
   Otherwise, let $\lambda^0 = \lambda_L - \varepsilon$ (where ε is a positive infinitesimal). Use standard linear programming sensitivity analysis techniques to solve the new (LPB(λ^0)),

to determine the new indices i and j, and to determine the new $[\lambda_L, \lambda_U]$. Go to Step 2.

Finiteness of the algorithm follows from the fact that the sequence of λ^0 values is monotone decreasing. Hence no pair (i, j) ever repeats. Optimality of the claimed solution to (PB) follows from Corollary 3.1 to Lemma 3. Given the structure of (LPB(λ)), the linear programming sensitivity analysis techniques can again be specialized so that this algorithm, too, can be implemented with $O(r)$ time complexity. Algorithmic details and the proof of their complexity result are given in Appendix B.

2. Example. We illustrate this method of determining lower and upper bounds on $h(\boldsymbol{\mu}, \mathbf{w})$ for the estimation of the all-points-connectedness reliability in network analysis. Let $G = (\mathscr{V}, \mathscr{E})$ denote an undirected network with node set $\mathscr{V}$ and arc set $\mathscr{E}$. Suppose that nodes operate perfectly but arcs fail randomly and independently, each with probability $1-p$. Let $m(p)$ denote the probability that all nodes in G are connected. The exact evaluation of $m(p)$ is known to belong to the $\#P$-complete class of problems, implying that no algorithm is known for solving this problem in time polynomial in $|\mathscr{V}|$ and $|\mathscr{E}|$. This limitation has encouraged analysts to apply the Monte Carlo method to the estimation of $m(p)$. In particular, Nel and Colbourn [5] recommend a Monte Carlo approach based on a partition of the set of all spanning trees of G due to Ball and Nemhauser [1], who show that one can represent the probability that all nodes are connected in the form

$$m(p) = p^{k-1} \sum_{i=0}^{l-k+1} H_i (1-p)^i, \tag{6}$$

where

$$H_i = |\mathscr{H}_i|, \qquad 0 \leqq i \leqq l-k+1,$$

$k = |\mathscr{V}|$, $l = |\mathscr{E}|$, and $\{\mathscr{H}_0, \mathscr{H}_1, \cdots, \mathscr{H}_{l-k+1}\}$ denotes a particular partition of the set of all spanning trees of G. A spanning tree is a set of $k-1$ arcs that connects all nodes. Using this formulation on each of n independent replications, Nel and Colbourn [5] randomly generate a spanning tree and determine the partition to which it belongs. They then demonstrate how to employ one version of the Ball and Provan [2] bounds together with these sample data to derive point estimates and confidence intervals for $\{H_i\}$ and for $m(p)$ for a range of values of the component reliability p. They illustrate their technique for the Advanced Research Projects Agency Network (ARPANET) in Fig. 1 with $k = 20$ nodes and $l = 32$ arcs. Our example consists of finding best-case and worst-case bounds for the variance of the point estimate for this network.

To place this problem in the context of this paper, let

$$x_i = \begin{cases} 1 & \text{if arc } i \text{ operates,} \\ 0 & \text{otherwise,} \quad 1 \leqq i \leqq l, \end{cases}$$

$$\mathbf{x} = (x_1, \cdots, x_l),$$

$$\mathscr{X} = \{0, 1\}^l = \text{set to which all } \mathbf{x} \text{ belong.}$$

Then

$$\mathscr{X}^* = \{\mathbf{x} \in \mathscr{X} \colon x_1 + \cdots + x_l = k-1 \text{ and all nodes are connected}\}$$

denotes the set of all spanning trees of G with size $\tau = |\mathscr{X}^*|$, and $\mathscr{H}_0, \mathscr{H}_1, \cdots, \mathscr{H}_{l-k+1}$

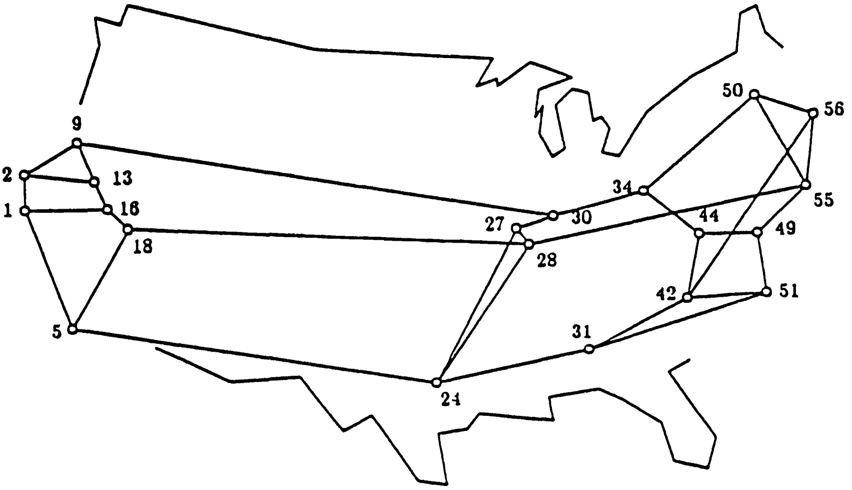


FIG. 1. *Skeleton of the* 1979 ARPANET.

is the partition of $\mathscr{X}^*$ of interest with

$$\phi_i(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{x} \in \mathscr{H}_i \\ 0, & \text{otherwise,} \end{cases}$$

$$w_i = p^{k-1}(1-p)^i,$$

and

$$\mu_i = H_i/\tau, \qquad i = 0, 1, \cdots, r = l - k + 1.$$

Suppose that we randomly draw $\mathbf{X}$ from $\mathscr{X}^*$. Then

$$Z = \tau[\phi_0(\mathbf{X})w_0 + \phi_1(\mathbf{X})w_1 + \cdots + \phi_r(\mathbf{X})w_r]$$

is an unbiased estimator of $m(p)$ in (6) with variance $\tau^2 h(\boldsymbol{\mu}, \mathbf{w})$ in (3). This is the essence of the Nel and Colbourn [5] approach.

Table 1 gives lower and upper bounds for $\{H_i\}$ based on the Ball-Provan method [2]. Note that these differ from the bounds given in [5] which are used to derive lower and upper bounds on $m(p)$, but not on each H_i. Since the lower and upper bounds are identical for H_i, $0 \leqq i \leqq 3$, there is no need to estimate these quantities. Accordingly, the worst-case variance is $\tau^2 h^*(\mathbf{w})$, where

$$h^*(\mathbf{w}) = \max_{\boldsymbol{\mu}} \left[\sum_{i=4}^{13} \mu_i w_i^2 - \left(\sum_{i=4}^{13} \mu_i w_i \right)^2 \right], \tag{7}$$

subject to

$$\mu_4 + \cdots + \mu_{13} = 1 - (H_0 + H_1 + H_2 + H_3)/\tau = \rho,$$

$$e_i \leqq \mu_i \leqq f_i, \qquad 4 \leqq i \leqq 13.$$

The suggested algorithm of § 1 and Appendix A continues to apply with α, defined in (A2), replaced everywhere in Appendix A by $\alpha^* = \alpha - 1 + \rho$. Table 1 also lists the lower and upper bounds for $\{\mu_i\}$.

Table 2 lists the resulting best-case and worst-case variance bounds apart from the factor τ^2. Observe that $3.90 \leqq h^*(\mathbf{w})/h_*(\mathbf{w}) \leqq 5.78$ for all values of p of interest. Also, the ratios $h^{**}(\mathbf{w})/h^*(\mathbf{w})$ make clear the benefit of solving (NLPW), suggesting that any worst-case sample size based on $\tau^2 h^{**}(\mathbf{w})$ would be at least 82.5 times greater than one based on $\tau^2 h^*(\mathbf{w})$.

TABLE 1
Bounds[1].

	Bounds on H_i		Bounds on μ_i	
i	Lower	Upper	e_i	f_i
0	1	1	$.987878\times 10^{-7}$	$.987878\times 10^{-7}$
1	19	19	$.187697\times 10^{-5}$	$.187697\times 10^{-5}$
2	190	190	$.187697\times 10^{-4}$	$.187697\times 10^{-4}$
3	1310	1310	$.129412\times 10^{-3}$	$.129412\times 10^{-3}$
4	1820	7107	$.179794\times 10^{-3}$	$.702085\times 10^{-3}$
5	6187	32148	$.611200\times 10^{-3}$	$.317583\times 10^{-2}$
6	18548	126141	$.183232\times 10^{-2}$	$.124612\times 10^{-1}$
7	50152	441066	$.495441\times 10^{-2}$	$.435720\times 10^{-1}$
8	122772	1401345	$.121284\times 10^{-1}$	.138436
9	256688	4105178	$.253576\times 10^{-1}$	.405542
10	307099	6369522	$.303376\times 10^{-1}$	.629231
11	0	5853352	0	.578240
12	0	5420570	0	.535486
13	0	5026666	0	.496573

[1] $\tau = 10122705$.

TABLE 2
Best-case and worst-case variance bounds.

				Basic indices for h^*		Basic variables	
p	$h_*(\mathbf{w})$	$h^*(\mathbf{w})$	$\dfrac{h^{**}(\mathbf{w})}{h^*(\mathbf{w})}$	i	j	μ_i	μ_j
.50	$.7418\times 10^{-17}$	$.4289\times 10^{-16}$	82.5	8	12	.138436	.249234
.60	$.8330\times 10^{-15}$	$.4091\times 10^{-14}$	148.6	8	12	.138436	.249234
.66	$.6963\times 10^{-14}$	$.3173\times 10^{-13}$	195.4	8	12	.119424	.268246
.70	$.2148\times 10^{-13}$	$.9413\times 10^{-13}$	226.4	7	12	.043572	.375542
.80	$.1102\times 10^{-12}$	$.4503\times 10^{-12}$	295.2	7	12	.043572	.375542
.90	$.3394\times 10^{-13}$	$.1339\times 10^{-12}$	340.7	6	12	.0124612	.414159
.92	$.1297\times 10^{-13}$	$.5094\times 10^{-13}$	346.3	6	12	.0124612	.414159
.94	$.2911\times 10^{-14}$	$.1140\times 10^{-13}$	350.7	6	12	.0124612	.414159
.96	$.2511\times 10^{-15}$	$.9815\times 10^{-15}$	353.9	5	12	.00317583	.424788
.98	$.2138\times 10^{-17}$	$.8349\times 10^{-17}$	355.7	5	12	.00317583	.424788
.99	$.1227\times 10^{-19}$	$.4791\times 10^{-19}$	356.2	5	12	.00317583	.424788

Notes:

1. $h^{**}(\mathbf{w}) = (\max_{4\leq i\leq 13} w_i - \min_{4\leq i\leq 13} w_i)^2/4$.

2. Variables with indices between i and j are set to their lower bounds; those with indices below i or above j are set to their upper bounds.

3. In all cases, the value of $\mu_j(=\mu_{12})$ is strictly between e_{12} and f_{12}. At $p = .66$, $\mu_i(=\mu_8)$ is strictly between e_8 and f_8; for all other cases, μ_i is at its upper bound f_i.

Note that the trivial worst-case bound $h^{**}(\mathbf{w})$ is evaluated after deleting the μ_i's for which $e_i = f_i$ in Table 1. Had this not been done, the resulting $h^{**}(\mathbf{w})$ would have been considerably greater in magnitude. The entry at $p = .66$ demonstrates a case at which two interior point assignments are made. For the true $\boldsymbol{\mu}$, $1.03 \leqq h^{*}(\mathbf{w})/\mathrm{h}(\boldsymbol{\mu}, \mathbf{w}) \leqq 1.28$ for all $p \in \{.5 + .01(i-1), 1 \leqq i \leqq 50\}$, revealing that, in this particular example, the bound is remarkably tight in spite of the fact that the individual bounds in Table 1 are not particularly tight. This observation strengthens the argument for computing the bound before sampling begins.

The results in Table 2 also show a limitation of this approach for estimating $m(p)$. Observe that the worst-case variance $\tau^2 h^*(\mathbf{w})$ exceeds unity for $p = .66$, .70, .80, .90, .92, and .94. In retrospect, this is not surprising when we rewrite (6) in the form

$$m(p) = \tau p^{k-1} \sum_{i=0}^{l-k+1} (H_i/\tau)(1-p)^i,$$

and recognize that observations on each trial of the experiment range over $[\tau p^{k-1}(1-p)^{l-k+1}, \tau p^{k-1}]$, which widens with increasing p. This suggests that an alternative approach based on generating random subgraphs and checking for connectedness may be a more efficient approach since the variance of each observation there never exceeds one-fourth. An adaptation of the Monte Carlo proposal in [3] to the all-points-connectedness problem would do just that.

Appendix A. Algorithmic details and complexity for worst-case variances.

A.1. Preliminaries. Let $\lambda \in [\underline{\lambda}, \bar{\lambda}]$ be given and let μ_i and μ_j with $i < j$ be optimal basic variables in (LPW(λ)). Let

$$\theta = w_i - w_j > 0, \tag{A1}$$

$$\alpha = 1 - \sum_{k=1}^{i-1} f_k - \sum_{k=i+1}^{j-1} e_k - \sum_{k=j+1}^{r} f_k, \tag{A2}$$

$$\beta = \sum_{k=1}^{i-1} w_k f_k + \sum_{k=i+1}^{j-1} w_k e_k + \sum_{k=j+1}^{r} w_k f_k. \tag{A3}$$

Then the optimal simplex tableau for (LPW(λ)) is given schematically in Table 3.

For the nonbasic columns, this schematic tableau only indicates the signs of the entries; their exact values are given by the formulas in the "Typical entry" column.

This tableau is optimal as long as λ satisfies

$$\begin{pmatrix} e_i \\ e_j \end{pmatrix} \leqq \frac{1}{\theta} \begin{pmatrix} -\alpha w_j - \beta + \lambda \\ \alpha w_i + \beta - \lambda \end{pmatrix} \leqq \begin{pmatrix} f_i \\ f_j \end{pmatrix},$$

that is, as long as $\lambda \in [\lambda_L, \lambda_U]$, where

$$\lambda_L = \max\{\lambda_{Li} = \theta e_i + \alpha w_j + \beta,\ \lambda_{Lj} = -\theta f_j + \alpha w_i + \beta\},$$

$$\lambda_U = \min\{\lambda_{Ui} = \theta f_i + \alpha w_j + \beta,\ \lambda_{Uj} = -\theta e_j + \alpha w_i + \beta\}.$$

TABLE 3

	Nonbasic at f	Basic	Nonbasic at e	Basic	Nonbasic at f	
	$\mu_1 \cdots \mu_{i-1}$	μ_i	$\mu_{i+1} \cdots \mu_{j-1}$	μ_j	$\mu_{j+1} \cdots \mu_r$	Typical entry
$z = \Phi(\lambda)$	$- \cdots -$	0	$+ \cdots +$	0	$- \cdots -$	$-(w_k - w_i)(w_k - w_j)$
$\mu_i = (1/\theta)(-\alpha w_j - \beta + \lambda)$	$+ \cdots +$	1	$+ \cdots +$	0	$- \cdots -$	$(w_k - w_j)/(w_i - w_j)$
$\mu_j = (1/\theta)(\alpha w_i + \beta - \lambda)$	$- \cdots -$	0	$+ \cdots +$	1	$+ \cdots +$	$(w_i - w_k)/(w_i - w_j)$

The quantities λ_L and λ_U result from substituting into (5) the values of μ_i^0 and μ_j^0 (with $\lambda = \lambda^0$) given in the tableau above. It is straightforward to verify the following result.

LEMMA A.1. *$\lambda_L = \lambda_{Li} \Leftrightarrow f_j + e_i \geqq \alpha$; $\lambda_L = \lambda_{Lj} \Leftrightarrow f_j + e_i \leqq \alpha$ and $\lambda_U = \lambda_{Ui} \Leftrightarrow f_i + e_j \leqq \alpha$; $\lambda_U = \lambda_{Uj} \Leftrightarrow f_i + e_j \geqq \alpha$.*

A.2. Moving from one basis to another. Suppose that Algorithm 1 has us decreasing λ.

1. If $\lambda_L = \lambda_{Li} \geqq \lambda_{Lj}$, then decreasing λ to λ_L introduces a primal degeneracy in row 1 (because μ_i is driven down to e_i).

(a) If $i \geqq 2$, then pivoting in row 1 and column $i-1$ yields an alternate optimal tableau for $\lambda = \lambda_L$ with μ_{i-1} basic at its upper bound f_{i-1}. Further small decreases in λ decrease μ_{i-1}, but row 1 remains primal feasible (i.e., $\mu_{i-1} > e_{i-1}$) for sufficiently small decreases.

(b) If $i = 1$, then decreasing λ below λ_L introduces a primal infeasibility in row 1 which cannot be removed by any pivot that maintains dual feasibility, i.e., $\lambda_L = \underline{\lambda}$.

2. If $\lambda_L = \lambda_{Lj} > \lambda_{Li}$, then decreasing λ to λ_L introduces a primal degeneracy in row 2 (because μ_j is driven up to f_j).

(a) If $j \geqq i+2$, then pivoting in row 2 and column $j-1$ yields an alternate optimal tableau for $\lambda = \lambda_L$ with μ_{j-1} basic at its lower bound e_{j-1}. Further small decreases in λ increase μ_{j-1}, but row 2 remains primal feasible (i.e., $\mu_{j-1} < f_{j-1}$) for sufficiently small decreases.

(b) If $j = i+1$ and $i \geqq 2$, pivoting in row 2 and column $i-1$ (and switching rows 1 and 2) yields an alternate optimal tableau in which μ_{i-1} is basic at its upper bound f_{i-1}, and the discussion in 1(a) applies to further decreases in λ.

(c) If $j = 2$ and $i = 1$, the discussion in 1(b) applies.

If we are increasing λ, analogous results hold. The result of this discussion leads to the following implementation of steps 2 and 3 of Algorithm 1, for which we need not explicitly compute any simplex tableaus:

2. Since $((w_i + w_j)/2) - \lambda_L < 0$, attempt to decrease λ below λ_L.

A. If $f_j + e_i \geqq \alpha$, and

(a) If $i \geqq 2$, then set

$$i' = i-1, \quad j' = j,$$
$$\alpha' = \alpha - e_i + f_{i-1},$$
$$\beta' = \beta + w_i e_i - w_{i-1} f_{i-1}.$$

(b) If $i = 1$, then $\lambda = \lambda_L = \underline{\lambda}$ is optimal for (PW).

B. If $f_j + e_i < \alpha$, and

(a) If $j \geqq i+2$, then set

$$i' = i, \quad j' = j-1,$$
$$\alpha' = \alpha - f_j + e_{j-1},$$
$$\beta' = \beta + w_j f_j - w_{j-1} e_{j-1}.$$

(b) If $j = i+1 \geqq 3$, then set

$$i' = i-1, \quad j' = j-1,$$
$$\alpha' = \alpha - f_{i+1} + f_{i-1},$$
$$\beta' = \beta + w_{i+1} f_{i+1} - w_{i-1} f_{i-1}.$$

(c) If $i = 1$ and $j = 2$, then $\lambda = \lambda_L = \underline{\lambda}$ is optimal for (PW).

3. Since $((w_i+w_j)/2)-\lambda_U>0$, attempt to increase λ above λ_U.

A. If $f_i+e_j \geqq \alpha$, and

(a) If $j \leqq r-1$, then set

$$i'=i, \quad j'=j+1,$$
$$\alpha'=\alpha-e_j+f_i,$$
$$B'=\beta+w_je_j-w_if_i.$$

(b) If $j=r$, then $\lambda=\lambda_U=\bar{\lambda}$ is optimal for (PW).

B. If $f_i+e_j<\alpha$, and

(a) If $j \geqq i+2$, then set

$$i'=i+1, \quad j'=j,$$
$$\alpha'=\alpha-f_i+e_{i+1},$$
$$\beta'=\beta+w_if_i-w_{i+1}e_{i+1}.$$

(b) If $j=i+1 \leqq r-1$, then set

$$i'=i+1, \quad j'=j+1,$$
$$\alpha'=\alpha-f_i+f_{i+2},$$
$$\beta'=\beta+w_if_i-w_{i+2}f_{i+2}.$$

(c) If $i=r-1$ and $j=r$, then $\lambda=\lambda_U=\bar{\lambda}$ is optimal for (PW).

A.3. Finding an initial basis. Apply the following algorithm.

```
ALGORITHM 3
0. Let $\delta_0=\sum_{k=1}^r e_k<1$. Set $i=0$.
1. (a) If $\delta_i<1$, let $\delta_{i+1}=\delta_i-e_{i+1}+f_{i+1}$. Set $i=i+1$ and return to step 1.
   (b) If $i=r$, then $\mu_{r-1}$ and $\mu_r$ are the initial basic variables.
   (c) If $\delta_i>1$ (and $i<r$), then $\mu_i$ and $\mu_r$ are the initial basic variables.
```

It should be pointed out that this algorithm for getting started is actually calculating the optimal basis for $\lambda=\bar{\lambda}$. Changing step 1(a) to $\delta_{i+1}=\delta_i-e_{r-i}+f_{r-i}$ and making corresponding changes in steps 1(b) and 1(c) calculates the optimal basis for $\lambda=\underline{\lambda}$. Equally simple modifications yield bases for values of λ well into the interior of $(\underline{\lambda}, \bar{\lambda})$.

A.4. Complexity of Algorithm 1. In the analysis, both additions and subtractions are grouped together as additions.

A.4.1. Initial basis (via Algorithm 3), $i, j, \alpha, \beta, \theta, \lambda_L$, and λ_U.

1. δ_0 requires $r-1$ additions.
2. Each repetition of step 1(a) requires one comparison and two additions.
3. Step 1(a) is repeated at most r times; i.e., for $i=0, 1, \cdots, r-1$.
4. Given the basic variables i and r, (A2) reduces to

$$\alpha=\begin{cases} 1-\sum\limits_{k=1}^{r-2} f_k=1-\delta_r+f_{r-1}+f_r & \text{if } i=r-1, \\ 1-\sum\limits_{k=1}^{i-1} f_k-\sum\limits_{k=i+1}^{r-1} e_k=1-\delta_i+f_i+e_r & \text{if } i<r-1. \end{cases}$$

Hence, the computation of α requires three additions. The computation of β, using (A3), requires $r-2$ multiplications and $r-3$ additions, and θ in (A1) requires one addition.

5. Given α, one addition and one comparison are needed to determine if $\lambda_L = \lambda_{Li}$ or $\lambda_L = \lambda_{Lj}$. Once this is known, the computation of λ_L requires two multiplications and two additions. The computation of λ_U is analogous.

6. Hence, finding the initial basis, $i, j, \alpha, \beta, \theta, \lambda_L$, and λ_U, requires at most $4r+6$ additions, $r+2$ multiplications, and $r+2$ comparisons.

A.4.2. Typical iteration.

1. Checking the current basis for optimality:
(a) For each basis, determining if $((w_i + w_j)/2) \in (\lambda_L, \lambda_U)$ requires one addition, one multiplication and two comparisons.
(b) For each basis after the first, determining if λ_L or λ_U is optimal requires an additional comparison.

2. Computing $i', j', \alpha', \beta', \theta', \lambda'_L$, and λ'_U if the current basis is not optimal:
(a) Each of i' and j' requires at most one addition.
(b) α' requires two additions.
(c) β' requires two multiplications and two additions.
(d) θ' requires one addition.
(e) Either $\lambda'_L = \lambda_U$ or $\lambda'_U = \lambda_L$. To compute the other member of $\{\lambda'_L, \lambda'_U\}$ requires three additions, two multiplications, and one comparison.

3. Hence, each iteration requires at most eleven additions, five multiplications, and four comparisons.

A.4.3. Number of iterations. Since the successive values of λ considered by the algorithm are either monotonically increasing or else monotonically decreasing, the successive values of i and j are similarly either monotonically nondecreasing or else monotonically nonincreasing (although at each iteration at least one of i or j will change). Hence the number of iterations is at most $2r-4$ (which occurs if initially $i=1$ and $j=2$, but the optimal basis has $i=r-1$ and $j=r$, or vice versa).

A.4.4. Finding the optimal solution, given $i, j, \alpha, \beta, \theta$, and λ. From Table 3, we see that the computation of μ_i and μ_j requires four multiplications and four additions.

A.4.5. Overall complexity.

	Additions	Multiplications	Comparisons
Initial basis	$4r+6$	$r+2$	$r+2$
Successive iterations	$11(2r-4)$	$5(2r-4)$	$4(2r-4)$
Optimal solution	4	4	0
Total	$26r-34$	$11r-14$	$9r-14$

Appendix B. Algorithmic details and complexity for best-case variances.

B.1. Preliminaries. The results from § A1 hold verbatim, with the exceptions that now the optimal solution has $\mu_1, \cdots, \mu_{i-1}$ nonbasic at their lower bounds, $\mu_{i+1}, \cdots, \mu_{j-1}$ nonbasic at their upper bounds, and $\mu_{j+1}, \cdots, \mu_r$ nonbasic at their lower bounds. In addition, we redefine

$$\alpha = 1 - \sum_{k=1}^{i-1} e_k - \sum_{k=i+1}^{j-1} f_k - \sum_{k=j+1}^{r} e_k, \tag{B1}$$

$$\beta = \sum_{k=1}^{i-1} w_k e_k + \sum_{k=i+1}^{j-1} w_k f_k + \sum_{k=j+1}^{r} w_k e_k. \tag{B2}$$

B.2. Moving from one basis to another. Recall that Algorithm 2 has us decreasing λ from $\bar{\lambda}$ to $\underline{\lambda}$.

1. If $\lambda_L = \lambda_{Lj} \geqq \lambda_{Li}$, then decreasing λ to λ_L introduces a primal degeneracy in row 2 (because μ_j is driven up to f_j).

   (a) If $j \leqq r-1$, then pivoting in row 2 and column $j+1$ yields an alternate optimal tableau for $\lambda = \lambda_L$ with μ_{j+1} basic at its lower bound e_{j+1}. Further small decreases in λ increase μ_{j+1}, but row 2 remains primal feasible (i.e., $\mu_{j+1} < f_{j+1}$) for sufficiently small decreases.

   (b) If $j = r$, then decreasing λ below λ_L introduces a primal infeasibility in row 2, which cannot be removed by any pivot that maintains dual feasibility, i.e., $\lambda_L = \underline{\lambda}$.

2. If $\lambda_L = \lambda_{Li} > \lambda_{Lj}$, then decreasing λ to λ_L introduces a primal degeneracy in row 1 (because μ_i is driven down to e_i).

   (a) If $i \leqq j-2$, then pivoting in row 1 and column $i+1$ yields an alternate optimal tableau for $\lambda = \lambda_L$ with μ_{i+1} basic at its upper bound f_{i+1}. Further small decreases in λ decrease μ_{i+1}, but row 1 remains primal feasible (i.e., $\mu_{i+1} > e_{i+1}$) for sufficiently small decreases.

   (b) If $i = j-1$ and $j \leqq r-1$, pivoting in row i and column $j+1$ (and switching rows 1 and 2) yields an alternate optimal tableau in which μ_{j+1} is basic at its lower bound e_{j+1}, and the discussion in 1(a) applies to further decreases in λ.

   (c) If $i = r-1$ and $j = r$, the discussion in 1(b) applies.

The preceding discussion leads to the following implementation of step 3 of Algorithm 2. Again, no simplex tableau is ever explicitly computed.

3A. If $f_j + e_i \leqq \alpha$ and

(a) If $j \leqq r-1$, then set

$$i' = i, \qquad j' = j+1,$$
$$\alpha' = \alpha - f_j + e_{j+1},$$
$$\beta' = \beta + w_j f_j - w_{j+1} e_{j+1}.$$

(b) If $j = r$, then $\lambda_L = \underline{\lambda}$, and $\boldsymbol{\mu}_B$ is optimal in (PB).

B. If $f_j + e_i > \alpha$, and

(a) If $i \leqq j-2$, then set

$$i' = i+1, \qquad j' = j,$$
$$\alpha' = \alpha - e_i + f_{i+1},$$
$$\beta' = \beta + w_i e_i - w_{i+1} f_{i+1}.$$

(b) If $i = j-1 \leqq r-2$, then set

$$i' = i+1, \qquad j' = j+1,$$
$$\alpha' = \alpha - e_{j-1} + e_{j+1},$$
$$\beta' = \beta + w_{j-1} e_{j-1} - w_{j+1} e_{j+1}.$$

(c) If $i = r-1$ and $j = r$, then $\lambda_L = \underline{\lambda}$ and $\boldsymbol{\mu}_B$ is optimal in (PB).

B.3. Finding an initial basis. Step 1 of Algorithm 3 is modified as follows:
1. (b) If $i=r$, then μ_1 and μ_r are the initial basic variables.
(c) If $\delta_i > 1$ (and $i < r$), then μ_1 and μ_i are the initial basic variables.

B.4. Complexity of Algorithm 2. This algorithm is essentially identical to Algorithm 1 except that

(i) We do not check the current basis for optimality, thereby saving one addition, one multiplication, and three comparisons.

(ii) At each basis, we compute

$$\varphi(\lambda_L) = \phi(\lambda_U) - (w_i + w_j)(\lambda_U - \lambda_L) - \lambda_L^2$$

and compare it to φ_B, thereby doing four additions, two multiplications, and one comparison.

(iii) For each basis with $\varphi(\lambda_L) < \varphi_B$, it takes four multiplications and four additions to compute $\boldsymbol{\mu}_B$ from the tableau with $\lambda = \lambda_L$.

Since these differences change the number of operations at each basis by a constant, and since there are at most $2r-4$ bases to consider, the overall time complexity of this algorithm is also $O(r)$, as claimed.

Acknowledgments. We are grateful to Professor Scott Provan for his advice on computing the Ball-Provan bounds and to Stephen Yarberry and Ben Hao Wang for their programming assistance.

REFERENCES

[1] M. O. BALL AND G. L. NEMHAUSER, *Matroids and a reliability analysis problem*, Math. Oper. Res., 4 (1979), pp. 132-143.

[2] M. O. BALL AND J. S. PROVAN, *Calculating bounds on reachability and connectedness in stochastic networks*, Networks, 13 (1983), pp. 253-278.

[3] G. S. FISHMAN, *A Monte Carlo sampling plan for estimating reliability parameters and related functions*, Networks, 17 (1987), pp. 169-186.

[4] K. G. MURTY, *Linear Programming*, John Wiley, New York, 1983.

[5] L. D. NEL AND C. J. COLBOURN, *Combining Monte Carlo estimates and bounds for network reliability*, Networks, 20 (1990), pp. 277-298.

SIAM J. Sci. Stat. Comput.
Vol. 13, No. 6, pp. 1361-1376, November 1992

STABILITY AND INSTABILITY IN THE COMPUTATION OF FLOWS WITH MOVING IMMERSED BOUNDARIES: A COMPARISON OF THREE METHODS*

CHENG TU†‡ AND CHARLES S. PESKIN†

Abstract. This paper describes thee different numerical methods for the computation of flows with moving immersed elastic boundaries. A two-dimensional incompressible fluid and a boundary in the form of a simple closed curve are considered. The inertia is assumed to be negligible and the Stokes equations are solved. The three methods are explicit, approximate-implicit, and implicit. The first two have been used before, but the implicit method is new in the context of flows with moving immersed boundaries. They differ only with respect to the computation of the boundary force. The results of the above methods at various values of the time-step size are compared in order to explore the numerical stability of the computation.

Key words. stability, computational fluid dynamics, immersed boundaries, Stokes flow

AMS(MOS) subject classifications. 65C20, 65N99, 76Z99

1. Introduction. The purpose of this paper is to present a new method for the study of flows with immersed elastic boundaries and to compare its stability with two other existing methods [10], [11]. The essential features of these methods are: (1) that the fluid computation is done on a fixed, regular computational lattice, (2) that the (Lagrangian) representation of the immersed boundary is independent of this lattice and involves a collection of moving material points, (3) that the immersed boundary acts on the fluid by means of a system of forces computed from the elastic stresses in the immersed boundary and applied to the nearby lattice points of the fluid with the help of a computational model of the Dirac δ-function, and (4) that the representative material points of the immersed boundary move at the local fluid velocity, which is obtained by interpolation from the nearby lattice points of the fluid. The same δ-function weights are used in the interpolation step as in the application of the boundary forces to the fluid. Methods of this general type have now been applied to blood flow in the heart [5], [6], [7], [9]; aquatic animal locomotion [1]; platelet aggregation during blood clotting [2]; and flows with suspended particles [3]. Much of the recent work in this area has been concerned with the supercomputer implementation of such methods in the three-dimensional case [4], [8], [12].

In all of the applications cited above, there has been a serious issue of numerical stability. Explicit computation of the boundary forces (i.e., the computation of these forces from the boundary configuration at the beginning of each time step) has typically led to explosively unstable results when computations with reasonable time steps have been tried.

The practical cure for this problem, first introduced in [10], has been to compute the boundary forces from an *estimate* of the boundary configuration at the end of the

* Received by the editors August 13, 1990; accepted for publication (in revised form) February 21, 1992. This work was supported by the National Institutes of Health under Research grant HL-17859. Computation was supported in part by the Minnesota Supercomputer Center and by Department of Energy contract DE-AC02-76ER03077 at the Courant Mathematics and Computing Laboratory, New York University.

† Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, New York 10012.

‡ Present address, Université Catholique de Louvain, Unité de Mécanique Appliquée, Place du Levant 2, P-1348 Louvain-la-Neuve, Belgium.

time step. For this to work, it is essential that the above-mentioned estimate take into account the influence of the (unknown) boundary forces themselves on the (unknown) boundary configuration at the end of the time step. Until now, the estimate that has been used for the boundary configuration at the end of the time step is based on the assumption that the force applied to the fluid by each boundary point acts exclusively on that point and not on any of the others. (This assumption is only made in the computation of the boundary forces. Once those forces are computed, they are applied to the fluid, and each element of force acts instantaneously everywhere, since the fluid is incompressible.)

In this paper, we compare three methods which differ only with regard to the computation of the boundary force. The three methods are:

(1) Explicit: boundary force computed from the boundary configuration at the *beginning* of the time step.

(2) Approximate-implicit: boundary force computed from an *estimate* of the configuration at the *end* of the time step.

(3) Implicit: boundary force computed from the configuration at the *end* of the time step.

Note that method (1) is essentially the forward-Euler method, while method (3), the principal subject of this paper, is essentially the backward-Euler method. Method (2), the one that has been used in practice, represents a compromise in which one hopes to achieve the good stability properties of the backward-Euler method at a smaller computational cost. In method (1), the boundary forces are obtained by evaluation of a definite formula; there are no equations to solve. In method (2), one must solve a nonlinear fixed-point problem for a self-consistent set of boundary forces and (estimated) final boundary configuration. Because only an estimate of the final boundary configuration is used, however, this nonlinear system may be formulated on the boundary itself, without reference to the fluid. Thus, method (2) leads to sparse systems of equations, since each boundary point is coupled only to its neighbors. In method (3), by contrast, the nonlinear fixed-point problem for the boundary forces involves the fluid-mediated interaction of each boundary point with every other boundary point. This leads to dense systems of equations which are much more expensive to solve.

The model problem that is used as a vehicle for the comparison of the three methods is as follows. We consider a two-dimensional incompressible fluid containing an immersed elastic boundary in the form of a simple closed curve. We assume that inertia is everywhere negligible, and so we use the Stokes equations, in which time enters in only as a parameter. Initially, the boundary is in the form of an ellipse, and it relaxes to a circular configuration. Area should be preserved during this transformation, since the fluid is incompressible. We compare the three methods for this problem at various values of the time-step parameter.

2. Assumptions and equations. We consider a two-dimensional viscous incompressible fluid containing an immersed elastic boundary. We assume that effects of inertia are negligible both for the fluid and for the immersed boundary. The boundary is in the form of a simple closed curve. Note that both the fluid inside and the fluid outside the immersed boundary curve are physically significant in the problem and influence the motion of the boundary. Although one might like the external region to extend to infinity, we put the entire problem in a periodic box. This keeps the domain finite while retaining the feature of translation invariance that the unbounded problem would have.

On the basis of the above assumptions, we can write the equations of motion that are valid for both the fluid and the immersed boundary as the Stokes equations

$$0 = -\nabla p + \eta \Delta \mathbf{u} + \mathbf{F}(\mathbf{x}, t), \tag{2.1}$$

$$0 = \nabla \cdot \mathbf{u}, \tag{2.2}$$

where $\mathbf{u}$ is the fluid velocity, p is the fluid pressure, η is the constant fluid viscosity, and $\mathbf{F}$ is the force density, which differs from zero only on the immersed boundary and which is infinite there; see below.

In order to calculate the force density $\mathbf{F}$ which arises from the elastic stress in the immersed boundary, we need to specify the material points of the boundary. Thus, we need a Lagrangian description of the boundary. Let $\mathbf{x} = \mathbf{X}(s, t)$ be the position of the material point whose label is s at time t. The parameter s indicates the unstressed length along the immersed boundary from some reference point denoted by $s = 0$. Let T be the tension in the immersed boundary. We assume that T obeys a possibly nonlinear generalization of Hooke's law

$$T = \sigma\left(\left|\frac{\partial \mathbf{X}}{\partial s}\right|\right); \tag{2.3}$$

then it can be verified [12] that the local density (with respect to the measure ds) of force applied by the boundary to the fluid is

$$\mathbf{f} = \frac{\partial}{\partial s}(T\boldsymbol{\tau}), \tag{2.4}$$

where $\boldsymbol{\tau}$ is the unit tangent to the boundary

$$\boldsymbol{\tau} = \frac{\partial \mathbf{X}}{\partial s}\bigg/\left|\frac{\partial \mathbf{X}}{\partial s}\right|. \tag{2.5}$$

To connect the boundary with the fluid we need to relate $\mathbf{F}$ and $\mathbf{f}$. Following [11], we can establish this correspondence using a Dirac δ-function:

$$\mathbf{F}(\mathbf{x}, t) = \int_0^{L_b} \mathbf{f}(s, t)\delta(\mathbf{x} - \mathbf{X}(s, t))\, ds, \tag{2.6}$$

where the integral is over the entire boundary and δ is a two-dimensional δ-function $\delta(\mathbf{x}) = \delta(x_1)\delta(x_2)$ where $\mathbf{x} = (x_1, x_2)$. The final assumption is that the fluid adheres to the boundary. Thus, we impose the no-slip condition

$$\frac{\partial \mathbf{X}}{\partial t}(s, t) = \mathbf{u}(\mathbf{X}(s, t), t). \tag{2.7}$$

Note that this plays the role of an equation of motion for the boundary and not the more standard role of a constraint on the fluid motion, since the boundary motion is unknown. We remark that $\mathbf{u}$ in (2.7) can be expressed in a form similar to (2.6) by using the definition of the δ-function

$$\mathbf{u}(\mathbf{X}(s, t), t) = \int_\Omega \mathbf{u}(\mathbf{x}, t)\delta(\mathbf{x} - \mathbf{X}(s, t))\, d\mathbf{x}. \tag{2.8}$$

We can summarize the equations of motion as follows: (2.1) and (2.2) are the Stokes equations of a viscous incompressible fluid with viscosity η. Equations (2.3)–(2.5) are elastic boundary equations enabling us to compute the boundary force density. Equations (2.6)–(2.8) are the connecting equations, which give the interaction between the boundary and the fluid.

3. Dimensionless form of the equations. It is useful to write these equations in terms of dimensionless variables. In this way once the solution for a particular flow with a particular immersed elastic boundary is known, we can obtain the solutions that are dynamically similar.

First, let us consider the equations describing the motion of the fluid. We arbitrarily choose a constant time t_0 (which will be specified at the end of this section) and a constant length L, which is the length of the regular square box on which the Stokes equations are solved. We then make the following change of variables:

$$t' = \frac{t}{t_0}, \quad \mathbf{x}' = \frac{\mathbf{x}}{L}, \quad \mathbf{u}' = \frac{\mathbf{u}t_0}{L}, \quad p' = \frac{pt_0}{\eta}, \qquad \mathbf{F}' = \frac{\mathbf{F}Lt_0}{\eta}.$$

Note that the variables denoted with the superscript $'$ are dimensionless for a two-dimensional fluid. Using these new variables we also have

$$\nabla = \frac{1}{L}\nabla', \qquad \Delta = \frac{1}{L^2}\Delta'.$$

Substituting the above into the fluid equations in the previous section gives

$$0 = -\nabla' p' + \Delta' \mathbf{u}' + \mathbf{F}', \tag{3.1}$$

$$0 = \nabla' \cdot \mathbf{u}'. \tag{3.2}$$

Next, we consider the equations for the elastic boundary. The particular choice of the function σ in this paper is

$$\sigma\left(\left|\frac{\partial \mathbf{X}}{\partial s}\right|\right) = \begin{cases} S_b\left(\left|\frac{\partial \mathbf{X}}{\partial s}\right| - 1\right), & \left|\frac{\partial \mathbf{X}}{\partial s}\right| \geqq 1, \\ 0, & \text{otherwise,} \end{cases}$$

where S_b is the stiffness of the elastic boundary. We make the following changes of variables:

$$\mathbf{X}' = \frac{\mathbf{X}}{L}, \quad s' = \frac{s}{L}, \quad T' = \frac{T}{S_b},$$

$$\sigma' = \frac{\sigma}{S_b}, \quad \boldsymbol{\tau}' = \boldsymbol{\tau}, \quad \mathbf{f}' = \frac{\mathbf{f}L}{S_b}.$$

Then (2.3)-(2.5) become

$$T' = \left(\left|\frac{\partial \mathbf{X}'}{\partial s'}\right| - 1\right), \tag{3.3}$$

$$\mathbf{f}' = \frac{\partial}{\partial s'}(T'\boldsymbol{\tau}'), \tag{3.4}$$

$$\boldsymbol{\tau}' = \frac{\partial \mathbf{X}'}{\partial s'} \bigg/ \left|\frac{\partial \mathbf{X}'}{\partial s'}\right|. \tag{3.5}$$

Finally, we consider the equations that describe the interaction of the boundary and the fluid. From the properties of the δ-function, we know that $\delta(Lx') = (1/L)\delta(x')$. Thus $\delta(\mathbf{x}) = (1/L^2)\delta(\mathbf{x}')$ for the two-dimensional δ-function.

Now we substitute the primed variables into (2.6)-(2.8) to obtain

$$\mathbf{F}'(\mathbf{x}', t') = \frac{S_b t_0}{\eta L} \int_0^{L_b/L} \mathbf{f}'(s', t')\delta(\mathbf{x}' - \mathbf{X}'(s', t'))\, ds', \tag{3.6}$$

$$\frac{\partial \mathbf{X}'}{\partial t'}(s', t') = \mathbf{u}'(\mathbf{X}'(s', t'), t'), \tag{3.7}$$

$$\mathbf{u}'(\mathbf{X}'(s', t'), t') = \int_\Omega \mathbf{u}'(\mathbf{x}', t')\delta(\mathbf{x}' - \mathbf{X}'(s', t'))\, d\mathbf{x}', \tag{3.8}$$

where L_b is the unstressed length of the boundary. If we pick t_0 such that $S_b t_0/\eta L = 1$, then the only parameter in this problem is L_b/L, which is dimensionless.

4. Discretization of the dimensionless equations. We now drop the primes in the dimensionless equations and discretize them for the numerical computation. The numerical scheme described below generalizes readily to the three-dimensional case, although the description of the immersed elastic boundary is then more complicated.

First, we introduce the notation that will be used to describe the numerical method. The Stokes equations are solved on a regular square box of length 1 with periodic boundary conditions. Let N be the number of lattice points in each direction; then $h = 1/N$ is the lattice width. We divide time into time steps of size Δt and use notation $\mathbf{u}_{ij}^n$ and $\mathbf{F}_{ij}^n$ for flow and force density evaluated at $t = n\Delta t$, $\mathbf{x} = (ih, jh)$. Note that all functions of $\mathbf{x}$ are periodic with period 1 in each direction, that is, all arithmetic involving lattice coordinates is modulo N.

We introduce the forward, backward, and centered difference operators as follows:

$$(D^+ g)_i = (g_{i+1} - g_i)/h,$$

$$(D^- g)_i = (g_i - g_{i-1})/h,$$

$$(D^0 g)_i = (g_{i+1} - g_{i-1})/2h.$$

We also use the subscript s to indicate the variable with respect to which the differencing is accomplished ($s = 1, 2$ denotes the two spatial dimensions), for example,

$$(D_1^+ \phi)_{ij} = (\phi_{i+1,j} - \phi_{ij})/h.$$

Moreover, let

$$\mathscr{D} \cdot \mathbf{u} = D_1^0 u_1 + D_2^0 u_2 \sim \nabla \cdot \mathbf{u}, \tag{4.1}$$

$$\mathscr{G} p = (D_1^0 p, D_2^0 p) \sim \nabla p, \tag{4.2}$$

$$\mathscr{L}\mathbf{u} = D_1^+ D_1^- \mathbf{u} + D_2^+ D_2^- \mathbf{u} \sim \Delta \mathbf{u}. \tag{4.3}$$

We discretize the Stokes equations with the above notation:

$$0 = -\mathscr{G} p + \mathscr{L}\mathbf{u} + \mathbf{F}, \tag{4.4}$$

$$0 = \mathscr{D} \cdot \mathbf{u}. \tag{4.5}$$

We now proceed to describe the notation used for discretization of the boundary equations. Recall that L_b here denotes $L_b' = L_b/L$, the dimensionless unstressed length of the boundary. Let N_b be the number of Lagrangian points used to represent the boundary. We define $\Delta s = L_b/N_b$ and use the notation $\mathbf{X}_k^n$ to denote the position of the kth point on the boundary at $t = n\Delta t$ where the subscript $k = 0 \cdots (N_b - 1)$. Since the boundary is a closed curve, the arithmetic on k is modulo N_b. According to the definition of the boundary tension T and the tangent vector $\boldsymbol{\tau}$, we discretize them as

$$T_{k+1/2} = \sigma(|\mathbf{X}_{k+1} - \mathbf{X}_k|/\Delta s) \tag{4.6}$$
$$= \begin{cases} \dfrac{|\mathbf{X}_{k+1} - \mathbf{X}_k|}{\Delta s} - 1, & \dfrac{|\mathbf{X}_{k+1} - \mathbf{X}_k|}{\Delta s} \geqq 1, \\ 0, & \text{otherwise}, \end{cases}$$

$$\boldsymbol{\tau}_{k+1/2} = \frac{\mathbf{X}_{k+1} - \mathbf{X}_k}{|\mathbf{X}_{k+1} - \mathbf{X}_k|}. \tag{4.7}$$

Finally, the discrete force density $\mathbf{f}_k$ is given by

$$\mathbf{f}_k = \frac{T_{k+1/2}\boldsymbol{\tau}_{k+1/2} - T_{k-1/2}\boldsymbol{\tau}_{k-1/2}}{\Delta s}. \tag{4.8}$$

Note that these equations define functions $\mathbf{f}_k = \mathbf{f}_k(\mathbf{X}_0, \mathbf{X}_1, \cdots, \mathbf{X}_{N_b-1})$ but that each $\mathbf{f}_k$ depends only on $\mathbf{X}_{k-1}$, $\mathbf{X}_k$, and $\mathbf{X}_{k+1}$.

We find that $\mathbf{f}_k$ can also be computed from an elastic energy function defined as

$$E(\mathbf{X}_0, \mathbf{X}_1, \cdots, \mathbf{X}_{N_b-1}) = \sum_{k=0}^{N_b-1} \mathscr{E}\left(\frac{|\mathbf{X}_{k+1} - \mathbf{X}_k|}{\Delta s}\right)\Delta s. \tag{4.9}$$

If $\mathscr{E}' = \sigma$, then grad $E = -\mathbf{f}$.

We now consider the equations connecting the fluid lattice and the boundary points. Since the positions of the boundary points generally do not coincide with those of the lattice points, we have to interpolate the velocity field from the fluid lattice to the boundary points and spread the boundary force from the boundary points to the nearby lattice points of the fluid. This problem is solved by introducing a sufficiently smooth approximation to the Dirac δ-function:

$$\delta_h(\mathbf{x}) = \delta_h(x_1)\delta_h(x_2), \tag{4.10}$$

where

$$\delta_h(x) = \begin{cases} \dfrac{1}{4h}\left(1 + \cos\dfrac{\pi x}{2h}\right), & |x| \leqq 2h, \\ 0, & |x| > 2h. \end{cases} \tag{4.11}$$

The reasons for the particular choice of δ-function are given in [11].

We define

$$\mathbf{U}_k = \sum_{i,j=0}^{N-1} \mathbf{u}_{ij}\delta_h(\mathbf{x}_{ij} - \mathbf{X}_k)h^2 \tag{4.12}$$

for the approximation to the identity (3.8).

Similarly, a discrete form of the force density (3.6) is given by

$$\mathbf{F}_{ij} = \sum_{k=0}^{N_b-1} \mathbf{f}_k(\mathbf{X}_0, \mathbf{X}_1, \cdots, \mathbf{X}_{N_b-1})\delta_h(\mathbf{x}_{ij} - \mathbf{X}_k)\Delta s. \tag{4.13}$$

Finally, the no-slip condition is discretized as

$$\frac{\mathbf{X}_k^{n+1} - \mathbf{X}_k^n}{\Delta t} = \mathbf{U}_k. \tag{4.14}$$

In (4.12)-(4.14) we have omitted the time index on certain quantities. Different choices of this index (e.g., n or $n+1$) will result in the different schemes that we consider below.

5. An FFT Stokes solver. In this section we will describe a method to solve the Stokes equations using the fast Fourier transform (FFT). Suppose we are given the force density **F**. We need a device to find the velocity **u** and the pressure p that satisfy the Stokes equations.

We consider a grid function ϕ and define the discrete Fourier transformation of ϕ as

$$(5.1)\qquad \hat{\phi}_{k_1k_2}=\sum_{j_1,j_2=0}^{N-1} e^{-i(2\pi/N)(j_1k_1+j_2k_2)}\phi_{j_1j_2},\qquad 0\leqq k_1,\,k_2\leqq N-1.$$

According to this definition, the discrete Fourier transform of the Stokes equations is as follows:

$$(5.2)\quad 0=\frac{-i}{h}\sin\left(\frac{2\pi k_s}{N}\right)\hat{p}_{k_1k_2}-\frac{4}{h^2}\sum_{s'=1}^{2}\sin^2\left(\frac{\pi k_{s'}}{N}\right)(\hat{u}_s)_{k_1k_2}+(\hat{F}_s)_{k_1k_2},\qquad s=1,2,$$

$$(5.3)\qquad 0=\sum_{s=1}^{2}\frac{i}{h}\sin\left(\frac{2\pi k_s}{N}\right)(\hat{u}_s)_{k_1k_2}.$$

Multiplying by

$$\frac{i}{h}\sin\left(\frac{2\pi k_s}{N}\right)$$

on both sides of (5.2) and summing over s from 1 to 2, then making use of (5.3), we get an algebraic equation for the Fourier transform of the pressure. The solution of this equation is:

$$(5.4)\qquad \hat{p}_{k_1k_2}=\frac{-\sum_{s=1}^{2}(i/h)\sin(2\pi k_s/N)(\hat{F}_s)_{k_1k_2}}{\sum_{s=1}^{2}(1/h^2)\sin^2(2\pi k_s/N)}.$$

Once $\hat{p}$ is known, (5.2) can be solved for $\hat{\mathbf{u}}$ as follows:

$$(5.5)\qquad (\hat{u}_s)_{k_1k_2}=\frac{-(i/h)\sin(2\pi k_s/N)\hat{p}_{k_1k_2}+(\hat{F}_s)_{k_1k_2}}{(4/h^2)\sum_{s'=1}^{2}\sin^2(\pi k_{s'}/N)}.$$

We remark that certain values of k_s will cause difficulty in (5.4) and (5.5). When $(k_1,k_2)=(0,0)$, $(0,N/2)$, $(N/2,0)$, or $(N/2,N/2)$, $\hat{p}_{k_1k_2}$ is undefined; see (5.4). In those cases, however, the value of $\hat{p}$ has no effect on $\hat{\mathbf{u}}$; see (5.5). Therefore, we may set $\hat{p}=0$ for these values of (k_1,k_2). In the case $(k_1,k_2)=(0,0)$, however, there is an additional problem that is not present in the other three cases, since the denominator in (5.5) is zero. To obtain a solution, the numerator must be zero also, and this means that $\hat{\mathbf{F}}_{00}=0$. Note that this is equivalent to

$$\sum_{i,j=0}^{N-1}\mathbf{F}_{ij}=0.$$

The latter condition follows directly from the discrete Stokes equations in a periodic domain by summing over all grid points. In our case, we know that this condition must be satisfied since we construct **F** from forces generated in links, since each link generates a pair of equal and opposite forces, and since total force is preserved by our δ-function method of spreading the forces onto the fluid grid. In this indeterminate case, we may set $\hat{\mathbf{u}}=0$, for there is an arbitrary constant in the general solution of the discrete Stokes equations.

In summary, the procedure to compute velocity $\mathbf{u}$ and pressure p from a given force density $\mathbf{F}$ is as follows: first, compute the Fourier transform of $\mathbf{F}$. This is accomplished by a fast Fourier transformation. We then use (5.4) to compute $\hat{p}$, and once $\hat{p}$ is found, we use (5.5) to compute $\hat{\mathbf{u}}$. Finally, the velocity $\mathbf{u}$ and the pressure p can be calculated by an inverse Fourier transformation on $\hat{\mathbf{u}}$ and $\hat{p}$, respectively, using an FFT.

6. The explicit method. We now describe the first of the three numerical methods for computing the boundary force in the presence of an elastic immersed boundary. This is the simplest among them and takes the smallest amount of computer time (per time step) and storage space. However, this method is not always stable. When the size of the time step is big, the boundary will explode.

The first step in the algorithm to compute $\mathbf{X}^{n+1}$ given $\mathbf{X}^n$ is to evaluate the force density. This is done using the energy function described in § 4:

$$\mathbf{f}_k^n = -\frac{\partial E}{\partial \mathbf{X}_k}(\mathbf{X}_0^n, \mathbf{X}_1^n, \cdots, \mathbf{X}_{N_b-1}^n). \tag{6.1}$$

Once the forces $\mathbf{f}_k^n$ have been computed, we use the δ-function to spread the force out on the computational lattice

$$\mathbf{F}_{ij}^n = \sum_{k=0}^{N_b-1} \mathbf{f}_k^n \delta_h(\mathbf{x}_{ij} - \mathbf{X}_k^n)\Delta s. \tag{6.2}$$

We next find the velocity at the lattice points by means of the FFT Stokes solver and use the δ-function again to interpolate the velocity to the boundary points

$$\mathbf{U}_k^n = \sum_{i,j=0}^{N-1} \mathbf{u}_{ij}^n \delta_h(\mathbf{x}_{ij} - \mathbf{X}_k^n) h^2. \tag{6.3}$$

Finally, we update the positions of the boundary points using

$$\mathbf{X}_k^{n+1} = \mathbf{X}_k^n + \Delta t \mathbf{U}_k^n. \tag{6.4}$$

Results will be given after the other methods have been described.

7. The approximate-implicit method. This method is similar to the foregoing except that here the force density is computed implicitly to avoid numerical instability. We define points $\mathbf{X}_0^{n+1,*}, \mathbf{X}_1^{n+1,*}, \cdots, \mathbf{X}_{N_b-1}^{n+1,*}$ by a system of equations of the form

$$\mathbf{X}_k^{n+1,*} = \mathbf{X}_k^n + \Delta t \lambda \mathbf{f}_k(\mathbf{X}_0^{n+1,*}, \mathbf{X}_1^{n+1,*}, \cdots, \mathbf{X}_{N_b-1}^{n+1,*}), \tag{7.1}$$

where λ is the magnitude of the velocity induced at a point by a unit force applied to that point. It can be estimated either by applying a unit force at a point and then measuring the displacement of that point after one time step or from the fundamental solution of the discrete Stokes equations. (This fundamental solution plays a critical role in the method that is described in the following section.) The quantity $\mathbf{X}^{n+1,*}$ is intended to approximate the positions of the boundary points at the end of the time step. The formula for $\mathbf{X}^{n+1,*}$, (7.1), takes into account the influence of the boundary force $\mathbf{f}$ but only in an approximate way. The approximation is that each element $\mathbf{f}_k$ is allowed to act only on the point $\mathbf{X}_k$. This approximation is removed in the implicit method that is the subject of the next section.

We now describe the method to solve the nonlinear fixed-point problem given by (7.1). We introduce the notation $\mathbf{X} = (\mathbf{X}_0^{n+1,*}, \mathbf{X}_1^{n+1,*}, \cdots, \mathbf{X}_{N_b-1}^{n+1,*})$; $\mathbf{X}^0 = (\mathbf{X}_0^n, \mathbf{X}_1^n, \cdots, \mathbf{X}_{N_b-1}^n)$; and $\mathbf{f} = (\mathbf{f}_0, \mathbf{f}_1, \cdots, \mathbf{f}_{N_b-1})$. Now the system in (7.1) can be rewritten as

$$\mathbf{X} = \mathbf{X}^0 + \Delta t \lambda \mathbf{f}(\mathbf{X}). \tag{7.2}$$

We can form a problem equilvalent to (7.2), which solves for the force density **f** instead of **X**. The motivation for this is to permit the consistent comparison to the implicit method discussed in the next section. Since $\mathbf{f} = -\text{grad } E$, we have

$$\begin{aligned}\mathbf{f}(\mathbf{X}) &= -\nabla E(\mathbf{X})\\ &= -\nabla E(\mathbf{X}^0 + \Delta t\lambda \mathbf{f}(\mathbf{X})).\end{aligned}$$

Treating **f** as a new independent variable, we then get

$$\mathbf{f} + \nabla E(\mathbf{X}^0 + \Delta t \lambda \mathbf{f}) = 0. \tag{7.3}$$

Next we consider the function

$$\phi(\mathbf{f}) = \frac{1}{2}(\mathbf{f}, \mathbf{f}) + \frac{1}{\Delta t\lambda} E(\mathbf{X}^0 + \Delta t \lambda \mathbf{f}). \tag{7.4}$$

If E is continuous and bounded from below, then ϕ has an absolute minimum at some point $\mathbf{f}^*$. That is, $\phi(\mathbf{f}^*) \leqq \phi(\mathbf{f})$ for all **f**. If E is also differentiable, then $\mathbf{f}^*$ is a solution of our fixed-point problem (7.3).

The numerical solution of (7.3) is found by Newton's method. Let $\mathbf{f}^m$ be the mth guess. (Here we use superscripts for the iteration number within Newton's method. This should not be confused with the time step, which is fixed during this discussion.) Then $\mathbf{f}^{m+1}$ is obtained by solving the following system of equations:

$$(\mathbf{I} + \Delta t \lambda H_E(\mathbf{X}^0 + \Delta t \lambda \mathbf{f}^m))(\mathbf{f}^{m+1} - \mathbf{f}^m) = -(\mathbf{f}^m + \nabla E(\mathbf{X}^0 + \Delta t \lambda \mathbf{f}^m)). \tag{7.5}$$

Here H_E is the Hessian matrix of E.

It can easily be verified that this is a periodic block-tridiagonal system (with 2×2 blocks, since each $\mathbf{f}_k$ is a 2-vector). A sufficient condition for this system to be positive definite is that σ and the derivative of σ be positive (see [12]). The solution of such a periodic, block-tridiagonal system is also described in [12]. The repeated solution of (7.5) results in a sequence $\mathbf{f}^m$ which converges quadratically to a force density $\mathbf{f}^*$ that solves (7.3).

We have just described how to compute the boundary force density implicitly. Once this new force is found, we use it to define the force density **F** that acts on the fluid, and we continue with the rest of the algorithm as in the explicit method.

We remark that the argument of δ_h used in this method is the same as in the explicit method. That is, it involves $\mathbf{X}^n$, not $\mathbf{X}^{n+1}$ or $\mathbf{X}^{n+1,*}$. This is because $\mathbf{X}^{n+1}$ is not yet known and $\mathbf{X}^{n+1,*}$ involves displacements from $\mathbf{X}^n$ that were not generated by an incompressible flow.

8. The implicit method. This new method is different from the above two methods because here the boundary force is computed from the unknown configuration at the end of the time step. It makes use of the fundamental solution of the Stokes equations to compute the fluid-mediated influence of one boundary point on another.

Ordinarily, the fundamental solution would be associated with a unit singular point force located at the origin:

$$\mathscr{F}^s = \mathbf{e}_s \delta(\mathbf{x}), \qquad s = 1, 2, \tag{8.1}$$

where $\mathbf{e}_s$ are the unit vectors in each of the two coordinate directions and $\delta(\mathbf{x})$ is a two-dimensional Dirac δ-function. We mentioned earlier, however, that the existence of the solution in a periodic domain requires that

$$\int_\Omega \mathbf{F}\, d\mathbf{x} = 0, \tag{8.2}$$

or in discrete form,

$$\sum_{i,j=0}^{N-1} \mathbf{F}_{ij} = 0. \tag{8.3}$$

Thus we need to subtract a uniform background force from $\mathcal{F}^s$ so that (8.2) will be satisfied. We define $\mathcal{F}^s$ as follows:

$$\mathcal{F}^s = (\delta(\mathbf{x}) - 1)\mathbf{e}_s, \tag{8.4}$$

or in the discrete case,

$$\mathcal{F}^s_{ij} = \left(\frac{1}{h^2}\delta_{i0}\delta_{j0} - 1\right)\mathbf{e}_s. \tag{8.5}$$

We note that any $\mathbf{F}$ that satisfies (8.3) can be written as

$$\mathbf{F}_{ij} = h^2 \sum_{i',j'=0}^{N-1} \sum_{s=1}^{2} \mathcal{F}^s_{i-i',j-j'}(F_s)_{i'j'}. \tag{8.6}$$

This can be shown by substituting (8.5) into (8.6) and using the definition of the Kronecker delta and (8.3).

If we let the solution corresponding to the singular force $\mathcal{F}^s$ be $(\mathbf{u}^s, p^s)$, then the velocity field $\mathbf{u}$ corresponding to $\mathbf{F}$ can be expressed in terms of $\mathbf{u}^s$ in the same way as (8.6) by linearity

$$\mathbf{u}_{ij} = h^2 \sum_{i',j'=0}^{N-1} \sum_{s=1}^{2} \mathbf{u}^s_{i-i',j-j'}(F_s)_{i'j'}. \tag{8.7}$$

Note that $\mathbf{u}^s$ does not depend on time. We can compute it by the FFT Stokes solver once and for all and store it in an $N \times N$ array $\mathbf{G}$ of 2×2 matrices. $\mathbf{G}$ is the discrete Green's function of the Stokes equations on a periodic domain. Each entry of $\mathbf{G}$ is a 2×2 matrix, since $\mathbf{u}^s_{ij}$ is a 2-vector for each s: $s=1$ corresponds to the fundamental solution for the singular point force applied in the x direction, while $s=2$ corresponds to that in which the singular point force is applied in the y direction. With the notation $\mathbf{G}$, we are able to write (8.7) as

$$\mathbf{u}_{ij} = h^2 \sum_{i',j'=0}^{N-1} \mathbf{G}(i-i', j-j')\mathbf{F}_{i'j'}. \tag{8.8}$$

We now seek an implicit method to compute the force density. It follows from (4.12) and (4.14) that

$$\mathbf{X}^{n+1}_k = \mathbf{X}^n_k + \Delta t h^2 \sum_{i,j=0}^{N-1} \mathbf{u}^{n+1}_{ij}\delta_h(\mathbf{x}_{ij} - \mathbf{X}^n_k). \tag{8.9}$$

Note that the argument of δ_h here is also $\mathbf{X}^n$, as in the previous two methods. In this sense, this method is not completely implicit. We call it the "implicit method" because the boundary force is computed from $\mathbf{X}^{n+1}$.

Now if we substitute (8.8) into (8.9), we get

$$\mathbf{X}^{n+1}_k = \mathbf{X}^n_k + \Delta t h^4 \sum_{i,j=0}^{N-1} \sum_{i',j'=0}^{N-1} \mathbf{G}(i-i', j-j')\mathbf{F}^{n+1}_{i'j'}\delta_h(\mathbf{x}_{ij} - \mathbf{X}^n_k). \tag{8.10}$$

From (4.13) we deduce

$$\begin{aligned}\mathbf{X}^{n+1}_k = \mathbf{X}^n_k + \Delta t \Delta s h^4 \sum_{k'=0}^{N_b-1} \sum_{i,j=0}^{N-1} \sum_{i',j'=0}^{N-1} \mathbf{G}(i-i', j-j')\delta_h(\mathbf{x}_{ij} - \mathbf{X}^n_k)\\ \cdot\, \delta_h(\mathbf{x}_{i'j'} - \mathbf{X}^n_{k'})\mathbf{f}_{k'}(\mathbf{X}^{n+1}_0, \mathbf{X}^{n+1}_1, \cdots, \mathbf{X}^{n+1}_{N_b-1}).\end{aligned} \tag{8.11}$$

Let

$$\tilde{\mathbf{G}}_{kl}=h^4\sum_{i,j=0}^{N-1}\sum_{i',j'=0}^{N-1}\mathbf{G}(i-i',j-j')\delta_h(\mathbf{x}_{ij}-\mathbf{X}_k^n)\delta_h(\mathbf{x}_{i'j'}-\mathbf{X}_l^n).$$

We can then rewrite (8.11) as

$$\mathbf{X}_k^{n+1}=\mathbf{X}_k^n+\Delta t\,\Delta s\sum_{l=0}^{N_b-1}\tilde{\mathbf{G}}_{kl}\mathbf{f}_l(\mathbf{X}_0^{n+1},\mathbf{X}_1^{n+1},\cdots,\mathbf{X}_{N_b-1}^{n+1}).\tag{8.12}$$

Let $\mathbf{X}=(\mathbf{X}_0^{n+1},\mathbf{X}_1^{n+1},\cdots,\mathbf{X}_{N_b-1}^{n+1})$; $\mathbf{X}^0=(\mathbf{X}_0^n,\mathbf{X}_1^n,\cdots,\mathbf{X}_{N_b-1}^n)$; and $\mathbf{f}=(\mathbf{f}_0^{n+1},\mathbf{f}_1^{n+1},\cdots,\mathbf{f}_{N_b-1}^{n+1})$. Let $\tilde{\mathbf{G}}$ be the matrix whose k, l entry is $\tilde{\mathbf{G}}_{kl}$. Equation (8.12) becomes

$$\mathbf{X}-\mathbf{X}^0-\Delta t\,\Delta s\,\tilde{\mathbf{G}}\mathbf{f}(\mathbf{X})=0.\tag{8.13}$$

This is again a nonlinear fixed-point problem. It is useful to put this problem in the form grad $\phi=0$ because then the linear problems that arise at each iteration of Newton's method will be symmetric. This can be done by using $\mathbf{f}$ as the independent variable instead of $\mathbf{X}$ and then by multiplying by $\tilde{\mathbf{G}}$ on the left. The result is

$$\tilde{\mathbf{G}}\mathbf{f}+\tilde{\mathbf{G}}\nabla E(\mathbf{X}^0+\Delta t\,\Delta s\,\tilde{\mathbf{G}}\mathbf{f})=0,\tag{8.14}$$

in which the left-hand side is the gradient of the function ϕ defined as follows:

$$\phi(\mathbf{f})=\frac{1}{2}(\mathbf{f},\tilde{\mathbf{G}}\mathbf{f})+\frac{1}{\Delta t\,\Delta s}E(\mathbf{X}^0+\Delta t\,\Delta s\,\tilde{\mathbf{G}}\mathbf{f}).\tag{8.15}$$

Now two comments relating to $\tilde{\mathbf{G}}$ are in order. First, we see that if we replace $\tilde{\mathbf{G}}$ by $\lambda I/\Delta s$, we obtain (7.3), the same problem that is solved when we use the approximate-implicit method. This implies that another (equivalent) method of estimating λ is to multiply the average value of the diagonal element of $\tilde{\mathbf{G}}$ by Δs.

Second, we note that $\tilde{\mathbf{G}}$ is positive semidefinite. This can be shown by applying $\mathbf{u}\,\cdot$ to the discrete form of the Stokes equations and summing over all the lattice points

$$0=h^2\sum_{i,j=0}^{N-1}\mathbf{u}_{ij}\cdot(-\mathscr{G}p_{ij})+h^2\sum_{i,j=0}^{N-1}\mathbf{u}_{ij}\cdot\mathscr{L}\mathbf{u}_{ij}+h^2\sum_{i,j=0}^{N-1}\mathbf{u}_{ij}\cdot\mathbf{F}_{ij}.\tag{8.16}$$

The first term after summation by part gives

$$h^2\sum_{i,j=0}^{N-1}(\mathscr{D}\cdot\mathbf{u})_{ij}p_{ij},$$

which is zero.

Operating in the same way on the second term yields

$$-h^2\sum_{i,j=0}^{N-1}(D_1^+\mathbf{u}_{ij}\cdot D_1^+\mathbf{u}_{ij}+D_2^+\mathbf{u}_{ij}\cdot D_2^+\mathbf{u}_{ij}).$$

This is always less than or equal to zero. It is equal to zero only if $\mathbf{u}$ is constant. But since

$$\sum_{i,j=0}^{N-1}\mathbf{u}_{ij}=0,$$

it is equal to zero only if $\mathbf{u}=0$.

Now we consider the third term:

$$\begin{aligned} h^2 \sum_{i,j=0}^{N-1} \mathbf{u}_{ij} \cdot \mathbf{F}_{ij} &= h^2 \sum_{i,j=0}^{N-1} \mathbf{u}_{ij} \cdot \sum_{k=0}^{N_b-1} \mathbf{f}_k(\mathbf{X}_0, \mathbf{X}_1, \cdots, \mathbf{X}_{N_b-1})\delta_h(\mathbf{x}_{ij} - \mathbf{X}_k)\Delta s \\ &= \Delta s \sum_{k=0}^{N_b-1} \mathbf{f}_k(\mathbf{X}_0, \mathbf{X}_1, \cdots, \mathbf{X}_{N_b-1}) \cdot h^2 \sum_{i,j=0}^{N-1} \mathbf{u}_{ij}\delta_h(\mathbf{x}_{ij} - \mathbf{X}_k) \\ &= \Delta s \sum_{k=0}^{N_b-1} \mathbf{f}_k(\mathbf{X}_0, \mathbf{X}_1, \cdots, \mathbf{X}_{N_b-1}) \cdot \mathbf{U}_k \\ &= \Delta s \sum_{k=0}^{N_b-1} \mathbf{f}_k(\mathbf{X}_0, \mathbf{X}_1, \cdots, \mathbf{X}_{N_b-1}) \cdot \sum_{l=0}^{N_b-1} \tilde{\mathbf{G}}_{kl}\mathbf{f}_l(\mathbf{X}_0, \mathbf{X}_1, \cdots, \mathbf{X}_{N_b-1}) \\ &= \Delta s\, \mathbf{f} \cdot \tilde{\mathbf{G}}\mathbf{f}. \end{aligned}$$

Note that it is important that we use the same δ-function for interpolating the velocity field from the fluid lattice to the boundary points and for spreading the boundary force from the boundary points to the nearby lattice points of the fluid. If different δ-functions were used, we would not have obtained the above proof. Combining the above results, we see that $\tilde{\mathbf{G}}$ is positive semidefinite, as was to be shown.

The foregoing argument shows only that $\tilde{\mathbf{G}}$ is positive *semi*definite. Does $\tilde{\mathbf{G}}$ have a nontrivial null space? By examining the proof that $\tilde{\mathbf{G}}$ is positive semidefinite we can see that $\tilde{\mathbf{G}}f = 0$ if and only if the velocity field $\mathbf{u}$ that results from the application of $\mathbf{f}$ is constant and hence 0. In fact, this can happen for nontrivial $\mathbf{f}$ when the boundary points are too close together in relation to the fluid mesh. Roughly speaking, this difficulty arises when there are four or more immersed boundary points per mesh width. Although we have no proof, our experience is that $\tilde{\mathbf{G}}$ is positive *definite* when such high densities of boundary points are avoided. The following method for finding the numerical solution of (8.14) applies to the situation in which $\tilde{\mathbf{G}}$ is positive definite.

As before, this numerical solution is found by Newton's method. Let $\mathbf{f}^m$ be the mth guess. Then $\mathbf{f}^{m+1}$ is obtained by solving the following system of equations:

$$(8.17) \quad [\tilde{\mathbf{G}} + \Delta t\, \Delta s\, \tilde{\mathbf{G}}H_E(\mathbf{X}^0 + \Delta t\, \Delta s\, \tilde{\mathbf{G}}\mathbf{f}^m)\tilde{\mathbf{G}}](f^{m+1} - \mathbf{f}^m) = -(\tilde{\mathbf{G}}\mathbf{f}^m + \tilde{\mathbf{G}}\nabla E(\mathbf{X}^0 + \Delta t\, \Delta s\, \tilde{\mathbf{G}}\mathbf{f}^m)).$$

As promised, this is a symmetric system of equations. Since it is positive definite (if high density of boundary points are avoided), we can use Cholesky factorization to find the solution. Once $\mathbf{f}^{n+1}$ is found, we update the position of the boundary points by

$$(8.18) \qquad \mathbf{X}^{n+1} = \mathbf{X}^0 + \Delta t\, \Delta s\, \tilde{\mathbf{G}}\mathbf{f}^{n+1}.$$

Note that this method makes no direct use of the Stokes solver during the computation for each time step, but it needs the Stokes solver to precompute the matrix $\mathbf{G}$.

Note also that this method is very expensive. First, computing $\tilde{\mathbf{G}}$ requires $O(N_b^2)$ operations even when $\mathbf{G}$ has been precomputed. (Only $O(1)$ operations per element of $\tilde{\mathbf{G}}$ are required because of the local character of δ_h.) Second, computing the matrix for solving the system of equations in (8.17) requires two matrix multiplications, each of which is $O(N_b^3)$ for a dense matrix. Third, solving the system of equations also requires $O(N_b^3)$. In the presence of many boundary points, this will imply heavy demands on both computer time and storage.

9. Results. The results of three sets of numerical experiments are given in Figs. 1, 2, and 3. In the experiments shown in Fig. 1, we compare the results of the three

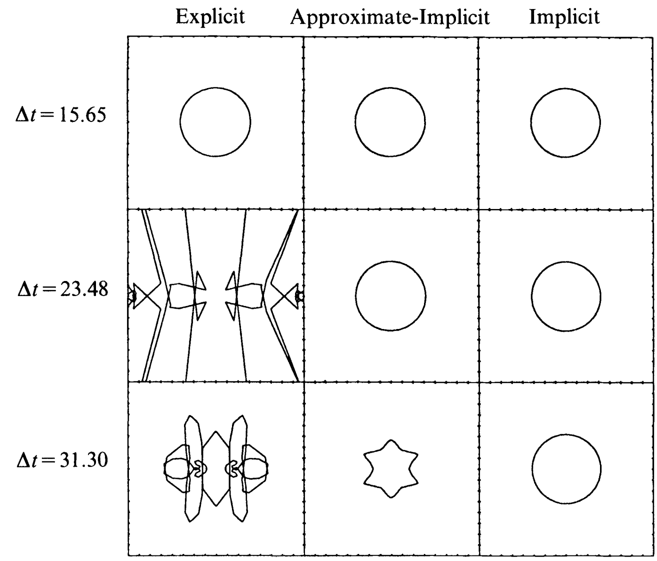


FIG. 1. *Results of the three methods with different step sizes. Only the configurations of the last time step are shown.*

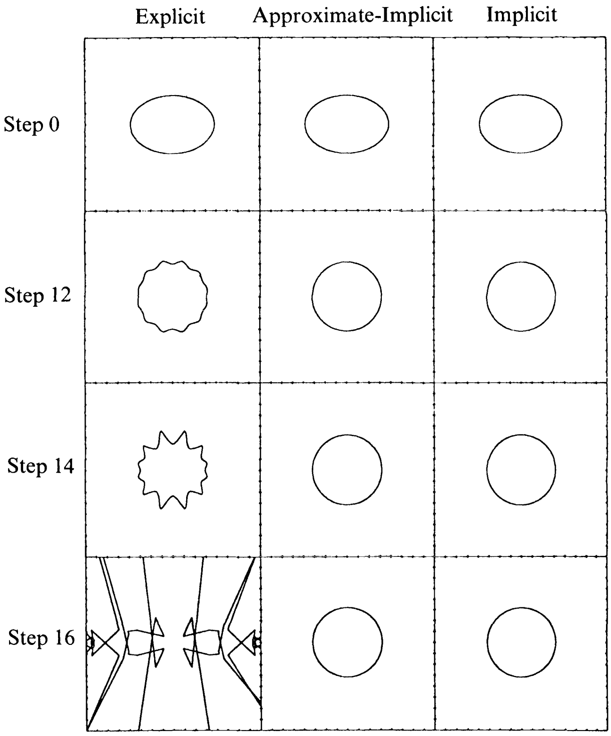


FIG. 2. *Results of the three methods with a fixed, intermediate step size. Time evolution of the boundary configuration in each case is depicted.*

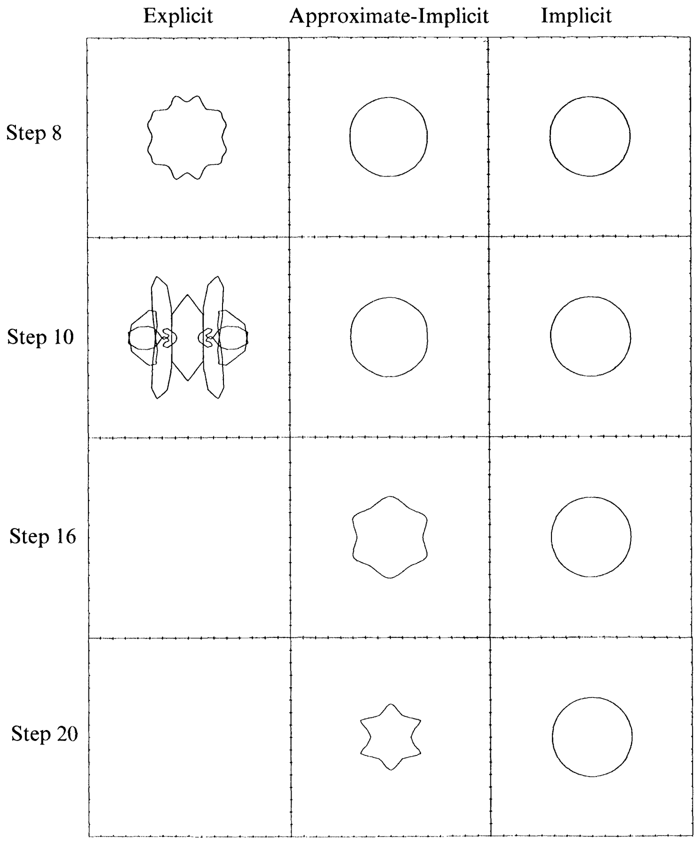


FIG. 3. *Results of the three methods with a fixed, large step size. Time evolution of the boundary configuration in each case is shown.*

TABLE 1
Areas inside the boundaries in Fig. 1.

	Explicit	Approximate-implicit	Implicit
$\Delta t = 15.65$	0.1244	0.1244	0.1246
$\Delta t = 23.48$	—	0.1236	0.1242
$\Delta t = 31.30$	—	0.0599	0.1236

TABLE 2
Areas inside the boundaries in Fig. 2.

Step No.	Explicit	Approximate-implicit	Implicit
0	0.1256	0.1256	0.1256
12	0.1222	0.1236	0.1242
14	0.1078	0.1236	0.1242
16	—	0.1236	0.1242

methods with different sizes of the time step parameter. Only the last step of each run is shown in Fig. 1. In Figs. 2 and 3, the time evolution is shown for all three methods at one fixed value of Δt. The area inside the boundary in each case is shown in Tables 1, 2, and 3.

The domain of the experiment is the 1×1 square box. We use a uniform square lattice with 64 points along each direction. The initial boundary configuration is an ellipse with 128 boundary points. Each experiment was allowed to run until the results diverged or until 20 time steps had been computed.

The pictures in the top row of Fig. 1 show the results of the three methods with a relatively small step size ($\Delta t = 15.65$) at time step 20. We find that in this circumstance all three methods are stable. The boundaries all converge to a circle. We see that the area inside the boundary is somewhat different in the three cases. The implicit method preserves the area better than the approximate-implicit method while the latter preserves the area better than the explicit method (the area within the initial boundary is 0.1256). In the middle row of Fig. 1 we depict the results of the three methods with a medium step size ($\Delta t = 23.48$). The implicit and approximate-implicit methods are shown at step 20 and the explicit method is shown at step 16, where it has already diverged. At this bigger step size, the explicit method is unstable. The boundary becomes very complex, diverging from the simple closed curve. However, the results for the other two methods are both stable and their boundaries converge to a circle, as in the case of a small step size. The results of the three methods with a relatively big step size ($\Delta t = 31.30$) are given in the bottom row of Fig. 1. The results are at step 20 except for the explicit method, which is at step 10. The explicit method is again unstable, with an explosive boundary diverging from the circle. At this high level of step size the approximate-implicit method shows signs of instability. While after 20 steps its boundary is still a simple closed curve, its star shape is a very poor approximation to a circle. Only the implicit method remains stable at this value of Δt.

We can obtain a more detailed comparison of the methods by observing the time evolution of the boundary configuration in each case. This is done in Fig. 2 for the intermediate step size ($\Delta t = 23.48$). The figure shows that both the implicit and approximate-implicit methods are stable and that their patterns of convergence over time are similar: the ellipse gradually becomes a circle. By contrast, the explicit method quickly diverges. By step 12, the boundary shows an oscillating pattern and eventually becomes a complicated curve.

The time evolution of the boundary configuration for the largest step size ($\Delta t = 31.30$) is shown in Fig. 3. At time step 0, the shapes are the same as in Fig. 2. The explicit method starts to show signs of instability at step 8 and quickly diverges at step 10. Beyond this step the results have exploded outside the boundary of the fluid domain. In the approximate-implicit method, the boundary becomes a star shape at step 16 and remains star-shaped at step 20. Only the implicit method is still stable in this case.

TABLE 3

Areas inside the boundaries in Fig. 3.

Step No.	Explicit	Approximate-implicit	Implicit
8	0.1157	0.1226	0.1239
10	—	0.1221	0.1238
16	—	0.1117	0.1237
20	—	0.0599	0.1236

10. Summary and conclusions. We have presented a new method for computing the boundary force in problems with an elastic immersed boundary and have compared its stability with two other existing methods. The results clearly rank these methods from the standpoint of stability. The implicit method has the best stability properties and the explicit method has the worst. The approximate-implicit method will be stable for a larger range of the time-step parameter than the explicit method but for a smaller range of that parameter than the implicit method. When viewed from the perspective of required computational time (per time step) and storage space, the methods have the reverse ordering. The explicit method is the most economical, the approximate-implicit comes next, and the implicit method is the most expensive. Thus there appears to be a trade-off between stability and cost of the computations.

Indeed, in its present form, the fully implicit method is probably too expensive for practical application. One purpose of the present investigation, however, was to determine whether substantial improvement in stability could be achieved through the use of the fully implicit approach, which had not previously been tried on this type of problem. The answer appears to be "yes," and this should serve as a motivation to develop more efficient implementations of the fully implicit scheme. Other challenges for the future include extending this work to the Navier–Stokes equations and to the three-dimensional case.

REFERENCES

[1] L. Fauci and C. S. Peskin, *A computational model of aquatic animal locomotion*, J. Comput. Phys., 77 (1988), pp. 85–108.

[2] A. L. Fogelson, *A mathematical model and numerical method for studying platelet adhesion and aggregation during blood clotting*, J. Comput. Phys. 56 (1984), pp. 111–134.

[3] A. L. Fogelson and C. S. Peskin, *A fast numerical method for solving the three-dimensional Stokes equations in the presence of suspended particles*, J. Comput. Phys. 79 (1988), pp. 50–69.

[4] S. Greenberg, D. M. McQueen, and C. S. Peskin, *Three-dimensional fluid dynamics in a two-dimensional amount of central memory*, in Wave Motion: Theory, Modeling, and Computation, A. J. Chorin, ed., Springer-Verlag, New York, 1987, pp. 85–146.

[5] D. M. McQueen, C. S. Peskin, and E. L. Yellin, *Fluid dynamics of the mitral valve: Physiological aspects of a mathematical model*, Amer. J. Physiol., 242 (1982), pp. H1095–H1110.

[6] D. M. McQueen and C. S. Peskin, *Computer-assisted design of pivoting-disc prosthetic mitral valves*, J. Thorac. Cardiovasc. Surg., 86 (1983), pp. 126–135.

[7] ———, *Computer-assisted design of butterfly bileaflet valves for the mitral position*, Scand. J. Thor. Cardiovasc. Surg., 19 (1985), pp. 139–148.

[8] ———, *A three-dimensional computational method for blood flow in the heart*: (II) *Contractile fibers*, J. Comput. Phys., 82 (1989), pp. 289–297.

[9] J. S. Meisner, D. M. McQueen, Y. Ishida, H. O. Vetter, U. Bortolotti, J. A. Strom, R. W. M. Frater, C. S. Peskin, and E. L. Yellin, *Effects of timing of atrial systole on* LV *filling and mitral valve closure: Computer and dog studies*, Amer. J. Physiol., 249 (1985), pp. H604–H619.

[10] C. S. Peskin, *Flow patterns around heart valves: A digital computer method for solving the equations of motion*, Ph.D. thesis, Department of Physiology, Albert Einstein College of Medicine, 1972.

[11] C. S. Peskin, *Numerical analysis of blood flow in the heart*, J. Comput. Phys., 25 (1977), pp. 220–252.

[12] C. S. Peskin and D. M. McQueen, *A three-dimensional computational method for blood flow in the heart*: (I) *Immersed elastic fibers in a viscous incompressible fluid*, J. Comput. Phys., 81 (1989), pp. 372–405.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 6, pp. 1377-1393, November 1992

007

NUMERICAL SOLUTION OF THE TIME-DEPENDENT AXISYMMETRIC BOUSSINESQ EQUATIONS ON PROCESSOR ARRAYS*

MICHAEL SCHÄFER†

Abstract. The paper deals with the numerical solution of time-dependent nonisothermal flow problems, governed by the axisymmetric incompressible Boussinesq equations, on array processors. Using finite difference methods for discretization and a pressure correction method in combination with a successive iteration process for linearization and decoupling of variables, the problem is approximated by a sequence of sparse linear systems. The parallel solution of these systems by preconditioned conjugate gradient (PCG) methods and multigrid (MG) methods is discussed. Numerical experiments on an array processor for a number of test problems, derived from a practical application in the field of crystal growth, are reported. Included are comparisons of the implicit Euler scheme and the Crank-Nicolson scheme for time discretization for different flows, as well as comparisons of PCG methods and MG methods.

Key words. viscous time-dependent flow, incompressible Boussinesq equations, finite differences, pressure correction methods, preconditioned conjugate gradient methods, multigrid methods, array processors

AMS(MOS) subject classifications. 65M05, 76D05

1. Introduction.

The numerical solution of time-dependent nonisothermal flow problems in cylindrical geometries is of practical interest in a variety of applications, especially in the field of material processing. Important examples are the common manufacturing techniques for semiconductor crystals, such as the widely used Czochralski method (see e.g., Derby and Brown [5]).

In this paper we consider the numerical solution of the time-dependent incompressible Boussinesq equations, which constitute an appropriate, widely accepted mathematical model for the flow problems considered here. We restrict ourselves to the axisymmetric case, but mention that a generalization of the examined methods to the fully three-dimensional case is straightforward.

It is well known that a numerical treatment of flow problems, especially in the time-dependent case, requires a large amount of computational work and storage capacity. Parallel computers, which are increasingly becoming available commercially, can help to treat such problems with the required accuracy in reasonable amounts of time.

Among the different concepts for parallel architectures, we concentrate here on the case of a two-dimensional mesh-connected array of processors working in single instruction multiple data (SIMD) mode (see, e.g., Hockney and Jesshope [10]), which seem to be a very natural architecture for the type of problems considered here. For the time being, the major advantages of SIMD architectures as against local memory multiple instruction multiple data (MIMD) machines (e.g., SUPRENUM, transputer arrays, etc.) are the larger numbers of processors and the much faster local data communication among the processors. The algorithms given in this paper take into account the considered architecture.

For space discretization, second-order finite differences on a staggered grid [7] are used. For time discretization, we consider the θ-method [4] including, in particular, the first-order implicit Euler (IE) scheme and the second-order Crank-Nicolson (CN)

* Received by the editors February 19, 1991; accepted for publication (in revised form) August 21, 1991.

† Lehrstuhl für Strömungsmechanik, University of Erlangen-Nürnberg, Cauerstr. 4, D-8520 Erlangen, Germany.

scheme. The solution of the resulting nonlinear algebraic systems, one for each timestep, is reduced to the solution of a sequence of linear systems by means of a pressure correction approach [11] and a successive iteration process. For the solution of these sparse linear systems, we consider basic iterative-method preconditioned conjugate gradient (PCG) methods and multigrid (MG) methods [6]. Special attention is paid to the basic iterative methods involved herein, which are the crucial components with regard to an efficient parallelization. We consider a parallel Gauss–Seidel method, based on a domain decomposition and a multicolor ordering of the unknowns, which is easy to implement and which results in efficient algorithms for the considered type of parallel architectures.

By various numerical experiments on a DAP 510, a SIMD machine with a 32×32 array of single-bit processors, we study the numerical behaviour and the computational efficiency of the considered algorithms. The test problems are derived from a model of Czochralski crystal growth, a practical application proposed in [18] as a benchmark problem. The tests include an investigation of the IE scheme and the CN scheme for different kinds of flows, as well as a comparison of the considered PCG methods and MG methods. The results show, as proved in [11] for two-dimensional isothermal flows for the CN scheme, that the pressure correction approach does not spoil the accuracy of the underlying time discretization scheme. Rather, if the driving forces of the flows are not too large, due to the higher order, the CN scheme is superior to the IE scheme. For large driving forces (e.g., high Rayleigh number flows), especially when Hopf bifurcation effects become apparent due to the better stability properties, the implicit Euler scheme clearly shows its advantages.

Concerning the sparse linear system solves, we already find that for relatively small problem sizes the multigrid methods are superior to the PCG methods. Clearly, due to the optimal order of MG methods, this superiority grows with the problem size. For smoothing in the MG methods, as well as for preconditioning in the CG methods, the proposed multicolor Gauss–Seidel (MCGS) method is more efficient than the Jacobi method, which is known as an optimally parallel algorithm.

2. Statement of the problem. We consider nonisothermal flows of an incompressible viscous fluid in a vertical circular cylinder of height H and radius R. The mathematical description of such flows can be derived from the basic conservation laws of continuum mechanics for mass, momentum, and energy. We assume the validity of the Boussinesq approximation [14] and an axisymmetric flow. Using a cylinder coordinate system (r, z) as shown in Fig. 1, the nondimensional balance equations for

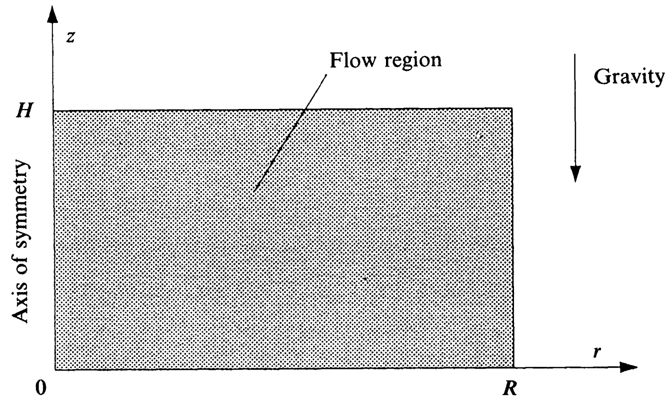


FIG. 1. *Problem configuration.*

the flow region can be written in the following compact form:

$$\begin{aligned} x_t+\mathbf{K}(x)x+\mathbf{G}p&=0,\\ Lx&=0, \end{aligned} \tag{1}$$

with

$$\mathbf{K}(x)=\begin{pmatrix} \mathbf{A}_u(u,w) & 0 & -\mathbf{B}(v) & 0\\ 0 & \mathbf{A}_w(u,w) & 0 & \mathbf{F}\\ \mathbf{B}(v) & 0 & \mathbf{A}_v(u,w) & 0\\ 0 & 0 & 0 & \mathbf{A}_T(u,w) \end{pmatrix},$$

$$\mathbf{G}=\begin{pmatrix}\mathbf{G}_r\\ \mathbf{G}_z\\ 0\\ 0\end{pmatrix},\quad \mathbf{L}=(\mathbf{L}_r,\mathbf{L}_z,0,0),$$

where $x=(u, w, v, T)^T$ and the differential operators are defined by

$$\begin{aligned} &\mathbf{A}_u(u,w)\phi=u\phi_r+w\phi_z-Pr(\Delta\phi-\phi/r^2),\\ &\mathbf{A}_w(u,w)\phi=u\phi_r+w\phi_z-Pr\Delta\phi,\\ &\mathbf{A}_v(u,w)\phi=\mathbf{A}_u(u,w)\phi,\\ &\mathbf{A}_T(u,w)\phi=u\phi_r+w\phi_z-\Delta\phi,\\ &\mathbf{G}_r\phi=\phi_r,\qquad \mathbf{G}_z\phi=\phi_z,\\ &\mathbf{L}_r\phi=(r\phi)_r/r,\qquad \mathbf{L}_z\phi=\phi_z,\\ &\mathbf{B}(v)\phi=v\phi/r,\qquad \mathbf{F}\phi=-RaPr\phi, \end{aligned}$$

for a scalar function ϕ. *Pr* and *Ra* are the Prandtl number and the Rayleigh number, respectively, and t denotes the time variable. A subscript denotes differentiation with respect to the subscript variable and Δ is the axisymmetric Laplace operator in cylindrical coordinates.

The (nondimensional) unknowns in (1) are the radial, axial, and azimuthal velocities u, w, and v; the temperature T; and the hydrostatic pressure p. In order to get a well-posed problem, the nonlinear system of partial differential equations (1) must be completed with boundary and initial conditions for the velocities and the temperature. On the symmetry axis $r=0$, we have the symmetry conditions

$$u=v=w_r=T_r=p_r=0. \tag{2}$$

We consider the numerical solution of (1), together with (2) and some boundary and initial conditions, on a parallel computer consisting of a mesh-connected array of $P_x \times P_y$ processors with local memories (Fig. 2). We assume that the processors work

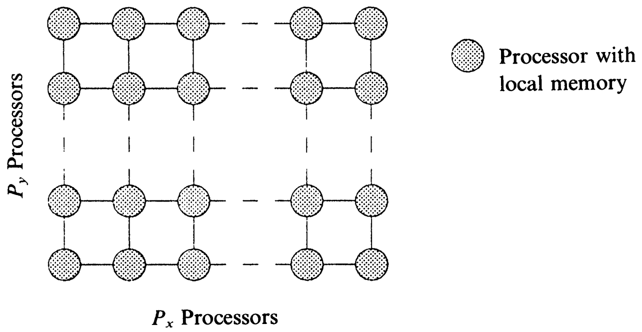


FIG. 2. *Processor array.*

in SIMD mode, which means that all processors perform the same operation at the same time, but on different data. As will be seen, for the type of problems considered here, the SIMD principle means no restriction with respect to the parallelization of the arithmetic operations, and it allows for a very fast local communication. Examples of currently available SIMD array computers are the distributed array processor (DAP), the connection machine, or the MasPar data-parallel computer.

3. Discretization and mapping. For the space discretization of (1), we use finite differences on a staggered grid, as introduced by Harlow and Welch [7]. The staggered arrangement of the unknowns and the grouping into cells is illustrated in Fig. 3 for an 8 × 4 grid. As discussed, for instance, in [15], a staggered grid has several advantages over an ordinary grid. Replacing the spatial derivatives by central differences and using standard second-order interpolation when needed (in consequence of the staggered grid), we obtain a system of ordinary differential equations for the unknown grid functions x_b and p_h:

$$\begin{aligned} x_{ht} + \mathbf{K}_h(x_h)x_h + \mathbf{G}_h p_h &= 0, \\ \mathbf{L}_h x_h &= 0. \end{aligned} \tag{3}$$

By Taylor expansion it can easily be shown that (3) is an $O(h^2)$ consistent approximation of (1), where $h > 0$ is a measure of the resolution of the grid (e.g., the maximum grid spacing). We remark that the boundary conditions for x are already included in (3). Because of the staggered grid, no artificial pressure boundary conditions are required. The initial conditions x_{0h} for (3) can be obtained as restrictions of the initial values for (1) to the grid points.

In order to allow an efficient mapping into our processor array, it is advantageous to take into account the size of the processor when choosing the grid. We assume here that the number of cells is given by

$$N_r \times N_z = k_r P_x \times k_z P_y$$

with some positive integers k_r and k_z. We subdivide the cell grid into $k_r \times k_z$ parts and map each part to one processor. As an example, Fig. 4 shows the mapping of an 8 × 4

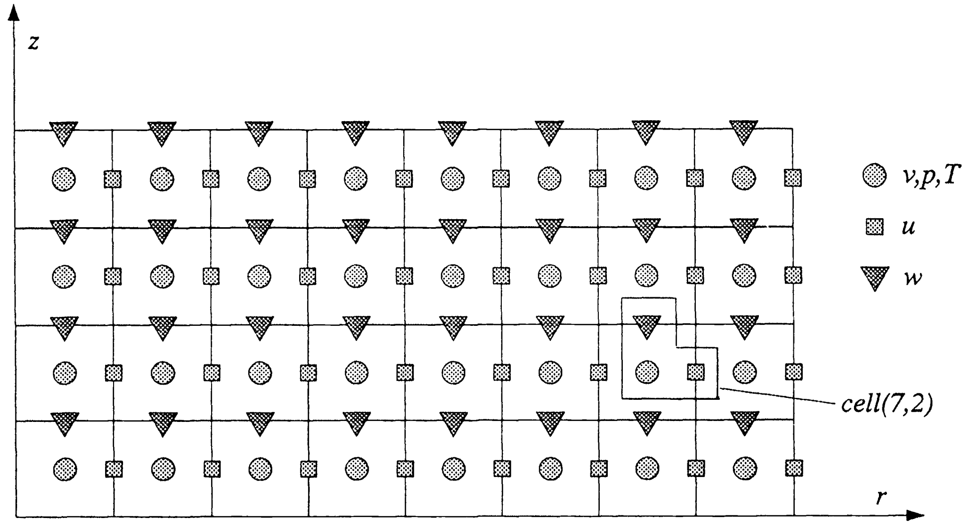


FIG. 3. *Staggered grid.*

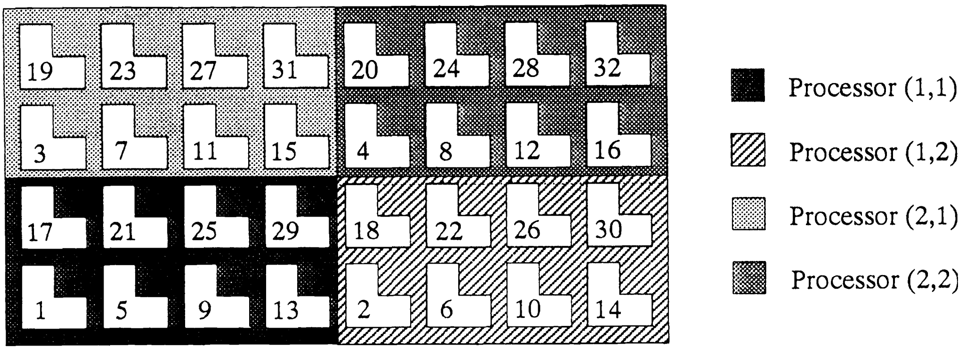


FIG. 4. *Distribution of the data to the processor array and multicolored numbering.*

cell grid onto a 2×2 processor array (the indicated numbering will be discussed later). For the time discretization, i.e., for the solution of the system (3), we consider the IE scheme and the CN scheme. In order to handle both schemes simultaneously, we formulate them in the setting of the θ-method [4]. The θ-method applied to (3) yields, starting from the initial values

$$x_h^0 = x_{0h}, \qquad p_h^0 = -(\mathbf{L}_h\mathbf{G}_h)^{-1}\mathbf{K}_h(x_h^0)x_h^0,$$

approximations x_h^n and p_h^n to x_h and p_h at the time levels $t_n = n\Delta t$, $n = 1, 2, \cdots$ from the solution of a nonlinear algebraic system of the form

$$\text{(4a)} \qquad x_h^n + \theta\Delta t[\mathbf{K}_h(x_h^n)x_h^n + \mathbf{G}_h p_h^n] = x_h^{n-1} + (\theta-1)\Delta t[\mathbf{K}_h(x_h^{n-1})x_h^{n-1} + \mathbf{G}_h p_h^{n-1}],$$

$$\text{(4b)} \qquad \mathbf{L}_h x_h^n = 0.$$

$\Delta t > 0$ is the timestep and, in general, the parameter θ can be in the interval [0, 1]. In the following, we only consider the cases $\theta = 1$, yielding the IE scheme, and $\theta = 0.5$, yielding the CN scheme. It can be shown by Taylor expansion that the CN scheme approximates (3) with a consistency error of $O(\Delta t^2)$, while the IE scheme is only $O(\Delta t)$ accurate. Both methods are unconditionally stable, but it is well known that for spatially nonsmooth solutions, the CN scheme may cause numerical oscillations [4]. The IE scheme does not show such behaviour. In § 7, we compare the two methods numerically for different kinds of problems.

4. Linearization and decoupling of variables. Each timestep with the θ-method requires the solution of the nonlinear algebraic system (4). Treating (4) directly, for instance with a Newton method, would be very inefficient, since the resulting linear systems would be very large and very poorly conditioned. We apply here a technique that reduces the solution of (4) to the solution of a sequence of smaller and better-conditioned linear systems. In a first step, we decouple the computation of x_h^n and p_h^n by means of a pressure correction approach that is based on a fractional step method due to Chorin [3]. We first compute an approximation $\tilde{x}_h^n$ to x_h^n from the system

$$\text{(5)} \qquad \tilde{x}_h^n + \theta\Delta t\mathbf{K}_h(\tilde{x}_h^n)\tilde{x}_h^n = x_h^{n-1} + (\theta-1)\Delta t\mathbf{K}_h(x_h^{n-1})x_h^{n-1} - \Delta t\mathbf{G}_h p_h^{n-1},$$

which results from (4a) by replacing the pressure term $\theta\Delta t\mathbf{G}_h p_h^n$ on the left side by $\theta\Delta t\mathbf{G}_h p_h^{n-1}$. Subtracting (5) from (4a) and neglecting the terms involving the operator $\mathbf{K}_h$ we obtain

$$\text{(6)} \qquad x_h^n = \tilde{x}_h^n - \theta\Delta t\mathbf{G}_h(p_h^n - p_h^{n-1}).$$

Applying the operator $\mathbf{L}_h$ to both sides of (6) together with (4b) yields the linear system

$$\theta\mathbf{L}_h\mathbf{G}_h p_h^n = \theta\mathbf{L}_h\mathbf{G}_h p_h^{n-1} + \frac{1}{\Delta t}\mathbf{L}_h\tilde{x}_h^n, \tag{7}$$

from which the new pressure p_h^n can be computed. Finally, x_h^n can be computed from (6).

Applying the same technique as in [11], where the above approach is used in combination with the Crank-Nicolson scheme for two-dimensional isothermal flows, it can be shown that the pressure correction (PC) method does not spoil the accuracy of the θ-method.

Summing up, solving (4) with the PC scheme requires the following steps:

PC1: Compute $\tilde{x}_h^n$ from (5).

PC2: Compute p_h^n from (7).

PC3: Compute x_h^n from (6).

The correction step PC3 consists of a direct computation and needs no further comment. Step PC2 requires the solution of a sparse system of linear equations. The coefficient matrix of the system is symmetric positive definite and it remains unchanged during the timestepping process. It can easily be verified [15] that the problem is equivalent to a discretization of a Poisson equation with Neumann boundary conditions.

Step PC1 requires the solution of a nonlinear algebraic system. We treat this problem by a successive iteration (SI) process involving a Gauss-Seidel technique, which results in a decoupling of the computation of the components $\tilde{u}_h^n$, $\tilde{w}_h^n$, $\tilde{v}_h^n$, and $\tilde{T}_h^n$ of $\tilde{x}_h^n$. The SI method is defined as follows:

SI1: $\tilde{x}_h^{n,0} = x_h^{n-1}$.

SI2: For $k = 1, \cdots, m$ compute $\tilde{x}_h^{n,k}$ from

$$\tilde{x}_h^{n,k} + \theta\Delta t\mathbf{S}_h(\tilde{x}_h^{n,k}, \tilde{x}_h^{n,k-1})\tilde{x}_h^{n,k} = x_h^{n-1} + (\theta - 1)\Delta t\mathbf{K}_h(x_h^{n-1})x_h^{n-1} - \Delta t\mathbf{G}_h p_h^{n-1}. \tag{8}$$

SI3: $\tilde{x}_h^n = \tilde{x}_h^{n,m}$.

In (8), the operator $\mathbf{S}_h(\tilde{x}_h^{n,k}, \tilde{x}_h^{n,k-1})$, an approximation to $\mathbf{K}_h(\tilde{x}_h^n)$, is defined by

$$\begin{pmatrix} \mathbf{A}_{uh}(\tilde{u}_h^{n,k-1}, \tilde{w}_h^{n,k}) & 0 & -\mathbf{B}_h(\tilde{v}_h^{n,k}) & 0 \\ 0 & \mathbf{A}_{wh}(\tilde{u}_h^{n,k-1}, \tilde{w}_h^{n,k-1}) & 0 & \mathbf{F}_h \\ \mathbf{B}_h(\tilde{v}_h^{n,k-1}) & 0 & \mathbf{A}_{vh}(\tilde{u}_h^{n,k-1}, \tilde{w}_h^{n,k-1}) & 0 \\ 0 & 0 & 0 & \mathbf{A}_{Th}(\tilde{u}_h^{n,k-1}, \tilde{w}_h^{n,k-1}) \end{pmatrix}.$$

Looking at this matrix we realize that the computation of $\tilde{x}_h^{n,k}$ in step SI2 can be performed by a back substitution process, which involves the successive solution of four linear systems for $\tilde{T}_h^{n,k}$, $\tilde{v}_h^{n,k}$, $\tilde{w}_h^{n,k}$, and $\tilde{u}_h^{n,k}$. We remark that the linearization of (5) by a Newton process, which would converge more rapidly, would not allow for such a decoupling. All components of $\tilde{x}_h^{n,k}$ would have to be computed simultaneously, resulting in a higher computational effort per iteration and higher memory requirements.

The number of iteration steps m in step SI2 either can be fixed (e.g., $m = 1$ may be sufficient), or it can be held variable by iterating until a stopping criterion such as

$$\frac{\|\tilde{x}^{n,k-1} - \tilde{x}^{n,k}\|}{\|\tilde{x}^{n,k}\|} < \varepsilon, \tag{9}$$

with a suitable norm $\|\cdot\|$ and some $\varepsilon > 0$ is reached. We investigate this question numerically in § 7.

In summation, we have reduced the solution of the nonlinear algebraic system (4) to the solution of $4m + 1$ sparse linear systems. So we are left with the problem of solving these systems in an efficient way on our processor array. This is the topic of § 5.

5. Parallel linear system solvers. For the solution of the sparse linear systems that arise in the algorithms of § 4, we consider MG methods and PCG methods, which rank among the most efficient methods for dealing with such problems. Let us denote by

$$\mathbf{M}y = b \tag{10}$$

the system that we wish to solve.

PCG methods [2] start with a transformation of (10) into an equivalent system

$$\mathbf{P}^{-1}\mathbf{M}y = \mathbf{P}^{-1}b, \tag{11}$$

with a nonsingular preconditioning matrix **P**. Applying a conjugate gradient algorithm to (11), we obtain a PCG method. For symmetric positive definite systems the classical conjugate gradient (CG) method of Hestenes and Stiefel [8] can be used. A generalization of this method to nonsymmetric systems is the conjugate gradient squared (CGS) method of Sonneveld [13].

The rate of convergence of PCG methods is determined by the ratio of the largest and smallest eigenvalues, the condition number, of the matrix $\mathbf{P}^{-1}\mathbf{M}$ [2]. The smaller the condition number the more rapid is the convergence. Therefore in order to obtain a rapidly convergent algorithm, **P** should be an approximation to **M** in some sense. On the other hand, since required one or more times in each step of a PCG method, the inverse of **P** should be as simple as possible. A compromise between the two extreme cases $\mathbf{P} = \mathbf{M}$ and $\mathbf{P} = \mathbf{Id}$ (identity matrix) must be found.

We concentrate here on basic iterative method preconditionings, as discussed, for instance, in [1]. In this approach, **P** is defined by

$$\mathbf{P}^{-1} = \sum_{i=0}^{s-1} (\mathbf{Id} - \mathbf{Q}^{-1}\mathbf{M})^i \mathbf{Q}^{-1},$$

with a positive integer s and a nonsingular matrix **Q** corresponding to a basic iterative method defined by a splitting $\mathbf{M} = \mathbf{Q} - \mathbf{R}$ of **M** and an iteration process of the form

$$y \leftarrow y - \mathbf{Q}^{-1}(\mathbf{M}y - b). \tag{12}$$

The resulting algorithm is known as s-step basic iterative PCG method. We shall return to the problem of choosing a suitable **Q** for the use on our processor array below.

The main components of MG methods, which are thoroughly treated in [6], are a sequence of coarsened grids, an interpolation procedure and a restriction procedure for the intergrid transfers, and an iterative method like (12) for smoothing on the different grids. As an example, we describe here the multigrid approach that is used for the solution of the pressure systems (7).

We coarsen the grid as indicated in Fig. 5 and choose the coarsest grid such that each subregion contains at least one grid point. Together with the induced mapping of the coarse-grid points (corresponding to that for the finest grid), this approach results in a uniform data distribution over the processor array for all grid levels and, therefore, ensures a full utilization of the processor array during the algorithm. It should be noted that for very large processor arrays, where, with the above strategy, the problem on the coarsest grid remains very large, it may be advantageous to put up with the full utilization and coarsen the grid below the numbers of processors.

The interpolation operator is illustrated on the right in Fig. 6. It results from piecewise linear interpolation and is of second order with respect to h. For restriction, we take the weighted restriction operator corresponding to the adjoint of the interpolation operator, shown on the left in Fig. 6. We remark that the seemingly natural and much easier to implement restriction operator, resulting from assigning each coarse-grid

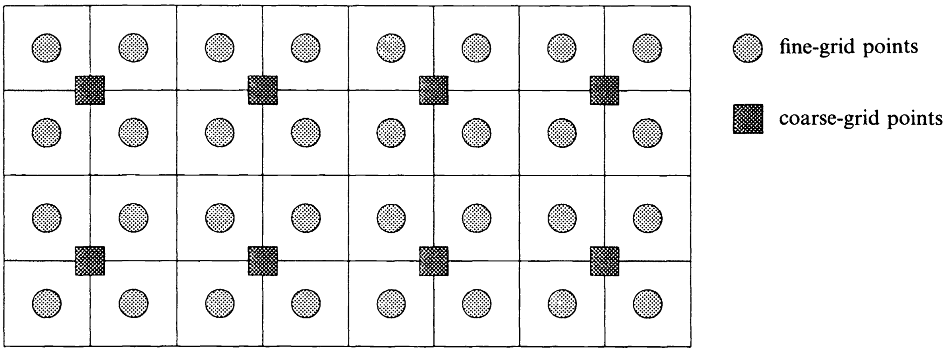


FIG. 5. *Grid coarsening.*

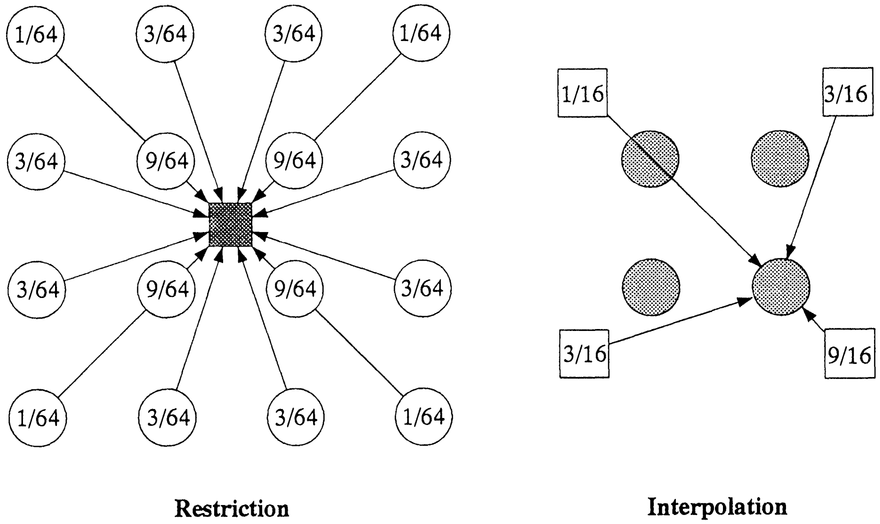


FIG. 6. *Second-order interpolation and corresponding weighted restriction.*

point the mean value of its four fine-grid neighbours, in connection with the corresponding piecewise constant first-order interpolation, would not be accurate enough to solve our second-order problems [17].

For the movement through the different grid levels, we use the standard V-cycle approach with a fixed number of presmoothing and postsmoothing steps [6]. The coarse-grid matrices are chosen according to the discretization on the respective grids. The solution on the coarsest grid is approximated by means of some steps of a PCG method. We found that the reduction in the required number of V-cycles when solving more exactly on the coarsest grid is too small to compensate for the additional work.

In order to obtain an efficient algorithm for a parallel computer, the crucial point for both PCG methods and MG methods is the choice of the iterative method (12). The parallelization of the other components of the methods, with the considered data mapping, is straightforward. Unfortunately, basic iterative methods that are very attractive with respect to their preconditioning or smoothing properties (for example, the powerful incomplete factorization methods [12], [16]) are inherently recursive and, therefore, not amenable to an efficient parallelization. It seems that a compromise between parallelism and convergence properties, which takes into account the type of problem and the available hardware, must be found.

The simple choice

$$\mathbf{Q}=\omega\mathbf{M}_D, \qquad 0<\omega\leqq 1,$$

where $\mathbf{M}_D$ denotes the diagonal matrix corresponding to the main diagonal of $\mathbf{M}$, results in the damped Jacobi (DJAC) method [6]. When used as a preconditioner, $\omega=1$ should be taken, and when used as a smoothing iteration, $\omega=0.5$ is a suitable choice. DJAC can be viewed as an ideally parallelizable algorithm. The degree of parallelism is equal to the number of grid points independent of the numbering of the unknowns.

As an improvement of DJAC, we consider here a parallel version of the Gauss-Seidel method, which can be described in the setting of multicolored basic iterative methods as discussed in Adams [1]. We choose a coloring that is induced by the number of processors and the problem size, as opposed to, for instance, the well-known red/black coloring [19], which is induced by the finite-difference discretization. The coloring strategy is as follows: all cells in the same subregion obtain a different color and all subregions are colored identically. Therefore, $k_r \times k_z$ colors are required, and the number of unknowns corresponding to the same color is equal to the number of processors. Numbering the unknowns color by color in the usual lexicographical way gives $\mathbf{M}$ a block structure such that each of the $k_r \times k_z$ diagonal blocks is a diagonal matrix, whose size equals the number of processors. As an example, Fig. 4 illustrates the multicolor numbering for an 8×4 grid and a 2×2 processor array.

With the usual splitting $\mathbf{M}=\mathbf{M}_L+\mathbf{M}_D+\mathbf{M}_U$ of $\mathbf{M}$ into the lower diagonal part $\mathbf{M}_L$, the diagonal part $\mathbf{M}_D$, and the upper diagonal part $\mathbf{M}_U$, the MCGS method is defined by

$$\mathbf{Q}=\mathbf{M}_D+\mathbf{M}_L.$$

The method is easy to implement and it is optimal concerning the utilization of the processor array. Since all points of the same color can be updated simultaneously, the degree of parallelism is equal to the number of processors. Concerning the convergence properties, the method lies somewhere between the Jacobi method and the lexicographical Gauss-Seidel method, depending on the number of subdomains (see [19, Ch. 4, Thm. 5.8]). If the processor array has the same size as the cell grid ($k_r=k_z=$ 1 color), the method is equivalent to the Jacobi method, and if we only have one processor ($k_r \times k_z = N_r \times N_z$ colors), it is equivalent to the lexicographical Gauss-Seidel method.

For use as a preconditioner, we also consider the symmetric variant (SMCGS) of the method, which is defined by

$$\mathbf{Q}=(\mathbf{M}_D+\mathbf{M}_U)\mathbf{M}_D^{-1}(\mathbf{M}_D+\mathbf{M}_L).$$

Using SMCGS, some symmetry properties of $\mathbf{M}$ carry over to $\mathbf{P}$, e.g., a symmetric positive definite $\mathbf{M}$ yields a symmetric positive definite $\mathbf{P}$, as is required when preconditioning the CG algorithm.

Concerning the data communication among the processors, we remark that only the summations in the innerproducts of the CG methods and the computations of some norm for checking the convergence require global communication. For all other components of the algorithms discussed above, only local communication is required. In § 8 we give a numerical comparison of the considered methods.

6. Test problems and implementation considerations. Our test problems are derived from a model of Czochralski crystal growth proposed in [18] as a benchmark problem. The (nondimensional) configuration consists of a rotating vertical cylindrical crucible of radius $r=1$ filled with a melt to a height $z=1$. The melt is bounded above by a rotating coaxial crystal of radius $r=0.4$. Concerning the boundary conditions the following assumptions are made:

- The surface of the melt between the crystal and the crucible is flat and free of shear stress with a linear temperature distribution from the crystal to the crucible wall;
- The crystal is isothermal with temperature $T=0$ and rotates with an angular velocity $v=g(t)$;
- The crucible rotates with an angular velocity $v=-g(t)/4$, the crucible wall is heated with a temperature $T=f(t)$ and the crucible base is a perfect insulator.

This leads us to the following set of boundary conditions:

$$\begin{aligned}
&u=w=0, \quad v=-g(t)/4, \quad T=f(t) \quad \text{for } r=1, \quad 0\leqq z\leqq 1,\\
&u=w=0, \quad v=-rg(t)/4, \quad T_z=0 \quad \text{for } 0\leqq r\leqq 1, \quad z=0,\\
&u_z=v_z=w=0, \quad T=(5r-2)f(t)/3 \quad \text{for } 0.4\leqq r\leqq 1, \quad z=1,\\
&u=w=T=0, \quad v=rg(t) \quad \text{for } 0\leqq r\leqq 0.4, \quad z=1.
\end{aligned}$$

The initial conditions are taken to be $u_0=w_0=v_0=T_0=0$.

Varying the Rayleigh number Ra and the functions f and g, this model allows us to investigate the numerical behaviour of the considered algorithms for the most relevant aspects of our flow problem: time-dependent thermal convection, time-dependent forced convection, and Hopf bifurcated oscillatory convection. We fix the Prandtl number to $Pr=0.05$, which is in the range of the values for semiconductor and metal melts (e.g., silicon, mercury, etc.).

The array processor that we use for our computations is an AMT DAP 510, which consists of a mesh-connected array of $P_x\times P_y=32\times 32$ single-bit processors working in SIMD mode (see, e.g., [10] for a detailed description of the hardware). The total memory size of the machine is 4 MBytes and the clock rate is 10 MHz. For the computations, we use 48-Bit arithmetic. In order to allow a classification of the performance, we remark that we can solve our test problems on the DAP 510 about 35 times faster than on a SUN 3/50 and about 1.2 times faster than on a CYBER 955.

For the space discretization, we use a uniform mesh with a grid spacing

$$h=\frac{1}{32k}, \qquad k=1,2,\cdots,$$

which results in a number of $N_r\times N_z=kP_x\times kP_y$ cells. Following the mapping strategy discussed in § 3 (see Fig. 4), we divide the cell grid in 32×32 subregions, and each processor obtains the data for the $k\times k$ cells contained in each subregion. We remark that the memory size of the DAP 510 only allows for calculations up to $k=4$. The mesh size of the coarsest grid in the multigrid methods is $h=1/32$. For all computations the stopping criterion for the linear system solvers is chosen to be

$$\frac{\|y^{k-1}-y^k\|_\infty}{\|y^k\|_\infty}<10^{-5},$$

where y^{k-1} and y^k are two consecutive iterates and $\|\cdot\|_\infty$ denotes the maximum norm. For the presentation of the results we introduce the stream function ψ, defined by

$$u=\psi_z/r, \qquad w=-\psi_r/r,$$

and $\psi=0$ on the boundary.

7. Comparison of discretization schemes. In the first instance, we study the convergence behaviour of the considered algorithms with respect to time. For this we consider our test problem in the time interval [0, 1] with the data sets

$$f(t)=\sin(\pi t/2),\quad g(t)=0,\quad Ra=500,$$

which results in a pure thermal convection flow driven by the temperature gradient between the crystal and the crucible wall, and

$$f(t)=0,\quad g(t)=10\sin(\pi t/2),\quad Ra=500,$$

which yields a pure forced convection flow driven by the rotation of the crucible and the crystal.

We want to compare the IE scheme and the CN scheme. For the number of iteration steps in the SI algorithm we consider the cases $m=1$, denoted by IE1 and CN1, and m according to (9) with the maximum norm and $\varepsilon=10^{-5}$, denoted by IE* and CN*. We fix the grid spacing to $h=1/64$.

In order to obtain reference values we first run CN* with the very small timestep $\Delta t=1/1000$. Next we compute the solution with IE1, IE*, CN1, and CN* for $\Delta t=1/10$, $1/20, 1/40, \cdots$. Figure 7 shows the results for the thermal convection problem. On the left-hand side plots of the relative errors of the maximum norms of the stream function ψ at $t=1$ as against the number of timesteps are shown, and on the right-hand side the relative errors are plotted against the required computing times. The corresponding results for the forced convection problem are shown in Fig. 8.

From the results we can conclude:

- As can be expected, due to the higher order, with the CN schemes the same accuracy can be reached faster, as with the corresponding IE schemes.

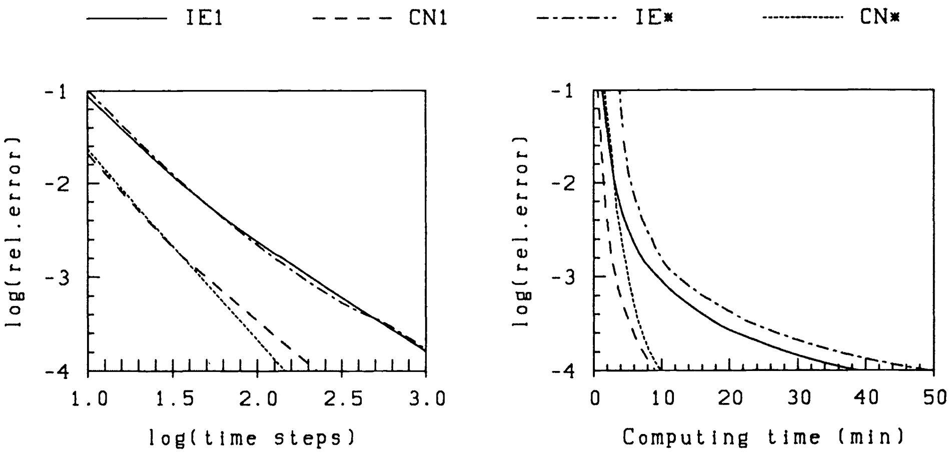


FIG. 7. *Convergence behaviour with respect to time for* IE1, IE*, CN1, *and* CN* *for the thermal convection problem.*

- For both problems the convergence behaviour of IE1 and IE* does not differ very much. Looking at the errors obtained with more than 100 timesteps, the linear convergence rate predicted by the theory can be observed for both schemes. Since one timestep with IE* takes longer than with IE1, it is not useful to choose $m>1$.

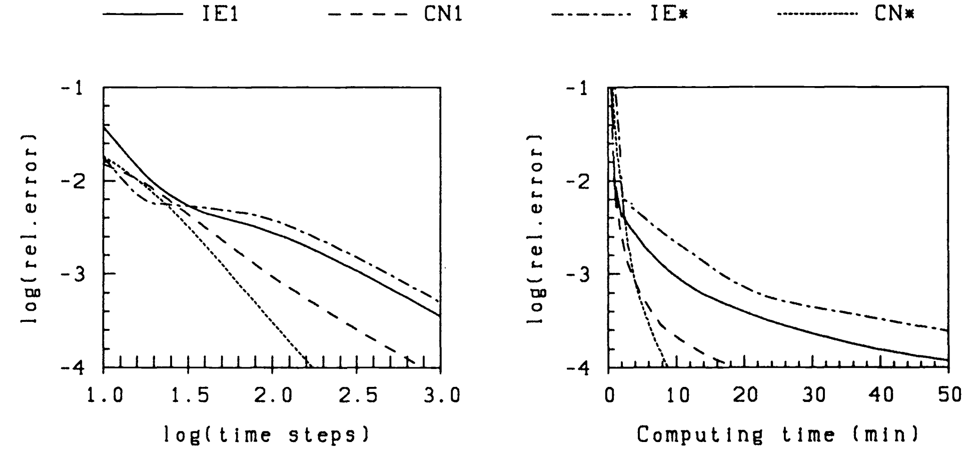


FIG. 8. *Convergence behaviour with respect to time for* IE1, IE*, CN1, *and* CN* *for the forced convection problem.*

- For both problems, CN* shows the quadratic convergence behaviour predicted by the theory for the Crank-Nicolson scheme. For the CN1 scheme the convergence order deteriorates with decreasing Δt. Due to the subdiagonal element $\mathbf{B}_h(\tilde{v}_h^n)$ in the matrix $\mathbf{K}_h(\tilde{x}_h^n)$ this effect is stronger for the forced convection problem. In this case, in order to improve the accuracy, it is preferable to choose $m > 1$ rather than a smaller Δt.

Now we investigate the behaviour of the IE and CN schemes when used for a high Rayleigh number problem having a Hopf bifurcated time-periodic solution. We consider our test problem in the time interval $[0, 0.4]$ with the data

$$f(t) = 1, \quad g(t) = 0, \quad Ra = 750{,}000.$$

We only present results for IE1 and CN1, because the schemes with variable m show the same behaviour. Figure 9 shows the variation of the maximal radial velocity with time obtained with the IE1 scheme using different values of Δt and h.

Let us first discuss the nature of the flow. Looking at the diagram for $\Delta t = 0.00005$ and $h = 1/128$, yielding the most accurate solution, we can see that after a startup phase with irregular oscillations, which is due to the incompatibility of the boundary conditions and the initial conditions (the initial temperature does not satisfy the boundary conditions on the crucible wall), the flow becomes periodic.

Now we turn to the numerical behaviour of the IE1 scheme. If Δt is chosen too large we do not find the periodic time dependency in the solution. Looking at the diagrams for $h = 1/128$, we see that with increasing Δt, the amplitude of the oscillations becomes smaller and smaller and finally disappears, such that we get a stationary solution corresponding to some mean value of the amplitude. For $h = 1/64$ this effect, which is due to the strong damping properties of the IE scheme, can also be observed.

If h is chosen too small we also get a stationary solution. Looking at the diagrams for $\Delta t = 0.00005$ we can see that by increasing h the amplitude of the oscillations decrease and disappear for $h = 1/32$. In this case, the stationary values for the maximal radial velocity are far away from the values for smaller h (observe the different scale in the diagrams for $h = 1/32$). This is because of spatial oscillations (also called wiggles) due to drastic changes of the solution in boundary layers at the rigid boundaries [4]. The oscillations can be suppressed by choosing a smaller h (the oscillations are no

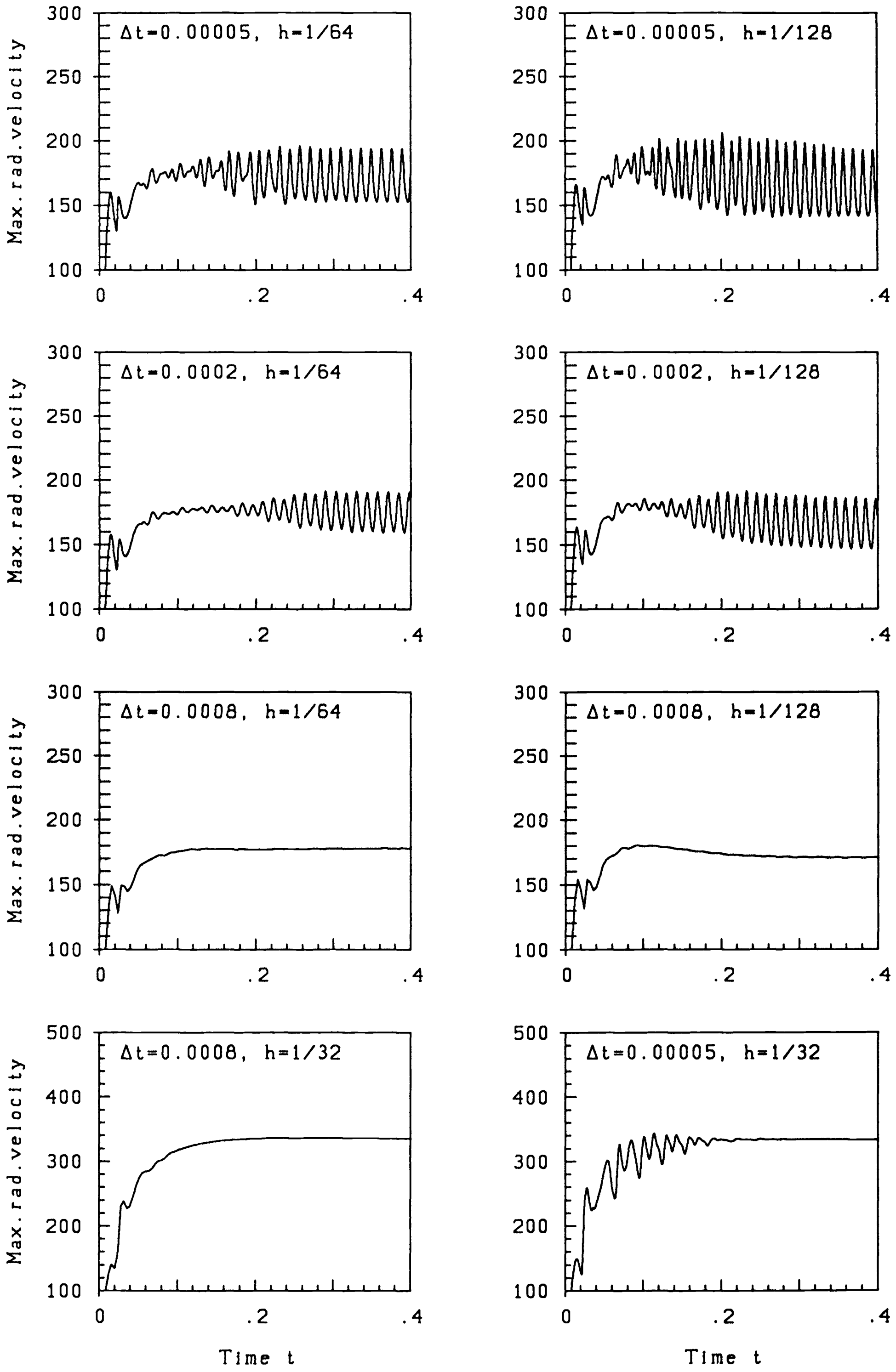


FIG. 9. *Maximal radial velocity versus time for the* IE1 *scheme with various* Δt *and* h.

longer apparent for $h=1/64$), a local refinement near the boundaries would be sufficient, or by choosing upwind differences for the first-order derivatives in the convective terms.

We were not able to compute a reasonable solution with the CN1 scheme. We tried stepsizes down to $\Delta t=0.000025$ and $h=1/128$ (due to the limited memory size, a smaller h is not possible on our DAP 510). As an example, Fig. 10 shows the maximal radial velocity as against time computed with $\Delta t=0.0001$ and $h=1/128$. For other values of Δt and h, we obtain similar results or, as in some cases, the program even terminates with an arithmetic overflow. While in the startup phase, the results are comparable to those for the IE1 scheme; they become chaotic when the flow enters the periodic phase. This effect seems to be due to the fact that the damping property of the CN scheme deteriorates for high-frequency components of permanently acting rough disturbances [4]. As suggested in [9], a damping by means of replacing some CN steps by IE steps or by shifting the CN scheme slightly to the implicit side may save the situation, but we have not yet investigated these techniques.

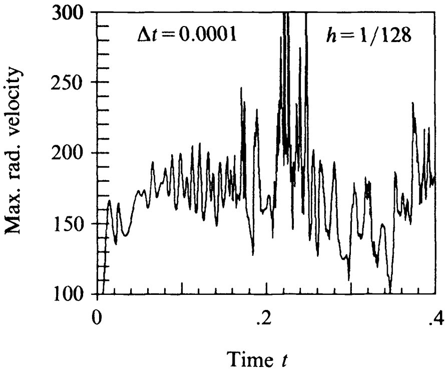


FIG. 10. *Maximal radial velocity versus time for the* CN1 *scheme with* $\Delta t=0.0001$ *and* $h=1/128$.

8. Comparison of linear system solvers. For a comparison of the sparse linear system solvers discussed in § 5, we consider the mixed convection problem in the time interval [0, 1] resulting with the data

$$f(t)=\sin(\pi t/2),\quad g(t)=10\sin(\pi t/2),\quad Ra=500.$$

We discretize with the CN1 algorithm and fix $\Delta t=1/80$.

TABLE 1
Numbers of iterations and timings for PCG *methods with* $h=1/32, 1/64, 1/128$.

	Iterations			Computing time (min)		
Method	$h=1/32$	$h=1/64$	$h=1/128$	$h=1/32$	$h=1/64$	$h=1/128$
CG	8476	16391	32916	0.9	6.4	49.1
CG-DJAC1	5456	10509	19026	0.6	4.4	31.0
CG-DJAC2	3275	5519	10373	0.6	3.6	26.2
CG-DJAC3	3561	6287	11443	0.8	5.5	39.0
CG-DJAC4	2408	4246	7435	0.7	4.7	32.0
CG-SMCGS1	3275	4842	8226	0.6	3.1	21.7
CG-SMCGS2	2408	3933	6191	0.7	4.0	25.7

TABLE 2
Numbers of iterations and timings for MG *methods with* $h = 1/64, 1/128$.

Method	Iterations		Computing time (min)	
	$h=1/64$	$h=1/128$	$h=1/64$	$h=1/128$
MG-DJAC1	447	561	1.4	3.4
MG-DJAC2	396	424	1.5	3.5
MG-MCGS1	387	352	1.3	2.4
MG-MCGS2	370	330	1.4	2.9

First we study the convergence behaviour of the PCG and MG methods with varying h. The methods are applied to the symmetric positive definite systems (7) arising in the timestepping process. Table 1 shows the results for the PCG methods. Indicated are the numbers of iterations and the computing times required to solve all 80 systems for $h = 1/32, 1/64, 1/128$ with the different preconditionings. The corresponding results for the MG method for $1/64, 1/128$ ($h = 1/32$ corresponds to the coarsest grid), and the different smoothing methods are given in Table 2. Concerning the notation, CG-BIs denotes the CG method preconditioned with s steps of the basic iterative (BI) method. MG-BIs denotes the MG method with s BI presmoothing steps. We note that for $h = 1/32$, the basic iterative methods DJACs and MCGSs, as well as the methods DJAC2s and SMCGSs, are equivalent. Several conclusions can be drawn from the results:

- All PCG methods are considerably faster than the unpreconditioned CG method. The saving of computational work due to the fewer iterations is much larger than the additional work due to the preconditionings.
- Only a small number of basic iterative method steps for preconditioning is efficient. The optimal value is $s = 2$ for the DJAC preconditioning and $s = 1$ for the SMCGS preconditioning. The reduction in the number of iterations when using larger values of s is too small to compensate for the additional work due to the larger number of basic iterative method steps.
- In agreement with theoretical results in [1], the DJAC preconditioning for odd $s > 1$ is not as efficient as for even s (CG-DJAC3 needs more iterations than CG-DJAC2).
- The SMCGS preconditioning is more efficient than the JAC preconditioning. For all h, CG-SMCGS1 is the fastest PCG method.
- The MG methods are more efficient than the PCG methods. The smaller h is, the more superior the MG method becomes.
- Using only one presmoothing step and one postsmoothing step is sufficient. The reduction in the number of iterations when using more steps is not large enough to compensate for the additional work.
- The MCGS smoothing is more efficient than the DJAC smoothing. The superiority grows with decreasing h. For all h, MG-MCGS1 is the fastest method.

Finally, we investigate the performance of the preconditioned CGS methods. We use the methods for the solution of the nonsymmetric systems arising during the timestepping process for our test problem. The numbers of iterations and the computing times for solving the 320 systems are given in Table 3 for the different preconditionings.

The results are very similar to those for the PCG method in the symmetric positive definite case. The reduction of the computing time by the preconditionings is larger in the nonsymmetric case. For instance, for $h = 1/128$ the ratio of the computing time

TABLE 3
Numbers of iterations and timings for PCGS *methods with* $h = 1/32, 1/64, 1/128$.

	Iterations			Computing time (min)		
Method	h = 1/32	h = 1/64	h = 1/128	h = 1/32	h = 1/64	h = 1/128
CGS	9869	18291	33295	1.7	10.8	75.9
CGS-DJAC1	2831	4848	8581	0.6	3.2	21.7
CGS-DJAC2	1606	2615	4494	0.4	2.9	19.3
CGS-DJAC3	1744	3355	6020	0.7	5.2	36.6
CGS-DJAC4	1274	2005	3396	0.7	4.0	26.6
CGS-SMCGS1	1606	2383	3698	0.4	2.6	16.7
CGS-SMCGS2	1274	1811	2797	0.7	3.4	21.2

for CGS to those for CGS-SMCGS1 amounts to 4.5, while the ratio is only 2.3 for CG and CG-SMCGS1.

9. Conclusion. We have presented a highly parallel algorithm for the numerical solution of time-dependent nonisothermal flow problems. If the driving forces of the flow are not too large, the proposed method with the Crank-Nicolson discretization is well suited to solving the type of problems considered here on SIMD array processors. If the driving forces become larger, the arising stability problems can be resolved by the use of the implicit Euler discretization. For the solution of the sparse linear systems occurring during the algorithm it was shown that already for relatively small problem sizes, MG methods are superior to PCG methods. For the involved basic iterative methods the multicolor block Gauss-Seidel method is preferable to the simple Jacobi method.

REFERENCES

[1] L. ADAMS, *M-step preconditioned conjugate gradient methods*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 452-463.

[2] O. AXELSSON AND V. BARKER, *Finite Element Solution of Boundary Value Problems*, Academic Press, Orlando, FL, 1984.

[3] A. CHORIN, *Numerical solution of the Navier-Stokes equations*, Math. Comp., 22 (1968), pp. 745-762.

[4] C. CUVELIER, A. SEGAL, AND A. V. STEENHOVEN, *Finite Element Methods and Navier-Stokes Equations*, D. Reidel, Dordrecht, the Netherlands, 1986.

[5] J. DERBY AND R. BROWN, *On the dynamics of Czochralski crystal growth*, J. Crystal Growth, 83 (1987), pp. 137-151.

[6] W. HACKBUSCH, *Multi-Grid Methods and Applications*, Springer-Verlag, Berlin, 1985.

[7] F. HARLOW AND J. WELCH, *Numerical calculation of time dependent viscous incompressible flow of fluids with free surface*, Phys. Fluids, 8 (1960), pp. 2183-2189.

[8] M. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 409-436.

[9] J. G. HEYWOOD AND R. RANNACHER, *Finite-element approximation of the nonstationary Navier-Stokes problem Part* IV: *Error analysis for second-order time discretization*, SIAM J. Numer. Anal., 27 (1990), pp. 353-384.

[10] R. W. HOCKNEY AND C. JESSHOPE, *Parallel Computers*: *Architecture, Programming and Algorithms*, Adam Hilger, Bristol, U.K., 1981.

[11] J. V. KAN, *A second-order accurate pressure-correction scheme for viscous incompressible flow*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 870-891.

[12] J. MEIJERINK AND H. VAN DER VORST, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148-162.

[13] P. SONNEVELD, CGS, *a fast Lanczos-type solver for nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 36-52.

[14] E. SPIEGEL AND G. VERONIS, *On the Boussinesq approximation for a compressible fluid*, Astrophys. J., 131 (1960), pp. 442-447.

[15] J. TEN THIJE BOONKKAMP, *The odd-even hopscotch pressure correction scheme for the incompressible Navier-Stokes equations*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 252-270.

[16] P. WESSELING, *Theoretical and practical aspects of a multigrid method*, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 387-407.

[17] ———, *Linear multigrid methods*, in Multigrid Methods, Chap. 2, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

[18] A. A. WHEELER, *Four Test Problems for the Numerical Simulation of Flow in Czochralski Crystal Growth*, J. Crystal Growth, 102 (1990), pp. 691-695.

[19] D. YOUNG, *Iterative Solution of Large Linear Systems*, Academic Press, New York, 1971.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 6, pp. 1394–1417, November 1992

008

STOCHASTIC BREAKTHROUGH TIME ANALYSIS OF AN ENHANCED OIL RECOVERY PROCESS*

HANS PETTER LANGTANGEN†

Abstract. Input data to mathematical models for petroleum reservoir flow are commonly subjected to a significant degree of uncertainty. In this paper, input data, such as porosity, absolute and relative permeabilities, and adsorption functions, are treated as stochastic quantities in a one-dimensional polymer flooding model. Tools from structural reliability theory are used to solve the governing stochastic partial differential equations. The statistics of the time T to water breakthrough in the production well are studied. In particular, the sensitivity of T to the input quantities are reported. Correlations between the input variables seem to have a considerable effect on their relative importance.

Key words. stochastic differential equations, polymer flooding, reliability methods, sensitivity analysis

AMS(MOS) subject classifications. 65C20, 65U05

1. Introduction. The input data to many mathematical models are subjected to significant uncertainty and may thereby cause unreliable simulation results. This is especially the case for the standard continuum models of multiphase/multicomponent porous media flow which are of particular importance in the petroleum recovery industry. Important decisions are made on the basis of results from mathematical models, and it is often desired to have measures of uncertainties available. The simplest approach to gaining insight into how input data uncertainties affect the solution is to carry out a deterministic sensitivity analysis [13]. Each input parameter is given a prescribed perturbation, expressing an "uncertainty," and the resulting perturbation of the output data is found by solving the equations entering the mathematical model. This approach has some deficiencies. The perturbation in input, reflecting some kind of uncertainty, is difficult to estimate. Often the perturbation corresponds to a statistical quantity such as one standard deviation. When using a deterministic model, the resulting perturbation in output can seldom be given any statistical significance. Another deficiency of deterministic sensitivity analysis is related to the fact that correlations between the input data cannot be taken into account. Changing one input parameter without changing others, according to some underlying physical structure among the input variables, may be unphysical. In practice, the relationship among input variables is seldom completely known, and covariances, which can be measured from physical experiments alone, may be an attractive way to represent such relationships. In order to incorporate information on correlations and to consistently compute statistics of the output data of a model, given input data statistics, one should use a *stochastic* mathematical model of the physical process.

In this paper, we are concerned with a stochastic flow model for a rather complicated oil recovery process where accurate measurements of the input data are almost impossible to obtain. Input data are modeled in terms of stochastic variables, or fields, and the governing stochastic partial differential equations are solved to produce statistics of an interesting output parameter in the model. Although there are very important applications for stochastic reservoir flow models, the major part of industrial reservoir simulations is based on deterministic analysis. In recent years,

* Received by the editors March 27, 1991; accepted for publication (in revised form) November 15, 1991.

† Department of Mathematics, University of Oslo, P. O. Box 1053, N-0316 Oslo 3, Norway.

statistical methods for describing reservoir geometry and reservoir parameters have emerged (see, e.g., [9]). Activity in the solution of flow models with stochastic input is also evolving [6], [17], [14], [18].

Also, in a stochastic simulator there are input data to the code that are subjected to uncertainties. These input data are usually associated with parameters entering statistical distributions, such as mean and standard deviation values. This paper is especially concerned with the sensitivity of the output parameters to perturbations in input parameters that enter the distributions.

The particular physical problem to be treated here is one-dimensional displacement of oil by an injected mixture of water and polymer. This is a widely used enhanced oil recovery (EOR) process. We study the output data in terms of the time to water breakthrough in the production well. The breakthrough time is an important parameter to petroleum engineers and decision makers. Moreover, it is also closely related to the total oil production until breakthrough. The mathematical model consists of a system of three, coupled, nonlinear, time-dependent partial differential equations. Uncertain input quantities, such as porosity, absolute and relative permeablity, and adsorption effects, are modeled as stochastic variables. The resulting stochastic partial differential equations must then be solved to find the statistics of the desired output quantities. Variants of Monte-Carlo simulation techniques are presently dominating for solving stochastic partial differential equations and applications to reservoir simulation can be found in [6] and [14]. In this paper, we employ an alternative approach by adapting methods from probabilistic structural analysis. The particular method to be used is the *first-order reliability method* (FORM). FORM analysis of finite-element discretized continua is emerging [12], and a feasibility study of possible applications to reservoir simulation has been performed [17]. The main reason for using FORM to solve stochastic differential equations is that it is efficient for the present type of problem, and that it gives sensitivity measures on how parameters in the distributions, like expectations and standard deviations, influence the statistical results. A general discussion of Monte-Carlo simulation and analytically based reliability methods, such as FORM, is given by Bjerager [5]. Contrary to most work on stochastic reservoir flow models, we do not focus on absolute permeability heterogeneities, but instead on the interplay between stochastic representations of many of the physical input parameters.

In §2, we present the governing partial differential equations. Section 3 discusses the FORM application to the present problem. Applications to porous media flow appear in §4. A short summary with conclusions is given in §5.

2. The governing equations. Polymer flooding is an EOR technique where a mixture of water and polymer is injected into the reservoir to displace oil. The appearance of the polymer causes the viscosity ratio between the injected and displaced fluid phases to decrease and hence enhance the recovery process [7]. The mathematical model for polymer flooding is based on the continuum mechanical equations for multiphase/multicomponent porous media flow. In the present case, we consider the flow of an oil phase and an aqueous phase in a porous medium. The phases are considered to be immiscible and incompressible. The aqueous phase consists of water and a polymer component which is totally miscible in water. Let $S(x,t)$ be the saturation of the aqueous phase, where x is the spatial coordinate and t denotes time. Only unidirectional flow is considered. Moreover, let $C(x,t)$ be the concentration of the polymer component in the aqueous phase. We may derive a system of partial differential equations governing S and C by applying Darcy's law and the principle of mass conservation. Neglecting capillary pressure effects results in the following equations

$$\phi\frac{\partial S}{\partial t}+v\frac{\partial F}{\partial x}=0, \tag{1}$$

$$\frac{\partial}{\partial t}\left[\phi SC+(1-\phi)A\right]+v\frac{\partial}{\partial x}\left[CF\right]=0, \tag{2}$$

where $0 < x < L$ and $t > 0$. We expect water to be injected in $x = 0$ (injection well) and oil to be produced in $x = L$ (production well). Moreover, we have v as the total Darcy filtration velocity (independent of x) and ϕ as the porosity. Furthermore, F is the fractional flow function [16] which depends on S and C. Adsorption of the polymer on the rock is modeled by the function A that depends on C. The two equations above determine S and C when v is known. This is the case if the velocity is controlled in the injection well. To model the influence of the absolute permeability adequately, we prescribe a pressure difference between the injection and production well. Then v will depend on the time-dependent pressure field P, which is governed by

$$\frac{\partial}{\partial x}\left(H\frac{\partial P}{\partial x}\right)=0. \tag{3}$$

We have $v = -H\partial P/\partial x$, where H is a function of S and C.

The fractional flow function has the form

$$F=\frac{k_{rw}}{k_{rw}+\mu k_{ro}}, \tag{4}$$

where k_{rw} and k_{ro} are the relative permeability of the aqueous and oil phases, respectively. Here $\mu = \mu_a/\mu_o$, where μ_a and μ_o are the viscosities of the aqueous and oil phases, respectively. Gravity effects are excluded in this paper. The viscosity of the aqueous phase depends on the polymer concentrations. In this work, we have used a linear relation between μ_a/μ_o and the concentration C (cf. Johansen, Tveito, and Winther [10], and the references therein):

$$\mu=\nu+\zeta\cdot 100C,$$

with $\nu = \mu_w/\mu_o$, where μ_w is the water viscosity.

The relative permeability curves have been written on the form

$$k_{rw}(S) = K_{wm}\left[\frac{S-S_{wr}}{1-S_{or}-S_{wr}}\right]^a, \tag{5}$$

$$k_{ro}(S) = K_{om}\left[\frac{1-S-S_{or}}{1-S_{wr}-S_{or}}\right]^b. \tag{6}$$

In these formulas K_{wm} and K_{om} are maximum values of k_{rw} and k_{ro}, respectively, S_{wr} is the irreducible saturation of the aqueous phase, and S_{or} is the irreducible oil saturation. The function H, which enters the pressure equation (3), has the form

$$H=\hat{K}\left(\frac{k_{rw}}{\mu_a}+\frac{k_{ro}}{\mu_o}\right).$$

Here $\hat{K}$ is the absolute permeability. However, it is convenient to define a characteristic absolute permeability κ, and to introduce $K = \hat{K}/\kappa$. Hereafter this K is referred to as the absolute permeability. The adsorption function is assumed to have the Langmuir form [2]

$$A = \frac{\xi \cdot 100C}{1 + \eta \cdot 100C},$$

where ξ and η are empirically determined constants.

As initial and boundary conditions for the equations (1), (2), and (3) we have used

$$\begin{aligned} S(x,0) &= S_{wr}, \\ S(0,t) &= 1 - S^0_{or}, \\ C(x,0) &= 0, \\ C(0,t) &= \begin{cases} \Lambda, & t < t_s, \\ 0, & t \geq t_s, \end{cases} \\ P(0,t) &= \Delta P, \\ P(L,t) &= 0, \end{aligned}$$

where S^0_{or} denotes S_{or} at $x = 0$. Only pressure differences, and not the pressure level, influence incompressible flow. The injection concentration Λ is usually small, typically $\Lambda = 0.01$.

If $S_{wr} = 0$, we can characterize the solution as follows. Without adsorption there is one shock in C, say, at $x = x_s$. C equals the constant Λ behind the shock, provided $t_s > t$. For $t_s < t$, the C profile becomes plug formed. With adsorption there are two shocks in S, one at $x = x_s$ and one for $x > x_s$. More details and examples can be found in [10]. Observe that the well-known Buckley–Leverett problem [16] is a special case of the polymer model where either C is constant or $\zeta = 0$.

In the stochastic analysis we need to solve the above inital-boundary value problem for a deterministic set of physical input parameters. This can generally only be accomplished by numerical methods. The elliptic pressure equation (3) is solved by a central finite-difference method. A first-order Godunov scheme is used for the two hyperbolic conservation laws (1) and (2). All equations are solved on a uniformly partitioned grid with N grid points. Let Δt be the timestep length and let Δx be the length of the intervals in the spatial grid. The stability condition for the Godunov scheme reads [10]

$$\Delta t \leq \Delta x \min_{0 \leq x \leq L} \left(\frac{\phi}{v \left| \frac{\partial F}{\partial S} \right|}, \frac{\phi}{v\xi} \right).$$

In each simulation we have chosen Δt as large as possible according to this stability criterion. The choice of Δx is discussed in §4.5.

The present numerical scheme leads to rather diffusive solutions for C, and the C shock usually extends over eight to ten intervals. If there is only a single shock in S, our scheme resolves this shock quite satisfactorily, that is, over about two grid intervals. When there are two shocks in S, the shock that has the same position as the shock in C is dispersed over eight to ten intervals. The shock in the front of the saturation profile is ordinarily resolved over a couple of grid intervals.

The output parameter to be studied in this paper is the time T to water breakthrough in the production well. For the present system of hyperbolic equations and boundary conditions, T can be defined mathematically as

$$T = \inf\{t : S(L,t) > S(L,0)\}.$$

This criterion is, of course, sensitive to the truncation error of the numerical method. However, the front shock in S is usually resolved over about two grid intervals and with a sufficiently small Δx, the errors can be made insignificant for the cases treated in this paper (see §4.5). If the numerical solution is guaranteed to be monotone, a more robust criterion is to let T equal the arrival time of the "rightmost" local minimum of $\partial S/\partial x$ at $x = L$. Numerical experiments showed that the differences between the two criteria are less than 1 percent for the numerical method and the parameter values of interest in this paper.

3. Stochastic analysis. Recently developed methods from structural reliability [15] will be employed for solving the the equations (1), (2), and (3) for S, C, and P, $0 < x < L$ and $0 < t \leq T$ when some of the input variables like ϕ, K, k_{rw}, and k_{ro} are modeled as stochastic quantities. Structural reliability methods have the advantage over sampling techniques (of the Monte-Carlo type) in that sensitivity of probabilities to variations in parameters in both the flow model and the input distributions are cheaply computed. In the problems studied herein reliability methods are also considerably more efficient than Monte-Carlo simulation. Reliability methods generally give accurate results when the computed probabilities are small. As we show below, it is likely that reliability methods may be accurate over the whole range of probabilities in the present reservoir flow problems.

Viewing the input data as a collection of real numbers, we choose r of these numbers to be modeled as stochastic variables. The stochastic variables are denoted by $\mathbf{X} \in \mathbf{R}^r$, $\mathbf{X} = (X_1, \cdots, X_r)^T$, and called *basic variables*. Given statistical information about $\mathbf{X}$ we want to establish statistics of the output parameter T. For this purpose FORM is used.

The next section reviews some of the basic theory. We also introduce notation and definitions and describe how the present problem can be formulated for FORM analysis.

Let $\mathbf{X} = (X_1, \cdots, X_r)^T$ be a vector of basic variables. We are interested in computing the probability of the event $\Pr\{g(\mathbf{X}) \leq 0\}$, where $g(\mathbf{X})$ is a *limit state function*. Let $\Omega \in \mathbf{R}^r$ be the set of points where $g \leq 0$ (the *failure domain*). Clearly, the probability can be computed as

$$\Pr\{g(\mathbf{X}) \leq 0\} = \int_\Omega f_{\mathbf{X}}(\mathbf{x}) d\mathbf{x}, \tag{7}$$

where $f_{\mathbf{X}}(\mathbf{x})$ is the joint probability density of the vector $\mathbf{X}$. We define the limit state function as $g = T - T_\omega$, giving $\Pr\{g \leq 0\} = \Pr\{T \leq T_\omega\}$, where $T_\omega = \omega\tilde{T}$. $\tilde{T}$ is the deterministic value of T when E[$\mathbf{X}$] is used as input to the governing equations (1), (2), and (3). Note that T must be computed by solving the differential equations from §2. There are two problems with using (7). First, it may be difficult to establish the joint density, and second, the evaluation of the integral, for example by quadrature rules, becomes very expensive even for moderate values of r. The first problem may be solved to some extent by using methods such as those in [11], discussed below. The second problem can be treated by FORM analysis.

In general, it is extremely difficult to assign a joint density to input variables in reservoir simulation. At most, we can hope to establish marginal distributions for each basic variable and perhaps some correlation coefficients. Let $\Theta_{X_i}(x_i)$ be the marginal distribution function of X_i. The associated density $d\Theta_{X_i}/dx_i$ is denoted by $\theta_{X_i}(x_i)$. If $\mathrm{Cov}[\cdot,\cdot]$ is the covariance operator and $\mathrm{D}[\cdot]$ is the standard deviation operator, the correlation coefficient is

$$\varrho(X_i, X_j) = \frac{\mathrm{Cov}[X_i, X_j]}{\mathrm{D}[X_i]\mathrm{D}[X_j]}.$$

The correlation matrix $\mathbf{R}$ is then $\{\mathbf{R}\}_{ij} = \varrho(X_i, X_j)$. The method used herein restricts the statistical information about the basic variables to only include Θ_{X_i}, $i = 1, \cdots, r$, and $\mathbf{R}$. We refer to [11] for extensions to cases where some or all joint distributions are prescribed.

3.1. FORM approximations to probabilities. Let $\langle \mathbf{x}^1, \mathbf{x}^2 \rangle$ be the Euclidian inner product of $\mathbf{x}^1, \mathbf{x}^2 \in \mathbf{R}^r$, and let $||\mathbf{x}^1|| = \sqrt{\langle \mathbf{x}^1, \mathbf{x}^1 \rangle}$ be the corresponding norm. In the case where the *failure surface*, defined by $g(\mathbf{X}) = 0$, is a hyperplane, and $f_{\mathbf{X}}(\mathbf{x})$ is a standardized, multivariate normal density,

$$f_{\mathbf{X}}(\mathbf{x}) = \frac{1}{(2\pi)^{r/2}} \exp\left(-\frac{1}{2}||\mathbf{x}||^2\right),$$

the integral in (7) can be calculated analytically:

$$\int_{\Omega} f_{\mathbf{X}}(\mathbf{x}) d\mathbf{x} = \Phi(-\beta). \tag{8}$$

Here Φ is the cumulative, univariate, standardized, normal distribution function, and β is the distance between $\mathbf{X} = \mathbf{0}$ and the (linear) failure surface $g = 0$. Let $\mathbf{x}^*$ be the point on $g = 0$ which is closest to the origin ($\mathbf{x}^*$ is called the *design point*). Then $\beta = ||\mathbf{x}^*||$.

To formulate a first-order reliability method, we consider a mapping $\mathbf{X} \rightarrow \mathbf{Z} \rightarrow \mathbf{Y}$, such that $\mathbf{Y} = (Y_1, \cdots, Y_r)^T$ is a vector of statistically independent, normally distributed variables with mean zero and unit variance. If the failure surface is linear in the $\mathbf{Y}$-space, the result (8) gives

$$\Pr\{g \leq 0\} = \Phi(-\beta), \qquad \beta = ||\mathbf{y}^*||, \tag{9}$$

with $\mathbf{y}^*$ as the design point in the $\mathbf{Y}$-space. In the general case where the failure surface is nonlinear, (9) holds approximately, i.e., $\Pr\{g \leq 0\} \approx \Phi(-\beta)$, provided the curvature of the surface at $\mathbf{y} = \mathbf{y}^*$ is small. FORM refers to this approximate method of calculating $\Pr\{g \leq 0\}$.

3.2. Transformation of the basic variables. In $\mathbf{Z}$-space the variables are assumed to be jointly normally distributed with correlation matrix $\mathbf{R}_0$, zero mean, and unit standard deviation. The mapping $\mathbf{Z} \rightarrow \mathbf{Y}$ will therefore be a linear transformation $\mathbf{Y} = \mathbf{L}_0^{-1}\mathbf{Z}$. $\mathbf{L}_0$ is here the lower triangular Cholesky decomposition of $\mathbf{R}_0$: $\mathbf{R}_0 = \mathbf{L}_0\mathbf{L}_0^T$. The mapping $\mathbf{X} \rightarrow \mathbf{Z}$ is given by

$$\Phi(z_i) = \Theta_{X_i}(x_i), \qquad i = 1, \cdots, r. \tag{10}$$

In addition, the $\mathbf{R}$ matrix must be transformed into $\mathbf{R}_0$. This may be done approximately by empirically based formulas (see [11]) for a wide range of distributions

Θ_{X_i}. However, we are exclusively concerned with normally or lognormally distributed variables in **X**-space. In this case, the transformation (10) and the formulas for transforming **R** to $\mathbf{R}_0$ can be simplified and made exact. Let $\{\mathbf{R}_0\}_{ij} = \varrho(Z_i, Z_j)$. If X_i is normally distributed with mean $\mathrm{E}[X_i]$ and standard deviation $\mathrm{D}[X_i]$, we have

$$Z_i = \frac{X_i - \mathrm{E}[X_i]}{\mathrm{D}[X_i]}.$$

If both X_i and X_j are normal variables, $\varrho(Z_i, Z_j) = \varrho(X_i, X_j)$. For a lognormally distributed variable X_i, the transformation (10) simplifies to

$$Z_i = \frac{\ln X_i - \mathrm{E}[\ln X_i]}{\mathrm{D}[\ln X_i]},$$

with

$$\mathrm{E}[\ln X_i] = \ln \mathrm{E}[X_i] - \frac{1}{2}\ln\left[\left(\frac{\mathrm{D}[X_i]}{\mathrm{E}[X_i]}\right)^2 + 1\right],$$

$$\mathrm{D}[X_i] = \sqrt{\ln\left[\left(\frac{\mathrm{D}[X_i]}{\mathrm{E}[X_i]}\right)^2 + 1\right]}.$$

Moreover, $\varrho(Z_i, Z_j) = \lambda\varrho(X_i, X_j)$, where

$$\lambda = \frac{\ln(1 + \varrho(X_i, X_j)\delta_i\delta_j)}{\varrho(X_i, X_j)\sqrt{\ln(1 + \delta_i^2)\ln(1 + \delta_j^2)}}, \tag{11}$$

when both X_i and X_j are lognormally distributed. The coefficient of variation, δ_i, is defined as

$$\delta_i = \frac{\mathrm{D}[X_i]}{\mathrm{E}[X_i]}.$$

If X_i is normally distributed and X_j is lognormally distributed, we have

$$\lambda = \frac{\delta_j}{\sqrt{\ln(1 + \delta_j^2)}}. \tag{12}$$

The formulas (11) and (12) are exact. We note that the joint normal distribution of **Z** is, of course, an assumption and implies a certain joint density of **X**. This joint density is generally different from the exact joint density which is unknown, but the two densities have the same correlation matrix **R**. The basic assumption is that the distribution is characterized to a large extent by second-order statistics. Note that uncorrelated basic variables imply stochastically independent variables when we use the transformation **X** → **Z** described above.

3.3. Determination of the design point. Let $g_y(\mathbf{y})$ be the limit state function in **Y**-space. The design point $\mathbf{y}^*$ is the solution of the constrained minimization problem

$$\mathbf{y}^* = \min_{g_y=0} ||\mathbf{y}||.$$

There are generally several local minima of the distance to the failure surface. The computation of a local minimum is carried out by a standard procedure [11]. In this work, we iterate in $\mathbf{X}$-space and linearize the transformation $\mathbf{X} \to \mathbf{Y}$ in each iteration. Define the vector $\widehat{\mathbf{M}} = (\hat{\mu}_1, \cdots, \hat{\mu}_r)^T$ and the matrix $\widehat{\mathbf{D}} = \operatorname{diag}(\hat{\sigma}_i, \cdots, \hat{\sigma}_r)$. Given an initial guess $\mathbf{x}^0 = (x_1^0, \cdots, x_r^0)^T$ for the design point in $\mathbf{X}$-space, we iterate for $\ell = 0, 1, \cdots$ as follows:

$$\hat{\sigma}_i^\ell = \frac{\varphi\left\{\Phi^{-1}\left[\Theta_{X_i}(x_i^\ell)\right]\right\}}{\theta_{X_i}(x_i^\ell)}, \qquad i = 1, \cdots, r, \tag{13}$$

$$\hat{\mu}_i^\ell = x_i^\ell - \hat{\sigma}_i^\ell \Phi^{-1}\left[\Theta_{X_i}(x_i^\ell)\right], \qquad i = 1, \cdots, r, \tag{14}$$

$$\mathbf{Q}^\ell = \widehat{\mathbf{D}}^\ell \mathbf{R}_0 \widehat{\mathbf{D}}^\ell, \tag{15}$$

$$\mathbf{x}^{\ell+1} = \widehat{\mathbf{M}}^\ell + \frac{\left\langle \mathbf{x}^\ell - \widehat{\mathbf{M}}^\ell, \nabla_x g(\mathbf{x}^\ell)\right\rangle - g(\mathbf{x}^\ell)}{\langle \nabla_x g(\mathbf{x}^\ell), \mathbf{Q}^\ell \nabla_x g(\mathbf{x}^\ell)\rangle} \mathbf{Q}^\ell \nabla_x g(\mathbf{x}^\ell), \tag{16}$$

$$\mathbf{y}^{\ell+1} = \mathbf{L}_0^{-1}\left[\widehat{\mathbf{D}}^\ell\right]^{-1}\left(\mathbf{x}^{\ell+1} - \widehat{\mathbf{M}}^\ell\right), \tag{17}$$

$$\beta^{\ell+1} = \sqrt{\langle \mathbf{y}^{\ell+1}, \mathbf{y}^{\ell+1}\rangle}. \tag{18}$$

In these equations, ∇_x is the gradient operator in $\mathbf{X}$-space, that is,

$$\nabla_x = \left(\frac{\partial}{\partial x_1}, \cdots, \frac{\partial}{\partial x_r}\right)^T.$$

φ denotes the univariate normal density function with zero mean and unit variance. Equation (17) shows that $\widehat{\mathbf{M}}$ and $\widehat{\mathbf{D}}$ can be interpreted as the instantaneous, or equivalent, means and standard deviations of $\mathbf{X}$. In fact, when $\mathbf{X}$ are jointly normally distributed, $\widehat{\mathbf{M}}$ and $\widehat{\mathbf{D}}$ contain the means and standard deviations, respectively. Since the value of g is a result of a numerical solution of partial differential equations, the gradient $\nabla_x g$ of g with respect to the basic variables must be computed numerically. In each iteration the algorithm above therefore requires $r+1$ evaluations of $g = T - T_\omega$. The iteration is stopped when $|\beta^{\ell+1} - \beta^\ell| \leq \epsilon_\beta$, where, for our purposes, $\epsilon_\beta = 0.01$ suffices. Having found β, we utilize the approximation

$$\Pr\{g(\mathbf{X}) \leq 0\} \approx \begin{cases} \Phi(-\beta), & g(\mathbf{x}^*) > 0, \\ \Phi(\beta), & g(\mathbf{x}^*) \leq 0. \end{cases}$$

We note that if $\mathbf{X}$ is jointly normally distributed, β coincides with the Hasofer–Lind reliability index [15].

3.4. Sensitivity measures. One of the attractive features of FORM is that we can compute sensitivity coefficients very efficiently. Let the transformation $\mathbf{X} \to \mathbf{Y}$ be denoted by $\mathbf{Y} = \mathbf{T}(\mathbf{X}; \mathrm{E}[\mathbf{X}], \mathrm{D}[\mathbf{X}])$. Then we have

$$\frac{\partial \beta}{\partial \mathrm{E}[\mathbf{X}]} = \frac{1}{\beta}\left\langle \mathbf{y}^*, \frac{\partial}{\partial \mathrm{E}[\mathbf{X}]}\mathbf{T}(\mathbf{x}^*; \mathrm{E}[\mathbf{X}], \mathrm{D}[\mathbf{X}])\right\rangle \tag{19}$$

and

$$\frac{\partial \beta}{\partial \mathrm{D}[\mathbf{X}]} = \frac{1}{\beta}\left\langle \mathbf{y}^*, \frac{\partial}{\partial \mathrm{D}[\mathbf{X}]}\mathbf{T}(\mathbf{x}^*; \mathrm{E}[\mathbf{X}], \mathrm{D}[\mathbf{X}])\right\rangle. \tag{20}$$

The limit state function may contain a set of parameters $\mathbf{r}$ besides the basic variables $\mathbf{X}$: $g = g(\mathbf{X}; \mathbf{r})$. In our case, $\mathbf{r}$ may be input variables that are not treated stochastically. It can be shown that

$$\frac{\partial \beta}{\partial \mathbf{r}} = \frac{\frac{\partial}{\partial \mathbf{r}} g(\mathbf{x}^*; \mathbf{r})}{\sqrt{\langle \nabla_x g(\mathbf{x}^*), \mathbf{Q}^* \nabla_x g(\mathbf{x}^*)\rangle}}, \tag{21}$$

where $\mathbf{Q}^*$ is the value of $\mathbf{Q}$ in the last iteration. Strictly speaking, the sensitivity coefficients above are approximately correct and approach the exact coefficients as $\beta \to \infty$. It is important to remark that the gradients of g with respect to limit state function parameters or distribution parameters must be determined numerically in our application. In practice, we are often interested in the sensitivity of $\Pr\{T \leq T_\omega\}$ to distribution or limit state function parameters. Such quantities are obtained straightforwardly:

$$\Upsilon_E \equiv \frac{\partial}{\partial \mathrm{E}[\mathbf{X}]}\Pr\{T \leq T_\omega\} = \varphi(\beta)\frac{\partial \beta}{\partial \mathrm{E}[\mathbf{X}]},$$

$$\Upsilon_D \equiv \frac{\partial}{\partial \mathrm{D}[\mathbf{X}]}\Pr\{T \leq T_\omega\} = \varphi(\beta)\frac{\partial \beta}{\partial \mathrm{D}[\mathbf{X}]},$$

$$\Upsilon_R \equiv \frac{\partial}{\partial \mathbf{r}}\Pr\{T \leq T_\omega\} = \varphi(\beta)\frac{\partial \beta}{\partial \mathbf{r}}.$$

Define the unit vector $\boldsymbol{\alpha} = (\alpha_1, \cdots, \alpha_r)^T$ as $\boldsymbol{\alpha} = \mathbf{y}^*/\beta$. The quantities $\alpha_1^2, \cdots, \alpha_r^2$ are termed *importance factors*, and since $\sum_{i=1}^r \alpha_i^2 = 1$, α_i^2 reflects the relative importance of $\mathbf{X}_i$ in determining β or $\Pr\{g \leq 0\}$.

Omission sensitivity factors reflect the error in β when one of the basic variables X_i is treated as a constant with value equal to, say, the median $\bar{m}_i$ of X_i. Generally, we have

$$\gamma_i \equiv \frac{1}{\sqrt{1-\alpha_i^2}} \to \frac{\beta(X_i = \bar{m}_i)}{\beta}, \qquad \alpha_i^2 \to 0. \tag{22}$$

Thus, when γ_i is close to unity, the variable X_i contributes little to β (or $\Pr\{g \leq 0\}$) and may be omitted from the stochastic analysis and instead replaced by its median value.

4. Results. In this section, we first present the exact analytical solution of a simplified oil recovery model. Then, some general considerations on the numerical search algorithm from §3 are reported. We discuss which of the input parameters in the problem should be modeled as stochastic variables and present a justification of the chosen values of statistical and physical parameters. A quantification of the numerical discretization errors is also given. Furthermore, we discuss the quality of the FORM approximations in the present application. Finally, the numerical method is applied to the hydrocarbon recovery model and the results are reported and discussed.

4.1. Analytical solution of a simplified model. With normally distributed variables and a linear limit state function g, $\Phi^{-1}(\beta)$ is the exact probability $\Pr\{T \leq T_\omega\}$. In this case, the numerical search algorithm converges to the exact solution in a single step. Let $\widehat{\mathbf{M}} = \mathrm{E}[\mathbf{X}]$, let $\widehat{\mathbf{D}} = \mathrm{diag}(\mathrm{D}[X_1], \cdots, \mathrm{D}[X_r])$, and let $\mathbf{R}_0$ be the correlation coefficient matrix. The exact FORM solution is then obtained by

$$\mathbf{Q} = \widehat{\mathbf{D}}\mathbf{R}_0\widehat{\mathbf{D}}, \tag{23}$$

$$\mathbf{x}^* = \widehat{\mathbf{M}} + \frac{\left\langle \mathbf{x}^0 - \widehat{\mathbf{M}}, \nabla_x g \right\rangle - g(\mathbf{x}^0)}{\langle \nabla_x g, \mathbf{Q}\nabla_x g\rangle} \mathbf{Q}\nabla_x g, \tag{24}$$

$$\mathbf{y}^* = \mathbf{L}_0^{-1} \left[\widehat{\mathbf{D}}\right]^{-1} \left(\mathbf{x}^* - \widehat{\mathbf{M}}\right), \tag{25}$$

$$\beta^* = \sqrt{\langle \mathbf{y}^*, \mathbf{y}^* \rangle}. \tag{26}$$

It is trivial to show that

$$\left\langle \mathbf{x}^0 - \widehat{\mathbf{M}}, \nabla_x g \right\rangle = g(\mathbf{x}^0) - \mathrm{E}[g].$$

Let us introduce the constant χ by

$$\chi = -\frac{\mathrm{E}[g]}{\langle \nabla_x g, \mathbf{Q}\nabla_x g\rangle}.$$

We then have

$$\mathbf{y}^* = \chi \mathbf{L}_0^T \widehat{\mathbf{D}} \nabla_x g.$$

The present polymer flooding model generally gives rise to nonlinear limit state functions. However, with a particular choice of the input parameters, it is possible to obtain a g that is linear in the basic variables. Let $\nu = 1$, $\zeta = 0$, $S_{wr} = S_{or} = 0$, $K_{wm} = K_{om} = 1$, and $a = b = 1$. Then the pressure field is constant throughout the spatial domain and the saturation fulfills the initial-boundary value problem

$$\phi S_t + cKS_x = 0, \quad S(x,0) = 0, \quad 0 < x < L, \quad S(0,t) = 1, \tag{27}$$

where c is a constant that depends on the pressure gradient κ and the viscosities. The general solution of this linear equation is

$$S(x,t) = 1 - \tilde{H}\left(x - \frac{cK}{\phi}t\right),$$

where $\tilde{H}(\cdot)$ is the Heaviside function. The corresponding breakthrough time becomes

$$T = \frac{\phi L}{cK}.$$

Assume that the porosity and the permeability are the only stochastic variables ($X_1 = \phi$, $X_2 = K$) and that both variables are normally distributed. Then the deterministic breakthrough time reads

$$\tilde{T} = \frac{\mathrm{E}[\phi]L}{\mathrm{E}[K]c}.$$

The limit state function has previously been defined as $g = T - T_\omega$, $T_\omega = \omega\tilde{T}$. Since only the failure surface $g = 0$ has any statistical significance, we may write $g = KT - K\omega\tilde{T}$ or

$$g(\phi, K) = \mathrm{E}[\phi]\frac{L}{c}\left(\frac{\phi}{\mathrm{E}[\phi]} - \omega\frac{K}{\mathrm{E}[K]}\right).$$

By introduction of the coefficients of variation $\delta_\phi = \mathrm{D}[\phi]/\mathrm{E}[\phi]$ and $\delta_K = \mathrm{D}[K]/\mathrm{E}[K]$, we achieve

$$\hat{\mathbf{D}}\nabla_x g = \mathrm{E}[\phi]\frac{L}{c}(\delta_\phi, -\omega\delta_K)^T.$$

The correlation coefficient matrix is written as

$$\mathbf{R} = \mathbf{R}_0 = \begin{pmatrix} 1 & \varrho \\ \varrho & 1 \end{pmatrix},$$

with lower triangular Cholesky factor

$$\mathbf{L}_0 = \begin{pmatrix} 1 & 0 \\ \varrho & \sqrt{1-\varrho^2} \end{pmatrix}.$$

This gives

$$\mathbf{y}^* = \tilde{\chi}\begin{pmatrix} \delta_\phi - \varrho\omega\delta_K \\ -\sqrt{1-\varrho^2}\omega\delta_K \end{pmatrix}, \tag{28}$$

where $\tilde{\chi} = \chi L\mathrm{E}[\phi]/c$.

Let us study the ratio of the importance factors

$$\frac{\alpha_1^2}{\alpha_2^2} = \left(\frac{1-\varrho\omega\hat{\delta}}{\sqrt{1-\varrho^2}\omega\hat{\delta}}\right)^2, \tag{29}$$

where $\hat{\delta} = \delta_K/\delta_\phi$. Without correlation and with $\delta_\phi = \delta_K$, the relative importance of ϕ vs. K depends on ω^{-2}. For small probabilities, ϕ is most important, while for probabilities close to 1, K is the dominating stochastic variable. It is obvious from the formula that the introduction of a correlation coefficient has significant influence on the relative importance of ϕ and K.

Although the physical and statistical simplifications that were required in this analytical model are highly questionable, the simple formula (29) gives valuable insight into the dynamics of a stochastic porous media flow model.

4.2. Performance of the search algorithm. When the partial differential equations are discretized, T can only be determined as $m\Delta t$, m being an integer. Numerical computations of limit state function derivatives, e.g.,

$$\frac{\partial g}{\partial x_1} \approx \frac{1}{\epsilon}[g(x_1+\epsilon, x_2, \cdots, x_r) - g(x_1, \cdots, x_r)],$$

may then result in a zero derivative if ϵ is so small that $g(x_1+\epsilon, x_2, \cdots, x_r) - g(x_1, \cdots, x_r) \leq \Delta t$. Even if ϵ is large enough to prevent vanishing derivatives, convergence problems of the search algorithm may occur. Of course, the value of ϵ when computing $\partial g/\partial x_i$ depends on the magnitude of x_i. We have scaled all basic variables such that their magnitudes are of order unity. In this work, we have obtained good results with $\epsilon = 0.1$ for all basic variables.

It is required that the entries in $\mathbf{R}$ must be chosen such that $\mathbf{R}_0$ becomes positive definite; for example, correlation coefficients $\varrho(X_i, X_j)$ close to ± 1 must be avoided, as these may result in $|\varrho(Z_i, Z_j)| > 1$. If a high correlation (e.g., $\varrho(X_i, X_j) \geq 0.7$) is assigned for many of the basic variables, convergence problems of the algorithm (13)–(18) may occur.

In general, the convergence rate of the search algorithm depends on ω. Usually, the algorithm requires about three iterations to converge. For ω values corresponding to very small probabilities or to probabilities close to 1, the convergence may be slower. When using FORM, convergence problems can sometimes be avoided by applying $\log g$ as limit state function instead of g. Such an approach was tested but not found necessary in the present study.

4.3. The choice of stochastic variables. Of the parameters entering the fractional flow function, it is physically reasonable to assume that ν and ζ can be treated as known constants with negligible uncertainty. On the contrary, ϕ, K_{wm}, K_{om}, S_{wr}, S_{or}, a, b, K, ξ, and η represent rock, or fluid–rock mixture, properties with a possibly significant inherent uncertainty. A deterministic sensitivity analysis of the present problem [13] indicated little sensitivity of T to perturbations in ξ and η. These two parameters are hence treated deterministically while the other parameters are represented by stochastic variables. If the reservoir is considered homogeneous, only one stochastic variable is needed to represent a stochastic physical parameter. However, in the general heterogeneous case, the physical parameters should be modeled as stochastic fields and then discretized to vectors of stochastic variables. The numerical examples in this section are limited to homogeneous reservoirs. We thus exclude the important effects of fine-scale permeability heterogeneity. The present model is therefore relevant when either the fine-scale permeability heterogeneities are negligible, or when some averaging procedures have been applied to approximate heterogeneity effects by an uncertain constant permeability value.

4.4. Statistical and physical parameters. The values of the rock parameters throughout the homogeneous reservoir are modeled by stochastic variables. The basic variables in the problem are defined as

$$X_1 = \phi, \quad X_2 = a, \quad X_3 = b, \quad X_4 = S_{wr},$$

$$X_5 = S_{or}, \quad X_6 = K_{wm}, \quad X_7 = K_{om}, \quad X_8 = K.$$

Adsorption parameters are taken as $\xi = 0.25$ and $\eta = 1.0$. Moreover, $\zeta = 2$ unless otherwise stated. In all calculations, we have used $L = 1000$ m, $\kappa = 10^{-11}$ m^2,

$\Lambda = 0.01$, $\nu = (\mu_w/\mu_o) = \frac{1}{4}$, and $\Delta P = 50$ GPa. Polymer is injected during the whole simulation ($t_s > T$).

In accordance with common practice, the porosity is assumed to be normally distributed, whereas the absolute permeability is assigned a lognormal distribution. The other basic variables can attain positive values only; some of them are also restricted to certain intervals, for example, $0 < K_{wm}, K_{om} \leq 1$ and $0 \leq S_{wr}+S_{or} \leq 1$. Due to lack of better knowledge, we let all variables except the porosity be lognormally distributed.

In order to achieve reliable conclusions regarding the present recovery process, the values of expectations, standard deviations, and correlations must be physically relevant. As an aid for evaluating the physical relevance of expectation and coefficient of variation values, we introduce an interval I_p^i for X_i such that $\Pr\{X_i \in I_p^i\} = p$. In Table 1 the intervals I_p^i, corresponding to a choice of $p = 0.997$, are listed for various values of δ_i. All values of $\mathrm{E}[X_i]$ and δ_i in Table 1 are considered physically relevant to the present displacement process; cf., for example, [1], [2], [3], [4], and [8].

TABLE 1
Relations between distribution parameters and intervals $I_{0.997}^i$, *where* $\Pr\{X_i \in I_{0.997}^i\} = 0.997$.

Quantity	Distribution	$\mathrm{E}[X_i]$	δ_i	$I_{0.997}^i$
ϕ	normal	0.2	1/10	[0.14, 0.26]
K	lognormal	1.0	1	[0.06, 8.60]
K	lognormal	1.0	1/2	[0.22, 3.69]
K	lognormal	1.0	1/10	[0.74, 1.34]
a, b	lognormal	2.0	1/10	[1.48, 2.68]
S_{wr}, S_{or}	lognormal	0.2	1/10	[0.15, 0.27]
K_{wm}, K_{om}	lognormal	0.6	1/10	[0.44, 0.81]
K_{wm}	lognormal	0.4	1/20	[0.34, 0.46]
K_{om}	lognormal	0.8	1/20	[0.69, 0.93]

Correlations among the basic variables do exist in the present problem, but numerical values of the correlation coefficients are difficult to assign. Lithologically similar samples show a linear relationship between porosity and the logarithm of the permeability [3], [1]. It is therefore widely accepted experimental evidence for setting $\varrho(\phi, \ln K)$ close to 1; here we use $\varrho(\phi, \ln K) = 0.8$. From both a numerical and a physical point of view, it may be interesting to investigate the impact of a stronger correlation between several of the basic variables, although the experimental data for such correlation coefficients are sparse. Based on a rough comparsion of different relative permeability curves in the literature, we make the assumption that increasing the curvature of k_{rw} is usually reflected in an increased curvature of k_{ro}. Moreover, a decrease in S_{wr} is assumed to be related to a decrease in S_{or}. This means that $\varrho(a, b) > 0$ and $\varrho(S_{or}, S_{wr}) > 0$. The latter two correlation coefficients are introduced in only one numerical example, since their physical relevance is not well documented. However, the results from such an example may reveal interesting features of the stochastic dynamics of the present flow model.

4.5. Numerical errors. It is important to identify which features of the results are due to numerical approximations and which are due to the mathematical model, that is, the physics in the problem. By running relevant examples on a sequence of grids with decreasing Δx, the sensitivity of the results to the discretization parameters can be demonstrated. We have chosen three test cases. In all cases, $\mathrm{E}[\phi] = \mathrm{E}[S_{wr}] = \mathrm{E}[S_{or}] = 0.2$, $\mathrm{E}[K_{wm}] = \mathrm{E}[K_{om}] = 0.6$, $\mathrm{E}[K] = 1$, and $\delta_i = 1/10$. Moreover, test cases

1 and 2 have E$[a]$ = E$[b] = 2$, while test case 3 has E$[a]$ = E$[b] = 1$. All basic variables are uncorrelated in test cases 1 and 3, while test case 2 has significant correlations: $\varrho(\phi, \ln K) = \varrho(S_{wr}, S_{or}) = \varrho(a, b) = 0.8$. The test cases are constructed to cover various expected numerical difficulties, such as strong correlations and linear relative permeability curves.

Table 2 presents the deterministic reference breakthrough time $\tilde{T}$ and the probability $\Pr\left\{T \leq 0.8\tilde{T}\right\}$ for $L/\Delta x = 40, 80, 160, 320$. It is seen that the errors in T are less than 3 percent for $L/\Delta x = 80$, while the errors in the probability are less than 1 percent. The variations of α_i^2 are not shown, but these were always less than 0.01. The fairly slow convergence of T as $\Delta x \to 0$ has little effect on the statistical results, since the latter is solely based on the failure surface $T = \omega\tilde{T}$. If the error δT in T is of the form $\delta T = pT$, the relative error p cancels in the expression for the failure surface.

Based on the results in Table 2 and our additional experience, the choice $L/\Delta x = 80$ seems to give sufficient accuracy. Variations of numerical parameters in the constrained optimization algorithm have less influence on the accuracy than Δx as long as these parameters are chosen as described in §§3 and 4.2.

TABLE 2

Variation of probabilities and deterministic breakthrough times with mesh partition, $p \equiv \Pr\{T \leq T_{0.8}\}$.

Test	$L/\Delta x = 40$		$L/\Delta x = 80$		$L/\Delta x = 160$		$L/\Delta x = 320$	
case	p	$\tilde{T}$	p	$\tilde{T}$	p	$\tilde{T}$	p	$\tilde{T}$
1	0.132	1440	0.120	1410	0.120	1390	0.114	1380
2	0.068	1440	0.071	1410	0.072	1390	0.069	1380
3	0.018	643	0.056	634	0.058	629	0.059	625

4.6. Examples of failure surfaces in reservoir flow.

Recall that the accuracy of FORM depends on the curvature of the failure surface $g = 0$ at the design point in **Y**-space. If the failure surface is approximately linear, $\Phi(-\beta)$ will be a very good approximation to $\Pr\{g \leq 0\}$. There may frequently be many local minima of the distance to the failure surface, and the computation of the probabilities is then more complicated and expensive [15].

To gain insight into the quality of the FORM approximations in the present problem, we have plotted contour lines of the limit state function when there are only *two* basic variables. Deterministic input parameters or expected values of stochastic variables have the following values: $\phi = 0.2$, $K = 1.0$, $a = b = 2.0$, $K_{wm} = K_{om} = 0.6$, $S_{wr} = S_{or} = 0.2$, $\xi = 0.25$, $\eta = 1.0$, and $\zeta = 2$. In addition, we have used $\delta_i = 1/3$. A deterministic calculation gave $\tilde{T} = 1410$ days.

Figure 1 shows g as a function of a and b in **X**-space and as a function of transformed variables in **Y**-space. A lognormal distribution is assigned to a and b. The variables are uncorrelated. At this point it would be natural to introduce additional symbols to distinguish deterministic and stochastic quantities. However, with the widely used notation employed here for the input parameters, no simple rule (e.g., upper case stochastic variables, lower case deterministic variables) seems applicable. In Fig. 2 we have plotted g as a function of S_{wr} and S_{or}. These basic variables are lognormally distributed with correlation coefficient $\varrho(S_{wr}, S_{or}) = 0.5$. Fig. 3 visualizes g as a function of K_{wm} and K_{om}. These two basic variables are uncorrelated and lognormally distributed. Finally, in Fig. 4, g is plotted as a function of ϕ and

K, where $\varrho(\phi, \ln K) = 0.8$, and ϕ is normally distributed while K has a lognormal distribution function.

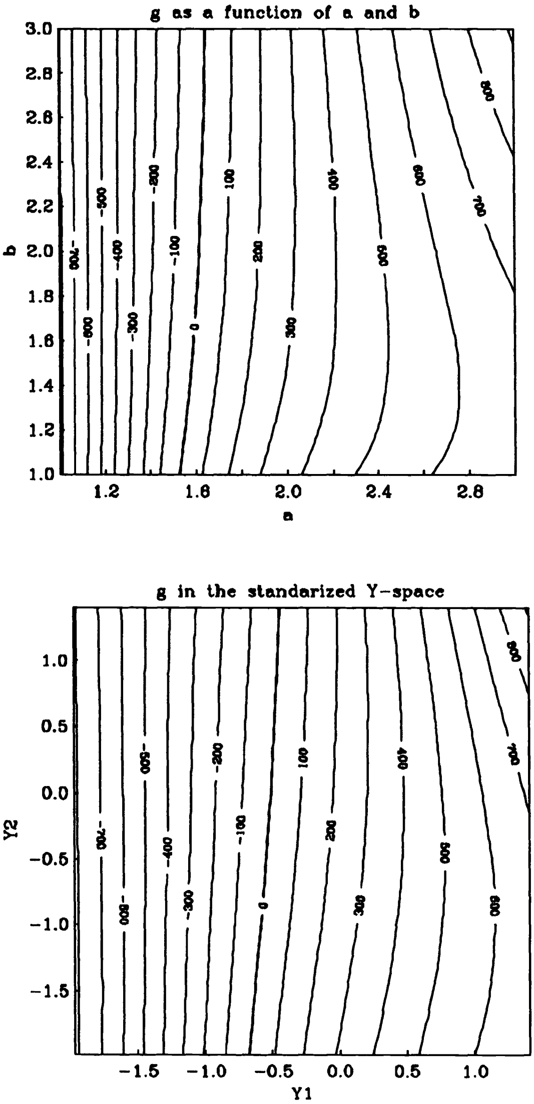


FIG. 1. *The limit state function $g = T - T_\omega$ as a function of $X_1 = a$ and $X_2 = b$ (top) and the corresponding transformed variables Y_1 and Y_2 (bottom). $\varrho(a, b) = 0$.*

The plots of $g(\mathbf{X})$ indicate how T varies with a, b, S_{wr}, S_{or}, K_{wm}, K_{om}, ϕ, and K, and can be of interest with respect to interpretation of the results from a deterministic simulator.

From the plot of g in $\mathbf{Y}$-space we see that the failure surfaces,which correspond to the contour lines of g, are approximately linear. This indicates satisfactory accuracy of FORM probabilities. We also see that there is not more than one local minimum

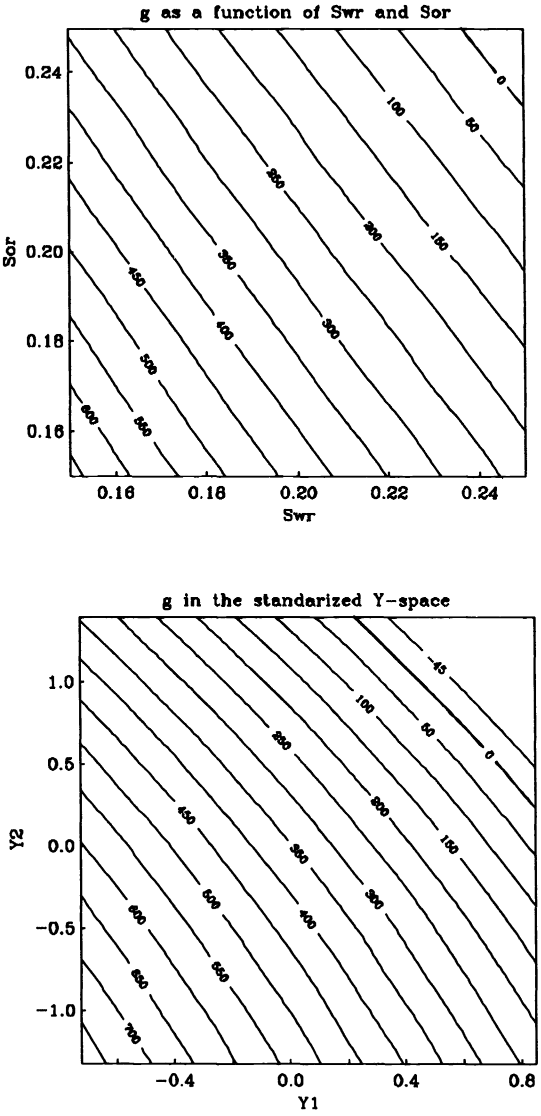


FIG. 2. *The limit state function $g = T - T_\omega$ as a function of $X_1 = S_{wr}$ and $X_2 = S_{or}$ (top) and the corresponding transformed variables Y_1 and Y_2 (bottom). $\varrho(S_{wr}, S_{or}) = 0$.*

of the distance to the failure surface, and that the formulas presented previously are, in this case, sufficient for calculating probabilities.

4.7. Stochastic polymer flooding. Some examples of FORM analysis applied to reservoir simulation are now given to indicate the potential of the proposed method. Most of the numerical examples are concerned with the computation of $\Pr\left\{T < 0.8\tilde{T}\right\}$, that is, the probability that the breakthrough time is underestimated by more than 20 percent when using a deterministic simulator. This probability is important for management decisions. We also give examples of complete probability

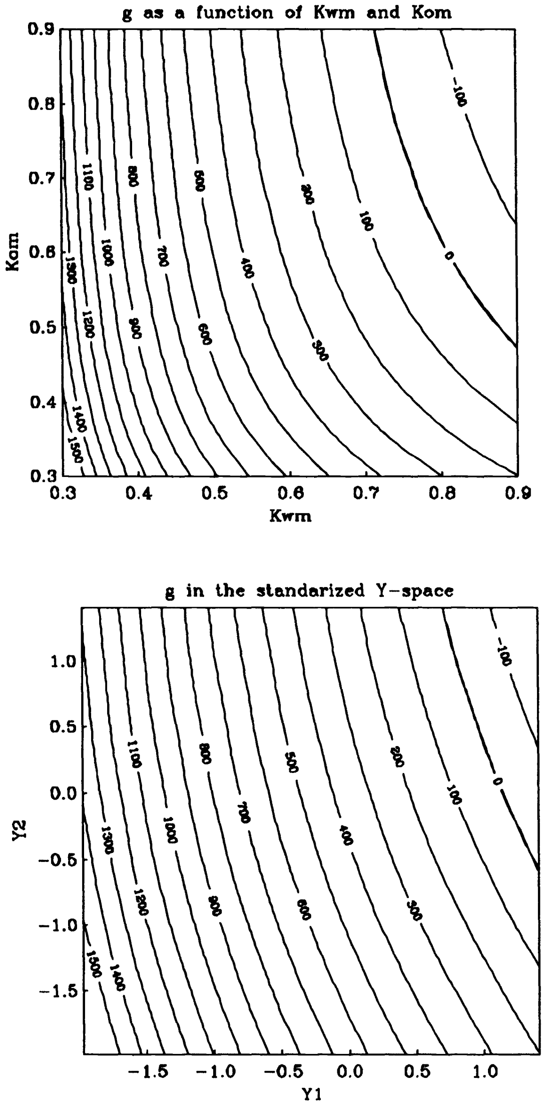


FIG. 3. *The limit state function $g = T - T_\omega$ as a function of $X_1 = K_{wm}$ and $X_2 = K_{om}$ (top) and the corresponding transformed variables Y_1 and Y_2 (bottom). $\varrho(K_{wm}, K_{om}) = 0$.*

distributions. All tables are written directly in the text processing format by the computer in order to minimize typing errors.

Selected results from the proposed stochastic model appear in Tables 3–9. Table 3 shows the statistical results for pure water flooding with uncorrelated basic variables. The corresponding polymer flooding problem gave the results presented in Table 4. There are minor differences in the statistical results. However, the deterministic breakthrough time is, as expected, significantly increased when polymer is injected. Imposing the physically relevant correlation coefficient $\varrho(\phi, \ln K) = 0.8$ on the problem

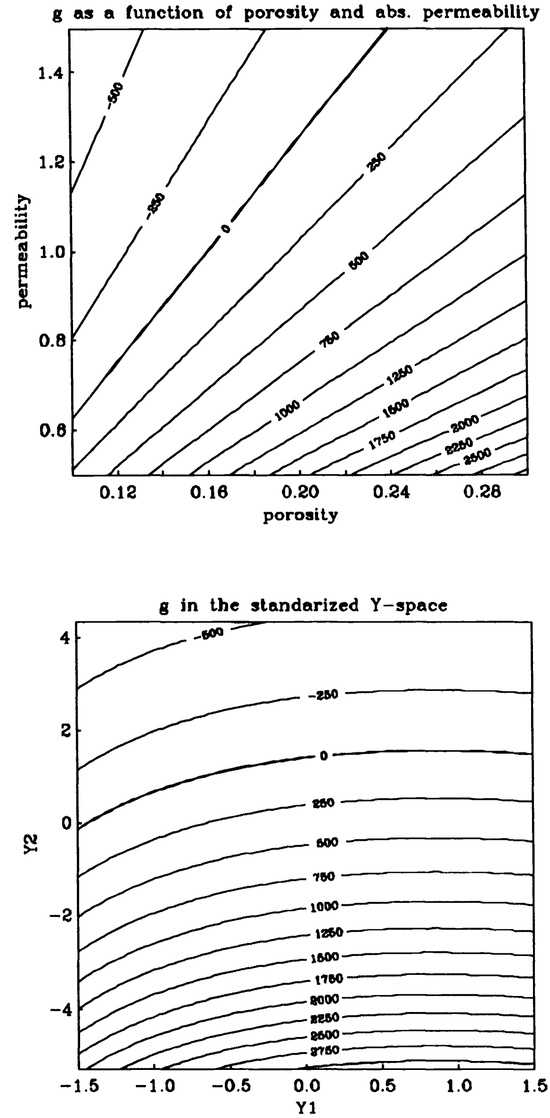


FIG. 4. *The limit state function $g = T - T_\omega$ as a function of $X_1 = \phi$ and $X_2 = K$ (top) and the corresponding transformed variables Y_1 and Y_2 (bottom). $\varrho(\phi, \ln K) = 0.8$.*

in Table 4 led to the results displayed in Table 5. Without correlations, ϕ has the largest importance factor. That is, ϕ is the input parameter that is most important to model as a stochastic variable. When ϕ and $\ln K$ are correlated, their importance factors decrease and become approximately equal, while the importance factor of S_{wr} increases significantly. It may be of interest to investigate the impact of additional correlations $\varrho(S_{wr}, S_{or}) = \varrho(a, b) = 0.8$. The corresponding results are presented in Table 6. In this case, S_{wr} is definitely the most important variable.

The coefficient of variations were fixed at 0.1 for ϕ, a, b, S_{wr}, S_{or}, and K in Tables 3–6. Although the associated standard deviations are physically reasonable for a ***homogeneous*** reservoir, it is interesting to investigate the effect of increasing the uncertainty in the absolute permeability. Table 7 corresponds to Table 4, with the exception that the coefficient of variation of the absolute permeability is five times larger in Table 7 than in Table 4. As expected, this leads to a larger importance factor of K. Less expected results arise, however, when the correlation $\varrho(\phi, \ln K) = 0.8$ is imposed (see Table 8). In this latter case ϕ is almost as important as K.

The stochastic analysis provides two different measures of input parameter importance: (1) the importance factor α_i^2, which reflects the importance of modeling a parameter as a random variable, and, (2) the Υ_E and Υ_D values, which give information about the parameters that are most important to measure accurately in laboratory experiments. It is seen from these tables that the sensitivity of $\Pr\{T \leq T_\omega\}$ to expectation values of the basic variables is significantly larger than the sensitivity to the standard deviations. We must also note that the degree of importance of a variable depends highly on the chosen importance measure: α_i^2, Υ_E, or Υ_D. For example, in the problem associated with the results in Table 7, K is the most important quantity to model as a stochastic variable, while ϕ has the most important expectation value. The sensitivity to expectation values is generally considerably less for K than for ϕ and S_{wr}.

A widely accepted assumption on the distribution functions is that ϕ follows a normal distribution while K is lognormally distributed. For the other basic variables, we have prescribed lognormal distributions, and it is therefore necessary to investigate the sensitivity of the results in the previous tables to variations in the type of distributions. If all the basic variables are assigned normal distribution functions in the physical problem associated with Table 5, we obtain the results shown in Table 9. It is evident that the particular choice of either normal or lognormal distributions has little influence on the results in this example. This observation increases the relevance of the analytical results obtained in §4.1, which required the permeability to be normally distributed.

TABLE 3

Buckley–Leverett flow (pure water flooding). No correlations, $\beta = 1.47$, $\Pr\{T < T_{0.80}\} = \Pr\{T < 659\} = 0.07$, *two iterations.*

Parameter	E[**X**]	D[**X**]	**x***	α_i^2	γ_i	Υ_E	Υ_D
ϕ	0.20	0.020	0.18	0.55	1.50	5.04	−0.33
a	2.00	0.200	1.93	0.04	1.02	0.15	−0.03
b	2.00	0.200	1.99	0.00	1.00	−0.01	0.00
S_{wr}	0.20	0.020	0.21	0.05	1.03	−2.41	0.03
S_{or}	0.20	0.020	0.21	0.06	1.03	−2.44	0.03
K_{wm}	0.40	0.020	0.40	0.02	1.01	−1.22	0.01
K_{om}	0.80	0.040	0.81	0.02	1.01	−0.54	0.00
K	1.00	0.100	1.07	0.26	1.16	−0.74	−0.08

The results in Tables 3–9 were restricted to the computations of $\Pr\{T \leq T_{0.8}\}$, i.e., $\omega = 0.8$. The sensitivity of the importance factors to variations in ω has also been investigated, and Table 10 displays an example of the extreme values of α_i^2 for ϕ and K when $\omega \in [0.5, 1.5]$. The physical problem corresponded to that in Tables 7 and 8. In contrast to the simplified analytical model in §3, ω has little impact on the relative ranking of the basic variables in the present example. Similar sensitivity studies of

TABLE 4

No correlations, $\beta = 1.36$, $\Pr\{T < T_{0.80}\} = \Pr\{T < 1496\} = 0.09$, *four iterations.*

Parameter	E[**X**]	D[**X**]	**x***	α_i^2	γ_i	Υ_E	Υ_D
ϕ	0.20	0.020	0.18	0.47	1.37	5.44	−0.30
a	2.00	0.200	1.92	0.07	1.03	0.23	−0.04
b	2.00	0.200	2.00	0.00	1.00	−0.03	0.00
S_{wr}	0.20	0.020	0.21	0.13	1.07	−4.26	0.02
S_{or}	0.20	0.020	0.21	0.07	1.04	−3.24	0.03
K_{wm}	0.40	0.020	0.410	0.05	1.02	−2.22	0.01
K_{om}	0.80	0.040	0.80	0.00	1.00	−0.06	0.00
K	1.00	0.100	1.06	0.22	1.13	−0.82	−0.07

TABLE 5

Same problem as in Table 4 *except that* $\varrho(\phi, \ln K) = 0.8$. $\beta = 1.98$, $\Pr\{T < T_{0.80}\} = \Pr\ \{T < 1496\} = 0.02$, *five iterations.*

Parameter	E[**X**]	D[**X**]	**x***	α_i^2	γ_i	Υ_E	Υ_D
ϕ	0.20	0.020	0.18	0.17	1.10	1.17	−0.06
a	2.00	0.200	1.84	0.16	1.09	0.13	−0.04
b	2.00	0.200	2.01	0.00	1.00	−0.01	0.00
S_{wr}	0.20	0.020	0.22	0.27	1.17	−2.15	−0.03
S_{or}	0.20	0.020	0.22	0.16	1.09	−1.66	−0.01
K_{wm}	0.40	0.020	0.41	0.08	1.04	−1.06	−0.01
K_{om}	0.80	0.040	0.80	0.00	1.00	−0.02	0.00
K	1.00	0.100	0.98	0.15	1.09	−0.26	0.02

TABLE 6

Same problem as in Table 5 *but with additional correlations*: $\varrho(\phi, \ln K) = \varrho(S_{wr}, S_{or}) = \varrho(a, b) = 0.8$. $\beta = 1.77$, $\Pr\{T < T_{0.80}\} = \Pr\{T < 1496\} = 0.04$, *seven iterations.*

Parameter	E[**X**]	D[**X**]	**x***	α_i^2	γ_i	Υ_E	Υ_D
ϕ	0.20	0.020	0.19	0.11	1.06	1.40	−0.05
a	2.00	0.200	1.90	0.07	1.03	0.12	−0.03
b	2.00	0.200	1.93	0.00	1.00	−0.01	0.00
S_{wr}	0.20	0.020	0.23	0.60	1.58	−4.59	−0.14
S_{or}	0.20	0.020	0.23	0.05	1.03	−1.34	−0.04
K_{wm}	0.40	0.020	0.41	0.06	1.03	−1.36	0.00
K_{om}	0.80	0.040	0.80	0.00	1.00	−0.05	0.00
K	1.00	0.100	0.98	0.11	1.06	−0.33	0.02

TABLE 7

Same problem as in Table 4 *except that the coefficient of variation of* K *is larger. No correlations,* $\beta = 0.70$, $\Pr\{T < T_{0.80}\} = \Pr\{T < 1496\} = 0.24$, *three iterations.*

Parameter	E[**X**]	D[**X**]	**x***	α_i^2	γ_i	Υ_E	Υ_D
ϕ	0.20	0.020	0.20	0.07	1.04	4.16	−0.05
a	2.00	0.200	1.97	0.01	1.01	0.19	−0.02
b	2.00	0.200	1.99	0.00	1.00	−0.01	0.00
S_{wr}	0.20	0.020	0.20	0.02	1.01	−3.52	0.07
S_{or}	0.20	0.020	0.20	0.01	1.01	−2.54	0.05
K_{wm}	0.40	0.020	0.40	0.01	1.00	−1.79	0.03
K_{om}	0.80	0.040	0.80	0.00	1.00	−0.08	0.00
K	1.00	0.500	1.22	0.88	2.85	−0.63	−0.02

Υ_E and Υ_D showed that the variation with ω was not significant. As $\omega \to 1$ (from both sides) Υ_E increased in absolute value for all basic variables. The variation of Υ_D with ω did not exhibit a regular pattern common to all X_i's.

TABLE 8

Same problem as in Table 5 *except that the coefficient of variation of K is larger.* $\varrho(\phi, \ln K) = 0.8, \beta = 0.85$, $\Pr\{T < T_{0.80}\} = \Pr\{T < 1496\} = 0.20$, *five iterations.*

Parameter	$\mathrm{E}[\mathbf{X}]$	$\mathrm{D}[\mathbf{X}]$	$\mathbf{x}^*$	α_i^2	γ_i	Υ_E	Υ_D
ϕ	0.20	0.020	0.21	0.41	1.30	−8.85	−0.28
a	2.00	0.200	1.97	0.02	1.01	0.20	−0.02
b	2.00	0.200	1.99	0.00	1.00	−0.03	0.00
S_{wr}	0.20	0.020	0.20	0.03	1.02	−4.00	0.07
S_{or}	0.20	0.020	0.20	0.02	1.01	−3.11	0.06
K_{wm}	0.40	0.020	0.40	0.01	1.01	−2.11	0.03
K_{om}	0.80	0.040	0.80	0.00	1.00	−0.09	0.00
K	1.00	0.500	1.30	0.51	1.43	−0.40	−0.04

TABLE 9

Same problem as in Table 5 *except that all variables are normally distributed.* $\varrho(\phi, \ln K) = 0.8, \beta = 1.95$, $\Pr\{T < T_{0.80}\} = \Pr\{T < 1496\} = 0.03$, *five iterations.*

Parameter	$\mathrm{E}[\mathbf{X}]$	$\mathrm{D}[\mathbf{X}]$	$\mathbf{x}^*$	α_i^2	γ_i	Υ_E	Υ_D
ϕ	0.20	0.020	0.18	0.17	1.10	1.22	−0.06
a	2.00	0.200	1.84	0.17	1.10	0.12	−0.04
b	2.00	0.200	2.02	0.00	1.00	−0.02	0.00
S_{wr}	0.20	0.020	0.22	0.24	1.14	−1.43	−0.08
S_{or}	0.20	0.020	0.22	0.16	1.09	−1.19	−0.06
K_{wm}	0.40	0.020	0.41	0.09	1.05	−0.88	−0.03
K_{om}	0.80	0.040	0.80	0.00	1.00	−0.03	0.00
K	1.00	0.100	0.98	0.17	1.10	−0.24	0.01

TABLE 10

Sensitivity of importance factors α_i^2 to variations in ω.

Quantity	$\varrho(\phi, \ln K) = 0$		$\varrho(\phi, \ln K) = 0.8$	
	$\omega = 0.5$	$\omega = 1.5$	$\omega = 0.5$	$\omega = 1.5$
ϕ	0.07	0.07	0.42	0.38
K	0.87	0.88	0.49	0.53

The continuous variation of $\Pr\left\{T \le \omega\tilde{T}\right\}$ with ω is demonstrated by the cumulative distribution function of T in Fig. 5 with the corresponding density plotted in Fig. 6. The physical problem had expectation values as in Tables 3–9, while the coefficient of variation of *all* variables equaled 0.1. The effect of correlations on the computed probabilities is most pronounced in the tails of the distributions.

The importance measures in the tables indicate that only a few basic variables contribute significantly to the calculation of the probabilities. For example, b and K_{om} could be replaced by their median values with negligible effect on the statistical results. A natural way to reduce the number of basic variables, and hence increase the efficiency of the computations, is to omit basic variables for which γ_i are less than, say, 1.03 after the first iteration. We have tested this approach, but found that the iterative procedure converged more slowly, so that no overall increase in efficiency was achieved. However, after an initial study of a problem we can simply omit some basic variables completely (if $\gamma_i \approx 1$) from the description. This approach enhances the efficiency considerably.

It must be mentioned that increasing the coefficient of variation of the basic variables may give rise to results that deviate from those presented here. Particularly, if $\mathrm{D}[a]$ is increased, for example, to $\frac{2}{3}$, the importance of a increases considerably. This

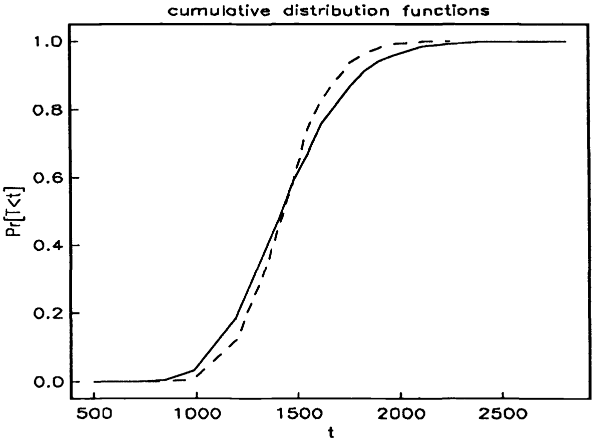


FIG. 5. *Probability distributions* $\Theta_T(t) = \Pr\{T \leq t\}$ *for the time* T *to water breakthrough. The solid line represents uncorrelated variables, while the dashed line represents correlated variables* $(\varrho(\phi, \ln K) = 0.8)$.

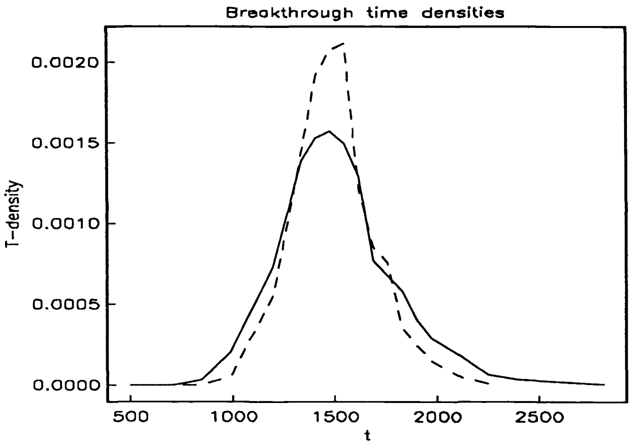


FIG. 6. *Probability densities corresponding to the distributions in Fig.* 5.

is probably because the breakthrough time is highly sensitive to variations in a when $a \sim 1$ [13] and, with $\mathrm{D}[a] = \frac{2}{3}$, a values in this regime have significant probabilities (e.g., $I^2_{0.997} = [0.72, 5.02]$).

5. Conclusion. The purpose of this paper has been to formulate and apply a one-dimensional mathematical model for polymer flooding where some of the geological input parameters were treated as stochastic variables. The time to water breakthrough in the production well constituted the investigated output parameter from the model. Using recently developed methods from structural reliability theory, the system of stochastic partial differential equations was solved as a sequence of the standard deterministic polymer flooding equations. Typically, about 30 deterministic problems must be solved to compute a point on the cumulative distribution curve for the breakthrough time in the present study. With only a few stochastic variables, the method is very efficient compared to Monte-Carlo simulation. An advantage of the method is the straightforward and efficient computation of the relative importance of stochastic variables and various input data, such as expectations and covariances. In one space dimension, the suggested numerical method seems appropriate. However, in higher space dimensions the discretization of heterogeneous stochastic fields implies that a large number of stochastic variables is introduced. In such cases, Monte-Carlo simulation is probably competitive.

One of the most serious deficiencies of a deterministic study is the problem with incorporation of the experimentally reported degree of functional relationship between the input parameters. The stochastic analysis in this paper showed that correlations between the stochastic variables had a significant effect on their relative importance in a stochastic analysis. The expectations of the residual saturation variables and the porosity were the most important input data to this stochastic simulator. On the other hand, the most important parameters to be modeled as stochastic variables were the porosity, the permeability, and the residual water saturation. The computed statistical quantities showed little sensitivity to the choice of either lognormal or normal distribution functions for the basic variables. It is emphasized that the numerical results that have been obtained here may be highly dependent on the probability level, the coefficients of variations, and the correlation coefficients, as was indicated by the simplified analytical solution to the problem.

If simulation methods of Monte-Carlo type are applied for further work concerning two- and three-dimensional problems, effort should be made to develop efficient procedures for calculating important measures of the kind used here. Such measures give valuable physical insight into the stochastic problem and provide information for increasing the efficiency by replacing unimportant stochastic quantities by deterministic values.

REFERENCES

[1] J. O. Aasen, J. K. Silseth, L. Holden, H. Omre, K. B. Halvorsen, and J. Høiberg, *A stochastic reservoir model and its use in evaluations of uncertainties in the results of recovery processes*, in North Sea Oil and Gas Reservoirs—II, The Norwegian Institute of Technology, Graham & Trotman, Norwell, MA, 1990.

[2] M. B. Allen, III, G. A. Behie, and J. A. Trangenstein, *Multiphase flow in porous media*, Lecture Notes in Engineering, Springer-Verlag, Berlin, New York, 1988.

[3] J. S. Archer and C. G. Wall, *Petroleum Engineering, Principles and Practice*, Graham & Trotman, Norwell, MA, 1986.

[4] K. Aziz and A. Settari, *Petroleum Reservoir Simulation*, Applied Science Publishers, London, 1979.

[5] P. Bjerager, *Probability computation methods in structural and mechanical reliability*, in Computational Mechanics of Probabilistic and Reliability Analysis, W. K. Liu and T. Belytschko, eds., Elme Press International, Switzerland, 1989.

[6] L. Y. Ding, R. K. Mehra, and J. K. Donnelly, *Stochastic modeling in reservoir simulation*, in Proc. 10th SPE Symposium on Reservoir Simulation, Texas, 1989, pp. 303–320.

[7] R. E. Ewing, *Problems arising in the modeling of processes for hydrocarbon recovery*, in The Mathematics of Reservoir Simulation, R. E. Ewing, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.

[8] A. Galli, D. Guérillot, C. Ravenne, and Heresim Group, *Combining geology, geostatistics and multiphase fluid flow for 3D reservoir studies*, in Proc. Second European Conference on the Mathematics of Oil Recovery, Arles, France, 1990, D. Guérillot and O. Guillon, eds. pp. 11–20.

[9] H. H. Haldorsen and E. Damsleth, *Stochastic modeling*, J. Petrol. Tech., (1990), pp. 404–412.

[10] T. Johansen, A. Tveito, and R. Winther, *A Riemann solver for a two-phase multicomponent process*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 846–880.

[11] A. Der Kiureghian and P.-L. Liu, *Structural reliability under incomplete probability information*, J. Engrg. Mech., 112 (1986), pp. 85–104.

[12] A. Der Kiureghian and J.-B. Ke, *The stochastic finite element method in structural reliability*, Probab. Engrg. Mech., 3 (1988), pp. 83–91.

[13] H. P. Langtangen, *Sensitivity analysis of an enhanced oil recovery process*, Appl. Math. Modelling, 15 (1991), pp. 467–474.

[14] F. Ma and M. S. Wei, *Monte Carlo simulation of linear two-phase flow in heterogeneous media*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 1053–1072.

[15] H. O. Madsen, S. Krenk, and N. C. Lind, *Methods of Structural Safety*, Prentice–Hall, Englewood Cliffs, NJ, 1986.

[16] D. Peaceman, *Fundamentals of numerical reservoir simulation*, Elsevier, Amsterdam, 1977.

[17] A. Selvig, *Feasibility study of possible applications of* PROBAN *in reservoir simulation*, Tech. Report, A.S. Veritas Research, Norway, 1988.

[18] P. J. Smith and C. E. Brown, *Stochastic modeling of two-phase flow in porous media*, in Proc. 57th SPE Annual Technical Conference, New Orleans, LA, 1982.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 6, pp. 1418–1432, November 1992

009

A FAST ALGORITHM TO SOLVE NONHOMOGENEOUS CAUCHY–RIEMANN EQUATIONS IN THE COMPLEX PLANE*

PRABIR DARIPA†

Abstract. An algorithm is provided for the fast and accurate computation of the solution of nonhomogeneous Cauchy–Riemann equations in the complex plane in the interior of a unit disk. The algorithm is based on the representation of the solution in terms of a double integral, some recursive relations in Fourier space, and fast Fourier transforms. The numerical evaluation of the solution at N^2 points on a polar coordinate grid by straightforward summation for the double integral would require $O(N^2)$ floating point operations per point. Evaluation of these integrals has been optimized in this paper giving an asymptotic operation count of $O(\ln N)$ per point on the average. In actual implementation, the algorithm has even better computational complexity, approximately of the order of $O(1)$ per point. The algorithm has the added advantage of working in place, meaning that no additional memory storage is required beyond that of the initial data. The performance of the algorithm has been demonstrated on several prototype problems. The algorithm has applications in many areas, particularly fluid mechanics, solid mechanics, and quasi-conformal mappings.

Key words. complex variable, nonhomogeneous Cauchy–Riemann equation, Beltrami equation

AMS(MOS) subject classifications. 65E05, 30C99, 65D30

1. Introduction.

Cauchy–Riemann (CR) equations arise in many areas including fluid mechanics, solid mechanics, and electrostatics. These equations are given by

$$\phi_x = \psi_y, \qquad \phi_y = -\psi_x, \tag{1.1}$$

where real variables ϕ and ψ are functions of x and y. The CR equations suggest that the complex function $u = \phi + i\psi$ is an analytic function of the complex variable $\sigma = x + iy$:

$$u = g(\sigma). \tag{1.2}$$

This equation can be equivalently written as $u_{\bar{\sigma}} = 0$. Henceforth we refer to this equation as the CR equation in the complex plane. The CR equation plays an important role for two main reasons: the first is the fact that these equations model many problems in many areas of applications; the second is that the power of complex variable theory may be used in solving these problems. It should be noted that the CR equation is linear and homogeneous. Thus it is not surprising that CR equations arise due to some linear approximations or some nonlinear transformations of the original equations that actually arise in applied fields.

An obvious extension of the CR equation would be to add a nonhomogeneous term to it:

$$u_{\bar{\sigma}} = h(u, \sigma, \bar{\sigma}), \tag{1.3}$$

where the complex function h may be a nonlinear function of u and depends explicitly on σ and $\bar{\sigma}$. This equation naturally reduces to a CR equation when h is identically zero. Note that this equation is nonlinear as well as nonhomogeneous. A well-known special case of (1.3) is the following Beltrami equation (see [1] and [2]):

$$u_{\bar{\sigma}} = \mu(u) u_{\sigma}, \tag{1.4}$$

* Received by the editors November 21, 1990; accepted for publication (in revised form) September 8, 1991. This research was partially supported by National Science Foundation grant DMS-8803669.

† Department of Mathematics, Texas A & M University, College Station, Texas 77843.

where μ may depend on u, σ, and $\bar{\sigma}$. This equation arises in many instances, most notably in the transformation of a quasi-linear partial differential equation to canonical form [2], in quasi-conformal mapping [7], and in compressible fluid flow [6]. In fact, recent work has been done by solving these equations in the real as well as complex plane (see [5] and [6]) in the context of compressible fluid flow and quasi-conformal mapping [7]. It may also arise in other applied areas.

In this paper we are interested in constructing a fast algorithm to solve an appropriate boundary value problem associated with the following linearized version of (1.3):

$$u_{\bar{\sigma}} = f(\sigma, \bar{\sigma}), \tag{1.5}$$

in the unit disk $|\sigma| \leq 1$. The nonanalytic function f is assumed to satisfy a Hölder condition with exponent α in a unit disk Ω: $|\sigma| \leq 1$. Henceforth we refer to this equation as the nonhomogeneous Cauchy–Riemann (NCR) equation. Below, we denote $f(\sigma, \bar{\sigma})$ by $f(\sigma)$.

In [6], Daripa introduced a numerical method to solve this problem. The method is based on splitting the solution of (1.5) as (see also [2])

$$u(\sigma) = u^p(\sigma) + u^a(\sigma), \tag{1.6}$$

where $u^a(\sigma)$ is the analytic part of the solution and $u^p(\sigma)$ is the nonanalytic part of the solution. The nonanalytic part $u^p(\sigma)$ is then given by

$$u^p(\sigma) = -\frac{1}{\pi} \int\int_{\Omega} \frac{f(\zeta)}{\zeta - \sigma} d\xi d\eta, \tag{1.7}$$

where $\zeta = \xi + i\eta$.

The analytic part admits a Taylor series representation which can be efficiently computed using the fast Fourier transform (FFT) (see [3], [4], [6], and [8]). However, there are two main difficulties in the computation of the nonanalytic part $u^p(\sigma)$. The first has to do with the singularity of the integrand in (1.7) at $\zeta = \sigma$. In [6, Appendix C], Daripa proposed a method of desingularizing the integral. This desingularized version of the integral can then be directly integrated, but with poor accuracy, as is explained in §5. The second has to do with the algorithmic complexity of straightforward integration. The straightforward computation of the double integral in (1.7) requires performing an integral with an operation count of the order of $O(N^2)$ for each node in the discretization. There are N^2 such nodes in the discretization of the domain and hence N^2 such integrals to be evaluated. Thus this method of evaluation is computationally very intensive. These are the main drawbacks of the method. However, our goal in this paper is to develop an efficient and accurate algorithm for evaluating this integral and to thus make this a desirable alternative to the above approach.

The process of evaluating these integrals is optimized in this paper, giving a net operation count of the order of $O(N^2 \ln N)$ for N^2 points. This algorithm has the added advantage of working in place, meaning that no additional memory storage is required beyond that of the initial data.

Our method is basically a recursive routine in Fourier space that divides the entire domain (the interior of the unit disk) into a collection of annular regions and expands the integral in Fourier series in angular direction with radius-dependent Fourier coefficients. A set of exact recursive relations are derived which are then used to produce the Fourier coefficients of the integral. These recursive relations involve appropriate scaling of one-dimensional integrals in annular regions. The desired integrals at all N^2 grid points are then easily obtained from the Fourier coefficients by the FFT.

The algorithm developed in this paper also provides a constructive analytical method for evaluating certain integrals that are otherwise difficult to evaluate. An example is provided in Appendix B.

The rest of the paper is structured as follows. Section 2 develops the mathematical foundation of the efficient algorithm to evaluate the particular solution $u^p(\sigma)$ within the unit disk. The fast algorithm and the algorithmic complexity are discussed in §3. In §4, an area integral is evaluated corroborating the algorithmic steps. Section 5 discusses the computational results in evaluating the particular solution using our algorithm of §4. In §6, Dirichlet problems associated with the NCR equation (1.5) and computations of their solution are discussed. We conclude in §7.

In a sequel, we shall apply our algorithm to equations of compressible fluid flow.

2. Mathematical foundation of the algorithm. In this section, we develop the theory needed to construct an efficient algorithm for evaluating the double integral that appears in (1.7).

In the following, we use the notations Ω_r: $|\sigma| \leq r < 1$, $\Omega_{\bar{r}}$: $\Omega \backslash \Omega_r$, and Ω_{ij}: $r_i \leq |\sigma| \leq r_j$. The following theorem is crucial for the later development of the algorithm.

THEOREM 2.1. *The particular solution of $u_{\bar{\sigma}} = f(\sigma)$ with $\sigma = re^{i\alpha}$ can be written as*

$$u^p(\sigma) = \sum_{n=-\infty}^{\infty} c_n(r) e^{in\alpha}, \tag{2.1}$$

where

$$c_n(r) = \begin{cases} \dfrac{1}{\pi} \displaystyle\iint_{\Omega_r} f(\zeta) \left(\frac{r}{\zeta}\right)^n \left(\frac{1}{\zeta}\right) d\xi d\eta, & n < 0, \\[2ex] -\dfrac{1}{\pi} \displaystyle\iint_{\Omega_{\bar{r}}} f(\zeta) \left(\frac{r}{\zeta}\right)^n \left(\frac{1}{\zeta}\right) d\xi d\eta, & n \geq 0. \end{cases} \tag{2.2}$$

Proof. If we introduce the notation

$$P_n(r, \zeta) = \int_0^{2\pi} \frac{e^{-in\alpha}}{\zeta - re^{i\alpha}} d\alpha, \tag{2.3}$$

then it follows from (1.7) and (2.1) that

$$-\pi c_n(r) = \frac{1}{2\pi} \iint_{\Omega} f(\zeta) P_n(r, \zeta) d\xi d\eta. \tag{2.4}$$

The integral in (2.3) can be evaluated using complex variables. It is an elementary exercise in complex variable theory to show that

$$P_n(r, \zeta) = -2\pi r^n S_n(\zeta), \tag{2.5}$$

where

$$S_n(\zeta) = -\delta(n)\zeta^{-(n+1)} + \begin{cases} \zeta^{-(n+1)}, & |\zeta| < |\sigma|, \\ 0.5\zeta^{-(n+1)}, & |\zeta| = |\sigma|, \\ 0, & |\zeta| > |\sigma|. \end{cases} \tag{2.6}$$

In (2.6), $\delta(n) = 0$ for $n < 0$ and $\delta(n) = 1$ for $n \geq 0$. Substitution of (2.5) into (2.4) yields the desired result, i.e., (2.2).

Remark 2.1. The above theorem can also be derived by first expanding $\frac{1}{\zeta-\sigma}$ in power of $\frac{\zeta}{\sigma}$ and $\frac{\sigma}{\zeta}$, followed by integration in the complex plane.

COROLLARY 2.1. *Suppose that $\zeta = \rho e^{i\theta}$ and $f(\zeta)$ has Fourier coefficients $f_n(\rho)$; then it easily follows that the coefficients $c_n(r)$ in* (2.2) *can be written as*

$$c_n(r) = \begin{cases} 2\displaystyle\int_0^r f_{n+1}(\rho)\left(\frac{r}{\rho}\right)^n d\rho, & n<0, \\ -2\displaystyle\int_r^1 f_{n+1}(\rho)\left(\frac{r}{\rho}\right)^n d\rho, & n\geq 0. \end{cases} \tag{2.7}$$

COROLLARY 2.2. *It follows directly from* (2.7) *that $c_n(1) = 0$ for $n \geq 0, c_n(0) = 0$ for all $n \neq 0$. It also follows from the Fourier expansion of $f(\zeta)$ that $f_n(0) = 0$ for $n \neq 0$, and $f_0(0) = f(0)$.*

COROLLARY 2.3. *Let $r_j > r_i$. Define*

$$c_n^{ij} = 2\int_{r_i}^{r_j} f_{n+1}(\rho)\left(\frac{R}{\rho}\right)^n d\rho, \tag{2.8}$$

where

$$R = \begin{cases} r_i, & n \geq 0, \\ r_j, & n < 0. \end{cases} \tag{2.9}$$

After some algebraic manipulation it follows from (2.7) *that*

$$c_n(r_j) = \left(\frac{r_j}{r_i}\right)^n c_n(r_i) + c_n^{ji}, \qquad n < 0, \tag{2.10}$$

and

$$c_n(r_i) = \left(\frac{r_i}{r_j}\right)^n c_n(r_j) - c_n^{ij}, \qquad n \geq 0. \tag{2.11}$$

COROLLARY 2.4. *Let $0 = r_1 < r_2 < r_3 \cdots < r_M = 1$. It follows from recursive applications of* (2.10) *and* (2.11) *and from using Corollary* 2.2 *that*

$$c_n(r_l) = \begin{cases} \displaystyle\sum_{i=2}^{l}\left(\frac{r_l}{r_i}\right)^n c_n^{i-1,i}, & \text{for } n<0 \text{ and } l = 2,\cdots,M, \\ -\displaystyle\sum_{i=l}^{M-1}\left(\frac{r_l}{r_i}\right)^n c_n^{i,i+1}, & \text{for } n\geq 0 \text{ and } l = 1,\cdots,M-1. \end{cases} \tag{2.12}$$

3. The fast algorithm. We construct the fast algorithm based on the theory of §2. The unit disk is discretized using $M \times N$ lattice points with M equidistant points in the radial direction and N equidistant points in the circular direction.

Initialization. Choose M and N. Define $K = \frac{N}{2}$.
Step 1. For $l \in [1, M]$ and $n \in [-K+1, K]$, compute the Fourier coefficients $f_n(r_l)$ of $f(\zeta)$ from the known values of $f(\zeta = r_j e^{2\pi i k/N}), j = 1, \cdots, M; k = 1, \cdots, N$.

Step 2. Compute $c_n^{i,i+1}, i \in [1, M-1]$ for $n \in [-K, K-1]$ using (2.8).

Step 3. Compute the Fourier coefficients $c_n(r_l), n \in [-K, K-1], l \in [1, M]$ using the relations (2.10) and (2.11).

```
set $c_n(r_M) = 0 \ \forall \ n \in [0, K-1]$
do $n = 0, \cdots, K-1$
    do $l = M-1, \cdots, 1$
    Use (2.11) of Corollary 2.4 to compute $c_n(r_l)$.
    $c_n(r_l) = \left(\frac{r_l}{r_{l+1}}\right)^n c_n(r_{l+1}) - c_n^{l,l+1}$
    enddo
enddo

set $c_n(r_1) = 0 \ \forall \ n \in [-K, -1]$
do $n = -K, \cdots, -1$
    do $l = 2, \cdots, M$
    Use (2.10) of Corollary 2.4 to compute $c_n(r_l)$
    $c_n(r_l) = \left(\frac{r_l}{r_{l-1}}\right)^n c_n(r_{l-1}) + c_n^{l-1,l}$
    enddo
enddo
```

Step 4. Finally compute $u^p(\sigma = r_j e^{2\pi i k/N}), j \in [1, M], k \in [1, N]$ using a truncated version of (2.1).

3.1. The algorithmic complexity. Here we consider the computational complexity of the above algorithm. We discuss the asymptotic operation count, the asymptotic time complexity, and asymptotic storage requirement, in that order. In Steps 1 and 4 above, there are 2M FFTs of length N and all other computations in Steps 2 and 3 are of lower order. With each FFT of length N contributing $N \ln N$ operations, the asymptotic operation count and hence the asymptotic time complexity is $O(MN \ln N)$.

The algorithm requires storing the MN Fourier coefficients $f_n(r_l)$ in Step 1, the MN Fourier coefficients $c_n(r_l)$ in Step 3, and the MN values of the desired u^p at MN grid points in Step 4. Therefore the asymptotic storage requirement is $O(MN)$.

Remark 3.1. The computation c_n^{ij} in Step 2 can be embedded within the inner do-loops of Step 3, thus avoiding the storage requirement for these. Note that we present the algorithm in the form shown above for the sake of clarity and without any sacrifice in the asymptotic time complexity.

4. Integral evaluation using the algorithmic steps. We illustrate the algorithmic steps mentioned above through an explicit example. There are three main reasons for doing this. The first is to give an explicit exposition of the operations in each of the algorithmic steps in detail, the second is to show the power of the algorithm to evaluate the integral analytically when function f is known explicitly in terms of the complex coordinates, and the third is to generate an explicit example to test the accuracy and the algorithmic complexity at each step of the algorithm by explicit numerical calculations. This is also useful for debugging various stages of the computation.

We first summarize our result in the following note, proof of which follows.

Note. If

$$f(\sigma) = \sigma^p \bar{\sigma}^q \tag{4.1}$$

where p and q are constants, so that $k = p - q$ is an integer, then the integral $u^p(\sigma)$ in (1.7) is given by the following.

For $k = p - q \geq 1$,

$$u^p(\sigma) = \begin{cases} \dfrac{\sigma^{(k-1)}}{q+1}(|\sigma|^{2(q+1)} - 1) & \text{if } q \neq -1, \\ 2\sigma^{k-1} \ln|\sigma| & \text{if } q = -1, \end{cases} \tag{4.2}$$

and, **For** $k = p - q < 1$,

$$u^p(\sigma) = \begin{cases} \dfrac{\sigma^{k-1}}{q+1}|\sigma|^{2(q+1)} & \text{if } q > -1, \\ \infty & \text{if } q \leq -1. \end{cases} \tag{4.3}$$

Proof. We prove this through the use of the algorithm described in §3.

Step 1. Since $\zeta = \rho e^{i\theta}$, we have from (4.1),

$$f_n = \begin{cases} \rho^{p+q} & \text{if } n = k, \\ 0 & \text{if } n \neq k. \end{cases} \tag{4.4}$$

Step 2. From (2.8) and (4.4)

$$c_n^{ij} = 0 \quad \text{for} \quad n \neq k - 1. \tag{4.5}$$

The nonzero coefficient c_{k-1}^{ij} is evaluated as follows according to the algorithm.

$$c_{k-1}^{i,i+1} = 2R^{k-1} \int_{r_i}^{r_{i+1}} \rho^{p+q+k-1} d\rho. \tag{4.6}$$

Since $k = p - q$, upon integration, (4.6) becomes

$$c_{k-1}^{i,i+1} = 2R^{k-1} \begin{cases} \dfrac{1}{2(q+1)} \left(r_{i+1}^{2(q+1)} - r_i^{2(q+1)}\right) & \text{if } q \neq -1, \\ \ln(r_{i+1}/r_i) & \text{if } q = -1. \end{cases} \tag{4.7}$$

Step 3. It follows from (2.10), (2.11), (4.5), and Corollary 2.2 that

$$c_n(r_j) = 0 \quad \text{for } n \neq k - 1, \quad \text{and} \quad j = 1, 2, \cdots, M. \tag{4.8}$$

The calculation of the coefficient $c_{k-1}(r_j), j = 1, 2, \cdots, M$, according to our algorithm, follows. There are two separate cases to be considered depending on whether $q = -1$ or $q \neq -1$.

Case 1. $q = -1$.

(i) **For** $k < 1$, upon using (4.7) in (2.12) we obtain, after some manipulation,

$$c_{k-1}(r_l) = 2r_\ell^{k-1} \sum_{i=2}^{\ell} \ln\left(\frac{r_i}{r_{i-1}}\right) = \infty, \tag{4.9}$$

where $0 \leq r_\ell \leq 1$. To arrive at the last equality in (4.9) we have used

$$\sum_{i=2}^{\ell} \ln\left(\frac{r_i}{r_{i-1}}\right) = \sum_{i=2}^{\ell} (\ln r_i - \ln r_{i-1}) = \ln r_\ell - \ln 0 = \infty. \tag{4.10}$$

(ii) **For** $k \geq 1$, upon using (4.7) in (2.12) we obtain, after some manipulation,

$$\begin{aligned} c_{k-1}(r_\ell) &= -2r_\ell^{k-1} \sum_{i=\ell}^{m-1} \ln\left(\frac{r_{i+1}}{r_i}\right) \\ &= 2r_\ell^{k-1} \ln(r_\ell), \end{aligned} \tag{4.11}$$

where $0 \leq r_\ell \leq 1$. To arrive at the last equality in (4.11) we have used

$$\begin{aligned} \sum_{i=\ell}^{m-1} \ln\left(\frac{r_{i+1}}{r_i}\right) &= (\ln(r_m) - \ln(r_{m-1})) + (\ln(r_{m-1}) - \ln(r_{m-2})) \\ &\quad + \cdots + (\ln r_{\ell+1} - \ln r_\ell) \\ &= \ln 1 - \ln r_\ell = -\ln r_\ell. \end{aligned} \tag{4.12}$$

Case 2. $q \neq -1$.

(i) **For** $k < 1$, upon using (4.7) in (2.12) we obtain, after some manipulation,

$$\begin{aligned} c_{k-1}(r_\ell) &= \frac{r_\ell^{k-1}}{q+1} \sum_{i=2}^{\ell} (r_i^{2(q+1)} - r_{i-1}^{2(q+1)}) \\ &= \frac{r_\ell^{k-1}}{q+1} (r_\ell^{2(q+1)} - r_1^{2(q+1)}) \\ &= \begin{cases} \frac{r_\ell^{k-1}}{q+1} r_\ell^{2(q+1)} & \text{if } q > -1, \\ \infty & \text{if } q < -1, \end{cases} \end{aligned} \tag{4.13}$$

where we have used

$$\sum_{i=2}^{\ell} (r_i^{2(q+1)} - r_{i-1}^{2(q+1)}) = r_\ell^{2(q+1)} - r_1^{2(q+1)}. \tag{4.14}$$

(ii) **For** $k \geq 1$, upon using (4.7) in (2.12) we obtain after some manipulation

$$\begin{aligned} c_{k-1}(r_\ell) &= -\frac{r_\ell^{k-1}}{q+1} \sum_{i=m-1}^{\ell} (r_{i+1}^{2(q+1)} - r_i^{2(q+1)}) \\ &= \frac{r_\ell^{k-1}}{q+1} (r_\ell^{2(q+1)} - 1), \end{aligned} \tag{4.15}$$

where $0 \leq r_\ell \leq 1$. To obtain the last equality in (4.15) we have used

$$\sum_{i=\ell}^{m-1} (r_{i+1}^{2(q+1)} - r_i^{2(q+1)}) = r_m^{2(q+1)} - r_\ell^{2(q+1)} = 1 - r_\ell^{2(q+1)}. \tag{4.16}$$

We can summarize the calculations of Step 3 by rewriting (4.9) and (4.13) into

(i) **For** $k < 1$,

$$c_{k-1}(r_\ell) = \begin{cases} \dfrac{r_l^{k-1}}{q+1} r_l^{2(q+1)} & \text{if } q > -1, \\ \infty & \text{if } q \leq -1, \end{cases} \tag{4.17}$$

and by rewriting (4.11) and (4.15) into

(ii) **For** $k \geq 1$

$$c_{k-1}(r_\ell) = \begin{cases} \dfrac{r_\ell^{k-1}}{q+1} (r_\ell^{2(q+1)} - 1), & q \neq -1, \\ 2r_\ell^{k-1} \ln(r_\ell), & q = -1. \end{cases} \tag{4.18}$$

Step 4. Using the Fourier coefficients given by (4.8), (4.9), (4.11), (4.13), and (4.15) in the Fourier series (2.1), we obtain our results (4.2) and (4.3).

Remark 4.1. The above method of integral evaluation is meant only to show the algorithmic steps explicitly. There are easier *analytical* methods to evaluate the above integral. For example, it is much easier to evaluate the coefficient c_{k-1} directly from (2.7).

Remark 4.2. We note that $u^p(\sigma)$ in the above example can be determined directly to within an arbitrary additive analytic function by simply integrating (4.1) with respect to the conjugate of the complex coordinate.

Remark 4.3. The above calculation is exact. In actual implementation, the errors will arise from finite truncation of Fourier series and approximate evaluation of the one-dimensional integrals. However, in this particular example there is only one mode, namely, $(p-q)$th mode, in $f(\sigma)$ (see (4.1)). Therefore, the error due to discrete Fourier approximation in this example is zero, provided this mode is included in the truncated Fourier series, i.e., $N > p$. In actual implementation, the errors due to numerical evaluation of the integrals depend on the method of integration and the values of p and q. In our implementation, we use trapezoidal rule and thus the error in Step 3 of the algorithm can be made zero if q is chosen to be zero in (4.1) (regardless of how many points are chosen in the radial integration).

The following two remarks have to do with the divergence of the double integral (1.7) for some specific choices for $f(\sigma)$. For the specific functional form (4.1) for $f(\sigma)$, the divergence of the double integral depends on the values of p and q.

Remark 4.4. From (4.3), it follows that $u^p(\sigma)$ blows up at all values of σ within the unit disk if $p - q - 1 < 0$ and $q \leq -1$, or equivalently, if $p < q + 1 \leq 0$.

Remark 4.5. From (4.2), it follows that $u^p(r = 0)$ blows up if $p - q - 1 \geq 0$ and $q = -1$, or equivalently, if $p \geq q + 1 = 0$.

5. Numerical results I. A computer program has been written which can compute the integral using the fast algorithm of this paper or using the naive method of directly integrating the double integral in [6, Appendix C, eq. (C.4)]. The program has been tested with various functions $f(\sigma)$ in (1.7). However, we present the performance of our fast algorithm using the example of §4. Since the exact value of the integral is available in this case, the relative maximum error in the numerical computation can easily be calculated.

Computations were carried out for different radial and angular grid spacings on a Cray Y-MP at Texas A&M University in single precision. The computations were performed in two ways: (i) using the fast algorithm, and (ii) using the direct method. We compare these two methods by monitoring CPU time and relative maximum error for various values of M and N. We summarize our numerical results below.

Computations were performed for various values of p and q such that $(p-q)$ is an integer. The results obtained with $q=0$ and with $N>p$ using the fast algorithm were accurate up to seven decimal places in single precision regardless of the number of radial grid points. In this calculation, the effect of the number of radial grid points is zero, as it should be according to Remark 4.3. The effect of the number of angular grid points is also zero since this number N has been chosen to be greater than p. Thus, the only error in this case is due to truncation error. However, there were no signs of truncation errors within seven decimal places in this case.

The results of the computations using $p=3$ and $q=1$ in the choice $f(\delta)$ is summarized in Table 1. The number of angular grid points N is kept constant at 17. The first column contains the number of radial grid points. The second and the third columns contain the CPU times T_{fast}, required by the fast algorithm of the present paper, and T_{dir}, required by the direct method, respectively. The fourth and fifth columns contain the maximum relative errors δ_{fast} and δ_{dir} in these two methods, respectively.

TABLE 1

CPU *times in seconds and maximum relative errors on Cray* Y-MP. *The terms within parentheses are approximate estimates.*

M	T_{fast}	T_{dir}	δ_{fast}	δ_{dir}
51	0.00913	1.61807	2.878941E-04	7.7745298E-02
101	0.01662	6.40785	7.1945221E-05	8.438147E-02
151	0.02408	14.3590	3.197717E-05	8.666666E-02
201	0.03157	25.41848	1.7987853E-05	8.78808483E-02
251	0.03987	39.65647	1.1513032E-05	8.88519315E-02
501	0.07786	158.16779	2.8797382E-06	8.9932327E-02
1001	0.15409	(632.67116)	7.2160367E-07	–

Remark 5.1. The fast algorithm of the present paper takes only 0.154 seconds of CPU time when $M=1001$. The CPU time when using the direct method with $M=1001$ is estimated by extrapolation (shown within parentheses in Table 1). It was not considered practical to use approximately 632 seconds of Cray CPU time to produce an exact value of this CPU time.

The following observations can be made about Table 1.

(i) The CPU time required by the fast algorithm increases linearly with M. In contrast, the CPU time required by the direct method increases quadratically with M.

(ii) The relative maximum error δ_{fast} decreases with increasing M. This is because the error in the numerical integration by the trapezoidal method in Step 2 decreases with increasing M.

(iii) The relative maximum error δ_{dir} decreases very slowly with increasing M. In this case, the function $f(\zeta)$ is very poorly resolved by only 17 points in the angular direction. Most of the error is probably due to this poor resolution.

(iv) The accuracy of the fast algorithm is remarkable. In contrast, the direct method has very poor accuracy.

The results of similar computations using $M = 101$ are shown in Table 2 for varying values of N. The first column contains the number of angular grid points N.

TABLE 2
CPU *times in seconds and maximum relative errors on Cray* Y-MP. *The terms within parentheses are approximate estimates.*

N	T_{fast}	T_{dir}	δ_{fast}	δ_{dir}
17	0.01662	6.40785	7.1945221E-05	8.438147E-02
33	0.03216	24.32479	7.1974139E-05	2.592902E-02
65	0.06680	93.81604	7.1972282E-05	6.968137E-03
129	0.13908	376.73983	7.1977357E-05	1.576032E-03
257	0.30031	(1500.00)	7.1972305E-05	6.520931E-04
513	0.64261	(6000.00)	7.1977943E-05	4.129121E-04
1025	1.36825	(24000.00)	7.1972308E-05	2.152631E-04

Remark 5.2. The Cray Y-MP CPU seconds shown within parentheses are approximate and were estimated from computations on the local MIPS computer. The corresponding errors were obtained on the local MIPS computer. The errors on Cray Y-MP and on the local MIPS computer for the same problem agree up to five decimal places.

The following observations can be made about Table 2.

(i) The CPU time required by the fast algorithm increases superlinearly with N and is less than the theoretical asymptotic estimate of $N \ln N$. In contrast, the CPU time required by the direct method increases quadratically with N.

(ii) The relative maximum error δ_{fast} does not change with changing N. This is expected since the values of N used are greater than $(p - q) = 2$ (see Remark 4.3).

(iii) The relative maximum error δ_{dir} decreases with increasing N. However, the accuracy of the fast algorithm is much better.

6. Dirichlet problems for nonhomogeneous Cauchy–Riemann equations. This section shows the application of our fast algorithm of §3 on a Dirichlet linear boundary value problem. Computations of nonlinear and other types of boundary value problems associated with (1.5) using our fast algorithm are under way and will be addressed elsewhere in detail. This section has been kept as brief as possible. We consider solving the following Dirichlet problem in the interior of a unit disk.

$$\text{(P)} \qquad \begin{cases} u_{\bar{\sigma}} = f(\sigma, \bar{\sigma}), & |\sigma| \leq 1, \\ \text{Real } [u(\sigma = e^{i\alpha})] = u_0(\alpha), & 0 \leq \alpha \leq 2\pi, \\ \text{Imag } [u(\sigma = 0)] = v_0. \end{cases}$$

We are interested in finding the solution of (P) in the entire domain.

It follows from the solution (1.6) of (1.5) that the above problem is equivalent to solving the following problem:

$$\text{(RP)}\qquad \begin{cases} u^a_{\bar\sigma} = 0, & |\sigma| \le 1, \\ \text{Real } [u^a(\sigma = e^{i\alpha})] = u_0(\alpha) - u^p(\alpha), & 0 \le \alpha \le 2\pi, \\ \text{Imag } [u^a(\sigma = 0)] = v_0 - u^p(\sigma = 0). \end{cases}$$

Once this problem has been solved, the solution of the problem (P) is constructed from (1.6). Therefore our method of solving (P) involves the following three steps:

(1) First, find the particular solution $u^p(\sigma)$ in the entire domain using the algorithm of the previous section. As shown previously, the algorithmic complexity of this stage of the computation is at worst $MN \ln N$ with MN grid points in the discretization of the unit disk.

(2) Second, construct the analytic function $u^a(\sigma)$ by solving the reduced problem (RP).

(3) Third, construct the solution of the original problem (P) by adding the above two solutions according to (1.6).

Note that the algorithmic complexity in solving the reduced problem (RP) must not exceed $MN \ln N$ so that the computational complexity of the entire calculation remains the same. There are many efficient algorithms that can be used to solve this problem. We use the following simple procedure in constructing the solution of the reduced problem (RP) at MN lattice points.

(i) The solution is expressed in terms of Taylor series

$$u^a(\sigma) = \sum_{n=0}^{\infty} c_n \sigma^n. \tag{6.1}$$

(ii) The FFT is used to construct the complex conjugate of the specified boundary values and the Fourier coefficients of the Taylor series. This step takes $N \ln N$ operations with N points on the boundary of the unit disk.

(iii) Next, the Fourier coefficients are used in constructing the solution $u^a(\sigma)$ at all interior points by using the FFT. This step takes $MN \ln N$ operations with M divisions in the radial direction.

Note that this approach has an asymptotic operation count of the order $MN \ln N$. Thus overall complexity is retained at the desired level.

Remark 6.1. More efficient algorithms can be constructed to solve the reduced problem (RP). We will implement better algorithms for this part of the calculation in the future.

6.1. Numerical results II. The following problems were numerically solved using the above steps.

Example 1.

$$\text{(E1)}\qquad \begin{cases} u_{\bar\sigma} = 1, & |\sigma| \le 1, \\ \text{Real } [u(\sigma = e^{i\alpha})] = 2\cos\alpha, & 0 \le \alpha \le 2\pi, \\ \text{Imag } [u(\sigma = 0)] = 0. \end{cases}$$

The exact solution to this problem is

$$u(\sigma) = \sigma + \bar\sigma, \qquad |\sigma| \le 1. \tag{6.2}$$

Example 2.

$$\text{(E2)} \qquad \begin{cases} u_{\bar{\sigma}} = -1, & |\sigma| \le 1, \\ \text{Real } [u(\sigma = e^{i\alpha})] = 0, & 0 \le \alpha \le 2\pi, \\ \text{Imag } [u(\sigma = 0)] = 0. & \end{cases}$$

The exact solution to this problem is

$$u(\sigma) = \sigma - \bar{\sigma}, \qquad |\sigma| \le 1. \tag{6.3}$$

Example 3.

$$\text{(E3)} \qquad \begin{cases} u_{\bar{\sigma}} = \sigma^k, & |\sigma| \le 1, \\ \text{Real } [u(\sigma = e^{i\alpha})] = \cos\alpha, & 0 \le \alpha \le 2\pi, \\ \text{Imag } [u(\sigma = 0)] = 0, & \end{cases}$$

where k is either 1 or 2. The exact solution to this problem is

$$u(\sigma) = \sigma + \bar{\sigma}\sigma^k - \sigma^{k-1}, \qquad |\sigma| \le 1. \tag{6.4}$$

The above three problems were numerically solved using our fast algorithm to obtain the solution at all $M \times N$ nodes with N=17 and M=101. The computed solutions were compared with the exact solutions for accuracy and the CPU times were recorded. These are shown in Table 3. The data for (E3) corresponds to $k = 2$ (see (E3) above). The first column contains the reference to one of the problems mentioned above. The second column contains the total CPU seconds (T_{tot}) required to solve these problems. The third and the fourth columns contain the breakup of T_{tot} into T_{h} (CPU seconds required to find $u^a(\sigma)$ at all interior points from the Taylor series (6.1)) and T_{nh} (CPU seconds required to find $u^p(\sigma)$ by evaluating the double integral using our algorithm of §3). The fifth column contains the maximum relative error δ.

TABLE 3
Computational results on boundary value problems (E1), (E2), *and* (E3) *using the fast algorithm.*

Examples	T_{tot}	T_{h}	T_{nh}	δ
E1	0.00932	0.00251	0.00681	5.6915101E-06
E2	0.00969	0.00269	0.00700	8.1740268E-06
E3	0.00932	0.00251	0.00681	7.3560102E-06

In Table 3, we note that our fast algorithm solves the boundary value problems within the entire unit disk with very good accuracy and within a fraction of a CPU second. The CPU seconds for computing the analytic part (T_{h}) is less than half of the CPU seconds T_{nh} needed to evaluate the particular solution $u^p(\sigma)$ at all interior points. Similar computations have been done on other types of boundary value problems with similar conclusions.

Remark 6.2. The computation of the analytic part can be accelarated by borrowing some of the ideas from [9], [12], and [13], which will be undertaken in the future.

7. Conclusions. The present work develops a fast algorithm to solve nonhomogeneous Cauchy–Riemann equations in the interior of a unit disk in the complex plane. It is based on computation of the particular solution from its Fourier coefficients. The recursive relations satisfied by these Fourier coefficients are derived, which is at the heart of the algorithm. The speedup provided by the algorithm is dramatic even for a moderate number of nodes in the domain. Our numerical experiments show that the actual CPU time requirements for the algorithm are even less than the asymptotic CPU estimate, which is very encouraging.

The algorithm has the limitation that the problem has to be solved within the interior of a unit disk. Therefore to construct a solution in a domain which is not circular, a conformal mapping of the domain to the interior of a unit disk must be done prior to the use of our algorithm. This should not be difficult, as there are many numerical methods these days to perform such conformal mapping (see [10] and [12]).

Prototype linear boundary problems have been solved here using the fast algorithm to provide some future directions. Nonlinear boundary value problems can be solved using this algorithm iteratively. Computations of nonlinear and other types of boundary value problems associated with the NCR equation using our fast algorithm are currently under way. The application of the algorithm is not limited to any particular field, thus its application to compressible fluid problems should be straightforward, since the compressible fluid flow equations already admit a formulation similar to (1.5) (see [6]). In fact, many compressible flow problems, including those solved in Woods [14] with the tangent gas approximation, can now be solved exactly, accurately, and very efficiently using the algorithm of this paper.

The algorithm presented here is suitable for implementation on a serial computer. However, the recursive set of equations of §2 has a structure suitable for implementation on a parallel computer, which will yield considerable savings in computational time depending on the number of processors. Construction of algorithms suitable for implementation on a parallel computer needs further work. Some of the ideas presented by Katzenelson [11] in connection with computational structure involving recursive relations may be useful here.

Appendix A. Application of the results of §2 in double integral evaluation. The results obtained in §2 provide an analytical technique to evaluate the double integral (1.8) with complicated $f(\zeta)$, which are otherwise difficult to evaluate analytically. We provide a prototype example.

Consider the following choice for the function $f(\zeta)$:

$$f(\zeta) = \zeta^p \sin\left(\frac{\zeta}{|\zeta|}\right) \quad \text{for} \quad p \geq 0. \tag{A.1}$$

Then the Fourier coefficients $f_n(r)$ are given by

$$f_n(r) = \frac{r^p}{2\pi}\int_0^{2\pi} e^{i(p-n)\theta}\sin(e^{i\theta})d\theta. \tag{A.2}$$

Using $\sigma = e^{i\theta}$, (A.2) reduces to

$$f_n(r) = \frac{r^p}{2\pi i}\oint \sigma^{p-n-1}\sin\sigma d\sigma. \tag{A.3}$$

Using the residues, we have from (A.3),

$$f_n(r) = r^p E(n-p), \tag{A.4}$$

where

$$E(n-p)=\begin{cases}0 & \text{if } (n-p)\le 0 \text{ or } (n-p)=\text{even integer},\\ \dfrac{1}{(n-p)!} & \text{if } (n-p)=1,5,9,\cdots,\\ \dfrac{-1}{(n-p)!} & \text{if } (n-p)=3,7,11,\cdots.\end{cases} \tag{A.5}$$

Using (A.4) in (2.8) we obtain

$$c_{n-1}^{i,i+1}=\begin{cases}0 & \text{for } n\le p,\\ \dfrac{2E(n-p)}{(n+p)}R^{n+1}(r_{i+1}^{n+p}-r_i^{n+p}) & \text{for } n>p.\end{cases} \tag{A.6}$$

It then follows from (2.12) that

$$c_{n-1}(r_j)=0 \quad \text{for} \quad n\le p, \text{ and } \quad j=1,2,\cdots,M. \tag{A.7}$$

For $n>p$, upon using (A.6) in (2.12) we obtain after some manipulation,

$$c_{n-1}(r_l)=\frac{2E(n-p)}{n+p}r_\ell^{n-1}(r_\ell^{n+p}-1). \tag{A.8}$$

Using (A.7), (A.8), and (A.5), we obtain

$$u^p(\sigma)=\sum_{n=p}^{\infty}\frac{2E(n+1-p)}{(n+p+1)!}r_\ell(r_\ell^{n+p+1}-1)e^{in\theta}. \tag{A.9}$$

Using $n-p=m$ and (A.5), we can rewrite (A.9) as

$$\begin{aligned}u^p(\sigma)&=\sum_{m=0}^{\infty}\frac{2}{(4m+1)!(4m+2p+1)}\sigma^{4m+p}(|\sigma|^{4m+2p+1}-1)\\&-\sum_{m=0}^{\infty}\frac{2}{(4m+3)!(4m+2p+3)}\sigma^{4m+p+2}(|\sigma|^{4m+2p+3}-1).\end{aligned} \tag{A.10}$$

Appendix B. Desingularization of $\tau^p=-\frac{1}{\pi}\int\int_\Omega f(\zeta)/(\zeta-\sigma)d\xi d\eta$. The desingularized version of this integral appears in Appendix C [6]. However, the equations (C.8) and (C.9) of that paper were in error and should be corrected, respectively, as (B.2) and (B.3) of this appendix. We prescribe here a much easier procedure than the one in [6] to desingularize this integral.

The above integral can be desingularized in the following manner:

$$\begin{aligned}u^p&=-\frac{1}{\pi}\int\int_\Omega\frac{f(\zeta)}{\zeta-\sigma}d\xi d\eta\\&=-\frac{1}{\pi}\int\int_\Omega\frac{f(\zeta)-f(\sigma)}{\zeta-\sigma}d\xi d\eta-\frac{f(\sigma)}{\pi}\int\int_\Omega\frac{d\xi d\eta}{\zeta-\sigma}.\end{aligned} \tag{B.1}$$

The second integral in (B.1) can be evaluated by using the integral evaluated in §4. With $p=0$ and $q=0$ in (4.1), we obtain the following from (4.3):

$$\int\int_\Omega\frac{d\xi d\eta}{\zeta-\sigma}=-\frac{\pi}{\sigma}|\sigma|^2. \tag{B.2}$$

From (B.1) and (B.2) we have

$$u^p = \bar{\sigma} f(\sigma) - \frac{1}{\pi} \int \int_\Omega \frac{f(\zeta) - f(\sigma)}{\zeta - \sigma} d\xi d\eta. \tag{B.3}$$

This desingularized version of the integral is suitable for numerical computation in the direct method (see also [6]).

Acknowledgments. The allocation of computer resources by the Texas A&M Supercomputer Center is gratefully acknowledged. It is a pleasure to thank Eric Grosse for making some useful suggestions that helped to reduce the length of the paper.

REFERENCES

[1] L. Bers, *Mathematical Aspects of Subsonic and Transonic Gas Dynamics*, John Wiley, New York, 1958.

[2] R. Courant and D. Hilbert, *Methods of Mathematical Physics, Vol.* II, John Wiley, New York, 1961.

[3] P. Daripa and L. Sirovich, *Exact and approximate gas dynamics using the tangent gas*, J. Comput. Phys., 62 (1986a), pp. 400–413.

[4] ———, *An inverse method for subcritical flows*, J. Comput. Phys., 63 (1986b), pp. 311–326.

[5] P. Daripa, *An exact inverse method for subcritical flows*, Quart. Appl. Math., XLVI (1988), pp. 505–526.

[6] ———, *On applications of a complex variable method in compressible flows*, J. Comput. Phys., 88(1990), pp. 337–361.

[7] ———, *On a numerical method for quasi-conformal grid generation*, J. Comput. Phys., 96(1991), pp. 296–307.

[8] P. R. Garabedian, *Supercritical Wing Sections* III, Springer-Verlag, New York, 1977.

[9] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulation*, J. Comput. Phys., 73 (1987), pp. 325–348.

[10] P. Henrici, *Applied and Computational Complex Analysis, Vol.* 3, John Wiley, New York, 1986.

[11] J. Katzenelson, *Computational structure of the N-body problem*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 787–815.

[12] S. T. O'Donnell and V. Rokhlin, *A fast algorithm for the numerical evaluation of conformal mapping*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 475–487.

[13] V. Rokhlin, *Rapid solution of integral equations of classical potential theory,* J. Comput. Phys., 60 (1985), pp. 187–207.

[14] L.C. Woods, *The Theory of Subsonic Plane Flow*, Cambridge University Press, New York, 1961.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 6, pp. 1433–1459, November 1992

010

PARALLEL IMPLEMENTATION OF THE *hp*-VERSION OF THE FINITE ELEMENT METHOD ON A SHARED-MEMORY ARCHITECTURE[*]

I. BABUŠKA[†], H. C. ELMAN[‡], AND K. MARKLEY[§]

Abstract. The costs incurred by an implementation of the *hp*-version of the finite element for solving two-dimensional elliptic partial differential equations on a shared-memory parallel computer are studied. For a collection of benchmark problems, the costs in CPU time of various individual subtasks performed by the finite element solver are systematically examined, including construction of local stiffness matrices, elimination of unknowns associated with element interiors, and global solution on element interfaces by a preconditioned conjugate gradient method. General observations are that the costs of the "naturally" parallel computations associated with local elements are significantly higher than any global computations, so that the latter do not represent a significant bottleneck to parallel efficiency. However, memory conflicts place some limitations on the sizes or number of local problems that can be handled efficiently in parallel.

Key words. finite element, *hp*-version, parallel, shared memory, domain decomposition

AMS(MOS) subject classifications. primary: 65F10, 65N20, 65W05

1. Introduction.

The finite element method is a standard computational tool for solving partial differential equations arising from engineering analysis. Variants include the standard *h*-version, which uses low-order basis functions and achieves accuracy by refining meshes [13]; the *p*-version, which uses a fixed mesh and achieves accuracy by increasing the order of the basis functions; and the *hp*-version, which combines these two approaches. See [7] for a survey and comprehensive list of references on the *p*- and *hp*-versions. For the first and last of these techniques, which divide domains into local elements and compute associated local stiffness operators, a large component of the required computations can be implemented very naturally on parallel architectures. In particular, for the *h*-version, *domain decomposition methods* (see, e.g., [8]–[12], [21], [22], [26], [27], [31]) group collections of elements into subdomains, or *super-elements*. The local stiffness operators associated with super-elements are independent of one another, so that they can be constructed in parallel on separate processors. Similarly, elimination of degrees of freedom internal to super-elements can be performed in parallel. For the *p*- and *hp*-version, we think of the space of high-order basis functions in each element as analogous to the grouping of *h*-elements into super-elements. Then, just as for domain decomposition methods, construction of local operators and partial elimination can be done in a natural way on independent processors. A combination of these points of view, with multiple high-order elements collected into super-elements, is also possible.

[*] Received by the editors December 3, 1990; accepted for publication (in revised form) September 9, 1991.

[†] Department of Mathematics and Institute for Physical Science and Technology, University of Maryland, College Park, Maryland 20742. The work of this author was supported by U.S. Office of Naval Research contract N00014-90-J-1030, and by National Science Foundation grant CCR-88-20979.

[‡] Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland 20742. The work of this author was supported by U.S. Army Research Office grant DAAL-0389-K-0016, and by National Science Foundation grants ASC-8958544 and CCR-8818340. Computer time was provided by the Advanced Computing Research Facility at Argonne National Laboratory.

[§] Department of Mathematics, Rice University, Houston, Texas 77001.

After the fully local computations have been performed, the result is a subproblem with unknowns on super-element interfaces. If an iterative method such as the conjugate gradient method (CG) is used to solve this subproblem, then much of the required computation is also local, so that there is a large amount of natural parallelism. However, these computations entail some interaction across super-element interfaces, and, in addition, CG requires some global computations. Moreover, convergence of such methods is often significantly accelerated by some type of global preconditioner [3], [9]–[12], [21], [22], [31], which may be less natural to implement in parallel. The effects of both super-element interactions and global operations on overall performance on parallel architectures is not well understood.

In this paper, we describe the results of an experimental study of an implementation of the *hp*-version of the finite element method for solving two-dimensional linear elliptic problems on a shared-memory parallel computer. We examined the computational costs of the various subtasks required by the *hp*-method, including:

- construction of local stiffness matrices;
- partial elimination of unknowns associated with purely local elements;
- CG iteration; and
- preconditioning derived from low-order elements.

Our goals were to determine how efficiently such computations can be done on parallel architectures, and what bottlenecks may exist that limit efficiency. Some particular issues considered were:

- the relative costs of the various individual subtasks;
- the effects of global operations, especially preconditioning, on overall performance and parallel efficiency;
- the overhead of using unassembled local matrices to perform the matrix-vector products required by CG; and
- whether there are any limitations associated with the "natural" subdivision of problems based on independent elements.

Our tests were performed on an Alliant FX/8, an eight-processor shared-memory computer with a fast cache memory and vector processors. In addition to examining the general issues of parallel implementation, we also considered the effects of the latter two architectural features. In general, we found that the dominant costs come from the local computations, especially the construction of local stiffness matrices, and that global computations required for fast convergence do not represent a significant bottleneck. However, there are some drawbacks to having local computations that operate on large sets of data, which appear to be architecture-related, derived from inefficient data movement for parallel processing.

An outline of the paper is as follows. In §2, we present the continuous model problem used for experiments, and we describe the *hp*-version of the finite element method used for the discretization. In §3, we give a high level description of the solution algorithm and a detailed description of our implementation. Section 4 contains the main results of the paper, a series of experimental results for a set of benchmark problems. These include overviews of iteration counts and CPU times, as well as several refinements of timing statistics showing where computational efforts are spent, and analyses of the effects of synchronization and vectorization. Finally, in §5, we summarize our observations and discuss their implications for computations on other classes of problems and parallel computers.

2. The model problem and its finite element solution. Consider the model problem

$$-\left(\frac{\partial}{\partial x}a\frac{\partial u}{\partial x}+\frac{\partial}{\partial y}b\frac{\partial u}{\partial y}\right)=f \quad \text{on } \Omega, \tag{1}$$

$$u=g_d \quad \text{on } \Gamma_D, \quad \frac{\partial u}{\partial n_c}=g_n \quad \text{on} \quad \Gamma_N. \tag{2}$$

Here, $\Omega \subset \mathbf{R}^2$ is a bounded domain with piecewise smooth (e.g., polygonal) boundary; $\Gamma = \Gamma_D \cup \Gamma_N$ is the boundary of Ω; a, b, f, g_d and g_n are functions that satisfy the usual conditions guaranteeing existence and uniqueness of the solution; and n_c is the conormal. We are interested in the weak solution of (1)–(2), i.e., $u \in H^1_D(\Omega) \equiv \{u \in H^1(\Omega) \,|\, u = 0 \text{ on } \Gamma_D\}$ such that

$$B(u,v) \equiv \int_\Omega a\frac{\partial u}{\partial x}\frac{\partial v}{\partial x}+b\frac{\partial u}{\partial y}\frac{\partial v}{\partial y}\,dx\,dy=\int_\Omega fv\,dx\,dy+\int_{\Gamma_N} g_n v\,ds \equiv F(v)$$

holds for any $v \in H^1_D(\Omega)$. $H^1(\Omega)$ denotes the usual Sobolev space.

We now give a general description of the *hp*-version of the finite element discretization of (1)–(2). Let $\mathcal{P} = \{\Omega^i\}$ denote a partitioning of Ω into open subdomains such that $\bar{\Omega} = \cup\bar{\Omega}^i$ (where $\bar{\Omega}$ denotes the closure of Ω). Assume that Ω^i is a curvilinear polygon, typically a triangle or quadrilateral. Let

$$\Gamma^i = \cup_{j=1}^{m_i}\Gamma^i_j$$

denote the set of (curved) open sides of Ω^i, and let

$$\Delta^i = \cup_{j=1}^{m_i}\Delta^i_j$$

denote its vertices. Assume that

1. $\bar{\Omega}^i \cap \bar{\Omega}^j = \bar{\Gamma}^i_k = \bar{\Gamma}^j_l$, i.e., $\bar{\Omega}^i$ and $\bar{\Omega}^j$ have common entire sides; or
2. $\bar{\Omega}^i \cap \bar{\Omega}^j = \Delta^i_k = \Delta^j_l$, i.e., $\bar{\Omega}^i$ and $\bar{\Omega}^j$ have one vertex in common; or
3. $\bar{\Omega}^i \cap \bar{\Omega}^j = \emptyset$.

The set $(\cup_{i,j}\Gamma^i_j) \cup (\cup_{i,j}\Delta^i_j)$ will be denoted the *frame* of the partitioning $\mathcal{P}$.

On every $\Omega^i \in \mathcal{P}$, we use a set of linearly independent functions $\Phi^i_j \in H^1(\Omega^i)$, $j = 1, \cdots, \rho_i$, called *shape functions*, which are divided into three categories:

Internal shape functions: $\mathcal{I} \subset \{\Phi^{(\mathcal{I})} \in H^1(\Omega^i) \,|\, \Phi^{(\mathcal{I})} = 0 \text{ on } \Gamma^i\}$.

Side shape functions: $\mathcal{S} \subset \{\Phi^{(\mathcal{S},j)} \in H^1(\Omega^i) \,|\, \Phi^{(\mathcal{S},j)} = 0 \text{ on } \Gamma^i - \Gamma^i_j\}$.

Nodal shape functions: $\mathcal{N} \subset \{\Phi^{(\mathcal{N},j)} \in H^1(\Omega^i) \,|\, \Phi^{(\mathcal{N},j)} = 0 \text{ on } \Delta^i_k, k \neq j\}$.

The following typical examples will be used in the sequel. Let $\mathcal{E} = (-1,1) \times (-1,1)$ be called the *standard element*. The nodes and sides of $\mathcal{E}$ are given by

$$\Delta_1 = (1,-1), \quad \Delta_2 = (1,1), \quad \Delta_3 = (-1,1), \quad \Delta_4 = (-1,-1),$$

$$\Gamma_1 = \{(\xi,\eta) \,|\, \xi = 1, |\eta| < 1\}, \qquad \Gamma_2 = \{(\xi,\eta) \,|\, |\xi| < 1, \eta = 1\},$$

$$\Gamma_3 = \{(\xi,\eta) \,|\, \xi = -1, |\eta| < 1\}, \qquad \Gamma_4 = \{(\xi,\eta) \,|\, |\xi| < 1, \eta = -1\}.$$

(See Fig. 1.) Let

$$\phi_j(\xi) = \sqrt{\frac{2j-1}{2}} \int_{-1}^{\xi} P_{j-1}(t)dt, \qquad j \geq 2, \tag{3}$$

where $P_j(t)$ is the Legendre polynomial of degree j [14]. Thus $\phi_j(\xi)$ is a polynomial of degree j and $\phi_j(\pm 1) = 0$. Two spaces, $\mathcal{Q}(p)$ and $\mathcal{Q}'(p)$, are defined as the span of the following shape functions on $\mathcal{E}$:

Internal shape functions. For $\mathcal{Q}(p)$,

$$\Phi_{jk}^{(\mathcal{I})}(\xi,\eta) = \phi_j(\xi)\phi_k(\eta), \qquad 2 \leq j,k \leq p;$$

and for $\mathcal{Q}'(p)$,

$$\Phi_{jk}^{(\mathcal{I})}(\xi,\eta) = \phi_j(\xi)\phi_k(\eta), \quad j,k \geq 2, \quad j+k \leq p.$$

Hence for $\mathcal{Q}(p)$, there are $(p-1)^2$ internal shape functions, and for $\mathcal{Q}'(p)$ there are $(p-2)(p-3)/2$ internal shape functions when $p \geq 4$, and none when $p < 4$.

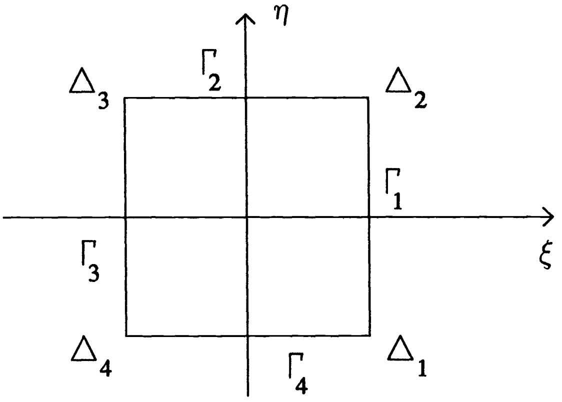


FIG. 1. *The standard element* $\mathcal{E}$.

Side shape functions. For both $\mathcal{Q}(p)$ and $\mathcal{Q}'(p)$, the side shape functions are given by

$$\begin{aligned} \Phi_j^{(\mathcal{S},1)}(\xi,\eta) &= \left(\frac{\xi+1}{2}\right)\phi_j(\eta), & \Phi_j^{(\mathcal{S},2)}(\xi,\eta) &= \phi_j(\xi)\left(\frac{\eta+1}{2}\right), \\ \Phi_j^{(\mathcal{S},3)}(\xi,\eta) &= \left(\frac{-\xi+1}{2}\right)\phi_j(\eta), & \Phi_j^{(\mathcal{S},4)}(\xi,\eta) &= \phi_j(\xi)\left(\frac{-\eta+1}{2}\right), \end{aligned} \qquad j = 2,\cdots,p.$$

Thus, there is a total of $4(p-1)$ side shape functions.

Nodal shape functions. For both $\mathcal{Q}(p)$ and $\mathcal{Q}'(p)$, the four nodal shape functions are given by

$$\begin{aligned} \Phi^{(\mathcal{N},1)}(\xi,\eta) &= \left(\frac{\xi+1}{2}\right)\left(\frac{-\eta+1}{2}\right), & \Phi^{(\mathcal{N},2)}(\xi,\eta) &= \left(\frac{\xi+1}{2}\right)\left(\frac{\eta+1}{2}\right), \\ \Phi^{(\mathcal{N},3)}(\xi,\eta) &= \left(\frac{-\xi+1}{2}\right)\left(\frac{\eta+1}{2}\right), & \Phi^{(\mathcal{N},4)}(\xi,\eta) &= \left(\frac{-\xi+1}{2}\right)\left(\frac{-\eta+1}{2}\right). \end{aligned}$$

Here, $\mathcal{Q}(p)$ contains all polynomials of degree p in each variable, and $\mathcal{Q}'(p)$ contains all polynomials of total degree p. For some choices of p, internal or side shape functions are not present, and the method can reduce to the standard h-version. Also, the spaces $\mathcal{Q}(p)$ and $\mathcal{Q}'(p)$ could be defined as the span of some other shape functions. For example, $\mathcal{Q}(p)$ is the span of all functions of the form $f_j(\xi)f_k(\eta)$ where $\{f_j(\xi)\}$ are Lagrange polynomials with interpolation points chosen to be the $(p+1)$ Gauss–Lobatto quadrature points [28]. Similar sets of standard shape functions can be defined on a triangular element.

Assume that $T_i(\xi,\eta)$ is a mapping of the standard element $\mathcal{E}$ onto Ω^i. Let $\mathcal{Q}^*$ denote either $\mathcal{Q}(p)$ or $\mathcal{Q}'(p)$, and let

$$\mathcal{V} = \{u \in H^1_D(\Omega) \,|\, u|_{\Omega^i}(x,y) = v(T_i^{-1}(x,y)),\ v \in \mathcal{Q}^*\}.$$

We impose on T_i the usual conditions of the finite element method, e.g., if Ω^i is a parallelogram, then T_i is a linear mapping. Thus the basis functions of $\mathcal{V}$ can easily be constructed using the three categories of standard shape functions. The finite element solution $u_{FE} \in \mathcal{V}$ is defined by

$$B(u_{FE},v) = F(v) \quad \text{for all } v \in \mathcal{V}. \tag{4}$$

Condition (4) uniquely defines u_{FE} except when $\Gamma_D = \emptyset$, in which case u_{FE} is determined up to a constant.

Accuracy of u_{FE} is achieved either by increasing the degree p of the shape functions, or by refining the partitioning $\mathcal{P}$. Consider the case where $\mathcal{P}$ partitions a rectangular domain Ω into an $m \times n$ rectangular grid composed of squares with side h. The following results contain typical error bounds for the finite element solution in the energy norm

$$\|u - u_{FE}\|_E \equiv B(u-u_{FE}, u-u_{FE})^{1/2} = \inf_{v\in\mathcal{V}} B(u-v,u-v)^{1/2}.$$

See [7] and references therein for further details.

THEOREM. (1) *Suppose h is such that $c_1h^{-1} \le m,\ n \le c_2h^{-1}$, and $u \in H^k(\Omega)$. Then*

$$\|u-u_{FE}\|_E \le C\frac{h^\alpha}{p^{k-1}}\|u\|_{H^k},$$

where $\alpha = \min\{k-1,p\}$ and C depends on k, c_1, c_2, and the discretization ($\mathcal{Q}(p)$ or $\mathcal{Q}'(p)$), but is independent of u, h, and p.

(2) *If u is analytic in $\bar{\Omega}$, then for any fixed $h > 0$,*

$$\|u-u_{FE}\|_E \le De^{-\beta p},$$

where D depends on u and h and $\beta > 0$ depends on the region in which u is analytic.

Thus for very smooth problems, the p and hp finite element solutions display exponential convergence.

For our investigation, we will restrict our attention to the case where $\Omega = (-1,1)\times(-1,1)$, $\mathcal{P}$ partitions Ω into a uniform $n\times n$ grid, T_i is a bilinear mapping, and $f=0$, $\Gamma_D = \emptyset$. We will consider the constant coefficient case $a = b = 1$. To ensure a unique solution, we will constrain the solution at the corners of $\partial\Omega$.

3. The solution algorithm and its implementation.

Formal specification of the finite element solution u_{FE} as a linear combination of the basis functions of $\mathcal{V}$ leads to a system of linear equations $S\alpha = s$, where S is the global stiffness matrix and α is the vector of coefficients of the basis functions. In this section, we present the algorithm used to solve this problem and describe the details of our implementation.

3.1. The solution algorithm.

Conceptually, the algorithm can be divided into four steps.

1. *Construction of the local stiffness matrices.* The global matrix has the form

$$S = \sum_i S_i,$$

where S_i is the local stiffness matrix associated with Ω^i. Formally, S_i is a large, sparse matrix with nonzero entries determined from shape functions in Ω^i. In the following, S_i will also be identified with the local matrix of order ρ_i given by the Gramm matrix $[B_{\Omega^i}(u,v)]$, where

$$B_{\Omega^i}(u,v) = \int_{\Omega^i} a\frac{\partial u}{\partial x}\frac{\partial v}{\partial x} + b\frac{\partial u}{\partial y}\frac{\partial v}{\partial y}\, dx\, dy, \tag{5}$$

and u and v range over all shape functions in Ω^i. The local contribution s_i to s is determined similarly from $\{F(v)\}$. For our study, we do not form S or s explicitly, but work directly with the local versions of S_i and s_i.

2. *Condensation of the local stiffness matrices.* The local stiffness matrices and right-hand sides have the form

$$S_i = \begin{bmatrix} A_i & B_i \\ B_i^T & C_i \end{bmatrix}, \qquad s_i = \begin{pmatrix} b_i \\ c_i \end{pmatrix}. \tag{6}$$

A_i corresponds to interactions among internal shape functions, C_i corresponds to interactions among side and nodal shape functions, and B_i corresponds to interactions between internal shape functions and side and nodal shape functions. Before solving for the unknowns associated with the frame of the partitioning $\mathcal{P}$, the unknowns associated with the interior Ω^i can be decoupled from the system. This process of *condensation*, or elimination of internal unknowns, entails computing the Schur complement

$$\tilde{C}_i = C_i - B_i^T A_i^{-1} B_i, \tag{7}$$

and modifying the right-hand side in a similar manner:

$$\tilde{c}_i = c_i - B_i^T A_i^{-1} b_i. \tag{8}$$

We will also normalize these quantities so that all local diagonal matrix entries are one, i.e.,

$$\hat{C}_i = D_i^{-1/2}\tilde{C}_i D_i^{-1/2}, \qquad \hat{c}_i = D_i^{-1/2}\tilde{c}_i, \tag{9}$$

where $D_i = \operatorname{diag}(\tilde{C}_i)$. This is equivalent to scaling the shape functions.

3. *Computation of the frame unknowns.* After the internal unknowns are eliminated, the result is a system of linear equations

$$\hat{S}\hat{\alpha} = \hat{s} \tag{10}$$

for the unknowns associated with the frame of $\mathcal{P}$. Here

$$\hat{S} = \sum_i \hat{C}_i, \qquad \hat{s} = \sum_i \hat{c}_i,$$

and $\hat{C}_i$ and $\hat{c}_i$ are determined from (9). We solve (10) using the preconditioned conjugate gradient method (PCG) [25]. For the preconditioner, we use the submatrix of $\hat{S}$ associated with *nodal* unknowns. That is, if the entries of $\hat{S}$ are arranged in the form

$$\begin{bmatrix} R & U \\ U^T & Q \end{bmatrix},$$

where Q corresponds to connections among nodal unknowns and R corresponds to connections among side unknowns, then the preconditioner for (10) is given by

$$\begin{bmatrix} I & 0 \\ 0 & Q \end{bmatrix}.$$

It is shown in [3] that the condition number of the resulting preconditioned matrix is

$$O(1 + (\log p)^2). \tag{11}$$

Hence the number of PCG iterations required for convergence is independent of h, and it grows very slowly as a function of p.

4. *Computation of the internal unknowns.* After the unknowns $\hat{\alpha}_i$ associated with the boundary Γ_i have been computed at step 3, the internal unknowns α_i associated with Ω^i can be computed by solving the system $A_i\alpha_i = b_i - B_i\hat{\alpha}_i$, using the Cholesky factorization of A_i computed in step 2.

3.2. Local stiffness matrix computations. Assume that the finite element discretization is made on an $n \times n$ element grid, and we have a parallel architecture with k processors where k divides n^2. The first two steps of the solution algorithm, construction of the local stiffness matrices and condensation, are naturally parallelizable. There are n^2 elements, so that there are n^2 local stiffness matrices to be constructed, each of which contains a subblock corresponding to a set of internal unknowns to be eliminated. All of these computations are obviously independent, so that they can be executed in parallel. Each processor has n^2/k elements assigned to it, for which it constructs the associated local stiffness matrices and then performs the corresponding condensation.

Consider the construction of the local stiffness matrices. The entries of S_i (6) are determined using (5). For general operators and domains, these entries must be computed by some quadrature rule that depends on the coefficients a and b, the shape of Ω^i, and the mapping T_i from $\mathcal{E}$ to Ω^i. In general, the resulting local stiffness matrix S_i is dense, although its nonzero structure may also be affected by these criteria. For rectangular elements, the shape functions specified in §2 have the form $\theta_j(x)\theta_k(y)$ where θ_j is a polynomial of degree j. Therefore, (5) simplifies to an expression of the form

$$B_{\Omega^i}(u,v) = \int\int a\theta_j'(x)\theta_l'(x)\theta_k(y)\theta_m(y) + b\theta_j(x)\theta_l(x)\theta_k'(y)\theta_m'(y)\,dxdy. \tag{12}$$

In this study, we are restricting our attention to the case where a and b are constant. For these problems, (12) further simplifies to

$$B_{\Omega^i}(u,v) = a\,I_x(j,k,l,m) + b\,I_y(j,k,l,m), \tag{13}$$

where

$$I_x(j,k,l,m) = I_1(j,l)I_0(k,m), \qquad I_y(j,k,l,m) = I_0(j,l)I_1(k,m), \tag{14}$$

and

$$I_0(s,t) = \int \theta_s\theta_t, \qquad I_1(s,t) = \int \theta'_s\theta'_t. \tag{15}$$

This reduces the costs of constructing S_i, since (15) can be computed in closed form, and many entries are zero. In particular, for both the $\mathcal{Q}(p)$ and $\mathcal{Q}'(p)$ discretizations, S_i has order $\rho_i = O(p^4)$, but only $O(p^2)$ entries are nonzero. (See Appendix.) For example, for the $\mathcal{Q}(p)$ discretization, if the rows and columns of S_i are ordered using the lexicographic ordering of shape functions

$$\Phi_{22}^{(\mathcal{I})}, \Phi_{32}^{(\mathcal{I})}, \cdots, \Phi_{p2}^{(\mathcal{I})}, \Phi_{23}^{(\mathcal{I})}, \cdots, \Phi_{pp}^{(\mathcal{I})},$$

$$\Phi_2^{(\mathcal{S},1)}, \cdots, \Phi_p^{(\mathcal{S},1)}, \Phi_2^{(\mathcal{S},2)}, \cdots, \Phi_p^{(\mathcal{S},2)}, \Phi_2^{(\mathcal{S},3)}, \cdots, \Phi_p^{(\mathcal{S},3)}, \Phi_2^{(\mathcal{S},4)}, \cdots, \Phi_p^{(\mathcal{S},4)},$$

$$\Phi^{(\mathcal{N},1)}, \Phi^{(\mathcal{N},2)}, \Phi^{(\mathcal{N},3)}, \Phi^{(\mathcal{N},4)},$$

then the nonzero structure for the $\mathcal{Q}(p)$ discretization is shown in Fig. 2.

In our code for constructing the local matrices, each entry of S_i is computed using (13)–(15), where the indices j, k, l, and m range over all values corresponding to the lower triangle of S_i. I_x, I_y, I_0, and I_1 can be thought of as representing FORTRAN functions, and (15) is computed in closed form using (23), (27), (28), and analogues for handling side and nodal shape functions. The routines corresponding to I_0 and I_1 are called only if the result is nonzero, as specified by (22). Thus the construction of S_i entails $O(1)$ scalar computations per entry, giving a total cost of $O(p^4)$. Computation of a zero entry entails several queries about the indices j, k, l, and m and at most three floating point operations; computation of a nonzero entry entails subroutine calls to I_x, I_y, I_0, and I_1, the latter two of which require on the order of 10 scalar floating point operations.

Now consider the other purely local operation, the condensation of the local stiffness matrix, to produce the Schur complement $\tilde{C}_i$ of (7). To perform this step, we compute the Cholesky factorization $A_i = L_iL_i^T$, and then compute $\tilde{B}_i = L_i^{-1}B_i$ and $C_i - \tilde{B}_i^T\tilde{B}_i$. These operations were implemented using (a slightly modified version of) off-the-shelf software from the BLAS2 subroutine library, which is designed to take advantage of vector architectures [17]. A general description of the algorithm is as follows. Assume that in (6), S_i has order ρ and A_i has order $\lambda \le \rho$. For any matrix M with the same dimensions as S_i, let

$$m_{\mu:\gamma,\nu} = (m_{\mu\nu}, m_{\mu+1,\nu}, \cdots, m_{\gamma,\nu})^T$$

denote the column vector consisting of entries μ through γ of the νth column of M, and let

$$M_\mu = \begin{cases} [m_{\sigma\tau}], \ \mu \le \sigma \le \rho, \ \nu \le \tau \le \mu - 1 & \text{if } \mu \le \lambda, \\ [m_{\sigma\tau}], \ \mu \le \sigma \le \rho, \ \nu \le \tau \le \lambda & \text{if } \mu > \lambda. \end{cases}$$

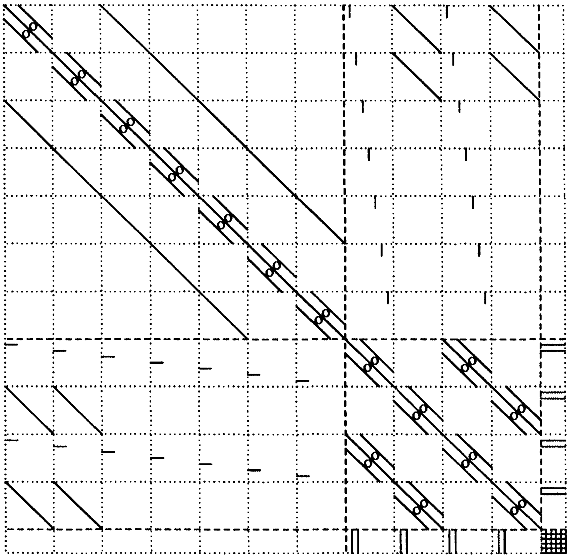


FIG. 2. *The nonzero structure of the local stiffness matrix for the $\mathcal{Q}(p)$ discretization. Solid lines indicate nonzeros; dashed lines delineate the sets $\mathcal{I}, S$, and $\mathcal{N}$; dotted lines delineate blocks of size $p-1$, and "o" identifies a zero band.*

M_μ is a submatrix of M in the lower left-hand corner of M. (See Fig. 3.) In the following code fragment, M initially contains the local matrix S_i of (6); as the computation proceeds, the contents of M are dynamically modified. In steps 1 through λ, the lower triangle of A_i is overwritten by the Cholesky factor L_i, and B_i^T is overwritten by $\tilde{B}_i^T$. In steps $\lambda+1$ through ρ, the lower triangle of C_i is overwritten with that of $\tilde{C}_i$.

(16)

```
for $\mu = 1$ to $\rho$ do
    $\nu_{max} \leftarrow \min\{\mu-1, \lambda\}$
    $m_{\mu:\rho,\mu} \leftarrow m_{\mu:\rho,\mu} - M_\mu m_{1:\nu_{max},\mu}$
    if $(\mu \le \lambda)$ then
        $m_{\mu\mu} \leftarrow \sqrt{m_{\mu\mu}}$
        $m_{\mu+1:\rho,\mu} \leftarrow m_{\mu+1:\rho,\mu}/m_{\mu\mu}$
    endif
enddo
```

Except for $m_{1:\nu_{\max},\mu}$, only the lower triangle of M is referenced. In the program used to implement this computation, only the lower triangle of M is stored, by column in *packed form.* That is, the contents of the array containing M are

$$m_{1,1}, m_{2,1}, \cdots, m_{\rho,1}, m_{2,2}, m_{3,2}, \cdots, m_{\rho,2}, m_{3,3}, \cdots m_{\rho,\rho}.$$

(Since the vector $m_{1:\nu_{\max},\mu}$ is, therefore, not available in contiguous storage, $m^T_{\mu,1:\nu_{\max}}$ is accumulated in a temporary vector at each step of the outer (μ) loop.) The sub-

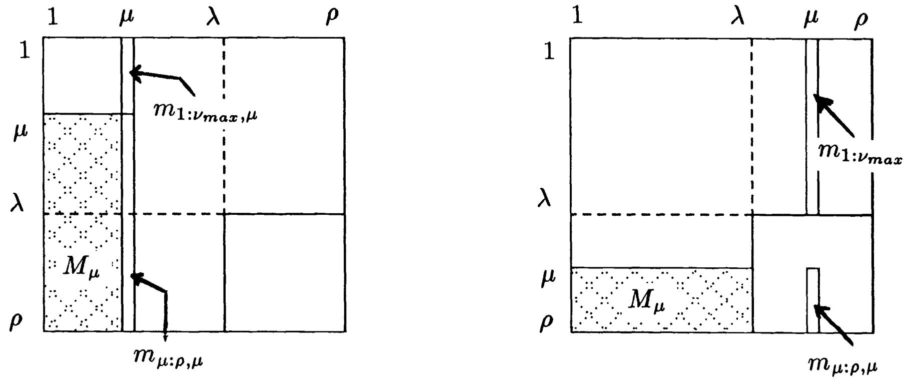


FIG. 3. *Submatrices and subvectors used for internal elimination. Quantities used for a Cholesky factorization step are shown on the left. Quantities used to compute the Schur complement are shown on the right.*

matrices and subvectors of M used in one step of internal elimination are depicted graphically in Fig. 3.

The algorithm (16) is essentially the "GAXPY" form of Gaussian elimination [19], [24], [25], in which the main large-scale operation at each step is a matrix-vector product

$$m_{\mu:\rho,\mu} \leftarrow M_\mu m_{1:\nu_{\max},\mu}. \tag{17}$$

The computations are arranged to take advantage of architectures with vector registers, by computing (17) as a linear combination of the columns of M_i. Our implementation is essentially that of the BLAS2 library [17].[1] In principle, the vector $m_{\mu:\rho,\mu}$ can be accumulated in one or more vector registers without being stored to memory until the computation (17) is complete.

For the $\mathcal{Q}(p)$ and $\mathcal{Q}'(p)$ discretizations, this implementation of the condensation step requires $O(p^4)$ floating point operations. This is determined by the cost of the matrix-vector product (17), and it is also strongly influenced by our choice of implementation. Consider the $\mathcal{Q}(p)$ discretization, where $\lambda = (p-1)^2$ and $\rho = (p+1)^2$. A feature of the BLAS2 software used for (17) is that only the nonzero entries of the vector $m_{1:\nu_{\max},\mu}$ are used for the linear combination of columns of M_μ. There are at most three such nonzeros for steps 1 through λ, and an average of $2.5(p-1)$ nonzeros for steps $\lambda+1$ through ρ. At step μ, the block M_μ contains $(p+1)^2 - (\mu-1)$ rows. Hence the number of floating point multiplications performed is approximately

$$\sum_{\mu=1}^{(p-1)^2} 3[(p+1)^2 - (\mu-1)] + \sum_{\mu=(p-1)^2+1}^{(p+1)^2} 2.5(p-1)[(p+1)^2 - (\mu-1)] \approx 1.5p^4 + 26p^3.$$

These computations are vectorized, but they do not take full advantage of sparsity, since M_μ is treated as a dense matrix. An implementation that operated only on the nonzeros of M_μ would require $O(p^2)$ operations.

[1] We used a slight modification of the subroutine DTPMV from the BLAS2 library. DTPMV computes a matrix-vector product $w \leftarrow Lv$ where L is a lower triangular matrix stored in packed form; our modification allows variations on outer loop counters to handle rectangular subblocks of L.

It is evident from this discussion that many factors contribute to the cost of construction and condensation of the local stiffness matrices. As we have noted, our program takes some advantage of the special structure of the test problem, but it does not make full use of sparsity in either construction or condensation. By way of comparison, consider the situation for more general problems, where the $O(p^4)$ entries of S_i are computed by applying a quadrature rule to (5) or, for rectangular domains, (12). Typically, $O(p)$ quadrature points are used in both the x and y coordinates, so that (ignoring the costs of function evaluations) (5) will require $O(p^6)$ floating point computations. For (12), this cost can be reduced to $O(p^5)$ by taking advantage of the tensor product structure [30]. The condensation is, asymptotically, an $O(p^6)$ computation. Of course, asymptotics also do not tell the whole story, since in general we work with relatively small values of p (on the order of 10); we expect asymptotic characterizations to be pessimistic for these values [4]. Both construction and condensation allow for significant amounts of vectorization, e.g., in the quadratures for construction and as in §3 for condensation. Our implementation is intended to take advantage of special problem structure in a "natural" way: in the matrix construction, by performing some computation for each entry, but not performing unnecessary quadratures; and in the condensation, by handling sparsity only in the manner inherited from standardized software. We believe that this implementation gives a plausible picture of the relative costs of construction and condensation; for more complex problems, absolute costs will be higher.

Step 4 of the solution algorithm, recovery of the internal unknowns, also entails purely local computations. However, because the solutions to our benchmark problems are identically zero, we did not experiment with this stage of the algorithm.

3.3. Computation of the frame unknowns. Next, consider the solution of the global linear system (10) by the preconditioned conjugate gradient method. We use the standard implementation of PCG, as described, for example, in [25, Algorithm 10.3.1]. Each step of the iteration requires a matrix-vector product by the coefficient matrix $\hat{S}$, a preconditioning solve of the form $w \leftarrow Q^{-1}v$, and a set of *vector operations* consisting of three inner products $\alpha \leftarrow v^T w$ and three daxpy's $w \leftarrow \alpha v_1 + v_2$. (This is one more inner product than specified in [25]. The extra one is used for a stopping test; see §4.1.)

The preconditioning operator and vectors required by PCG are represented with global indices; implementation issues associated with these quantities are discussed in §3.4. In this section, we focus on the matrix-vector product, which shares some of the purely local character of the local stiffness matrix computations. The global matrix $\hat{S}$ is not constructed explicitly; instead the matrix-vector product is computed as

$$w = \sum_i w_i = \sum_i \hat{C}_i v_i = \hat{S} v, \tag{18}$$

where the sum is taken over all elements, and $\hat{C}_i$ is the Schur complement associated with an individual element. The vector v_i is gathered from a globally indexed vector v, the local matrix-vector product $w_i = \hat{C}_i v_i$ is performed, and then w_i is used to update the global vector w. Hence the steps required for the local matrix-vector product are

a. *Index and copy*: determine the indices in v corresponding to v_i and copy v_i from v.

b. *Arithmetic*: $w_i = C_i v_i$.

c. *Update*: accumulate w_i into w.

For the parallel implementation, each processor performs this computation on all elements assigned to it. Ignoring any memory conflicts, all processors can read from the global vector v and compute the local matrix-vector product independently, so that steps a and b can be implemented with a high degree of parallelism. However, for the program to be correct, no more than one processor can write into a given location of w at any time. We enforce this by synchronizing all writes into w, so that execution of step c by different processors is performed serially. (See §3.4 for the method used to achieve this.) For the arithmetic step b, recall that only the lower triangle of $\hat{C}_i$ is stored, by column. The actual computation has the form shown in the following code fragment, in which $\gamma = \rho - \lambda$ is the order of $\hat{C}_i$, and for the sake of simplicity, the subscript i is omitted:

(19)

```
for $\mu = 1$ to $\gamma$ do
    $w \leftarrow w + v_\mu \hat{c}_{\mu:\gamma,\mu}$
    $w_\mu \leftarrow w_\mu + (\hat{c}_{\mu+1:\gamma,\mu})^T \hat{c}_{\mu+1:\gamma,\mu}$
enddo
```

That is, multiplication by the lower triangle is done using a linear combination of the columns, and the additional inner product and accumulation for the upper triangular multiplication are performed in the same loop. No extraneous vector writes (of w_i) or reads (of columns of $\hat{C}_i$) need be performed. (Thus this is also essentially a BLAS2-type computation [17].) For our test problems, approximately 50 percent of the entries of $\hat{C}_i$ are nonzero; however, as in the condensation, $\hat{C}_i$ is treated as a dense matrix.

3.4. Other coding conventions. We conclude this section with an outline of our coding conventions. All code was written in Alliant FX/8 FORTRAN and compiled using the global "-O" optimization switch. All vectorizable loops were preceded with the Alliant compiler directives `VECTOR` and (for global inner products) `ASSOC`. Thus all computations are fully vectorized. Although parallelism on the Alliant FX/8 can be achieved using compiler constructs, the compiler does not permit easy control of individual processors, and it also does not permit data-driven synchronization of the type required for the matrix-vector product. To circumvent these difficulties, we implemented all the local computations described above using the scheduling program SCHEDULE [20]. For a local computation such as construction of the local stiffness matrix, SCHEDULE runs on k processors by initiating k processes consisting of n^2/k matrix constructions. This is a relatively simple use of this software, which has the property that any overhead associated with it is amortized over n^2/k large-scale computations. Compiler-generated parallelism is explicitly prevented using Alliant FORTRAN compiler directives (i.e., `NOCONCUR`). Synchronization of the updates required by the matrix-vector products in PCG is enforced using SCHEDULE's *lockon* and *lockoff* primitives, which force processes to spin-wait when access to w is restricted. Finally, to handle SCHEDULE's requirement that multiple copies of subroutines be used simultaneously, all compilation was done with the "-recursive" switch.

Computations not discussed in detail above are the construction and factorization of the preconditioner, and the preconditioning and vector operations (inner products and scalar-vector products) performed during the PCG iteration. All these computations are *global* operations, in the sense that they are concerned with global quantities associated with the nodal and side unknowns. Constructing the preconditioner consists of assembling the (global) nodal operator Q from entries of the local stiffness matrices, and then computing the Cholesky factorization $Q = LL^T$. The factorization was performed using band elimination [25]. The preconditioning operation consists of forward-solves $w \leftarrow L^{-1}v$ and backsolves $w \leftarrow L^{-T}v$. The preconditioning and vector operations all contain a large amount of natural parallelism, but not at the element level. As a result, parallelism for these tasks was handled using Alliant compiler directives (CONCUR).

4. Experimental results. In this section, we present the results of a series of numerical tests of the algorithm of §3. For several problems, we give a general overview of costs, and then we show how these costs are broken down by individual computational tasks. Our objectives are both to show how the methods perform, and to understand what aspects of algorithms and computer architecture affect performance. In particular, we examine the influences of local and global computations required by algorithms, and of architectural considerations such as number of processors, vectorization, and cache memory. We remark that we are not examining the issue of accuracy of the computed solution here. Correlations between accuracy requirements and cost will be discussed in a subsequent report [5].

4.1. Machine independent results. For most results presented in this section, we used benchmark problems with the $\mathcal{Q}(p)$ discretization on $n \times n$ grids, with $p = 4$, 8, and 16, and $n = 4$, 8, 16, and 32. In addition, we have observed [5] that often the $\mathcal{Q}(p)$ and $\mathcal{Q}'(p)$ discretizations provide solutions of comparable accuracy when $p_{\mathcal{Q}} \approx \sqrt{2}p_{\mathcal{Q}'}$, so that we examined the $\mathcal{Q}'(p)$ discretization with $p = 6$, 11, and 23, on the same grids. The choices for degree represent moderate, large, and very large values. The values $p = 16$ for $\mathcal{Q}(p)$ and $p = 23$ for $\mathcal{Q}'(p)$ are larger than values typically used in practice and are studied primarily to see trends in the data; these values were not considered on the 32×32 grid. Tables 1 and 2 show the number of global and local unknowns of various types associated with these problems.[2]

In all experiments, the problems were posed with $s \equiv 0$, so that the solution to (10) is $\hat{\alpha} \equiv 0$. The stopping criterion for the PCG iteration was based on the relative error in the energy norm,

$$(\hat{\alpha}^{(j)}, \hat{S}\hat{\alpha}^{(j)})^{1/2}/(\hat{\alpha}^{(0)}, \hat{S}\hat{\alpha}^{(0)})^{1/2} \leq .5 \times 10^{-3},$$

where $\{\hat{\alpha}^{(j)}\}$ are the PCG iterates and the initial guess $\{\hat{\alpha}^{(0)}\}$ is a vector of random numbers between 0 and 1. Table 3 shows the number of iterations required to reach this stopping criterion. Note that these iteration counts are consistent with condition numbers of the form (11).

4.2. Overview of timing results. We first give a general overview of CPU times needed to solve these benchmark problems. For all experiments, reported times are in seconds, and they represent averages over three runs. The timings were determined from the "user time" returned by the Unix function *etime*; the measurements exclude timing overhead [1]. Speedup is defined to be the ratio of CPU time using one processor to CPU time on multiple processors; the same program was used in all experiments, so that the timings on one processor include a small amount of overhead associated with the scheduler.

[2] For simplicity of programming, we constrained the four vertices of $\partial\Omega$ by adding a constant to the diagonal entries of $\hat{C}_i$ associated with these vertices. This corresponds to constraining the vertices by spring supports. It does not influence any results discussed below.

TABLE 1
Number of global unknowns for benchmark problems.

		$Q(p)$			$Q'(p)$		
		Internal	Frame	Total	Internal	Frame	Total
$p = 4$ (Q) $p = 6$ (Q')	4×4 grid	144	145	289	96	225	321
	8×8 grid	576	513	1089	384	801	1185
	16×16 grid	2304	1921	4225	1536	3009	4545
	32×32 grid	9216	7425	16641	6144	11649	17793
$p = 8$ (Q) $p = 11$ (Q')	4×4 grid	784	305	1089	576	425	1001
	8×8 grid	3136	1089	4225	2304	1521	3825
	16×16 grid	12544	4097	16641	9216	5729	14945
	32×32 grid	50176	15873	66049	36864	22209	59073
$p = 16$ (Q) $p = 23$ (Q')	4×4 grid	3600	625	4225	3360	905	265
	8×8 grid	14400	2241	16641	13440	3249	16689
	16×16 grid	57600	8449	66049	53760	12257	66017

TABLE 2
Number of unknowns in each element for benchmark problems.

	$Q(p)$			$Q'(p)$			
	Internal	Frame	Total		Internal	Frame	Total
$p = 4$	9	16	25	$p = 6$	6	24	30
$p = 8$	49	32	81	$p = 11$	36	44	80
$p = 16$	225	64	289	$p = 23$	210	92	302

TABLE 3
Iteration counts for benchmark problems.

	$Q(p)$			$Q'(p)$		
	$p = 4$	$p = 8$	$p = 16$	$p = 6$	$p = 11$	$p = 23$
4×4	16	20	27	13	17	24
8×8	15	19	26	13	17	20
16×16	14	18	23	13	16	20
32×32	14	18	–	12	16	–

Tables 4 and 5 show timing statistics and speedups for the $Q(p)$ and $Q'(p)$ discretizations, respectively, for the entire solution procedure. Table 6 shows the efficiencies of these computations, defined to be the ratio of speedup to number of processors. (Table 6 was computed with raw data, so there are some differences between these numbers and those obtainable from the rounded quantities shown in Tables 4 and 5.) These results show that, in general, speedups are higher for both larger values of p and for larger grid sizes. For the $Q(p)$ shape functions, efficiency on eight processors ranges from 40 percent for the smallest grid (4×4) and polynomial degree ($p = 4$), where scheduling overhead is high; to a maximum of 85 percent, corresponding to maximum speedup of slightly under 7. There are some examples of slight declines in efficiency when p increases from 8 to 16. The $Q'(p)$ shape functions incur larger costs and they have slightly higher efficiencies, but otherwise they produce qualitatively similar results to those for the $Q(p)$ shape functions.

Because the $Q(p)$ basis functions have lower costs, we restrict our attention to them in the sequel. Table 7 shows a breakdown of costs and speedups of several of the individual tasks performed by the solution algorithm, for $n = 16$ and three values of p. This data corresponds to the third row of each block row in Table 4. The computations are broken into three large-scale steps, consisting of the construction

TABLE 4
Timings and speedups for Q-type shape functions.

		Timings				Speedups			
		Number of processors				Number of processors			
		1	4	6	8	1	4	6	8
$p=4$	4 × 4 grid	0.71	0.27	0.24	0.22	1.0	2.6	3.0	3.2
	8 × 8 grid	2.55	0.80	0.62	0.53	1.0	3.2	4.1	4.8
	16 × 16 grid	9.94	2.87	2.13	1.77	1.0	3.5	4.7	5.6
	32 × 32 grid	41.56	11.53	8.56	7.07	1.0	3.6	4.9	5.9
$p=8$	4 × 4 grid	3.76	1.10	0.88	0.69	1.0	3.4	4.3	5.4
	8 × 8 grid	14.65	3.96	2.90	2.26	1.0	3.7	5.1	6.5
	16 × 16 grid	57.98	15.39	10.80	8.60	1.0	3.8	5.4	6.7
	32 × 32 grid	234.37	61.73	43.17	34.61	1.0	3.8	5.4	6.8
$p=16$	4 × 4 grid	37.19	10.09	7.93	5.83	1.0	3.7	4.7	6.4
	8 × 8 grid	147.92	40.21	28.98	22.37	1.0	3.7	5.1	6.6
	16 × 16 grid	587.83	156.81	111.23	87.07	1.0	3.7	5.3	6.8

TABLE 5
Timings and speedups for $Q'(p)$ shape functions.

		Timings				Speedups			
		Number of processors				Number of processors			
		1	4	6	8	1	4	6	8
$p=6$	4 × 4 grid	0.92	0.31	0.27	0.24	1.0	3.0	3.4	3.8
	8 × 8 grid	3.49	1.02	0.78	0.64	1.0	3.4	4.5	5.5
	16 × 16 grid	13.91	3.89	2.82	2.29	1.0	3.6	4.9	6.1
	32 × 32 grid	56.10	15.22	11.08	9.02	1.0	3.7	5.1	6.2
$p=11$	4 × 4 grid	4.23	1.19	0.96	0.74	1.0	3.6	4.4	5.7
	8 × 8 grid	16.68	4.46	3.23	2.52	1.0	3.7	5.2	6.6
	16 × 16 grid	65.97	17.39	12.31	9.63	1.0	3.8	5.4	6.9
	32 × 32 grid	266.09	69.85	48.81	38.81	1.0	3.8	5.5	6.9
$p=23$	4 × 4 grid	50.47	13.66	10.68	7.76	1.0	3.7	4.7	6.5
	8 × 8 grid	201.69	53.93	38.80	30.16	1.0	3.7	5.2	6.7
	16 × 16 grid	796.02	211.54	148.83	117.94	1.0	3.8	5.3	6.7

TABLE 6
Overall efficiency.

		$Q(p)$			$Q'(p)$		
		Processors			Processors		
		4	6	8	4	6	8
$p=4$ (Q) $p=6$ (Q')	4 × 4 grid	65%	48%	40%	73%	56%	49%
	8 × 8 grid	80	68	60	86	75	68
	16 × 16 grid	87	78	70	90	82	76
	32 × 32 grid	90	81	74	92	84	78
$p=8$ (Q) $p=11$(Q')	4 × 4 grid	86	71	69	89	74	72
	8 × 8 grid	93	84	81	94	86	83
	16 × 16 grid	94	90	84	95	89	86
	32 × 32 grid	95	91	85	95	91	86
$p=16$(Q) $p=23$(Q')	4 × 4 grid	92	78	80	92	79	81
	8 × 8 grid	92	85	83	94	82	84
	16 × 16 grid	94	88	84	94	89	84

and condensation of the local stiffness matrices, the construction and factorization of the nodal preconditioning matrix, and the preconditioned conjugate gradient iteration for computing the nodal and side unknowns. The first of these steps entails purely local computations, the second is associated with the global (nodal) mesh, and the third requires both local and global computations. We see the following trends in this data:

— The costs are dominated by local stiffness matrix computations (construction and elimination). Since these are purely local, they are very highly parallelizable, and we see speedups on eight processors of 7.47, 7.25, and 6.86, respectively, for $p = 4$, 8, and 16 (efficiencies of 93 percent, 91 percent, and 86 percent). Thus efficiencies are generally high, although there is a decline as p increases.

— The construction and factorization of the (nodal) preconditioning matrix represents a small percentage of the overall cost.

— The PCG iteration also represents a small percentage of the computation, although it is more costly than the construction of the preconditioner. The speedups achieved for this part of the computation are smaller than those of the local matrix computations, but they are strictly increasing as p increases.

TABLE 7
Breakdown of timing costs and speedups for $Q(p)$ shape functions on a 16×16 grid.

	Timings				Speedups			
$p = 4$	Number of processors				Number of processors			
	1	4	6	8	1	4	6	8
Construct / condense LSM	6.27	1.61	1.10	0.84	1.0	3.9	5.7	7.5
Construct / factor precon.	0.45	0.13	0.12	0.11	1.0	3.5	3.8	4.0
PCG iteration	3.22	1.13	0.91	0.82	1.0	2.8	3.5	3.9
Complete computation	9.94	2.87	2.13	1.77	1.0	3.5	4.7	5.6

$p = 8$	Number of processors				Number of processors			
	1	4	6	8	1	4	6	8
Construct / condense LSM	48.59	12.52	8.58	6.70	1.0	3.9	5.7	7.3
Construct / factor precon.	0.50	0.14	0.13	0.12	1.0	3.6	3.8	4.2
PCG iteration	8.89	2.73	2.09	1.77	1.0	3.3	4.3	5.0
Complete computation	57.98	15.39	10.80	8.60	1.0	3.8	5.4	6.7

$p = 16$	Number of processors				Number of processors			
	1	4	6	8	1	4	6	8
Construct / condense LSM	555.79	147.68	104.14	81.04	1.0	3.8	5.3	6.9
Construct / factor precon.	0.67	0.19	0.17	0.16	1.0	3.5	3.9	4.2
PCG iteration	31.37	8.94	6.92	5.87	1.0	3.5	4.5	5.3
Complete computation	587.83	156.81	111.23	87.07	1.0	3.7	5.3	6.8

Remark 1. The preconditioning operations and the CG vector operations were implemented in parallel using compiler directives, and we did not limit the number of processors on which these computations were performed. As a result, the timings for four and six processors are underestimates. However, as we show below, the contributions of both these operations to overall cost are small, so that these timings do give a reasonable picture of performance.

Figures 4 and 5 show the asymptotic behavior of the timings from Table 4, as functions of n and p, respectively, in loglog scale. Both figures reflect the fact that costs are dominated by the local matrix computations. Thus for any fixed p, there are n^2 independent local constructions and condensations, and the costs grow like n^2. For fixed n and large p, growth is slightly slower than $O(p^4)$, the asymptotic cost; for example, the line segment between $p = 8$ and $p = 16$, for one processor and $n = 16$,

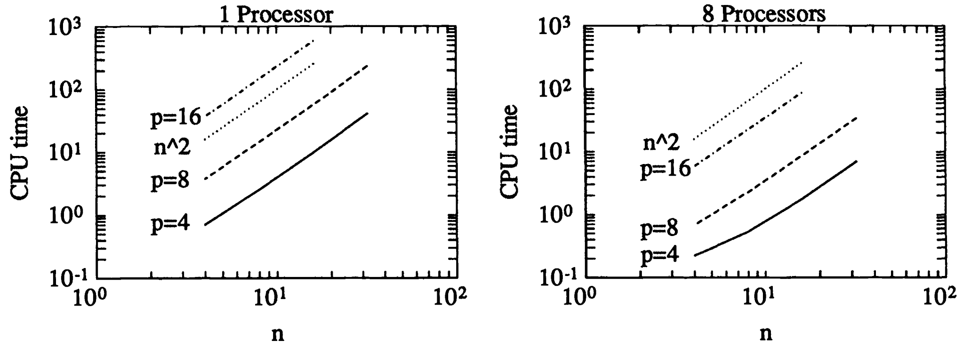


FIG. 4. *CPU times as functions of n, on* loglog *scale.*

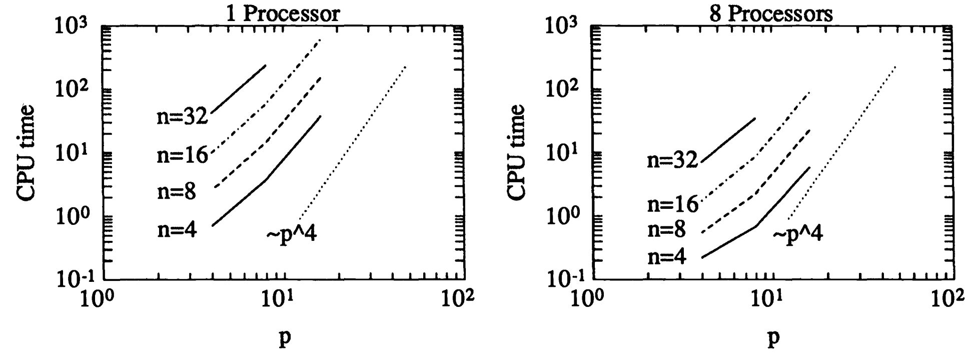


FIG. 5. *CPU times as functions of p, on* loglog *scale.*

has slope 3.34. For small p, growth is closer to $O(p^{2.5})$ because of the larger cost of computing the $O(p^2)$ nonzero entries.

4.3. Refined breakdown of costs. We now refine and elaborate on the timing results of Tables 4–7, showing how subsidiary steps of the tasks represented in Table 7 compare in cost. For these refined statistics, we compute costs on a single processor and supplement these results with discussion of how synchronization and memory conflicts affect performance on multiple processors.

First, consider the local stiffness matrix computations, i.e., construction and condensation of the local matrices. Table 8 shows the CPU times for each of these two steps on one processor, for $n = 16$ and $p = 4$, 8, and 16.[3] The results indicate that construction of the local matrices is more expensive than condensation. Since the matrix construction often involves a less regular set of computations than the condensation, we expect this phenomenon to be more pronounced for more general problems.

[3] To prevent calls to the timer from affecting measurements, the data in Tables 8 and 9 was generated separately from that in Tables 5 and 7, and Table 10 was produced separately from all of these. This is why these tables do not contain identical subtotals for identical computations.

TABLE 8

Breakdown of timing costs of local stiffness matrix computations on one processor, for a 16×16 *grid.*

	$p = 4$	$p = 8$	$p = 16$
Construct	3.98	33.24	366.28
Condense	2.58	15.06	184.99
Total	6.56	48.30	551.27

As noted above (see Table 7), although the local stiffness matrix computations corresponding to different elements are independent of one another, parallel efficiency declines as p increases. From Table 7, it is evident that efficiency also goes down as the number of processors grows. Figure 6 shows a more detailed picture of these phenomena. The curves represent speedups of the local matrix computations on four and eight processors, for $n = 8$, 12, and 16, and $p = 4$ through 18 in increments of 2. (As above, each curve represents average CPU times over three runs.) Here, the two sets of curves in each part of the figure correspond to two versions of the condensation step. The first version is the one used for all experiments described thus far; as shown in §3, it takes some advantage of sparsity of the local matrices. The second version takes no advantage of sparsity during the condensation, so that the computation (7) is performed as though all participating matrices are dense.[4] Thus this version is considerably more expensive. The same procedure for constructing the local matrices, as described in §3, was used with both versions of the condensation. The results of these figures show that there is indeed a decline in speedup as p increases, especially for the more costly version of condensation; in addition, efficiency is greater on four processors than on eight processors.

To understand these issues, it is necessary to examine the computer architecture in more detail. A feature of the Alliant FX/8 is that data moves between main memory and computational elements (i.e., processors) through a cache memory. Main memory and cache memory are connected by two buses, and cache memory and computational elements are connected by a crossbar switch with four paths to the cache [1], [18], [29]. Thus there are two sources of delay associated with movement of data between memory and processors: (i) it will take longer for data to move between processors and main memory than between processors and cache memory; (ii) when multiple processors are used, there will be contention for the buses (between main and cache memories), and possibly some delay in moving data through the crossbar switch (between cache memory and processors). We expect the crossbar switch to be less of a bottleneck than the buses [29]. However, if more than two processors attempt to move data to or from main memory, some processors will have to wait for access to the buses. Consequently, contention for the buses will tend to increase any delays caused by the need to move data between main and cache memories.

Although it is difficult to prove rigorously that these observations provide a complete explanation of parallel performance, the results of Fig. 6 are consistent with them. The machine used for these experiments has a cache memory with 512K bytes, or 64K double precision words. For the local stiffness matrix computations, each processor works with one block of storage of size equal to the number of entries in the upper triangle of the local stiffness matrix (approximately $(p+1)^4/2$). Consequently, k such blocks of storage fit into the cache provided $k \times (p+1)^4/2 \leq 65536$, which gives

[4] This entails removing checks for zero entries in the vector $m_{1:\nu_{\max},\mu}$ for the matrix-vector product (17).

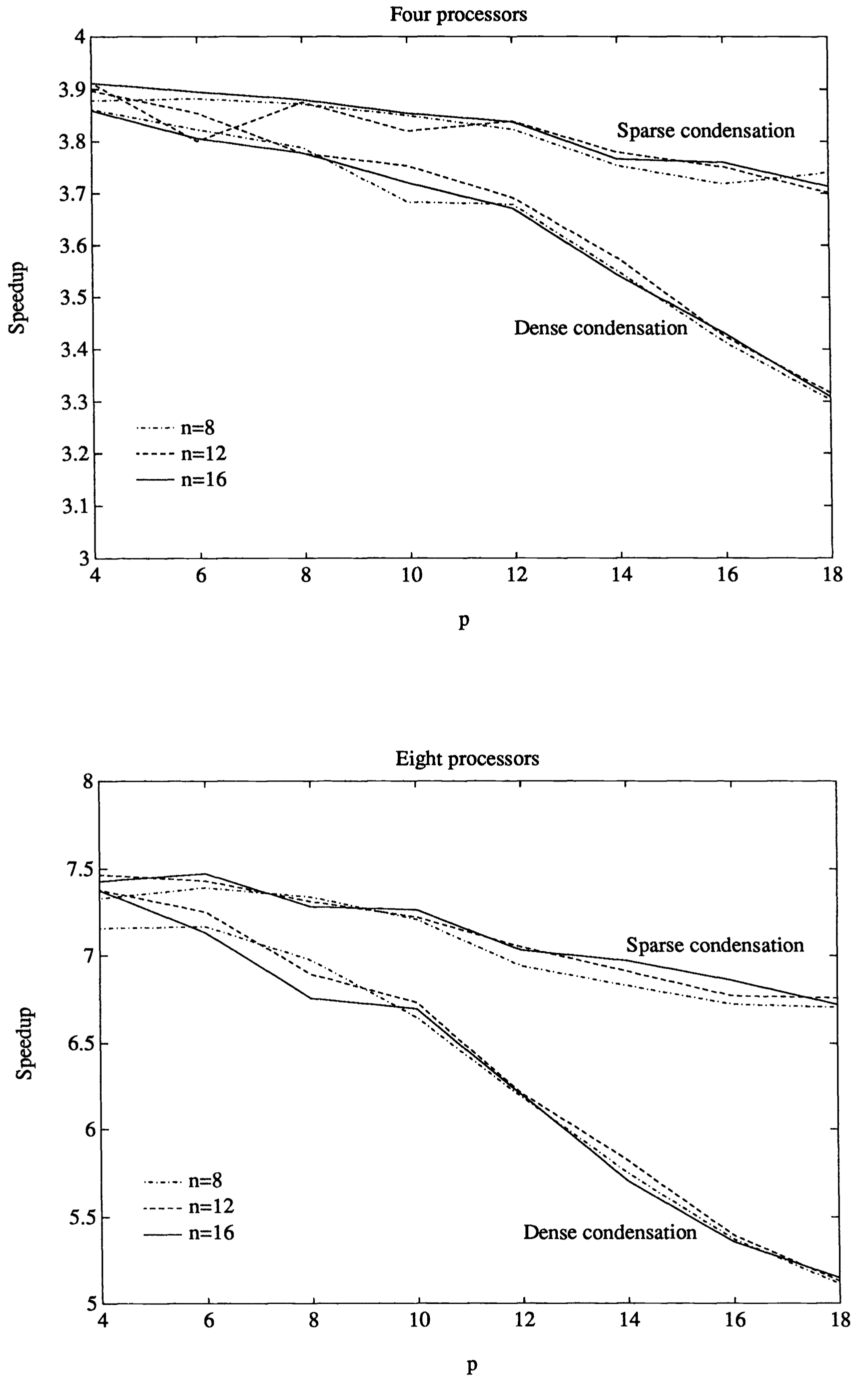


FIG. 6. *Speedups of local stiffness matrix computations for three meshes, on four and eight processors.*

$p \leq 12$ for $k = 4$ and $p \leq 10$ for $k = 8$. Examination of Fig. 6 shows steeper declines in speedups at precisely these values of p. The delays are greater for dense condensation, which we attribute to the additional memory references required for this version. We suspect that the (less pronounced) drops in speedup for smaller problems are partly explained by the lesser delays associated with the crossbar switch. In addition, in general we have no control over how data is distributed between cache memory and main memory, and it may be that the cache is not used with maximum efficiency even when all local matrices could fit into it. Thus there may be some additional congestion between the two levels of memory even for smaller problems. Finally, note that Fig. 6 suggests that speedup is essentially unaffected by the number of elements. This is substantiated by the following simple analysis. Let c_s denote the cost of processing a single local stiffness matrix in a serial computation, and assume that the parallel cost is increased to $c_p = (1 + \delta)c_s$, where δ (which may increase with k) reflects the delay due to memory contention. Then the speedup on k processors is

$$\frac{n^2 c_s}{n^2/(kc_p)} = k/(1 + \delta),$$

i.e., it is independent of the number of elements.

Remark 2. The last observation contrasts with our previous statement that speedups are somewhat lower for very coarse grids, i.e., $n = 4$, especially when p is also small. (See Tables 4 and 5.) For such small problems, smaller speedups are the result of scheduling overhead, which is amortized over a relatively small amount of computation.

Remark 3. The effects of memory conflicts of the type discussed above can also be diminished by implementing the condensation with BLAS3-type constructs [16], which are designed to use cache memory efficiently.

The PCG iteration does not contribute as much to overall cost as the local computations. Nevertheless, consideration of its individual steps still reveals some interesting properties of the particular form of the matrix-vector product (18), as well as some differences between global and local operations. We group the iterations into four subsidiary operations: matrix-vector products $w \leftarrow \hat{S}v$; preconditioning solves $w \leftarrow Q^{-1}v$; and two types of vector operations, inner products $\alpha \leftarrow v^T w$, and daxpy's $w \leftarrow \alpha v_1 + v_2$, of which there are three each. Table 9 shows a breakdown of the costs of these individual operations on a 16×16 grid, using one processor, and Table 10 further refines the details of the matrix-vector product. Here, "arithmetic" refers to the computation (19) used to perform the matrix-vector product; "index/copy" refers to the identification of global locations of local vectors and the copying of entries of global vectors (v in (18)) to local vectors (v_i); and "synchronize/copy" refers to the copy from w_i to w, plus the execution of the synchronization functions that prevent simultaneous writes to shared locations of w. "I/O" refers to the cost of copying the local stiffness matrix from one memory location to another.[5]

These results indicate that the matrix-vector product dominates the PCG iteration. This is largely a consequence of floating point operation counts (multiplications

[5] This cost is an artifact of the program design, which allows either in-core or out-of-core storage for local stiffness matrices, but in both cases requires that the local matrix be explicitly read from some source. This data movement could have been avoided, so that the cost of the matrix-vector is artificially high. It does not, however, alter our general conclusions. The cost of I/O would be much higher with out-of-core techniques.

TABLE 9

Breakdown of timing costs of local stiffness and PCG *computations on one processor, for a* 16×16 *grid.*

	$p = 4$	$p = 8$	$p = 16$
Matrix-vector product	2.92	8.31	30.19
Precondition	0.23	0.30	0.44
Inner product	0.05	0.13	0.34
Daxpy	0.06	0.17	0.43
Total	3.26	8.91	31.40

TABLE 10

Breakdown of timing costs of matrix-vector product, for a 16×16 *grid.*

	$p = 4$	$p = 8$	$p = 16$
Arithmetic	1.86	5.65	21.17
Index/copy	0.63	0.95	1.89
Synchronize/copy	0.30	0.67	1.37
I/O	0.26	1.21	5.45
Total	3.05	8.48	29.88

and additions), which are summarized as follows:

$$\begin{aligned}&\text{matrix-vector product} : 32n^2p^2,\\ &\text{CG-vector operations} : 24n^2p,\\ &\text{preconditioning solves} : 4n^3.\end{aligned}$$

The first line of Table 11 shows the ratios of operation counts for the matrix-vector products to operation counts for the other two steps, for $n = 16$ and several values of p; the data reveals the dominance of the matrix-vector product. The second line of the table shows the analogous ratios of CPU times, where the timing data for the CG-vector and preconditioning operations comes from Table 9, and the data for the matrix-vector product is from the "Arithmetic" entry of Table 10. We see that the results for operation counts agree qualitatively with those for CPU times, but they do not tell the whole story. For example, in the comparison of matrix-vector product and CG-vector operations, the ratios for operation counts are smaller than those for CPU times, indicating that the CG-vector operations are implemented more efficiently. Other factors that affect performance are vector startups and vector lengths. Each of the n^2 local matrix-vector products requires $4p$ vector startups, giving a total of $4n^2p$ startup overhead for the matrix-vector product; this contrasts with just six vector startups for the CG-vector operations. In addition, we are using only the lower triangle of the matrix $\hat{C}_i$, so that some of the vectors used in (19) are smaller than the Alliant's basic vector length of 32. Hence, although all these computations are vectorizable, performance for the matrix-vector is somewhat lower than for the CG-vector operations. The effect of startup overhead will be diminished on multiple processors, since some startups will be performed in parallel. In the comparison of matrix-vector and preconditioning steps, we see that for $p \geq 8$, the cost of the preconditioning (band)-solve is higher than the operation counts predict (i.e., the ratios of CPU times are smaller than the ratios of operation counts); this is because its typical vector length is the bandwidth n, or 16 for these data, i.e., less than 32.

A second general issue, which limits the speedups achieved by the PCG computations (Table 7), is the need to synchronize the results of local matrix-vector products

TABLE 11
Ratios of costs of PCG *operations on a* 16 × 16 *grid.*

	Matrix-vector product vs. CG-vector operations			Matrix-vector product vs. preconditioning		
	p =4	p =8	p =16	p =4	p =8	p =16
Ratio of operation counts	5.7	10.7	21.3	4.0	32.0	128.0
Ratio of CPU times	16.2	16.7	27.7	8.0	18.0	48.0

in forming the global product (18). Consider the following analysis. Let c_p denote the fully parallel part of the local matrix-vector product, consisting of the "index/copy" and "arithmetic" and "I/O" steps; here, we are ignoring any memory conflicts that may exist in these steps. Let c_s denote the serial part, consisting of the "synchronize/copy" steps. The cost of the global matrix-vector product on one processor is $n^2(c_p + c_s)$. In a parallel computation, processes often have to wait for access to w, and the wait can be as long as $(k-1)c_s$. Suppose every local matrix-vector product waits this long. The cost of the parallel computation is then approximately

$$\frac{n^2}{k}(c_p + (k-1)c_s),$$

so that the speedup is approximately

$$\frac{n^2(c_p + c_s)}{\frac{n^2}{k}(c_p + (k-1)c_s)} = k\left(\frac{c_p + c_s}{c_p + (k-1)c_s}\right). \tag{20}$$

Since the matrix-vector product dominates the PCG iteration, we also take (20) as a measure of the speedup achievable by PCG. Table 12 compares the values of (20) (where c_p and c_s are taken from Table 10) with the actual speedups from Table 7. The results suggest that (20) is a good indicator of the qualitative behavior of the PCG iteration. We attribute the fact that the accuracy of the model decreases with additional processors to added memory conflicts.

TABLE 12
Comparison of speedup model with actual speedups of the PCG *iteration, for a* 16 × 16 *grid.*

	$p = 4$		$p = 8$		$p = 16$	
Processors	Model	Actual	Model	Actual	Model	Actual
4	3.35	2.84	3.45	3.26	3.66	3.51
6	4.32	3.53	4.56	4.26	5.07	4.54
8	5.06	3.95	5.43	5.01	6.27	5.34

Remark 4. Although this model is pessimistic in the sense that $(k-1)c_s$ may be a long waiting time, we have observed empirically that processors do not reach the synchronization step in a fixed order, so that there are large delays for many local computations. It is also possible to decrease synchronization overhead using better bookkeeping techniques to identify specific locations of w that are available.

4.4. Comments on performance. Finally, we discuss the performance, in terms of floating point operations, of parts of the code that are both vectorized and implemented in parallel. Consider the condensation of the local stiffness matrices, which entail Cholesky factorization and computation of the Schur complement. To simplify operation counts, in the experiments considered here we used the dense version of condensation, as described in the discussion of Fig. 6. For the $Q(p)$ discretization, the floating point operation (multiplications and additions) counts are:

$$\text{Factor } A_i = L_i L_i^T : \tfrac{1}{3}(p-1)^6 + (p-1)^4 - \tfrac{4}{3}(p-1)^2$$

$$\text{Compute } \hat{B}_i = L_i^{-1} B_i : 4(p-1)^2((p-1)^2+1)p$$

$$\text{Compute } \hat{C}_i = C_i - \hat{B}_i^T \hat{B}_i : 4p(4p+1)(p-1)^2.$$

CPU times on one processor, for $n = 16$ and $p = 8$ and 16, are 28.4 and 874.5 seconds, respectively, giving performances in millions of floating point operations per second (Mflops) of 1.55 for $p = 8$ and 2.35 for $p = 16$. Based on the speedups for the local computations from Fig. 6 (6.75 for $p = 8$ and 5.36 for $p = 16$), this gives performance estimates on eight processors of 10.5 and 12.6 Mflops, respectively. Similar results were also obtained for the matrix-vector product (18)–(19), with a maximum rate of three Mflops on one processor (for local matrices of order approximately 500).

By way of contrast, the LINPACK benchmark (for dense elimination with dense matrices of order 100) on a single processor of an Alliant FX/4 is 2.1 Mflops [15], and on eight processors, the BLAS2 kernels with arguments in main memory achieve 18–20 Mflops [24, p. 81]. Hence our performance on vectorized code is at best comparable to that of the BLAS2 kernels. Note that this is lower than performance achieved for a variety of matrix operations reported, e.g., in [24], where BLAS3-type blocking strategies lead to performance of upwards of 30 Mflops. There are several reasons for this. First, despite the fact that (17) and (19) are designed to avoid unnecessary stores of the accumulating products $m_{\mu:\rho,\nu}$ and w, examination of the generated assembler code reveals that the actual computations are not performed efficiently. For example, in principle, the outer loop of (17) requires a daxpy with one argument (columns of M_μ) taken from memory, but no loads or stores to memory; the actual code performs one store, one load, and a daxpy with one argument in memory. Thus there are three times as many memory references as are necessary, leading to a degradation of performance on the order of 50 percent. Second, we have little control over management of the cache memory; we suspect that because there are many local matrices being processed, they are likely to be located in main memory rather than cache memory. The good performances exhibited in [24] were achieved using hand-coded assembler [23]. We believe that better performance of the techniques under consideration here can be obtained using more sophisticated coding techniques. However, since these tasks do not have the dominant cost of the overall computation, further tuning will not affect our conclusions, and we have not pursued this issue. For more complex problems, e.g., where local stiffness matrices are constructed by (vectorized) quadrature, it would be imperative to implement the quadratures efficiently.

5. Conclusions. In this paper, we have examined the computational costs of an implementation of the *hp*-version of the finite element method on the Alliant FX/8, a shared-memory parallel computer. Our main conclusions are as follows:

1. Costs are dominated by the local computations, i.e., construction of local stiffness matrices and condensation of these matrices for elimination of internal unknowns.

2. Global operations, particularly the preconditioning associated with nodal unknowns, contribute a relatively small amount to overall cost.

3. Communication and synchronization costs associated with the use of unassembled local stiffness matrices for CG iteration do not greatly degrade performance, and their effects are understood.

4. The likelihood of memory conflicts places limitations on the sizes and number of local problems that can be handled efficiently. We expect this problem to be ameliorated through the use of more sophisticated coding techniques such as those available in the BLAS3 [16] or LAPACK [2] libraries, but we do not expect it to disappear entirely.

Thus the "natural parallelism" associated with the decomposition of problems by elements can be exploited in a straightforward manner to get good speedups, provided the size or number of local problems are not too large. These conclusions apply to a particular type of architecture, a shared-memory machine with a relatively small number of processors. We expect similar results to apply to other machines in this class, e.g., the CRAY-2.

We now discuss how we expect our observations to carry over to other classes of problems and computers.

1. *Different problem coefficients or domain topologies.* As long as the element grid is topologically rectangular, the general methodology described here should be applicable. As shown in §3, if the coefficients of (1) are more complex, or if the domain is less regular, then the fully local computations are more expensive, and we expect these costs to be more dominant. For highly anisotropic problems (e.g., large a/b) or discretizations with very flat rectangular elements, there may be an increase in PCG iteration counts, but we suspect this will not offset the dominance of the local computations.

2. *Use of the h-version.* If the h-version is used for discretization, then the analogue of the solution method presented here is domain decomposition, with local super-elements consisting, e.g., of p^2 elements. In this case, for PCG iterations to display convergence rates independent of problem size, it is necessary to perform some type of modification of the super-element side and internal shape functions [3], [6], [11]. We expect conclusions similar to those above to hold for such methodologies. An alternative for achieving fast convergence is to use standard shape functions, but to apply a relatively fine nodal preconditioner [27]. For such a strategy, a larger percentage of computational effort is devoted to the sparse matrix factorization and solves associated with preconditioning than we have observed.

3. *Adaptive methods.* In contrast to the methodology considered here, where all operators were computed "from scratch," the hp-method is often implemented in a hierarchical manner, where higher-order elements are used to supplement previously computed low-order operators. In this case, the local computations will be somewhat less dominant. Many issues along these lines, such as load balancing if different order basis functions are used in different elements, as well as mesh refinement strategies, remain open.

4. *Shared-memory computers with more processors.* Increasing the number of processors decreases the costs of the local stiffness matrix computations more than those of the other computations. For example, for all of the problems of Table 7, we estimate that increasing the number of processors by as much as a factor of four (to 32) will decrease the local costs significantly, but the effect on the costs of PCG will be small. In such a scenario, for $p \geq 8$ local costs will still dominate. In light of the fact that our local costs are artificially low, we expect our conclusions to apply for shared-memory machines with on the order of 50 processors. However, for this to be borne out, it will be necessary for the local matrix computations to be implemented so that memory conflicts do not limit efficiency.

5. *Local-memory computers.* We do not attempt to make a precise statement about this class of architectures, but to outline some of the issues. The memory conflicts associated with local matrix computations on shared-memory computers should not be a factor on local-memory machines, so that we expect the local computations to be more efficient on the latter class of architectures. The matrix-vector products entail exchanges of data corresponding to super-element boundaries, but we expect the effect of this overhead to be similar to that of the synchronization required by the shared-memory implementation. It is necessary to implement the global preconditioner and other CG operations efficiently.

Appendix. We outline the properties of the shape functions that give rise to sparse local stiffness matrices for constant coefficient problems. Consider the representation

$$S_i = \begin{bmatrix} (\mathcal{I},\mathcal{I}) & (\mathcal{I},\mathcal{S}) & (\mathcal{I},\mathcal{N}) \\ (\mathcal{S},\mathcal{I}) & (\mathcal{S},\mathcal{S}) & (\mathcal{S},\mathcal{N}) \\ (\mathcal{N},\mathcal{I}) & (\mathcal{N},\mathcal{S}) & (\mathcal{N},\mathcal{N}) \end{bmatrix}, \tag{21}$$

where each entry represents a block matrix containing all terms (5) in which the shape functions come from the indicated set. Thus, for example, the block $(\mathcal{S},\mathcal{I})$ contains all terms in which $u \in \mathcal{S}$ and $v \in \mathcal{I}$. (In (6), A_i corresponds to $(\mathcal{I},\mathcal{I})$.) From properties of the Legendre polynomials, it can be shown that the following relations hold:

(22)

$$\begin{array}{llll}
\text{In } (\mathcal{I},\mathcal{I}): & B_{\Omega^i}(\Phi^{(\mathcal{I})}_{jk},\Phi^{(\mathcal{I})}_{lm}) \neq 0 & \text{iff} & (j = l \text{ or } j = l \pm 2) \text{ and } k = m; \text{ or} \\
 & & & (k = m \text{ or } k = m \pm 2) \text{ and } j = l. \\
\text{In } (\mathcal{S},\mathcal{I}): & \left.\begin{array}{r} B_{\Omega^i}(\Phi^{(\mathcal{S},1)}_{j},\Phi^{(\mathcal{I})}_{lm}) \neq 0 \\ B_{\Omega^i}(\Phi^{(\mathcal{S},3)}_{j},\Phi^{(\mathcal{I})}_{lm}) \neq 0 \end{array}\right\} & \text{iff} & j = m \text{ and } (l = 2 \text{ or } l = 3). \\
 & \left.\begin{array}{r} B_{\Omega^i}(\Phi^{(\mathcal{S},2)}_{j},\Phi^{(\mathcal{I})}_{lm}) \neq 0 \\ B_{\Omega^i}(\Phi^{(\mathcal{S},4)}_{j},\Phi^{(\mathcal{I})}_{lm}) \neq 0 \end{array}\right\} & \text{iff} & j = l \text{ and } (m = 2 \text{ or } m = 3). \\
\text{In } (\mathcal{N},\mathcal{I}): & B_{\Omega^i}(\Phi^{(\mathcal{N},j)},\Phi^{(\mathcal{I})}_{lm}) \equiv 0. & & \\
\text{In } (\mathcal{S},\mathcal{S}): & B_{\Omega^i}(\Phi^{(\mathcal{S},l)}_{j},\Phi^{(\mathcal{S},m)}_{k}) \neq 0 & \text{iff} & l - m \text{ is even and } (j = k \\
 & & & \text{or } j = k \pm 2). \\
\text{In } (\mathcal{N},\mathcal{S}): & B_{\Omega^i}(\Phi^{(\mathcal{N},k)},\Phi^{(\mathcal{S},l)}_{j}) \neq 0 & \text{iff} & j = 2 \text{ or } j = 3. \\
\text{In } (\mathcal{N},\mathcal{N}): & B_{\Omega^i}(\Phi^{(\mathcal{N},j)},\Phi^{(\mathcal{N},k)}) \neq 0. & &
\end{array}$$

Relations in $(\mathcal{I},\mathcal{S})$, $(\mathcal{I},\mathcal{N})$, and $(\mathcal{S},\mathcal{N})$ are determined from symmetry.

As an example of how (22) is established, consider the entries of $(\mathcal{I},\mathcal{I})$. Let $(f,g) \equiv \int_{-1}^{1} f(\xi)g(\xi)d\xi$. The Legendre polynomials satisfy [14]

$$(P_j, P_k) = \begin{cases} 1/(2j+1) & \text{if } j = k, \\ 0 & \text{if } j \neq k; \end{cases} \tag{23}$$

$$P_j(\xi) = \frac{1}{2j+1}\left(P'_{j+1}(\xi) - P'_{j-1}(\xi)\right); \tag{24}$$

$$P_j(-1) = (-1)^j. \tag{25}$$

From (3) and (13)–(15), we have

$$B_{\mathcal{E}}(\Phi^{(\mathcal{I})}_{jk}, \Phi^{(\mathcal{I})}_{lm}) = a\,(\phi_j, \phi_l)(\phi'_k, \phi'_m) + b\,(\phi'_j, \phi'_l)(\phi_k, \phi_m). \tag{26}$$

Consequently, (3), (24), and (25) imply that

$$\phi_j(\xi) = \frac{1}{\sqrt{2(2j-1)}}\,(P_j(\xi) - P_{j-2}(\xi)),$$

so that

$$(\phi_j, \phi_l) = \begin{cases} [(P_j, P_j) + (P_{j-2}, P_{j-2})] \,/\, [2(2j-1)] & \text{if } j = l, \\ -(P_j, P_j) \,/\, [2\sqrt{(2j-1)(2j+3)}] & \text{if } j = l-2, \\ -(P_{j-2}, P_{j-2}) \,/\, [2\sqrt{(2j-1)(2j-5)}] & \text{if } j = l+2, \\ 0 & \text{otherwise.} \end{cases} \tag{27}$$

Moreover,

$$\phi'_k(\xi) = \sqrt{\frac{2k-1}{2}}\,P_{k-1}(\xi),$$

so that

$$(\phi'_k, \phi'_m) = \begin{cases} [(2k-1)/2]\,(P_{k-1}, P_{k-1}) & \text{if } k = m, \\ 0 & \text{otherwise.} \end{cases} \tag{28}$$

Thus the first term of (26) is nonzero if and only if $j = l$ or $j = l \pm 2$ and $k = m$; the second term is handled in an identical way.

Acknowledgment. We thank Kyle Gallivan for several helpful discussions.

REFERENCES

[1] Alliant Computer Systems Corporation, Littleton, MA, *FX/FORTRAN Programmer's Handbook*, March 1987.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, LAPACK: *A portable linear algebra library for high-performance computers*, Tech. Report CS-90-105, Computer Science Department, University of Tennessee, Knoxville, TN, 1990.

[3] I. Babuška, A. Craig, J. Mandel, and J. Pitkäranta, *Efficient preconditionings for the p-version finite element method in two dimensions*, SIAM J. Numer. Anal., 28 (1991), pp. 624–661.

[4] I. Babuška and H. C. Elman, *Some aspects of parallel implementation of the finite element method on message passing architectures*, J. Comp. Appl. Math., 27 (1989), pp. 157–187.

[5] ———, *Performance of the hp-version of the finite element method with various elements*, Tech. Report, University of Maryland Institute for Physical Science and Technology, College Park, MD, 1991.

[6] I. Babuška, M. Griebel, and J. Pitkäranta, *The problem of selecting the shape functions for a p-type finite element*, Internat. J. Numer. Methods Engrg., 28 (1989), pp. 1891–1908.

[7] I. Babuška and M. Suri, *The p- and h-p versions of the finite element method, an overview*, Comput. Methods Appl. Mech. Engrg., 80 (1990), pp. 5–26.

[8] P. E. Bjørstad and O. B. Widlund, *Iterative methods for the solution of elliptic problems on regions partitioned into substructures*, SIAM J. Numer. Anal., 23 (1986), pp. 1097–1120.

[9] J. H. Bramble, J. E. Pasciak, and A. H. Schatz, *The construction of preconditioners for elliptic problems by substructuring*, I, Math. Comp., 47 (1986), pp. 103–134.

[10] ———, *The construction of preconditioners for elliptic problems by substructuring*, II, Math. Comp., 49 (1987), pp. 1–16.

[11] ———, *The construction of preconditioners for elliptic problems by substructuring*, III, Math. Comp., 51 (1988), pp. 415–430.

[12] ———, *The construction of preconditioners for elliptic problems by substructuring*, IV, Math. Comp., 53 (1989), pp. 1–24.

[13] P. G. Ciarlet, *The Finite Element Method for Elliptic Problems*, North–Holland, Amsterdam, 1978.

[14] S. D. Conte and C. de Boor, *Elementary Numerical Analysis, An Algorithmic Approach*, McGraw–Hill, New York, 1980.

[15] J. J. Dongarra, *Performance of various computers using standard linear equations software*, Tech. Report CS-89-85, Computer Science Department, University of Tennessee, Knoxville, TN, 1989.

[16] J. J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling, *A set of level* 3 *basic linear algebra subprograms*, ACM Trans. Math. Software, 16 (1990), pp. 1–17.

[17] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson, *An extended set of* FORTRAN *basic linear algebra subprograms*, ACM Trans. Math. Software, 14 (1988), pp. 1–17.

[18] J. J. Dongarra and I. S. Duff, *Advanced Architecture Computers*, Tech. Report 57, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 1989 (Revision 2).

[19] J. J. Dongarra, F. G. Gustavson, and A. Karp, *Implementing linear algebra algorithms for dense matrices on a vector pipeline machine*, SIAM Rev., 26 (1984), pp. 91–112.

[20] J. J. Dongarra, D. C. Sorensen, K. Connolly, and J. Patterson, *Programming methodology and performance issues for advanced computer architectures*, Parallel Comput., 8 (1988), pp. 41–58.

[21] M. Dryja, *A method of domain decomposition for three-dimensional finite element elliptic problems*, in Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Periaux, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988, pp. 43–61.

[22] M. D. Dryja and O. B. Widlund, *Some domain decomposition algorithms for elliptic problems*, in Iterative Methods for Large Linear Systems, D. R. Kincaid and L. J. Hayes, eds., Academic Press, San Diego, CA, 1990, pp. 273–291.

[23] K. A. Gallivan, Personal communication, 1990.

[24] K. A. Gallivan, R. J. Plemmons, and A. H. Sameh, *Parallel algorithms for dense linear algebra computations*, SIAM Rev., 32 (1990), pp. 54–135.

[25] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Second Edition, The Johns Hopkins University Press, Baltimore, MD, 1989.

[26] W. D. Gropp and D. E. Keyes, *A comparison of domain decomposition techniques for elliptic partial differential equations and their parallel implementation*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 166–202.

[27] ———, *Parallel performance of domain-decomposed preconditioned Krylov methods for PDEs with adaptive refinement*, Tech. Report RR-773, Computer Science Department, Yale University, New Haven, CT, 1990.

[28] A. T. Patera, *Advances and future directions of research on spectral methods*, in Computational Mechanics: Advances and Trends, A. K. Noor, ed., AMD-Vol. 75, American Society of Mechanical Engineers, New York, 1987, pp. 411–427.

[29] P. Stenström, *Reducing contention in shared-memory multiprocessors*, IEEE Comput., 21 (1988), pp. 26–37.

[30] A. Weiser, S. C. Eisenstat, and M. H. Schultz, *On solving elliptic equations to moderate accuracy*, SIAM J. Numer. Anal., 17 (1980), pp. 908–929.

[31] O. B. Widlund, *Iterative substructuring methods; algorithms and theory for problems in the plane*, in Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Periaux, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1988, pp. 113–128.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 6, pp. 1460–1469, November 1992

011

A SEMICOARSENING MULTIGRID ALGORITHM FOR SIMD MACHINES [*]

J. E. DENDY, JR.[†], M. P. IDA[‡], AND J. M. RUTLEDGE[§]

Abstract. A semicoarsening multigrid algorithm suitable for use on single instruction multiple data (SIMD) architectures has been implemented on the CM-2. The method performs well for strongly anisotropic problems and for problems with coefficients jumping by orders of magnitude across internal interfaces. The parallel efficiency of this method is analyzed, and its actual performance is compared with its performance on some other machines, both parallel and nonparallel.

Key words. multigrid, parallel computing

AMS(MOS) subject classifications. 65N20, 65W05

1. Introduction. Some previous papers have examined multigrid methods for their suitability for calculation on machines with SIMD architectures. Frederickson and McBryan [FM] developed and analyzed a method that was designed to keep all the processors busy. Decker analyzed [D1] the performance on SIMD machines of more traditional multigrid methods. Both of these papers were restricted to Poisson's equation with periodic boundary conditions; neither paper attempted to address the sort of problem we are interested in, namely,

$$-\nabla \cdot (D(x,y)\nabla U(x,y)) + \sigma(x,y)U(x,y) = F(x,y) \tag{1.1}$$

in a bounded region Ω of R^2, where $D = (D^1, D^2)$, D^i is positive, $i = 1, 2$, and D^i, σ, and F are allowed to be discontinuous across internal boundaries Γ of Ω; moreover, $D_1 \gg D_2$ and $D_1 \ll D_2$ in different subregions of Ω is possible.

Another method was advocated by Hackbusch [H] and was shown to be robust for constant coefficient, periodic, anisotropic problems. It can be argued that this method, or at least its precursor, may be found in [T]. We refer to this method as the Brandt–Hackbusch–Ta'assan method, if only to arrive at the acronym BHT. The BHT method, like the Frederickson–McBryan algorithm, preserves the busyness of the processors, and has the added advantage of only needing point relaxation for anisotropic problems. However, as shown in §4, the BHT method, as described by Hackbusch, does not handle problems like (1.1), and it is unclear how to give the method this capability.

Both the Frederickson–McBryan and BHT methods have the presumed advantage of keeping all the processors busy. However, idleness of processors is unimportant; what is important is the convergence factor per machine cycle. If keeping all the processors busy led to a significantly smaller convergence factor, then the busyness of processors would indeed be important. Also, busyness of processors may be an important issue only on grids with less than one point per processor (virtual processor ratio (VP ratio) less than 1). When the VP ratio is greater than 1, the CM-2 makes use of virtual processors. In effect, serial do loops (VP loops) are created on each

[*] Received by the editors March 25, 1991; accepted for publication (in revised form) October 25, 1991. The work of the first two authors was performed under the auspices of the U.S. Department of Energy under contract W-7405-ENG-36 and was partially supported by the Center for Research on Parallel Computation through National Science Foundation Cooperative Agreement CCR-8809615.

[†] Theoretical Division, Los Alamos National Laboratory, Los Alamos, New Mexico 87545.

[‡] California Institute of Technology, Pasadena, California 91125.

[§] Chevron Oil Field Research Company, P. O. Box 446, La Habra, California 90631.

processor. Thus, when the VP ratio is greater than 1, "busy processor" methods actually incur a substantial computational penalty. Since moderate-size problems easily exceed VP ratios of 1 on today's machines, "busy processor" methods would seem to be, at best, methods for the future.

What about the more traditional methods of dealing with (1.1)? The first multigrid method to handle such problems successfully was given in [ABDP] and expanded in [D2]; it used standard coarsening (discussed below), interpolation induced by the operator, Galerkin coarsening, and alternating red-black line relaxation. An alternative was first discussed in [DMRRS] for three-dimensional problems. (However, we must point out that the robustness of line relaxation coupled with semicoarsening for constant coefficient anisotropic problems was first reported in [W].) The method discussed in this paper is the two-dimensional analogue of the method in [DMRRS]; it uses semicoarsening in y, interpolation induced by the operator, Galerkin coarsening, and red-black line relaxation by lines in x. Additionally, the method in this paper uses a technique due to Schaffer [S]; without this technique, the semicoarsening method would not be competitive. The method discussed in this paper is largely the same as the method given in [SW]. This fact should not be too surprising, since both papers had their genesis in a code written by Dendy.

One potential liability of the method considered in this paper is the necessity to perform line relaxation. The BHT method, were it robust, would avoid this difficulty. The suggestion was made in [B] that anisotropies could be avoided by the use of local grid refinement, under the assumption that physical problems are isotropic and that anisotropies arise from nonuniform gridding. One way to avoid nonuniform gridding is to use local grid refinement. In [D3], it was shown how to generalize [D2] to the case of local grid refinement. However, many person-years have been invested in codes that do not use local grid refinement, and all these codes would have to be rewritten to use this approach. Moreover, it is not clear how local grid refinement algorithms will perform on SIMD machines. Finally, there are important physical problems which are strongly anisotropic; an example is petroleum reservoir engineering [L]. For these problems, local grid refinement, although it may be desirable for other reasons, does not lead to isotropic problems on the local grids. Thus it appears that the issue of anisotropic problems must be directly attacked, not avoided.

Yet another method, due to Mulder [M], seems to have possibilities. The idea is that each grid has two offspring, one obtained by semicoarsening in x, the other by semicoarsening in y. When two offspring are of the same size in x and y, they are declared to be the same offspring. This method is discussed further in §2.

2. Standard versus semicoarsening. In [ABDP] and [D2] standard coarsening was used; that is, given a Cartesian grid, the coarser grid is obtained by deleting the even x- and y-lines. In this paper semicoarsening is used; that is, the coarser grid is obtained by deleting the even y-lines. To handle general anisotropic situations with standard coarsening seems to require alternating line relaxation, whereas with semicoarsening only, line relaxation by lines in x is required. (We note that there are some situations in both cases that cannot be handled by these choices of relaxation [ABDP], but for our purposes, these cases are pathological.) What is the sequential relaxation work for the two methods? Given an $nx \times ny$ grid, the relaxation work for semicoarsening is of the order of $2(nx)(ny)(1+1/4+\cdots) = (8/3)(nx)(ny)$. For semicoarsening, the relaxation work is of the order of $(nx)(ny)(1+1/2+\cdots) = 2(nx)(ny)$.

To do this same counting argument for a SIMD architecture requires a short discussion of the assumptions. We first consider the case where the VP ratio is less than or equal to 1. In the simplest model for the CM-2, it is important that the arrays be "compatible," that is, of the same size; otherwise, great inefficiencies in communication result. Thus, given a fine grid and a coarser grid, it is assumed that the data for each grid are stored in compatible arrays. Thus every other row of the coarse grid matrix contains no useful information. Moreover, on a relaxation sweep, a mask is employed which makes the processors for these rows idle. The assumption, therefore, is that for every grid, the amount of work required to solve the collection of tridiagonal systems on that grid depends only on the number of x- or y-points on the finest grid. Thus the relaxation work in the standard coarsening case is $2W(nx)\log_2 ny$, and the relaxation work in the semicoarsening case is $W(nx)\log_2 ny$, where $W(nx)$ is the work to solve an $nx \times ny$ tridiagonal system. (There are $\log_2 ny$ grids, and on each grid the relaxation work is $2W(nx)$ or $W(nx)$, respectively.) If sparse Gaussian elimination (a.k.a. the Thomas algorithm) is used, $W(nx) = O(nx)$. If straightforward cyclic reduction is used, $W(nx) = O(\log_2 nx)$.

The above argument is correct for the tridiagonal solver currently implemented in CMSSL (CM Scientific Subroutine Library). However, we can write a tridiagonal solver that yields the following relaxation work estimate for the standard coarsening case:

$$\begin{aligned} 2\left(\log_2 nx + \log_2 \frac{nx}{2} + \cdots\right) &= 2(\log_2 nx + (\log_2 nx - 1) + \cdots) \\ &= 2\left(\log_2 nx \log_2 ny - \frac{1}{2}(\log_2 ny - 1)(\log_2 ny)\right) \\ &\simeq \log_2 nx \log_2 ny. \end{aligned}$$

The point is that when we know that, for example, every other processor is idle, the cyclic reduction algorithm can be started further along, eliminating every fourth point instead of every second point. For a VP ratio less than 1, however, communication dominates computation, so it is unlikely that this improved algorithm will be twice as fast as using the CMSSL tridiagonal solver, particularly since the latter is written in carefully optimized assembly language.

When the VP ratio is greater than 1, a hybrid algorithm [J] becomes the algorithm of choice for solving the tridiagonal systems; this algorithm performs the traditional Thomas algorithm sequentially on the serial loop part of the i-index on each physical processor and uses cyclic reduction to solve between physical processors; it is in fact implemented in the tridiagonal solver in CMSSL. This algorithm is also used in [SW]. It is ironic that the same algorithm is the efficient one for these different architectures (SIMD and MIMD (multiple instruction, multiple data)); however, the algorithm used in [SW] is actually a SPMD (single program, multiple data) algorithm.

When the VP ratio is much greater than 1, the serial work on each processor dominates, and the work estimate for relaxation reverts to the serial case. In any case, the semicoarsening algorithm appears to be more efficient for all VP ratios than the standard coarsening algorithm. The semicoarsening algorithm is also conceptually simpler, particularly in three dimensions, when the alternative to coarsening in z and performing xy-plane relaxations (using multigrid) [DMRRS] is standard coarsening, performing alternating plane relaxation [D5]. Bandy and Brickner [BB], however, are investigating the standard coarsening approach on the CM-2; hence, we should eventually be able to compare directly the two approaches.

We briefly discuss how the interpolation operators are derived, even though the description in [SW] is excellent. Let us denote the interpolation operator from the coarse grid G^{k-1} to the fine grid G^k by I^k_{k-1}. (The coarse grid operator L^{k-1} is given by Galerkin coarsening from the fine grid operator L^k by forming $(I^k_{k-1})^* L^k (I^k_{k-1})$. We are interested in five-point or nine-point discretizations of (1.1); hence, we want the coarse grid operators also to be five- or nine-point operators.) In [ABDP], [D1], and [D3], I^k_{k-1} was described as follows: At coarse grid points coinciding with fine grid points, I^k_{k-1} is just the identity. At a fine grid point lying vertically between two coarse grid points, let the template of the operator be given by

$$\begin{pmatrix} NW & N & NE \\ W & C & E \\ SW & S & SE \end{pmatrix}. \tag{2.1}$$

Then I^k_{k-1} at $v_{i,j}$ is given by $av_{i,j-1} + bv_{i,j+1}$, where

$$a = -(SW + S + SE)/(W + C + E) \quad \text{and} \quad b = -(NW + N + NE)/(W + C + E).$$

That is, we think of summing away the x-dependence to obtain a three-point relation. A problem with this approach, when using standard coarsening, is that if $\rho = C - NW - N - NE - W - E - SW - S - SE$ is small, then instead of using $W + C + E$ in (2.1), we should use $SW + S + SE + NW + N + NE$ instead; this point is discussed in [D3]. With standard coarsening, results are relatively insensitive to switching between formulas based on the size of ρ; however, for semicoarsening this is not the case. With standard coarsening, interpolation is also being performed in the x-direction; hence, there is a possibility of coefficient variations being averaged out in that direction. In the semicoarsening case, some mechanism for averaging in the x-direction is apparently needed. Schaffer [S] also came to this conclusion and discovered the following scheme: Let

$$A^- v^- + A^0 v^0 + A^+ v^+ = 0$$

be the equation that would give the row $v^0 = (v_{i,j}, i = 1, \cdots, nx)$ in terms of the rows $v^- = (v_{i,j-1}, i = 1, \cdots, nx)$ and $v^+ = (v_{i,j+1}, i = 1, \cdots, nx)$. Then

$$v^0 = -(A^0)^{-1}(A^- v^- + A^+ v^+). \tag{2.2}$$

Unfortunately, use of (2.2) would lead to a nonsparse interpolation, leading to nonsparse coarse grid operators. Schaffer's idea is to assume that

$$-(A^0)^{-1}A^- \quad \text{and} \quad (-A^0)^{-1}A^+$$

can each be approximated by diagonal matrices in the sense that B^- and B^+ are diagonal matrices such that

$$-(A^0)^{-1}A^- e = B^- e \quad \text{and} \quad (-A^0)^{-1}A^+ e = B^+ e,$$

where e is the vector $(1, \cdots, 1)$. To find B^- and B^+ requires just two tridiagonal solves. The interpolation formula using B^- and B^+ is

$$v^0 = B^- v^- + B^+ v^+.$$

At first blush it would appear that the SIMD relaxation work of Mulder's algorithm [M] is comparable to the SIMD relaxation work of our method, since the number of grids in Mulder's method, if $nx = ny$, is approximately $(\log_2 ny)^2$. In [NR], however, Van Rosendale and Naik (to be identified with the author of [D1]) show that the subgrids in the Mulder method can be organized so that relaxation on all grids simultaneously (concurrent relaxation) can be done efficiently. This approach leads to a degradation in convergence factor as well as processor-to-processor communication between grids for interpolation and residual weighting. In two dimensions [NR], the grids can be packed in such a way that communication between grids is efficient, but then the efficiency of relaxation suffers. Nevertheless, implementation of the Mulder method on the CM-2 is planned, as is a comparison with the method of this paper.

3. Implementation. Implementation issues are complicated by the fact that we are aiming at a moving target. The first version of this paper was written when the compiler on the CM-2 was the bit-serial version. Subsequently, this version was replaced by the slicewise compiler. For some time there will continue to be improvements in the compiler, operating system, and CMSSL. Rather than delay publication of this paper indefinitely, we have chosen to report the current status, and to try to guess what the effects of future developments will be.

In the first version of this paper we discussed the inefficiencies present in the large VP ratio case when compatible arrays for intergrid communication are assumed. Let us denote the i-index as the tridiagonal solver direction and the j-index as the multigrid coarsening direction. If the j-index is declared parallel, then the code compiles so that the VP loop on a physical processor always remains the same size, regardless of the coarse grid size. Thus, if the j-index is declared such that the VP loop size is 64, it remains so, instead of decreasing to 32, 16, etc., thus reflecting the inactive j-direction grid elements on coarser grids. One way to avoid this difficulty is to code the VP loop by splitting the j-index into parallel and serial parts; this kind of splitting was done for the CM-2 implementation (by Gyan Bhanot of Thinking Machines) of Jameson's FLOW 67 code, a timestepping multigrid code. In the first version of this paper, this splitting was done for the relaxation routine only, since the coding for this splitting is extremely cumbersome. Moreover, this splitting does not solve the problem of wasted storage.

A better solution than splitting the j-index into parallel and serial parts is to have arrays that are not compatible on fine and coarser grids, and to use temporary arrays to achieve compatibility. For definiteness, assume that a fine grid array A is $nx \times ny$, and that a coarser grid array B is $nx \times \frac{ny}{2}$. If it is desired that A communicate with B, every other row of A needs to be placed in a temporary $nx \times \frac{ny}{2}$ array C. This is an intraprocessor move of data and can be accomplished efficiently with a routine written by Brickner. This solution has two difficulties associated with it. The first is that it creates a ragged array data structure not supported by FORTRAN 8X. (A and B in the example are really $D(k, \cdots)$ and $D(k-1, \cdots)$.) This difficulty has been cured by a routine written in C which does dynamic storage allocation. With each array $D(k, \cdots)$ is associated an array descriptor that contains the information on the dimensions of $D(k, \cdots)$. Thus the FORTRAN 8X compiler can be fooled into accepting a ragged array data structure. The second difficulty is that the current compiler aggressively monitors array layouts to assure that arrays are evenly distributed on processors. In many applications, this aggressiveness is a good strategy; however, in this application, because the semicoarsening leads to rectangular (as opposed to square) arrays, it can lead to reallocation to different processors of points which need to communicate and which should be on the same processor. We attempted to bypass this reallocation by writing routines which essentially informed the compiler to leave our arrays alone.

Unfortunately, the compiler still intervened when creating temporary arrays, yielding not only inefficiencies but also wrong answers. Our current remedy has been to code at a lower level than FORTRAN 8X; this remedy solves the problem of the compiler trying to take control of the layout, but does not generate as efficient code per processor as the compiler is capable of generating.

One important aspect of this work has been to identify this compiler shortcoming. (We must temper these whinings with the observation that the slicewise compiler was created in an incredibly short time.) Ours has not been the only application in which it is desirable for the programmer to take away control of the layouts from the compiler, and indeed, new versions of the compiler have been promised which will provide this capability. In the standard coarsening case, however, we may not need this new compiler capability; in this case, the compiler's choice of layout may be acceptable.

A final comment for the large VP ratio case is that line relaxation performs with nearly the same efficiency as point relaxation, since most of the work is serial work done on each processor. For large problems, the work performed on the grids with the VP ratio less than or equal to 1 is a small part of the overall calculation. Related to this issue is the question of when the coarse grids calculation should be done on the front-end machine. For a powerful front-end machine, the coarse grids may have to be fairly fine before it even pays to invoke the CM-2's power. This issue is addressed further in §4.

There are at least two versions of the Thomas algorithm. One computes the LU-decomposition on the tridiagonal matrix as it is needed. Other versions save the LU-decomposition (one such version was exploited in [ABDP] to avoid expensive divides on the CDC-7600). There is an analogous situation with respect to cyclic reduction. In the first version of this paper, we found that a version that saves the LU-decomposition ran two times faster on the CM-2 than a version that recomputes the LU-decomposition. However, for cyclic reduction, the LU-decomposition must be stored at each level of the parallel reduction. The result, for the i-index, is that the requirement for storage is proportional to $nx(\log_2 nx)$, where nx is the number of i-grid points. However, for the hybrid version [J], the storage requirement of the LU-decomposition is just proportional to nx, assuming that we do not save the two-cyclic LU-decomposition needed for the processor boundary grid points. (For high VP ratios, this assumption is reasonable since the cyclic reduction part of the tridiagonal solves is a small fraction of the overall computational time.)

4. Results. Many authors present gigaflop rates as a figure of merit while others report on speedup (of many processors compared to a single processor). While both these measures are useful in comparing the improved running speed of a specific algorithm, they can be misleading in determining the most efficient algorithm to solve a given problem. Point Jacobi, for example, applied to solve a discretization of (1.1), has an impressive gigaflop rate and speedup factor; however, it cannot compete with the multigrid algorithm of this paper since its convergence factor is abysmally near 1 for large problems while the multigrid convergence factor stays nicely bounded away from 1. Hence what we concentrate on in this paper is actual timing data. The convergence factors for various problems for the multigrid algorithm are reported in detail in [SW] and need not be repeated here. Finally, it is impossible to give speedup data for a CM-2 since it is impossible to access just one processor; we do, however, compare performance of one-quarter of a 2048-processor machine with one-quarter of a 1024-processor machine.

We present timing comparisons of several machines in Table 1. The timing results are given as seconds per V-cycle and were obtained by running five V-cycles (including setup time) and dividing by 5. Thus the timing results are independent of the difficulty of the problem run. All of the CM-2 timings reported in this section were done using one-quarter of a 2048-Weitek-processor machine or one-quarter of a 1024-Weitek-processor machine. (These timings may be used to address, at least partially, the issues of speedup and scalability.) The front end for the CM-2 was a Sun 4/90. The iPSC/2 machine had 64 nodes of 386-type processors; several configurations of these processors were considered for each problem size; here we have reported the timings only for the best [SW].

TABLE 1
Time per V-cycle on three machines.

Size of problem	iPSC/2	CRAY Y-MP	CM-2, one-quarter of 1024 processors	CM-2, one-quarter of 2048 processors
32 × 32	0.3	0.01	––	––
64 × 64	0.7	0.04	0.65	0.77
128 × 128	2.0	0.09	0.99	0.80
256 × 256	––	0.27	1.84	1.79
512 × 512	––	0.95	4.55	3.04
1024 × 1024	––	3.69	++	8.11
2048 × 2048	––	––	++	25.39

++ too large
–– no information

The timing results on the CRAY Y-MP were obtained in a time-sharing environment; with a dedicated Y-MP, we could have easily run problems larger than 2048 × 2048; however, if we had used all of a 2048-Weitek processor CM-2 we could have also run problems larger than 4096 × 4096. The results on the Y-MP show that great gains are easily made from vectorization for the smaller problems, but for the larger problems, the asymptote of time being linearly proportional to problem size has nearly been reached. The code used on the CRAY Y-MP uses only standard FORTRAN and is run only in single-processor mode. Presumably, speedups could be obtained using the multiprocessor capability of the Y-MP, but this is not considered here. The 512 × 512 and 1024 × 1024 cases were too large for the current loader and used a memory manager, to some obvious disadvantage in performance.

In the first version of this paper, we compared two versions of the hybrid tridiagonal solver: a version which recomputes the LU factorization on each call and a version which saves the LU factorization; the latter was about two times faster than the former. The figures in Table 1 use the CMSSL tridiagonal solve routine. This routine, which recomputes the LU factorization on each call, is as fast as the faster of our two versions since it is written in carefully optimized assembly language. If a factor-of-two speedup can be expected from the CMSSL tridiagonal solver which saves LU factorizations (not yet available), then we can expect to see nearly a factor-of-two decrease in the timings in Table 1 for the CM-2.

We report in Table 2 the effect of changing the size of the coarsest grid direct solve. The direct solve is done with a band solver on the front end with the LU-decomposition of that matrix being precomputed once. The problem in Table 2 has 256 × 256 grid points. (The timings in Table 2 use an earlier version of our code and should therefore only be considered in relation to each other and not to the timings in Table 1.) Note that there is a minimum for both the case of saving and not saving the cyclic reduction LU-decomposition. The reason for this is that on the coarser grids, so few Weitek processors are active that the front end (which is considerably faster than one Weitek processor) is more efficient, even when the time to transfer the data from the CM-2 to the front end is taken into account.

TABLE 2
Time per V*-cycle varying coarsest grid size.*

Size of coarsest grid	Number of grid levels	Recompute LU	Save LU
256 × 1	9	8.57	3.22
256 × 4	7	6.53	2.66
256 × 8	6	5.87	2.69
256 × 16	5	5.98	3.50
256 × 32	4	8.41	6.62

One other possibility for gaining efficiency from this algorithm is to make use of the multiwire (multiple NEWS communication) library written by R. Brickner and documented in [CM]. The routines in this library allow one to do simultaneous communication and computation in each index of an array. We have investigated the use of this library in a preliminary way, but have postponed further work until the other issues above have been satisfactorily resolved.

Let us now consider the BHT method; first, for the problem

$$\begin{cases} -\Delta U + U = 1 & \text{in } (0,1)\times(0,1), \\ U & \text{doubly periodic.} \end{cases}$$

On a 16 × 16 grid we compare the result of using the method of [D4] (standard coarsening, operator-induced interpolation, Galerkin coarsening, and red-black point relaxation) to the method of [H]. (We use the method in [D4] because we have not yet extended the method in this paper to handle periodic boundary conditions.) Both methods achieve an average convergence factor, over ten V-cycles, of 0.05 per V-cycle.

Now let us consider the problem

$$\begin{cases} -\nabla\cdot(D\nabla U) + U = F & \text{on } (0,16)\times(0,16), \\ U & \text{doubly periodic,} \end{cases} \tag{4.1}$$

where D and F are as shown in Fig. 1. Using a 16 × 16 grid, we again compare [D4] to [H]; the average convergence factor per V-cycle is 0.06 vs 0.54, respectively. If we modify the method in [H] to attempt to use the same operator-induced interpolation used in [D4], we obtain an average convergence factor of 0.09 instead. The problem is that within the context of the method in [H], it is no longer clear what operator-induced interpolation should be.

Finally, let us comment that we believe that some multigrid method is likely to be the fastest algorithm for solving problems like (1.1) on SIMD machines. We intend to substantiate this belief by timing the algorithm of this paper against some possible competitors, such as preconditioned conjugate gradient methods. In this comparison, the convergence factor per unit time must be considered for various problems, since clearly conjugate gradient with no conditioning will win against multigrid on SIMD machines for well-conditioned elliptic problems. In one sense the contest is over before it starts, since we already have an example in which a preconditioned conjugate gradient method stagnates badly, but for which the method of this paper is robust.

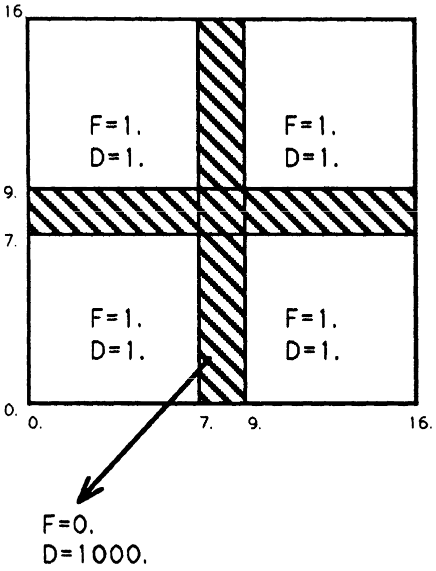


FIG. 1. *Diffusion coefficients and right-hand side for* (4.1).

Nevertheless, we hope to perform a comparison for a set of problems with a wide range of difficulty.

5. Conclusions. In this paper we have examined several multigrid methods in an attempt to find one that performs well on SIMD machines for problems with rough and anisotropic coefficients. We chose a semicoarsening multigrid algorithm for implementation on the CM-2 and have shown that it does perform well on that machine. We expect even better performance from this algorithm as compiler, operating systems, and library improvements become available.

Note added in proof. Recent advances in the version of BHT using operator-induced interpolation have led to an improved average convergence factor per V-cycle for (4.1): .09 per V-cycle instead of the .34 per V-cycle reported in this paper.

REFERENCES

[ABDP] R. E. ALCOUFFE, A. BRANDT, J. E. DENDY, JR., AND J. W. PAINTER, *The multi-grid method for the diffusion equation with strongly discontinuous coefficients*, SIAM J. Sci. Statist. Comput. 2(1981), pp. 430–454.

[B] A. BRANDT, *Multi-level adaptive solutions to boundary-value problems*, Math. Comp., 31(1977), pp. 333–390.

[BB] V. BANDY AND R. BRICKNER, private communication.

[D1] N. DECKER, *On the parallel efficiency of the Frederickson–McBryan multigrid*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 208–220.

[D2] J. E. DENDY, JR., *Black box multigrid*, J. Comp. Phys., 48(1982), pp. 366–386.

[D3] ———, *A priori local grid refinement in the multigrid method*, in Elliptic Problems Solvers II, G. Birkhoff and A. Schoenstadt, eds., Academic Press, New York, 1984, pp. 439–452.

[D4] ———, *Black box multigrid for periodic and singular problems*, Appl. Math. Comput., 25 (1988), pp. 1–10.

[D5] ———, *Two multigrid methods for three-dimensional problems with discontinuous and anisotropic coeffcients*, SIAM Sci. Statist. Comput., 8 (1987), pp. 673–685.

[DMRRS] J. E. Dendy, Jr., S. F. McCormick, J. W. Ruge, T. F. Russell, and S. Schaffer, *Multigrid methods for three-dimensional petroleum reservoir simulation*, in Proc. Tenth Symposium on Reservoir Simulation, Houston, TX, February 6–8, 1989, pp. 19–25.

[FM] P. O. Frederickson and O. A. McBryan, *Normalized convergence rates for the* PSMG *method*, SIAM J. Sci. Statist. Comput., 12 (1981), pp. 221–229.

[H] W. Hackbusch, *The frequency decomposition multigrid method, Part* I: *Application to anisotropic equations*, Numer. Math., 56(1989), pp. 229–245.

[J] S. L. Johnsson, *Solving tridiagonal systems on ensemble architectures*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 354–392.

[L] L. W. Lake, *The origins of anisotropy*, J. Petrol. Technology, April 1988, pp. 395–396.

[M] W. A. Mulder, *A new multigrid approach to convection problems*, J. Comput. Phys., 83 (1989), pp. 303–329.

[NR] N. Naik and J. Van Rosendale, *The improved robustness of multigrid solvers based on multiple semicoarsened grids*, SIAM J. Numer. Anal., 31 (1993), to appear.

[S] S. Schaffer, private communication, manuscript.

[SW] R. A. Smith and A. Weiser, *Semicoarsening multigrid on a hypercube*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 1314–1329.

[T] S. Ta'assan, *Multigrid methods for highly oscillatory problems*, Ph. D. thesis, Weizmann Institute of Science, Rehovot, Israel, 1984.

[W] G. Winter, *Fourienanalyse zur Konstruktion schneller MGR-Verfahren*, Ph. D. thesis, Rheinischen Friedrich-Wilhelms-Universität zu Bonn, Bonn, Germany, 1982.

SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 6, pp. 1470–1487, November 1992

012

A MINIMUM-PHASE LU FACTORIZATION PRECONDITIONER FOR TOEPLITZ MATRICES*

TA-KANG KU† AND C.-C. JAY KUO†

Abstract. A new preconditioner is proposed for the solution of an $N \times N$ Toeplitz system $T_N\mathbf{x} = \mathbf{b}$, where T_N can be symmetric indefinite or nonsymmetric, by preconditioned iterative methods. The preconditioner F_N is obtained based on factorizing the generating function $T(z)$ into the product of two terms corresponding, respectively, to minimum-phase causal and anticausal systems and is therefore called the minimum-phase LU (MPLU) factorization preconditioner. Due to the minimum-phase property, $||F_N^{-1}||$ is bounded. For rational Toeplitz matrices T_N with generating function $T(z) = A(z^{-1}/Bz^{-1}) + C(z)/D(z)$, where $A(z)$, $B(z)$, $C(z)$, and $D(z)$ are polynomials of orders p_1, q_1, p_2, and q_2, it is shown that the eigenvalues of $F_N^{-1}T_N$ are repeated exactly at 1 except at most α_F outliers, where α_F depends on p_1, q_1, p_2, q_2, and the number w of the zeros of $\tilde{T}(z) = A(z^{-1})D(z) + B(z^{-1})C(z)$ outside the unit circle. A preconditioner K_N in circulant form generalized from the symmetric case is also presented for comparison.

Key words. LU factorization, minimum phase, preconditioned iterative methods, preconditioner, Toeplitz, Laurent Padé approximation

AMS(MOS) subject classifications. 65F10, 65F15

1. Introduction.

Toeplitz matrices arise in many signal processing applications. To solve a general $N \times N$ Toeplitz system of equations $T_N\mathbf{x} = \mathbf{b}$, direct inverse algorithms based on Levinson recurrence [24] with $O(N^2)$ operations have been studied intensively in the past [13], [19], [32], [36]. Superfast algorithms with $(N \log^2 N)$ complexity have also been proposed [1], [3], [4], [12]. Although the computational complexity of these algorithms is lower than that of Gaussian elimination with pivoting, i.e., $O(N^3)$, their stability is still an issue when applied to indefinite or nonsymmetric T_N. It has been shown that these algorithms may become unstable if T_N is not symmetric positive definite (SPD) and well conditioned [5], [11]. A stable extension of the Levinson algorithm to general Toeplitz matrices has recently been studied by Chan and Hansen [9], [10].

In this research, we consider the use of preconditioned iterative methods for solving general Toeplitz systems $T_N\mathbf{x} = \mathbf{b}$ to reduce the computational complexity as well as to avoid the numerical instability. Various preconditioners in circulant form have been used in the the preconditioned conjugate gradient (PCG) algorithm [6], [8], [18], [20], [30] to solve SPD Toeplitz systems. All the preconditioners can be inverted via fast transform algorithms with $O(N \log N)$ operations. Besides, the spectra of the preconditioned Toeplitz matrices have such a nice clustering property that the PCG method converges superlinearly for T_N generated by a positive function in the Wiener class [7], [20]. Although it is possible to generalize this preconditioning technique to general Toeplitz matrices in a straightforward way (see §4), the focus of this paper is to develop a novel approach to construct a general Toeplitz preconditioner based on an approximate LU factorization. The resulting preconditioned systems are then solved by various iterative methods, such as the generalized minimal residual (GMRES) [28] and the conjugate gradient squared (CGS) [29].

* Received by the editors February 19, 1991; accepted for publication (in revised form) October 7, 1991. This work was supported by the University of Southern California Faculty Research and Innovation Fund and by a National Science Foundation Research Initiation Award (ASC-9009323).

† Signal and Image Processing Institute and Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, California 90089-2564 (tkku@sipi.usc.edu and cckuo@sipi.usc.edu).

The idea of constructing the LU factorization preconditioner can be simply stated as follows. Consider a banded Toeplitz matrix T_N with a finite-order generating function $T(z) = \sum_{n=-s}^{r} t_n z^{-n}$. Let us assume that $T(z)$ has no zeros on the unit circle. The $T(z)$ can be factorized into the product $T(z) = z^j L(z^{-1})U(z)$, where $L(z^{-1})$ and $U(z)$ have all zeros inside and outside the unit circle, respectively. We associate z^j, $L(z^{-1})$, and $U(z)$ with a shift matrix S_N, lower and upper triangular banded Toeplitz matrices L_N and U_N, correspondingly, and the product $F_N = S_N L_N U_N$ is the desired preconditioner for T_N. The above factorization procedure has been used frequently in the context of digital signal processing [26] to design the minimum-phase causal (or maximum-phase anticausal) linear filter. The F_N is therefore called the minimum-phase LU (MPLU) factorization preconditioner. To generalize the MPLU preconditioning technique to full Toeplitz matrices, we first obtain an approximating rational generating function for the original one with the Laurent Padé approximation. Since a rational Toeplitz matrix can be transformed to a banded matrix which is nearly Toeplitz, the appropriate MPLU preconditioner can also be constructed. A similar factorization procedure was used by Trench to derive a stable direct method for solving rational Toeplitz systems [35].

The condition number of the preconditioner F_N is bounded due to the minimum-phase factorization property. Thus, for well-conditioned Toeplitz matrices T_N, the condition number of the preconditioned matrix $A_N = F_N^{-1} T_N$ is also bounded so that the system $A_N \mathbf{x} = F_N^{-1}\mathbf{b}$ can be stably solved by iterative algorithms. One obvious choice is to form the SPD normal system $A_N^T A_N \mathbf{x} = A_N^T F_N^{-1}\mathbf{b}$, and to solve the resulting system by the CG method (known as the CGN method [17]). Thus, for well-conditioned nonsymmetric Toeplitz systems, numerical stability is easily obtained by using preconditioned iterative methods.

The spectral clustering properties of the MPLU-preconditioned Toeplitz matrix $F_N^{-1} T_N$ are studied for both banded and rational T_N. We prove that, for rational T_N with generating function $T(z) = A(z^{-1})/B(z^{-1}) + C(z)/D(z)$, where $A(z)$, $B(z)$, $C(z)$, and $D(z)$ are polynomials of orders p_1, q_1, p_2, and q_2, the eigenvalues of $F_N^{-1} T_N$ are repeated exactly at 1 except at α_F outliers, where α_F depends on p_1, q_1, p_2, q_2, and the number w of the zeros of $\tilde{T}(z) = A(z^{-1})D(z) + B(z^{-1})C(z)$ outside the unit circle. A direct consequence of these spectral properties is that the appropriate preconditioned iterative methods converge in at most $\alpha_F + 1$ iterations. This result should be compared to that of the circulant-preconditioned rational Toeplitz matrix $K_N^{-1} T_N$. In [23], we proved that the eigenvalues of $K_N^{-1} T_N$, except α_K outliers, are clustered in the disk centered at 1 with radius ϵ_K, where the clustering radius ϵ_K is proportional to the magnitude of the last elements used to construct the circulant preconditioner. It is clear that $\epsilon_K \geq \epsilon_F = 0$. Furthermore, if $w = \max(p_2, q_2)$, it can be shown that $\alpha_K = 2\alpha_F$ (see Theorems 3 and 4) so that the MPLU preconditioner provides better spectral clustering properties for a faster convergence rate. If $w \neq \max(p_2, q_2)$, we may have $\alpha_K < \alpha_F$ and $\epsilon_K \approx 0$ so that the circulant preconditioner K_N gives a faster convergence rate. In general, the MPLU preconditioner F_N has a better or a comparable convergence rate compared to the circulant preconditioner K_N.

Since the MPLU preconditioner F_N is a product of the shift matrix S_N and triangular banded Toeplitz matrices L_N and U_N, the preconditioning step $\mathbf{z} = F_N^{-1}\mathbf{r}$ can be achieved with a computational complexity proportional to $O(N)$ only. The total computational complexity for solving a rational Toeplitz system by MPLU-preconditioned iterative methods is $O(N)$, which is lower than the $O(N \log N)$ operations required by the circulant-preconditioned iterative methods and is in the same order as that required by several direct methods [14], [15], [33], [34], [35]. However, there is a drawback of the MPLU preconditioner in the context of parallel processing. That is, the MPLU preconditioning has to be performed sequentially, whereas the circulant preconditioning can be easily parallelized.

The outline of this paper is as follows. In §2, the procedure to construct the MPLU preconditioner for banded Toeplitz matrices is described, and the spectral properties of the preconditioned banded Toeplitz matrices are examined. In §3, the MPLU preconditioning technique is generalized to full Toeplitz matrices, including both rational and nonrational cases, and the spectral properties of the MPLU-preconditioned rational Toeplitz matrices are studied. In §4, we compare the MPLU preconditioner with the circulant preconditioner K_N. Finally, numerical results are given in §5 to assess the efficiency of the MPLU preconditioner.

2. MPLU preconditioner for banded Toeplitz matrices.

Consider a sequence of $m \times m$ banded Toeplitz matrices T_m, $m = 1, 2, \cdots$, generated by a polynomial

$$T(z) = \sum_{n=-s}^{r} t_n z^{-n}. \tag{2.1}$$

Let us assume that function $T(z)$ has no zeros on the unit circle, i.e.,

$$T(e^{i\theta}) \neq 0 \quad \forall \theta. \tag{2.2}$$

The system of linear equations

$$T_N \mathbf{x} = \mathbf{b} \tag{2.3}$$

can be solved by various iterative methods. To accelerate the convergence rate, a preconditioner P_N is introduced to solve the preconditioned system

$$P_N^{-1} T_N \mathbf{x} = P_N^{-1} \mathbf{b}, \tag{2.4}$$

where P_N is the preconditioner used to approximate T_N.

2.1. Construction of the preconditioner.

We can use a direct method to factorize T_N,

$$T_N = \mathcal{L}_N \mathcal{U}_N, \tag{2.5}$$

where $\mathcal{L}_N$ and $\mathcal{U}_N$ are lower and upper triangular matrices, respectively. The exact factorization (2.5) with the Schur-type algorithm requires $O(Nr + Ns)$ operations for banded T_N [14], [33]. If T_N is not SPD, the numerical stability of these algorithms cannot be guaranteed. Instead of performing the exact factorization, we propose to factorize T_N approximately as

$$T_N \approx S_N L_N U_N = F_N, \tag{2.6}$$

where S_N is a shift matrix and L_N and U_N are, respectively, lower and upper triangular banded Toeplitz matrices. Our objectives include that the approximate factorization (2.6) be achieved by a stable algorithm with operations independent of N, that F_N approximate T_N well, and that $||F_N^{-1}||$ be bounded. Then, the F_N can be used as a preconditioner in preconditioned iterative methods.

To derive the approximate factorization, it is convenient to consider the problem in the Z-transform domain and ignore the boundary effect arising in a Toeplitz system. When T_N is banded with lower bandwidth r and upper bandwidth s, its generating function can be expressed as

$$T(z) = \sum_{n=-s}^{r} t_n z^{-n} = t_{-s} z^s \prod_{i=1}^{d} (1 - z_i z^{-1}), \tag{2.7}$$

where $d = r + s$ and z_i is a root of $T(z)$. From (2.2), we know that $|z_i| \neq 1$. If $T(z)$ has w zeros outside the unit circle, we can factorize $T(z)$ as

$$T(z) = z^{s-w} L(z^{-1}) U(z), \tag{2.8}$$

where

$$L(z^{-1}) = \prod_{|z_i|<1} (1 - z_i z^{-1}), \qquad U(z) = t_{-s} \prod_{|z_i|>1} (z - z_i).$$

Note that the above factorization has a special feature, namely, all zeros of $L(z^{-1})$ (or $U(z)$) are inside (or outside) the unit circle. The following example is used to illustrate the factorization procedure.

Example 1. Let T_N be an $N \times N$ tridiagonal Toeplitz matrix with $t_1 = 1.5$, $t_0 = -6.5$, and $t_{-1} = 2$. Then, we have that

$$T(z) = 1.5z^{-1} - 6.5 + 2z = 2z(1 - 0.25z^{-1})(1 - 3z^{-1}) = L(z^{-1})U(z),$$

where

$$L(z^{-1}) = 1 - 0.25z^{-1}, \qquad U(z) = 2z - 6.$$

Since $r = s = w = 1$ in this example, the term z^{s-w} in (2.8) is equal to 1.

Let us associate the right-hand side of the factorization (2.8) with the following matrices:

$$L(z^{-1}) \longleftrightarrow L_N, \qquad U(z) \longleftrightarrow U_N, \qquad z^{s-w} \longleftrightarrow S_N \equiv E_N^{s-w}, \tag{2.9}$$

where L_N and U_N are $N \times N$ lower and upper triangular Toeplitz matrices with generating functions $L(z^{-1})$ and $U(z)$, respectively, E_N is the $N \times N$ unit row-shift matrix,

$$E_N = [e_N, e_1, e_2, \cdots, e_{N-1}],$$

and e_n is the $N \times 1$ unit vector with the nth element equal to 1 and zeros elsewhere. It is straightforward to verify that

$$E_N^{-1} = [e_2, e_3, \cdots, e_N, e_1]$$

and that E_N^k is the product of E_N (or E_N^{-1}) $|k|$ times for positive (or negative) integer k. The premultiplication of E_N (or E_N^{-1}) with an $N \times N$ matrix is equivalent to the circular upshift (or downshift) of its rows by one.

Then, the product of S_N, L_N, and U_N is used as the desired preconditioner

$$F_N = S_N L_N U_N = E_N^{s-w} L_N U_N. \tag{2.10}$$

Its inverse

$$F_N^{-1} = U_N^{-1} L_N^{-1} S_N^{-1} = U_N^{-1} L_N^{-1} E_N^{w-s}$$

can be performed effectively with $O(N)$ operations due to the special structures of S_N, L_N, and U_N. The factorization (2.8) has been frequently used in the context of digital signal processing [26] to design the minimum-phase causal (or maximum-phase anticausal) linear filter, which is by definition a system characterized by a lower (or upper) triangular matrix with a stable inverse. Thus we call F_N defined by (2.10) the MPLU factorization preconditioner.

2.2. Spectral properties. The minimum-phase factorization procedure guarantees that $||F_N^{-1}||$ is bounded, which is proved in the following theorem.

THEOREM 1. *Let T_N be a banded Toeplitz matrix with lower bandwidth r and upper bandwidth s satisfying condition* (2.2), *and L_N and U_N be obtained from the minimum-phase factorization* (2.8)–(2.10). *Then, the* 1-, 2-, *and* ∞-*norms of F_N^{-1} and F_N are bounded for all N.*

Proof. It is well known that there exists an isomorphism between the ring of the power series $G(z^{-1}) = \sum_{n=0}^{\infty} g_N z^{-n}$ and the ring of semi-infinite lower triangular Toeplitz matrices with $g_0, g_1, \cdots, g_N, \cdots$ as the first column, and that the power series multiplication is isomorphic to matrix multiplication [15]. With this isomorphism, we know that L_N^{-1} is a lower triangular Toeplitz matrix whose first column $\tau_0, \tau_1, \cdots, \tau_n, \cdots$ can be obtained from the coefficients of the power series, i.e.,

$$\frac{1}{L(z^{-1})} = \prod_{|z_i|<1} \frac{1}{(1 - z_i z^{-1})} = \sum_{n=0}^{\infty} \tau_n z^{-n}.$$

It is clear that $\sum_{n=0}^{\infty} |\tau_n|$ is bounded if and only if all poles of $1/L(z^{-1})$ are inside the unit circle, which is guaranteed by the minimum-phase factorization (2.8).

Since $T(z)$ has no zeros on the unit circle,

$$|z_i| \le 1 - \beta \quad \text{or} \quad 1 + \beta \le |z_i| < \infty,$$

where $\beta > 0$. Thus

$$||L_N^{-1}||_1 = ||L_N^{-1}||_\infty = \sum_{n=0}^{N-1} |\tau_n| \le \sum_{n=0}^{\infty} |\tau_n| \le \prod_{|z_i|<1} \frac{1}{1 - |z_i|} \le \beta^{-(d-w)}.$$

The 2-norm of L_N is bounded by

$$||L_N^{-1}||_2 \le (||L_N^{-1}||_1 ||L_N^{-1}||_\infty)^{1/2} = \sum_{n=0}^{N-1} |\tau_n| \le \beta^{-(d-w)}.$$

Similar arguments can be used to prove that $||U_N^{-1}||_2 \le \beta^{-w}$. Since $||E_N||_2 = ||E_N^{-1}||_2 = 1$, we have

$$||F_N^{-1}||_2 \le ||L_N^{-1}||_2 ||U_N^{-1}||_2 \le \beta^{-d},$$

which is independent of N. Besides, since $||L_N||_1 = ||L_N||_\infty$, we have

$$||L_N||_2 \leq (||L_N||_1||L_N||_\infty)^{1/2}.$$

Similarly, $||U_N||_2$ is bounded and $||F_N||_2 \leq ||L_N||_2||U_N||_2$ is also bounded. □

A direct consequence of the above theorem is that the condition number of preconditioner F_N is bounded for all N.

If $L(z^{-1})$ (or $U(z)$) is not chosen according to (2.8) so that there exist zeros of the polynomial $L(z^{-1})$ (or $U(z)$) with magnitude greater (or less) than one, i.e., nonminimum-phase factorization, we can easily check that $||L_N^{-1}||_2$ (or $||U_N^{-1}||_2$) is unbounded for asymptotically large N. For example, if we choose

$$\tilde{L}(z^{-1}) = 1 - 3z^{-1}, \qquad \tilde{U}(z) = 2z - 0.5,$$

for L_N and U_N in Example 1, the product $L_N U_N$ leads to an ill-conditioned matrix whose smallest eigenvalue converges to zero for asymptotically large N. Thus the minimum-phase factorization is crucial for the stability of the preconditioning step $\mathbf{z} = F_N^{-1}\mathbf{r}$.

Next, we study the spectral properties of $F_N^{-1}T_N$. For F_N to be a good preconditioner, it is desirable that $F_N^{-1}T_N$ has clustered eigenvalues. In Theorem 2 we will prove that it has only a finite number of eigenvalues different from 1. To derive this theorem, we need two lemmas.

LEMMA 1. *Let T_N be a banded Toeplitz matrix with lower bandwidth r and upper bandwidth s, where $r + s = d < N$, generated by $T(z)$ which has w zeros outside the unit circle. Then, for L_N and U_N obtained by the minimum-phase factorization* (2.8) *and* (2.9), *$L_N U_N$ is a banded Toeplitz matrix generated by $z^{w-s}T(z)$ with lower bandwidth $d - w$ and upper bandwidth w except its northwest $(d - w) \times w$ block.*

Proof. This lemma can be proved with definitions and direct matrix multiplication. □

Lemma 1 basically states that the product $L_N U_N$ is a nearly banded Toeplitz matrix. Despite the fact that T_N and $L_N U_N$ have the same total bandwidth d, they do not have the same lower bandwidth and upper bandwidth unless $w = s$. By shifting the rows of $L_N U_N$ circularly, we are able to construct another nearly banded Toeplitz matrix $F_N = E_N^{s-w} L_N U_N$, which has the same lower and upper bandwidth as T_N.

LEMMA 2. *Let T_N be a banded Toeplitz matrix with lower bandwidth r and upper bandwidth s, where $r + s = d < N$, generated by $T(z)$, which has w zeros outside the unit circle. Then, the matrix $F_N = E_N^{s-w} L_N U_N$ defined in* (2.10) *is a nearly banded Toeplitz matrix. Elements of matrices T_N and F_N are identical except for the following*:

(1) *the northwest $r \times s$ block when $s = w$;*

(2) *the northwest $r \times w$ block and the northeast $(w - s) \times r$ block when $s < w$;*

(3) *the northwest $r \times w$ block, the southwest $(s - w) \times s$ block, and the southeast $(s - w) \times (d - w)$ block when $s > w$.*

Proof. When $s = w$, it can be directly verified that $F_N = L_N U_N$ is a banded Toeplitz matrix generated by $T(z)$ with lower bandwidth r and upper bandwidth s except the northwest $r \times s$ block. When $s < w$, recall that the rows of $F_N = E_N^{s-w} L_N U_N$ are obtained from those of $L_N U_N$ with circularly downward-shift $w - s$ rows so that the last $w - s$ rows in $L_N U_N$ become the the first $w - s$ rows of F_N and the first $N - (w - s)$ rows in $L_N U_N$ become the last $N - (w - s)$ rows of F_N. By using Lemma 1, we can clearly see that F_N is a banded Toeplitz with lower bandwidth r

and upper bandwidth s generated by $T(z)$ except the northwest $r \times w$ block and the northeast $(w-s) \times r$ block. Similarly, we can prove the case $s > w$. □

Lemma 2 tells us that $\triangle E_N = F_N - T_N$ is a zero matrix except for at most three small blocks. Based on this lemma, we characterize the spectral properties of $F_N^{-1} T_N$ in Theorem 2.

THEOREM 2. *Let T_N be a banded Toeplitz matrix with lower bandwidth r and upper bandwidth s, where $r+s = d < N$, generated by $T(z)$ which has w zeros outside the unit circle. Then, there are at most α_F eigenvalues of $F_N^{-1}T_N$ not equal to 1, where*

$$\alpha_F = \begin{cases} \min(r,s), & s = w, \\ \min(r, 2w-s), & s < w, \\ \min(d-w, s), & s > w. \end{cases} \tag{2.11}$$

Proof. Since we have

$$F_N^{-1}T_N = F_N^{-1}(F_N - \triangle E_N) = I_N - F_N^{-1}\triangle E_N,$$

where I_N denotes the $N \times N$ identity matrix, the eigenvalue 1 of $F_N^{-1}T_N$ corresponds to the eigenvalue 0 of $F_N^{-1}\triangle E_N$, and the number of eigenvalues of $F_N^{-1}T_N$ not equal to 1 is determined by the rank of $\triangle E_N$. Notice that the rank of a matrix is bounded by the number of nonzero rows or columns, and the rank of the sum of two matrices is bounded by the sum of their individual ranks. All nonzero elements in $\triangle E_N$ are inside the blocks given by Lemma 2. When $s = w$, since all nonzero elements of $\triangle E_N$ are in the first r rows or the first s columns, the rank of $\triangle E_N$ is bounded by $\min(r, s)$. When $s < w$, we have $w - s \le d - s = r$. Since all nonzero elements of $\triangle E_N$ are either in the first r rows or in the union of the first w columns and the first $w - s$ rows, the rank of $\triangle E_N$ is bounded by $\min(r, 2w - s)$. When $s > w$, since all nonzero elements of $\triangle E_N$ are either in the union of the first r and the last $s - w$ rows or in the union of the first w columns and the last $s-w$ rows, the rank of $\triangle E_N$ is bounded by $\min(d - w, s)$. The proof is completed. □

Example 2 illustrates the above theorem.

Example 2. Consider the following $N \times N$ banded Toeplitz matrices with $N \ge 4$,

$$\begin{array}{lll} T_{N,1} & [(r,s) = (3,0)]: & t_3 = 2,\ t_2 = -5,\ t_1 = 6,\ t_0 = -2, \\ T_{N,2} & [(r,s) = (2,1)]: & t_2 = 2,\ t_1 = -5,\ t_0 = 6,\ t_{-1} = -2, \\ T_{N,3} & [(r,s) = (1,2)]: & t_1 = 2,\ t_0 = -5,\ t_{-1} = 6,\ t_{-2} = -2, \\ T_{N,4} & [(r,s) = (0,3)]: & t_0 = 2,\ t_{-1} = -5,\ t_{-2} = 6,\ t_{-3} = -2. \end{array}$$

$T(z)$ has zeros $0.5 + 0.5i$, $0.5 - 0.5i$, and 2, so that $w = 1$. For these matrices, the MPLU factorization results in the same L_N and U_N defined by the generating sequences

$$\begin{array}{llll} l_0 = 1, & l_1 = -1, & l_2 = 0.5, & l_n = 0 \quad n \ne 0,1,2, \\ u_0 = 4, & u_{-1} = -2, & & u_n = 0 \quad n \ne 0,-1. \end{array}$$

To illustrate Theorem 2, we list values of d, s, r, w, and α_F in Table 1.

Since $F_N^{-1}T_N$ has only at most $\alpha_F + 1$ distinct eigenvalues, appropriate preconditioned iterative methods, such as GMRES and CGS, converge in at most $\alpha_F + 1$ iterations with exact arithmetic (see Test Problems 1 and 4 in §5).

TABLE 1
An example to illustrate Theorem 2.

	d	r	s	w	α_F
$T_{N,1}$	3	3	0	1	2
$T_{N,2}$	3	2	1	1	1
$T_{N,3}$	3	1	2	1	2
$T_{N,4}$	3	0	3	1	2

3. Preconditioning full Toeplitz matrices.

In this section, we generalize the MPLU preconditioning technique to full Toeplitz matrices. It is known that a rational Toeplitz system $T_N\mathbf{x} = \mathbf{b}$ is equivalent to a banded system

$$\tilde{T}_N\tilde{\mathbf{x}} = \tilde{\mathbf{b}}, \tag{3.1}$$

where matrix $\tilde{T}_N$ is a banded Toeplitz matrix except for a northwest block. Thus we construct a MPLU preconditioner for $\tilde{T}_N$ as if it were an exact banded Toeplitz matrix and determine the solution of the rational Toeplitz system from (3.1) by preconditioned iterative methods. The equivalent MPLU preconditioner for rational Toeplitz matrix T_N is also a product of a shift matrix and triangular banded Toeplitz matrices, and can be inverted with $O(N)$ operations. The MPLU preconditioning scheme can also be generalized to nonrational Toeplitz matrices. The basic idea is to approximate the full Toeplitz matrix with a rational Toeplitz matrix, and then construct the MPLU preconditioner for the rational Toeplitz matrix.

3.1. Rational Toeplitz matrices.

Toeplitz matrices with a rational generating function can be transformed to banded ones [15]. We describe the transformation briefly as follows. Let the generating function of T_N be of the form

$$T(z) = \frac{A(z^{-1})}{B(z^{-1})} + \frac{C(z)}{D(z)}, \tag{3.2}$$

where $A(z)$, $B(z)$, $C(z)$, and $D(z)$ are polynomials in z with orders p_1, q_1, p_2, and q_2, respectively. Note that a special case of (3.2) is $A(z) = C(z)$ and $B(z) = D(z)$, which leads to a symmetric rational Toeplitz matrix of order (p, q) with $p_1 = p_2 = p$ and $q_1 = q_2 = q$. By applying the isomorphism between the ring of the power series and the ring of semi-infinite triangular Toeplitz matrices, we have the relationship

$$T_N = L_aL_b^{-1} + U_cU_d^{-1},$$

where L_a (or L_b) is an $N \times N$ lower triangular Toeplitz matrix with the first N coefficients in $A(z^{-1})$ (or $B(z^{-1})$) as its first column and U_c (or U_d) is an $N \times N$ upper triangular Toeplitz matrix with the first N coefficients in $C(z)$ (or $D(z)$) as its first row. Since power series multiplication is commutative, we have

$$\tilde{T}_N = L_bT_NU_d = L_aU_d + L_bU_c, \tag{3.3}$$

where $\tilde{T}_N$ is a banded and nearly Toeplitz matrix characterized by the following lemma.

LEMMA 3. *Let T_N be the $N \times N$ Toeplitz matrix generated by $T(z)$ in (3.2); the corresponding $\tilde{T}_N$ obtained from (3.3) is a banded Toeplitz matrix with lower bandwidth $r = \max(p_1, q_1)$ and upper bandwidth $s = \max(p_2, q_2)$ generated by*

$$\tilde{T}(z) = A(z^{-1})D(z) + B(z^{-1})C(z), \tag{3.4}$$

except for the northwest $r \times s$ block.

Proof. Consider $N \times N$ Toeplitz matrices L_a and U_d, where L_a is lower triangular with lower bandwidth p_1 generated by $A(z^{-1})$ and U_d is upper triangular with upper bandwidth q_2 generated by $D(z)$. We can verify that the product L_aU_d is a banded Toeplitz matrix generated by $A(z^{-1})D(z)$, except for its northwest $p_1 \times q_2$ block. This result can be easily generalized to the sum of two such products, i.e., $\tilde{T}_N = L_aU_d + L_bU_c$, and the proof is completed. □

Through (3.3), the system $T_N\mathbf{x} = \mathbf{b}$ is transformed to an equivalent system

$$\tilde{T}_N\tilde{\mathbf{x}} = \tilde{\mathbf{b}},$$

where $\mathbf{x} = U_d\tilde{\mathbf{x}}$ and $\tilde{\mathbf{b}} = L_b\mathbf{b}$. We then use the procedure described in §2.1 to construct the MPLU preconditioner $\tilde{F}_N$ for $\tilde{T}_N$ as if it were an exact banded Toeplitz matrix. The following theorem characterizes the spectral properties of $\tilde{F}_N^{-1}\tilde{T}_N$.

THEOREM 3. *Let T_N be the $N \times N$ rational Toeplitz matrix generated by $T(z)$ in* (3.2), *and $\tilde{F}_N$ the* MPLU *preconditioner constructed with respect to $\tilde{T}(z)$ in* (3.4). *In addition, $r = \max(p_1, q_1)$, $s = \max(p_2, q_2)$, and w denotes the number of zeros of $\tilde{T}(z)$ outside the unit circle. Then, when $r + s = d < N$, there are at most α_F eigenvalues of $\tilde{F}_N^{-1}\tilde{T}_N$ not equal to* 1, *where*

$$\alpha_F = \begin{cases} \min(r, s), & s = w, \\ \min(r, 2w - s), & s < w, \\ \min(d - w, 2s - w), & s > w. \end{cases}$$

Proof. By Lemma 3, $\tilde{T}_N$ is a banded Toeplitz matrix with generating function $\tilde{T}(z)$ except for the northwest $r \times s$ block. The $\tilde{F}_N$ is a banded Toeplitz matrix with generating function $\tilde{T}(z)$ except for the blocks described in Lemma 2. Define $\triangle\tilde{T}_N = \tilde{F}_N - \tilde{T}_N$. We can use arguments similar to those in proving Theorem 2 to determine the bound of the rank of $\triangle\tilde{T}_N$ and, hence, the number of eigenvalues of $\tilde{F}_N^{-1}\tilde{T}_N$ not equal to 1. □

Since $\tilde{F}_N^{-1}\tilde{T}_N$ has only at most $\alpha_F + 1$ distinct eigenvalues, appropriate preconditioned iterative methods converge in at most $\alpha_F + 1$ iterations with exact arithmetic (see Test Problems 2 and 5 in §5). Note that $\tilde{F}_N$ is a preconditioner for $\tilde{T}_N$ rather than T_N. However, since $\tilde{T}_N$ is related to T_N via (3.3), the equivalent preconditioner for matrix T_N is

$$F_N = L_b^{-1}\tilde{F}_NU_d^{-1} = L_b^{-1}E_N^{\tilde{s}-\tilde{w}}\tilde{L}_N\tilde{U}_NU_d^{-1},$$

where $\tilde{s}$, $\tilde{w}$, $\tilde{L}_N$, and $\tilde{U}_N$ are obtained with respect to $\tilde{T}(z)$ $(= A(z^{-1})D(z)+B(z^{-1})C(z))$. Thus the preconditioning step can be implemented as

$$F_N^{-1}\mathbf{r} = U_d\tilde{U}_N^{-1}\tilde{L}_N^{-1}E_N^{\tilde{w}-\tilde{s}}L_b\mathbf{r},$$

for arbitrary $\mathbf{r}$ with $O(N)$ operations.

3.2. Nonrational Toeplitz matrices. When T_N is generated by a nonrational function $T(z)$, we use the Laurent *Padé* approximation [4], [16] to approximate $T(z)$ with a certain rational function

$$T'(z) = \frac{A'(z^{-1})}{B'(z^{-1})} + \frac{C'(z)}{D'(z)},$$

where $A'(z)$, $B'(z)$, $C'(z)$, and $D'(z)$ are polynomials in z with orders p_1, q_1, p_2, and q_2, respectively. The coefficients of $A'(z)$, $B'(z)$, $C'(z)$, and $D'(z)$ are chosen such that

$$T_+(z^{-1})B(z^{-1}) - A(z^{-1}) = O(z^{-(p1+q1+1)}),$$
$$T_-(z)D(z) - C(z) = O(z^{p2+q2+1}),$$

where

$$T_+(z^{-1}) = ct_0 + \sum_{n=1}^{\infty} t_n z^{-n},$$
$$T_-(z) = (1-c)t_0 + \sum_{n=1}^{\infty} t_{-n} z^{n},$$

with given c. We then construct the preconditioner $\tilde{F}'_N$ with respect to $\tilde{T}'(z)$ $(= A'(z^{-1})D'(z) + B'(z^{-1})C'(z))$ or, equivalently, use F'_N $(= (L'_b)^{-1}\tilde{F}'_N(U'_d)^{-1})$ as preconditioner for T_N. Since $T(z) \neq T'(z)$, the eigenvalues of $(F'_N)^{-1}T_N$ are not repeated at, but are clustered around, 1. Also, we find from our experiment that small values of p_1, q_1, p_2, and q_2 such as 2, 3, or 4 are sufficient to give good performance (see Test Problems 3 and 6 in §5).

4. Comparison of factorization and circulant preconditioners. Various preconditioners in circulant form have been proposed for symmetric Toeplitz matrices [6], [8], [18], [20], [30]. All these preconditioners can be inverted effectively via fast transform algorithms with $O(N \log N)$ operations. This preconditioning technique can be easily generalized to nonsymmetric Toeplitz matrices. In the following, we discuss the generalization of the preconditioner $K_{1,N}$ [20] proposed by the authors to the nonsymmetric case.

Let T_N be an $N \times N$ Toeplitz matrix

$$T_N = \begin{bmatrix} t_0 & t_{-1} & \cdot & t_{-(N-2)} & t_{-(N-1)} \\ t_1 & t_0 & t_{-1} & \cdot & t_{-(N-2)} \\ \cdot & t_1 & t_0 & \cdot & \cdot \\ t_{N-2} & \cdot & \cdot & \cdot & t_{-1} \\ t_{N-1} & t_{N-2} & \cdot & t_1 & t_0 \end{bmatrix}.$$

We define a $2N \times 2N$ circulant matrix using elements of T_N as

$$R_{2N} = \begin{bmatrix} T_N & \triangle T_N \\ \triangle T_N & T_N \end{bmatrix}, \tag{4.1}$$

where

$$\triangle T_N = \begin{bmatrix} t_N & t_{N-1} & \cdot & t_2 & t_1 \\ t_{-(N-1)} & t_N & t_{N-1} & \cdot & t_2 \\ \cdot & t_{-(N-1)} & t_N & \cdot & \cdot \\ t_{-2} & \cdot & \cdot & \cdot & t_{N-1} \\ t_{-1} & t_{-2} & \cdot & t_{-(N-1)} & t_N \end{bmatrix}.$$

Since the augmented circulant system

$$\begin{bmatrix} T_N & \triangle T_N \\ \triangle T_N & T_N \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{b} \end{bmatrix}$$

is equivalent to

$$(T_N + \triangle T_N)\mathbf{x} = \mathbf{b},$$

the $(T_N + \triangle T_N)^{-1}\mathbf{b}$ can be computed efficiently via fast Fourier transform (FFT) so that

$$K_N = T_N + \triangle T_N$$

can be used as a preconditioner for T_N. Note that K_N is also a circulant matrix.

When T_N is a symmetric Toeplitz matrix generated by a positive function in the Wiener class, it can be proved [7], [20] that the eigenvalues of the circulant-preconditioned Toeplitz matrix are clustered around 1 except at a finite number of outliers. When T_N is additionally rational of order (p, q), the eigenvalues of $K_N^{-1}T_N$ are clustered between $(1 - \epsilon_K, 1 + \epsilon_K)$ except at $\alpha_K = 2\max(p, q)$ outliers, where $\epsilon_K = O(|t_N|)$ [22], [31]. A special case of Theorem 3 is that when T_N is a symmetric rational matrix, we have $r = s = w$ and $\alpha_F = \max(p, q) = \frac{1}{2}\alpha_K$. A more general result applicable to the nonsymmetric case has been obtained under the following two conditions:

$$\sum_{-\infty}^{\infty} |t_n| \leq B_T < \infty, \tag{4.2}$$

$$|T(e^{i\theta})| = \left|\sum_{-\infty}^{\infty} t_n e^{-in\theta}\right| \geq \mu_T > 0 \quad \forall \theta. \tag{4.3}$$

THEOREM 4. *Let T_N be a rational Toeplitz matrix generated by $T(z)$ of order (p_1, q_1, p_2, q_2) as given by* (3.2), *and satisfying* (4.2) *and* (4.3). *For sufficiently large N, the preconditioned Toeplitz matrix $K_N^{-1}T_N$ has the following two properties:*

$P(1)$: The number of outliers is at most $\alpha_K = 2\min(r, s)$.

$P(2)$: There are at least $N - \eta$ eigenvalues confined in the disk centered at 1 *with radius $\epsilon_K = O(|t_N| + |t_{-N}|)$.*

Proof. See [23] for the proof. □

The spectral properties of $F_N^{-1}T_N$ and $K_N^{-1}T_N$ for rational T_N are compared as follows. One main difference is that the eigenvalues except at outliers are exactly repeated at 1 for $F_N^{-1}T_N$ but only clustered around 1 for $K_N^{-1}T_N$, i.e., $\epsilon_K \geq \epsilon_F = 0$. Another difference is the number of outliers that are, by definition, the eigenvalues not converging to 1 for asymptotically large N. Asymptotically, ϵ_K converges to 0 and the CGS method with preconditioners F_N and K_N converges in at most $\alpha_F + 1$ and $\alpha_K + 1$ iterations, respectively. For finite N, $\epsilon_K \neq 0$ and the performance of K_N are determined by both the number of the outliers α_K and the clustering radius ϵ_K. Although it may happen that $\alpha_K < \alpha_F$, the MPLU preconditioner, in general, provides a faster or a comparable convergence rate since $\epsilon_K \geq \epsilon_F = 0$.

The preconditioning step $F_N^{-1}\mathbf{r}$ can be accomplished with $O(N)$ operations by permutation, forward- and back-substitution, since F_N is a product of a shift matrix, lower- and upper-triangular banded Toeplitz matrices. In comparison, the preconditioning step $K_N^{-1}\mathbf{r}$ requires $O(N \log N)$ operations via FFT. Hence, in terms of computational complexity per iteration, preconditioner F_N is slightly better. However, note that $F_N^{-1}\mathbf{r}$ must be implemented sequentially, whereas $K_N^{-1}\mathbf{r}$ can be easily parallelized via the parallelism provided by FFT.

5. Numerical results. Our numerical experiments include both SPD and nonsymmetric Toeplitz matrices with banded, rational, and nonrational generating sequences. The SPD problems are solved by the PCG method. For nonsymmetric systems, there exist numerous iterative algorithms for their solution [2], [27]. As suggested by [25], we applied the preconditioned version of three iterative methods, i.e., CGN, GMRES, and CGS, for our numerical experiments. We observed that GMRES and CGS converge faster than CGN, and that CGS outperforms GMRES by a factor of 1 to 2 for all test problems. Since our focus is on the preconditioners rather than the iterative methods, only results solved by the CGS iteration are reported. All experiments are performed with $N = 32$, $\mathbf{b} = (1, \cdots, 1)^T$, and zero initial guess.

TEST PROBLEM 1 (symmetric banded Toeplitz system). The generating function is

$$T(z) = z^{-4} + 3z^{-3} + 4z^{-2} + 7z^{-1} + 11 + 7z + 4z^2 + 3z^3 + z^4.$$

The convergence history of the PCG method with preconditioners F_N and K_N is plotted in Fig. 1. We clearly see that the 2-norm of the residual is significantly reduced in four iterations for both F_N and K_N and that F_N performs slightly better than K_N. We point out that $F_N^{-1}T_N$ and $K_N^{-1}T_N$ have four and eight outliers, respectively. However, for this test problem, the outliers of $K_N^{-1}T_N$ are related in pairs and it takes only $\frac{1}{2}\alpha_K$ iterations to eliminate these α_K outliers. A similar kind of convergence behavior for $K_N^{-1}T_N$ was reported in [20]. In general, preconditioners F_N and K_N have a similar performance for symmetric banded Toeplitz matrices.

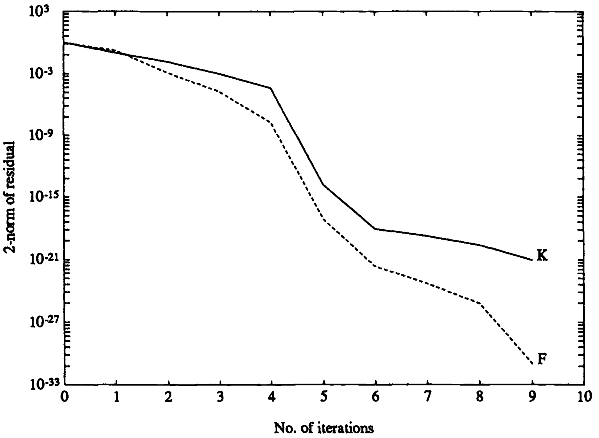


FIG. 1. *The convergence history of the* PCG *method with preconditioners* F_N *and* K_N *for Test Problem* 1.

TEST PROBLEM 2 (symmetric rational Toeplitz system). The generating function is

$$T(z) = \frac{(1-0.2z^{-1})(1+0.3z^{-1})(1-0.5z^{-1})}{(1-0.3z^{-1})(1+0.5z^{-1})(1-0.7z^{-1})} + \frac{(1-0.2z)(1+0.3z)(1-0.5z)}{(1-0.3z)(1+0.5z)(1-0.7z)}.$$

Since T_N is symmetric ($r = s = w = 3$), the eigenvalues of $F_N^{-1}T_N$ are repeated at 1 except at three outliers (see Theorem 3), and the eigenvalues of $K_N^{-1}T_N$ are clustered around 1 except at six outliers (see Theorem 4). The convergence history of the PCG method with preconditioners F_N and K_N is plotted in Fig. 2. Since $F_N^{-1}T_N$ has four distinct eigenvalues, the PCG method with preconditioner F_N converges in four iterations. However, although $K_N^{-1}T_N$ has six outliers, it only requires three iterations to eliminate the outliers. The convergence rate after the first three iterations depends on the clustering radius ϵ_K. It is clear that preconditioner F_N performs better than preconditioner K_N.

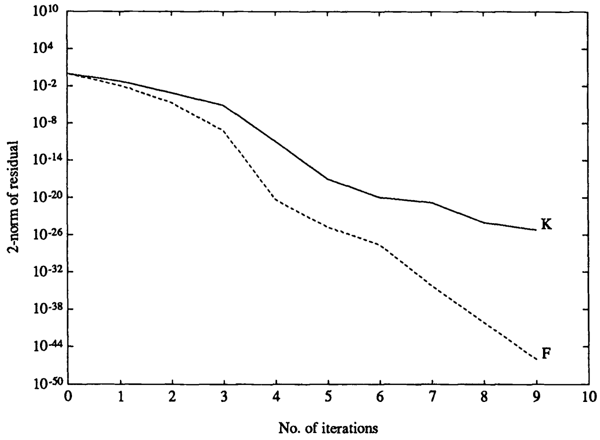


FIG. 2. *The convergence history of the* PCG *method with preconditioners F_N and K_N for Test Problem* 2.

TEST PROBLEM 3 (symmetric nonrational Toeplitz system). The generating sequence is

$$t_n = \begin{cases} 2, & n = 0, \\ 1/(1 + |n|), & n \neq 0, \end{cases}$$

and the corresponding generating function is

$$T(z) = T_+(z^{-1}) + T_+(z),$$

where

$$T_+(z^{-1}) = \sum_{n=0}^{\infty} \frac{z^{-n}}{1+n}.$$

Consider the *Padé* approximant of order (p, q), i.e., $A'_p(z^{-1})/B'_q(z^{-1})$, to $T_+(z^{-1})$. Preconditioners $F_{p,q,N}$ are then constructed with respect to

$$T'_{p,q}(z) = \frac{A'_p(z^{-1})}{B'_q(z^{-1})} + \frac{A'_p(z)}{B'_q(z)}.$$

In our experiment, (p, q) is chosen to be $(3, 3)$ and $(4, 4)$. The convergence history of the PCG method with preconditioners $F_{3,3,N}$, $F_{4,4,N}$, and K_N is plotted in Fig. 3. All these preconditioners converge at a similar rate.

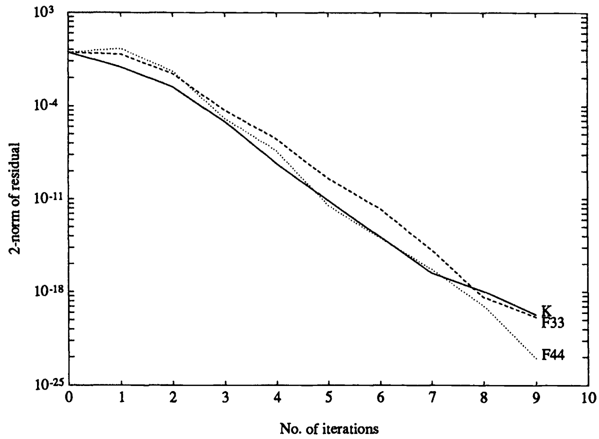


FIG. 3. *The convergence history of the* PCG *method with preconditioners* $F_{3,3,N}, F_{4,4,N}$*, and* K_N *for Test Problem* 3.

TEST PROBLEM 4 (nonsymmetric banded Toeplitz system). The generating function is

$$T(z) = -z^{-3} + 2z^{-2} + 9z^{-1} + 4 - 2z - 3z^2 + z^3,$$

so that T_N is a banded Toeplitz matrix with lower bandwidth $r = 3$ and upper bandwidth $s = 3$. Note also that $T(z)$ has $w = 4$ zeros outside the unit circle. The convergence history of the CGS method with preconditioners F_N and K_N is plotted in Fig. 4. According to Theorem 2, $F_N^{-1}T_N$ has three eigenvalues different from 1 and, consequently, the CGS method with preconditioner F_N converges in four iterations. According to Theorem 4, $K_N^{-1}T_N$ has six eigenvalues not equal to 1 so that the CGS method with preconditioner K_N converges in seven iterations. We see clearly that the CGS method with preconditioner F_N converges faster.

TEST PROBLEM 5 (nonsymmetric rational Toeplitz system). The generating function is

$$T(z) = \frac{(1-0.2z^{-1})(1+0.3z^{-1})(1-0.5z^{-1})}{(1-0.7z^{-1})(1+0.5z^{-1})} + \frac{1+2z}{(1.5-z)(2+z)(2-z)}.$$

We can transform T_N into a banded matrix with $r = s = w = 3$. From Theorems 3 and 4, we know that $F_N^{-1}T_N$ has only three eigenvalues not equal to 1 and the eigenvalues of $K_N^{-1}T_N$ are clustered around 1 except six outliers. The convergence history of the CGS method with preconditioners F_N and K_N is plotted in Fig. 5. The CGS method with preconditioner F_N performs better.

TEST PROBLEM 6 (nonsymmetric nonrational Toeplitz system). Let T_N be a nonsymmetric Toeplitz matrix with generating sequence

$$t_n = \begin{cases} 1/\log(2-n), & n \leq -1, \\ 1/\log(2-n) + 1/(1+n), & n = 0, \\ 1/(1+n), & n \geq 1. \end{cases}$$

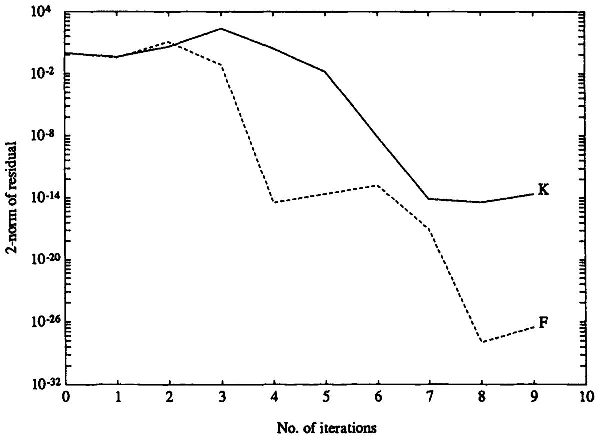


FIG. 4. *The convergence history of the* CGS *method with preconditioners* F_N *and* K_N *for Test Problem* 4.

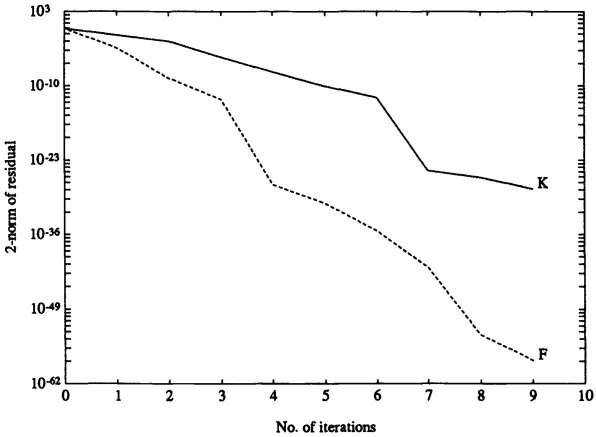


FIG. 5. *The convergence history of the* CGS *method with preconditioners* F_N *and* K_N *for Test Problem* 5.

The corresponding causal and anticausal generating functions can be written as

$$T_+(z^{-1}) = \sum_{n=0}^{\infty} \frac{z^{-n}}{1+n},$$

$$T_-(z) = \sum_{n=0}^{\infty} \frac{z^n}{\log(2+n)}.$$

Let the *Padé* approximants of order (p,q), to $T_+(z^{-1})$ and $T_-(z)$, be $A'_p(z^{-1})/B'_q(z^{-1})$ and $C'_p(z)/D'_q(z)$, respectively. We construct preconditioner $F'_{p,q,N}$ for

$$T'_{p,q}(z) = \frac{A'_p(z^{-1})}{B'_p(z^{-1})} + \frac{C'_q(z)}{D'_q(z)},$$

TABLE 2
Numbers of iterations required for the CGS *method.*

N	$F_{2,2,N}$	$F_{3,3,N}$	$F_{4,4,N}$	K_N
32	6	5	5	11
64	8	7	6	11
128	9	8	7	13

with $p = q = 2, 3, 4$. The convergence history of the CGS method with preconditioners $F_{p,q,N}$ and K_N is plotted in Fig. 6. To understand the asymptotical behavior of the preconditioned CGS method, we also performed experiments for this test problem with $N = 64, 128$. The numbers of iterations required with preconditioners $F_{p,q,N}$, $p = q = 2, 3, 4$, and K_N satisfying $||\mathbf{b} - T_N\mathbf{x}||_2 < 10^{-15}$ are summarized in Table 2 for different N. Note that the numbers of iterations required for all preconditioners increase slightly as N becomes larger. However, preconditioners $F_{p,q,N}$, $p = q = 2, 3, 4$ perform better than preconditioner K_N.

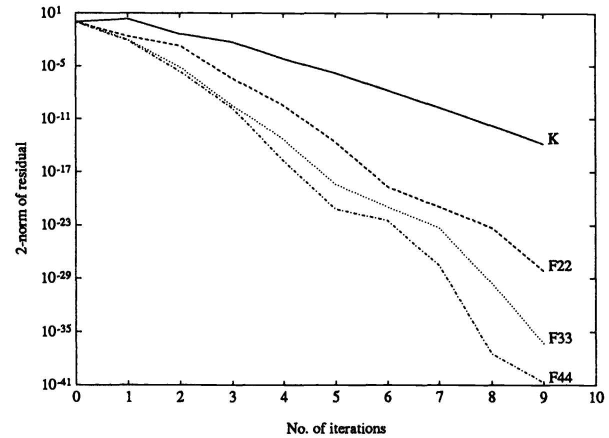


FIG. 6. *The convergence history of the* CGS *method with preconditioners* $P_{p,q,N}, p = q = 2, 3, 4$, *and* K_N *for Test Problem* 6.

6. Conclusion. In this paper, we applied the minimum-phase factorization technique to Toeplitz generating functions and obtained a new Toeplitz preconditioner called the MPLU preconditioner. This preconditioning technique is applicable to both banded and full Toeplitz matrices. We characterized the spectral properties of the MPLU preconditioned Toeplitz matrices, and showed that most of their eigenvalues are repeated exactly at unity for rational Toeplitz matrices. Thus an $N \times N$ rational Toeplitz system can be solved by preconditioned iterative methods with $O(N)$ complexity. We also demonstrate the superior performance of the MPLU preconditioner over another Toeplitz preconditioner in circulant form with several numerical examples, including both rational and nonrational cases.

Although our discussion on the MPLU factorization preconditioner has primarily focused on real nonsymmetric Toeplitz systems, its application to complex nonhermitian Toeplitz systems can be generalized in a straightforward way. However, the MPLU factorization preconditioning technique cannot be easily extended to higher-dimensional Toeplitz systems, such as block Toeplitz matrices. This is due to the absence of the fundamental theorem of algebra for multivariate polynomials. In contrast, higher-dimensional Toeplitz matrices can be preconditioned with higher-dimensional circulant matrices. See [21] for the two-dimensional case. Another limitation of the MPLU preconditioner is that it is not as easily parallelizable as the preconditioners in circulant form.

REFERENCES

[1] G. S. AMMAR AND W. B. GRAGG, *Superfast solution of real positive definite Toeplitz systems*, SIAM J. Matrix Anal. Appl., 9 (1988), pp. 61–76.

[2] S. F. ASHBY, T. A. MANTEUFFEL, AND P. E. SAYLOR, *A taxonomy for conjugate gradient methods*, SIAM J. Numer. Anal., 27 (1990), pp. 1542–1568.

[3] R. R. BITMEAD AND B. D. ANDERSON, *Asymptotically fast solution of Toeplitz and related systems of equations*, Linear Algebra Appl., 34 (1980), pp. 103–116.

[4] R. P. BRENT, F. G. GUSTAVSON, AND D. Y. YUN, *Fast solution of Toeplitz systems of equations and computations of Padé approximations*, J. Algorithms, 1 (1980), pp. 259–295.

[5] J. R. BUNCH, *Stability of methods for solving Toeplitz systems of equations*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 349–364.

[6] R. H. CHAN, *Circulant preconditioners for Hermitian Toeplitz system*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 542–550.

[7] R. H. CHAN AND G. STRANG, *Toeplitz equations by conjugate gradients with circulant preconditioner*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 104–119.

[8] T. F. CHAN, *An optimal circulant preconditioner for Toeplitz systems*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 766–771.

[9] T. F. CHAN AND P. C. HANSEN, *A look-ahead Levinson algorithm for indefinite Toeplitz systems*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 490–506.

[10] ———, *A look-ahead Levinson algorithm for general Toeplitz systems*, Tech. Report CAM 90-11, Department of Mathematics, University of California, Los Angeles, CA, 1990; IEEE Trans. Signal Process, to appear.

[11] G. CYBENKO, *The numerical stability of the Levinson–Durbin algorithm for Toeplitz systems of equations*, SIAM J. Sci. Statist. Comput., 1 (1980), pp. 303–319.

[12] F. DE HOOG, *A new algorithm for solving Toeplitz systems of equations*, Linear Algebra Appl., 88/89 (1987), pp. 123–138.

[13] P. DELSARTE AND Y. V. GENIN, *The split Levinson algorithm*, IEEE Trans. Acoust. Speech Signal Process., ASSP-34 (1986), pp. 470–478.

[14] B. W. DICKINSON, *Efficient solution of linear equations with banded Toeplitz matrices*, IEEE Trans. Acoust. Speech Signal Process., ASSP-27 (1979), pp. 421–422.

[15] ———, *Solution of linear equations with rational Toeplitz matrices*, Math. Comp., 34 (1980), pp. 227–233.

[16] W. B. GRAGG, *The Padé table and its relation to certain algorithms of numerical analysis*, SIAM Rev., 14 (1972), pp. 1–63.

[17] M. R. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 409–436.

[18] T. HUCKLE, *Circulant and skew-circulant matrices for solving Toeplitz matrix problems*, in Copper Mountain Conf. Iterative Methods, Copper Mountain, CO, 1990; SIAM J. Matrix Anal. Appl., 13 (1992), pp. 746–762.

[19] T. KAILATH, A. VIEIRA, AND M. MORF, *Inverse of Toeplitz operators, innovations, and orthogonal polynomials*, SIAM Rev., 20 (1978), pp. 106–119.

[20] T. K. KU AND C. J. KUO, *Design and analysis of Toeplitz preconditioners*, Tech. Report 155, Signal and Image Processing Institute, University of Southern California, Los Angeles, CA, May 1990; IEEE Trans. Signal Process., 14 (1992), pp. 129–141.

[21] ———, *On the spectrum of a family of preconditioned block Toeplitz matrices*, Tech. Report 164, Signal and Image Processing Institute, University of Southern California, Los Angeles, CA, Nov. 1990; SIAM J. Sci. Statist. Comput., 13 (1992), pp. 948–966.

[22] ———, *Spectral properties of preconditioned rational Toeplitz matrices*, Tech. Rep. 163, Signal and Image Processing Institute, University of Southern California, Los Angeles, CA, Sep. 1990; SIAM J. Matrix Anal. Appl., 14 (1993), to appear.

[23] ———, *Spectral properties of preconditioned rational Toeplitz matrices: The nonsymmetric case*, Tech. Report 175, Signal and Image Processing Institute, University of Southern California, Los Angeles, CA, Apr. 1991; SIAM J. Matrix Anal. Appl., 14 (1993), to appear.

[24] N. LEVINSON, *The Wiener* RMS *error criterion in filter design and prediction*, J. Math. Phys., 25 (1947), pp. 261–278.

[25] N. M. NACHTIGAL, S. C. REDDY, AND L. N. TREFETHEN, *How fast are nonsymmetric matrix iterations?*, in Copper Mountain Conf. Iterative Methods, Copper Mountain, CO, 1990; SIAM J. Matrix Anal. Appl., 13 (1992), pp. 778–795.

[26] A. V. OPPENHEIM AND R. W. SCHAFER, *Discrete-Time Signal Processing*, Prentice–Hall, Englewood Cliffs, NJ, 1989.

[27] Y. SAAD, *Krylov subspace methods on supercomputers*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 1200–1232.

[28] Y. SAAD AND M. H. SCHULTZ, GMRES: *A generalized minimum residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.

[29] P. SONNEVELD, CGS, *A fast Lanczos-type solver for nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 36–52.

[30] G. STRANG, *A proposal for Toeplitz matrix calculations*, Stud. Appl. Math., 74 (1986), pp. 171–176.

[31] L. N. TREFETHEN, *Approximation theory and numerical linear algebra*, in Algorithms for Approximation II, M. Cox and J. C. Mason, eds., Chapman, London, 1988.

[32] W. F. TRENCH, *An algorithm for the inversion of finite Toeplitz matrices*, J. Soc. Indust. Appl. Math., 21 (1964), pp. 515–523.

[33] ———, *Inversion of Toeplitz band matrices*, Math. Comp., 28 (1974), pp. 1089–1095.

[34] ———, *Solution of systems with Toeplitz matrices generated by rational functions*, Linear Algebra Appl., 74 (1986), pp. 191–211.

[35] ———, *Toeplitz systems associated with the product of a formal Laurent series and a Laurent polynomial*, SIAM J. Matrix Anal. Appl., 9 (1988), pp. 181–193.

[36] S. ZOHAR, *The solution of a Toeplitz set of linear equations*, J. ACM, 21 (1974), pp. 272–276.

SIAM J. Sci. Stat. Comput.
Vol. 13, No. 6, pp. 1488–1501, November 1992

DERIVATION OF EFFICIENT, CONTINUOUS, EXPLICIT RUNGE–KUTTA METHODS*

BRYNJULF OWREN† AND MARINO ZENNARO‡

Abstract. Continuous, explicit Runge–Kutta methods with the minimal number of stages are considered. These methods are continuously differentiable if and only if one of the stages is the FSAL evaluation. A characterization of a subclass of these methods is developed for orders 3, 4, and 5. It is shown how the free parameters of these methods can be used either to minimize the continuous truncation error coefficients or to maximize the stability region. As a representative for these methods the fifth-order method with minimized error coefficients is chosen, supplied with an error estimation method, and analysed by using the DETEST software. The results are compared with a similar implementation of the Dormand–Prince 5(4) pair with interpolant, showing a significant advantage in the new method for the chosen problems.

Key words. Runge–Kutta, interpolant, continuous, optimal

AMS(MOS) subject classification. 65L05

1. Introduction.

We consider the first-order system of differential equations

$$y'(x) = f(x, y(x)), \quad y(x_0) = y_0, \quad x_0 \le x \le x_e, \tag{1}$$

where $f : \mathbf{R} \times \mathbf{R}^m \to \mathbf{R}^m$ defines m generally nonlinear equations, and y_0 is the m-vector of initial values. We are interested in a continuous approximation to the solution $y(x)$ for $x \in [x_0, x_e]$. A possible way to obtain such an approximation is to apply a continuous, explicit Runge–Kutta method (CERK method) as proposed by the authors in [16]. One may expect such a method to be appropriate in the case when (1) is nonstiff. CERK methods may also be used for solving certain types of functional differential equations, like delay differential equations, for which retarded arguments in the right-hand side of (1) might be estimated by continuous extensions over previous subintervals. The continuous approximation $u(x)$ is obtained on $[x_0, x_e]$ by using a mesh $\{x_0 < x_1 < \cdots < x_N = x_e\}$ and computing polynomials $u_{n+1}(x_n + \theta h_n)$, $n = 0, \ldots, N-1$, such that $u(x) = u_{n+1}(x_n + \theta h_n)$ for $x_n \le x = x_n + \theta h_n \le x_{n+1}$ where $h_n = x_{n+1} - x_n$. The continuity assumption on $u(x)$ requires that $u_n(x_n) = u_{n+1}(x_n) := y_n$, $n = 1, \ldots, N-1$. The general form of u_{n+1} is

$$K_i = f\left(x_n + c_i h_n, y_n + h_n \sum_{j=1}^{i-1} a_{ij} K_j\right), \qquad i = 1, \ldots, s,$$

$$u_{n+1}(x_n + \theta h_n) = y_n + h_n \sum_{i=1}^{s} b_i(\theta) K_i, \qquad \theta \in [0, 1],$$

where $b_i(\theta)$, $i = 1, \ldots, s$, are polynomials of degree $\le d$ for some positive integer d. We shall also require $c_i = \sum_{j=1}^{i-1} a_{ij}$, and $b_i(0) = 0$ for $i = 1, \ldots, s$; the last condition is necessary for continuity. Henceforth we shall denote by A the strictly lower triangular matrix defined by the coefficients a_{ij}. Observe that a conventional Runge–Kutta

* Received by the editors November 26, 1990; accepted for publication (in revised form) October 11, 1991.

† Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A4.

‡ Dipartimento di Matematica Pura ed Applicata, Università dell'Aquila, I–67100 L'Aquila, Italy.

method is obtained by putting $y_{n+1} = u_{n+1}(x_n + h_n)$. In fact, a CERK method is equivalent to a Runge–Kutta method supplied with an interpolant. On surveying the literature one finds such interpolants for most of the commonly used Runge–Kutta formulas, e.g., Shampine [17], [18]; Dormand and Prince [6], [7]; Calvo, Montijano, and Rández [4]; and Horn [13]. Enright, Jackson, Nørsett, and Thomsen [8] provides a general technique for constructing interpolants to a Runge–Kutta formula while Zennaro [20] discusses natural continuous extensions of Runge–Kutta methods which are especially suited for functional differential equations when certain restrictions are imposed on the mesh. In a recent paper Verner [19] elaborates on differentiable interpolants of higher order.

An important issue in many of these papers has been whether or not the continuous approximation should yield the same order of consistency in the interior of the step as at the end-points. Following [16] we shall define the uniform order, or simply the order of a CERK method, as the greatest integer p for which

$$\max_{0\leq\theta\leq 1} |y_{n+1}(x_n + \theta h_n) - u_{n+1}(x_n + \theta h_n)| = O(h_n^{p+1}) \tag{2}$$

where $y_{n+1}(x)$ is the local solution to the initial value problem $y'_{n+1}(x) = f(x, y_{n+1}(x))$, $y_{n+1}(x_n) = y_n$. Here $|\cdot|$ can be any norm on $\mathbf{R}^m$. Of course, the order q at the end-points satisfies $q \geq p$ and in this paper we shall not impose the possible additional requirement that $q > p$ for any of our methods. We shall always require (see [16]) that the degree d of the polynomials $b_i(\theta)$, $i = 1, \ldots, s$, satisfies $d \leq p$. Again, according to [16], for each order p one may define the order barrier as the smallest number of stages CEN(p) that a CERK method satisfying (2) can have. These order barriers were derived for $p \leq 5$. It was found that CEN(1) = 1, CEN(2) = 2, CEN(3) = 4, CEN(4) = 6, and CEN(5) = 8. The first examples of fifth-order CERK methods with only eight stages were developed. It should be made clear that these new continuous methods with the minimal number of stages may only be expected to be cheap when the continuous approximation is required along the entire interval of integration. In some applications the continuous approximation is needed only occasionally, e.g., when it is used to locate discontinuities; see, e.g., [9]. In such cases it is recommended that one use a discrete method that can be supplied with a continuous extension, possibly at more than the minimal cost.

In this paper, we consider CERK methods with CEN(p) stages for $p = 3, 4, 5$ with the additional property that they are $C^{(1)}$ continuous. We give a complete recipe for the construction of some if not all such methods, and we present pairs of formulas with optimized error constants and show how the regions of absolute stability may be optimized. Finally, we present some results based on tests made with the DETEST package [10]. These tests are made with a fifth-order representative of the CERK methods described in this paper, displaying the properties of the underlying discrete method.

2. Preliminary results. Henceforth we shall make extensive use of the theory of rooted trees and order conditions developed by Butcher [1], [2]. From [16] we find the continuous version of the order conditions

$$\sum_{j=1}^{s} b_j(\theta)\Phi_j(t) = \frac{\theta^{\rho(t)}}{\gamma(t)} \quad \text{for all trees } t \text{ such that } \rho(t) \leq p, \tag{3}$$

where $\Phi_j(t)$ is the jth elementary weight for the tree t, $\rho(t)$ is the order of t, and $\gamma(t)$ is a coefficient depending on the tree t. For each $r \geq 1$, let n_r be the number of trees such that $\rho(t) = r$. Thus a CERK method of order p must satisfy N_p conditions where $N_p = \sum_{r=1}^{p} n_r$. We number the N_p trees t increasingly in terms of $\rho(t)$, such that $\rho(t_i) > \rho(t_j)$ only if $i > j$. This ordering is not unique. By putting $z_j(\theta) := b_j'(\theta)$, $j = 1, \ldots, s$, (3) can be written as

$$(4) \qquad \sum_{j=1}^{s} \phi_{ij} z_j(\theta) = \frac{\rho(t_i)\theta^{\rho(t_i)-1}}{\gamma(t_i)}, \qquad i = 1, \ldots, N_p,$$

where $\phi_{ij} = \Phi_j(t_i)$. Moreover, by writing

$$z_j(\theta) = \sum_{k=0}^{p-1} z_{jk}\theta^k \quad \text{and} \quad \frac{\rho(t_i)\theta^{\rho(t_i)-1}}{\gamma(t_i)} = \sum_{l=0}^{p-1} q_{il}\theta^l,$$

and by defining the $N_p \times s$ matrix $\Phi := ((\phi_{ij}))$, the $s \times p$ matrix $Z := ((z_{jk}))$, and the $N_p \times p$ matrix $Q := ((q_{il}))$, (4) becomes

$$\Phi Z = Q.$$

The $N_p \times s$ matrix Φ depends on the $s \times s$ matrix A of the coefficients of the RK method, whereas the $N_p \times p$ matrix Q is independent of A. For convenience, we introduce the mappings

$$F_p : \bigcup_{s\geq 1} \mathcal{L}(\mathbf{R}^s, \mathbf{R}^s) \longrightarrow \bigcup_{s\geq 1} \mathcal{L}(\mathbf{R}^s, \mathbf{R}^{N_p}) \quad \text{such that } F_p(A) := \Phi,$$

and

$$G_p : \bigcup_{s\geq 1} \mathcal{L}(\mathbf{R}^s, \mathbf{R}^s) \longrightarrow \bigcup_{s\geq 1} \mathcal{L}(\mathbf{R}^{s+p}, \mathbf{R}^{N_p}) \quad \text{such that } G_p(A) := \Phi|Q,$$

where $\Phi|Q$ is the $N_p \times (s + p)$ matrix obtained by attaching the rows of Q to the rows of Φ. It was pointed out in [16] that a CERK method is of order p if and only if $\text{rank}(F_p(A)) = \text{rank}(G_p(A))$. These methods constitute a set

$$\begin{aligned}\mathcal{M}^p := \{A \in \textstyle\bigcup_{s\geq 1} \mathcal{L}(\mathbf{R}^s, \mathbf{R}^s) | \ & A \text{ is strictly lower triangular,} \\ & \text{rank}(F_p(A)) = \text{rank}(G_p(A))\}.\end{aligned}$$

Consider the subset of $\mathcal{M}^p$,

$$\mathcal{M}_*^p := \{A \in \mathcal{M}^p | \text{rank}(F_p(A)) = s\},$$

where s is the order of the matrix A. It was proved in [16] that a necessary condition for a CERK method of order p to have CEN(p) stages is that $A \in \mathcal{M}_*^p$. Thus, henceforth we shall only consider methods for which $A \in \mathcal{M}_*^p$.

3. Optimal $C^{(1)}$ approximations. A property possessed by all explicit Runge–Kutta methods is that the first stage of the step from x_{n+1} to x_{n+2} is given by $K_1 = f(x_{n+1}, y_{n+1})$. Many methods take advantage of this property by also using this stage in the previous step from x_n to x_{n+1}. In the literature, this reusable stage is sometimes referred to as the FSAL (first same as last) evaluation. Because the methods we consider are explicit, the FSAL evaluation cannot be involved in the end-point approximation, but it may be used to obtain an error estimate or for the continuous approximation. It turns out that there is a close connection between the reusable stage being included in the CERK method and the uniform approximation being continuously differentiable. We shall consider CERK methods where the last stage K_s is the FSAL evaluation, i.e., we impose the *stage reuse conditions*

$$(5) \qquad c_s = 1 \quad \text{and} \quad a_{sj} = b_j(1), \quad j = 1, \dots, s.$$

For the discussion that follows we need the following lemma.

LEMMA 1. *Let* $A \in \mathcal{M}^p$ *define a* CERK *method with stage reuse. Then* $\Phi := F_{p+1}(A) := ((\phi_{ij}))$ *is such that*

$$\phi_{is} = \frac{\rho(t_i)}{\gamma(t_i)}, \qquad i = 1, \dots, N_{p+1}.$$

Proof. The lemma is proved by induction on the row index. The result holds for $i = 1$ since the first row corresponds to the only tree, τ, of order 1 and since $\gamma(\tau) = \rho(\tau) = \phi_{1s}(\tau) = 1$. Then assume that the lemma is true for all i such that $i \leq n - 1$. The nth condition corresponds to a tree t_n which either has the form $[t_{n'}]$ for some tree $t_{n'}$ of order $\rho(t_n) - 1$ or the form $[t_{\nu_1}, \dots, t_{\nu_u}]$ for $u\,(\geq 2)$ trees t_{ν_i} where $1 \leq \rho(t_{\nu_i}) \leq \rho(t_n) - 2$ and $\rho(t_n) = 1 + \sum_{i=1}^{u} \rho(t_{\nu_i})$. In the latter case, with $t_{n_i} = [t_{\nu_i}]$, we immediately obtain from the definition of γ [3, p. 88] that

$$\gamma(t_n) = \rho(t_n) \prod_{i=1}^{u} \frac{\gamma(t_{n_i})}{\rho(t_{n_i})}$$

such that

$$\phi_{ns} = \prod_{i=1}^{u} \phi_{n_i s} = \prod_{i=1}^{u} \frac{\rho(t_{n_i})}{\gamma(t_{n_i})} = \frac{\rho(t_n)}{\gamma(t_n)}.$$

In the former case, we get

$$\phi_{ns} = \sum_{j=1}^{s-1} a_{sj} \phi_{n'j} = \sum_{j=1}^{s-1} b_j(1) \phi_{n'j} = 1/\gamma(t_{n'}) = \frac{\rho(t_n)}{\gamma(t_n)},$$

where we have applied the stage reuse conditions along with the order conditions at $\theta = 1$. Finally, observe that the induction works for all n such that $\rho(t_n) \leq p+1$ since we only used the order condition corresponding to $t_{n'}$ and $\rho(t_{n'}) = \rho(t_n) - 1$. □

With this result, the following theorem is now easy to prove.

THEOREM 2. *Let* $A \in \mathcal{M}_*^p$ *be a* CERK *method with stage reuse. Then the global continuous approximation is continuously differentiable.*

Proof. It is sufficient to prove that $u'(x_0) = K_1$ and that $u'(x_0 + h) = K_s$. By Lemma 1, the last column of $F_p(A)$ is equal to the right-hand side of (4) evaluated at $\theta = 1$. Moreover, the first column of $F_p(A)$ equals the right-hand side of (4) evaluated at $\theta = 0$. Since, by assumption, the columns of $F_p(A)$ are linearly independent, we obtain $z_i(1) = \delta_{is}$ and $z_i(0) = \delta_{i1}$ such that $u'(x_0) = K_1$ and $u'(x_0 + h) = K_s$. □

It is of interest to know whether there exist CERK methods with stage reuse having a total of CEN(p) stages. It is easy to prove that such methods cannot exist for $p \leq 2$ under the assumption that the degree d of the polynomial weights does not exceed p. We have not been able to answer this question for general $p > 2$, but we shall see that such methods exist for $p = 3, 4, 5$. During this discussion we shall sometimes impose some additional conditions, which we shall refer to as the simplifying assumptions; see, e.g., [3, p. 195],

$$\sum_{j=1}^{i-1} a_{ij}c_j = \frac{1}{2}c_i^2, \qquad i = 3, \ldots, s. \tag{6}$$

We begin by considering the case $p = 3$. By combining the four continuous order conditions with (5), we find that a third-order CERK method with four stages and stage reuse can be constructed in the following way:

- Choose c_2 and c_3 with $c_2 \neq 0$, $c_3 \neq 0$, $c_2 \neq c_3$, and $c_2 \neq \frac{2}{3}$ in order that $A \in \mathcal{M}_*^3$.
- Put $c_4 = 1$ and compute a_{32}, a_{42}, and a_{43} from the formulas

$$a_{32} = \frac{c_3(c_3 - c_2)}{c_2(2 - 3c_2)},$$
$$a_{42} = \frac{3c_3 - 2}{6c_2(c_3 - c_2)}, \qquad a_{43} = \frac{2 - 3c_2}{6c_3(c_3 - c_2)}.$$

- The continuous weights are then given by

$$\begin{aligned}
b_1(\theta) &= (1 - 2a_{41})\theta^3 + (3a_{41} - 2)\theta^2 + \theta, \\
b_2(\theta) &= -a_{42}(2\theta^3 - 3\theta^2), \\
b_3(\theta) &= -a_{43}(2\theta^3 - 3\theta^2), \\
b_4(\theta) &= \theta^3 - \theta^2.
\end{aligned}$$

Observe that this continuous extension of the three-stage discrete method above is nothing but the cubic Hermite interpolant based on the end-points x and $x + h$.

Considering order $p = 4$ we shall restrict ourselves to the methods derived in [16] having CEN(4) $= 6$ stages. These methods satisfy the simplifying assumptions (6) with $s = 6$ and, using the notation of [3], we let the row space of $G_4(A)$ be spanned by the rows corresponding to the order conditions arising from the trees

$$\tau,\ [\tau],\ [\tau^2],\ [_2\tau]_2,\ [\tau^3],\ [_3\tau]_3. \tag{7}$$

Combining these conditions with (5) we find that one may choose $c_2 \neq 0$ and c_3, c_4, c_5 nonzero and distinct. Then if $6c_3c_4 - 4(c_3 + c_4) + 3 \neq 0$, the remaining coefficients are uniquely determined. If $c_3 = \frac{1}{2}$ and $c_4 = 1$, there exists a one-parameter family of methods (with a_{54} arbitrary). Given c_2, c_3, c_4, c_5, one may use the following procedure to obtain a fourth-order CERK method with stage reuse having six stages.

- Put $c_6 = 1$, $a_{62} = 0$, and compute a_{63}, a_{64}, a_{65} from the linear system

$$\sum_{j=3}^{5} a_{6j}c_j^{k-1} = \frac{1}{k}, \qquad k = 2, 3, 4.$$

By assumption, there exists a unique solution.

- Consider the equation

$$12a_{65}a_{54}c_4(c_4 - c_3) = 1 - 2c_3,$$

arising from the condition corresponding to the tree $[_3\tau]_3$. If $a_{65} \neq 0$ this equation can be satisfied in any case and thereby a_{54} is uniquely determined. This requires that $6c_3c_4 - 4(c_3 + c_4) + 3 \neq 0$. If $a_{65} = 0$, then $c_3 = \frac{1}{2}$ leading to $c_4 = 1$, in which case a_{54} is arbitrary.

- We obtain the remaining coefficients from the formulas

$$\begin{aligned}
a_{32} &= \frac{c_3^2}{2c_2},\\
a_{42} &= \frac{(3c_3 - 2c_4)c_4^2}{2c_2c_3}, \qquad a_{43} = \frac{(c_4 - c_3)c_4^2}{c_3^2},\\
a_{52} &= \frac{(3c_3 - 2c_5)c_5^2 + 6a_{54}c_4(c_4 - c_3)}{2c_2c_3},\\
a_{53} &= \frac{(c_5 - c_3)c_5^2 - a_{54}c_4(3c_4 - 2c_3)}{c_3^2}.
\end{aligned}$$

The continuous weights are found by solving the linear 6×6 system of equations arising from the order conditions corresponding to the trees (7).

Also for the order-five case we shall impose the simplifying assumptions (6) with $s = \mathrm{CEN}(5) = 8$ and the row space of $G_5(A)$ is assumed to be spanned by the rows corresponding to the trees

$$\tau,\ [\tau],\ [\tau^2],\ [_2\tau]_2,\ [\tau^3],\ [_3\tau]_3,\ [\tau^4],\ [_4\tau]_4. \tag{8}$$

For a detailed discussion of such fifth-order CERK methods with eight stages, see [16]. Now, combining these conditions with the stage reuse conditions (5) and the assumption $A \in \mathcal{M}_*^5$, one finds after some long but straightforward algebra that $c_2 \neq 0$, $c_3 \neq 0$, $c_6 \neq 0$, $c_7 \neq 0$, and $a_{54} \neq 0$ can be chosen subject to the constraints $c_6 \neq c_3$, $c_6 \neq 2c_3$, $c_7 \neq c_3$, $c_7 \neq 2c_3$, $c_7 \neq c_6$, $c_8 \neq 2c_3$, i.e., $c_3 \neq \frac{1}{2}$ and $\frac{2}{3}c_3^2 - \frac{3}{4}c_3 + \frac{1}{5} \neq c_6(c_3 - \frac{1}{2})^2$. The following procedure provides the remaining coefficients:

- Put $c_5 = c_4 = 2c_3$.
- Put $c_8 = 1$ and $a_{82} = 0$. Compute a_{86} and a_{87} from the formulas

$$a_{86} = -\frac{\frac{2}{3}c_3^2 - \frac{3}{4}c_3 + \frac{1}{5} - c_7(c_3 - \frac{1}{2})^2}{c_6(c_6 - c_3)(c_6 - 2c_3)(c_7 - c_6)}, \qquad a_{87} = \frac{\frac{2}{3}c_3^2 - \frac{3}{4}c_3 + \frac{1}{5} - c_6(c_3 - \frac{1}{2})^2}{c_7(c_7 - c_3)(c_7 - 2c_3)(c_7 - c_6)}.$$

Observe that $a_{87} = 0$ would violate the assumption $\frac{2}{3}c_3^2 - \frac{3}{4}c_3 + \frac{1}{5} \neq c_6(c_3 - \frac{1}{2})^2$.

- We may now compute a_{76} from the formula

$$a_{76} = \frac{\frac{1}{3}c_3^2 - \frac{1}{4}c_3 + \frac{1}{20}}{a_{87}c_6(c_6 - c_3)(c_6 - 2c_3)}.$$

- Now we turn to the sixth stage, which is independent of the stage reuse conditions

$$\begin{aligned}
a_{62} &= \frac{c_6^2(2c_3 - c_6)}{2c_2c_3}, \qquad a_{63} = \frac{c_6^2(c_6 - c_3)}{3c_3^2},\\
a_{64} &= \frac{c_6^2(c_6 - c_3)(2c_3 + a_{54} - c_6)}{12a_{54}c_3^2}, \qquad a_{65} = \frac{c_6^2(c_6 - c_3)(c_6 - 2c_3)}{12a_{54}c_3^2}.
\end{aligned}$$

- The remaining coefficients of the seventh stage are given by

$$a_{75} = \frac{c_7^2(c_7-c_3)(c_7-2c_3) - a_{76}c_6(c_6-c_3)(3c_6-4c_3)}{12a_{54}c_3^2},$$

$$a_{74} = \frac{\frac{1}{12}c_7^2(c_7-c_3)(c_7-2c_3) + a_{76}c_6(c_3(\frac{1}{2}c_7-\frac{1}{3}c_3) - c_6(\frac{1}{2}c_7-\frac{1}{3}c_6))}{c_3^2(c_7-2c_3)} - a_{75},$$

$$a_{73} = \frac{\frac{2}{3}c_7^2(c_7-c_3)(c_7-2c_3) - a_{76}c_6(\frac{4}{3}(c_6-4c_3)(c_6-\frac{1}{2}c_3) + c_6c_7)}{c_3^2(c_7-2c_3)} - 4(a_{74}+a_{75}).$$

a_{72} is obtained from (6).

- We next give formulas for the remaining coefficients of the eighth stage:

$$a_{85} = \frac{(1-c_3)(1-2c_3) - a_{86}f_6 - a_{87}f_7}{12a_{54}c_3^2},$$

where $f_i = 4c_i^3 - 9c_i^2c_3 + 4c_ic_3^2 + 2c_2c_3a_{i2}, \subset= 6,7$;

$$a_{84} = \frac{\frac{1}{4}(1-c_3)(1-2c_3) - (1-2c_3+2a_{54})a_{85}c_3^2 - a_{86}g_6 - a_{87}g_7}{(1-2c_3)c_3^2},$$

where $g_i = \frac{1}{4}c_i(3c_i-2c_3)(1-2c_3) - \frac{1}{4}c_i^2 + \frac{1}{3}(c_i^3 + a_{i2}c_2c_3)$, $i = 6,7$;

$$a_{83} = \frac{1 - c_3 - a_{86}c_6(3c_6-2c_3) - a_{87}c_7(3c_7-2c_3)}{c_3^2} - 8(a_{84}+a_{85}).$$

- The remaining coefficients are independent of the stage reuse conditions

$$a_{32} = \frac{c_3^2}{2c_2}, \quad a_{42} = -\frac{2c_3^2}{c_2}, \quad a_{43} = 4c_3,$$

$$a_{52} = \frac{2c_3(3a_{54}-c_3)}{c_2}, \qquad a_{53} = 4c_3 - 8a_{54}.$$

The polynomials $b_1(\theta), \ldots, b_8(\theta)$ are found by solving the linear 8×8 system of equations arising from the order conditions corresponding to the trees (8).

Remark. An interesting question is whether there exist methods among this fifth-order class such that their first six stages define a discrete RK formula of order five. This would clearly require $a_{87} = b_7(1) = 0$, a case which is excluded in the formulas above because of the assumption $\frac{2}{3}c_3^2 - \frac{3}{4}c_3 + \frac{1}{5} \neq c_6(c_3 - \frac{1}{2})^2$. However, it can be shown that this condition is also necessary for the existence of eight-stage fifth-order CERK methods with stage reuse based on the simplifying assumptions and the choice of linearly independent rows of $G_5(A)$ made above.

4. Error estimation. Variable step Runge–Kutta methods are usually supplied with an error estimation device. We shall be concerned with the strategy based on embedded formulas (see [12] for details) and we shall see how this strategy can be adapted to pairs of CERK methods. Such a pair will be denoted by CERK(p,q), which means that the integration is proceeded by a continuous method of order p as described in the previous section, while a discrete formula of order q is used to obtain an error estimate at the end-point of each step. It is obvious that we must require $p \neq q$ and it is customary to assume that $|p-q| = 1$, but it is not clear whether one should have $q > p$ or $p > q$. Most implementations of the discrete pairs proposed by Fehlberg [11] are of the former type, while in the methods of Dormand and Prince [5] the intention is to impose $p = q + 1$ (local extrapolation). We will pay most of our attention to the latter type of pairs, mainly for two reasons. First, we have the following negative result for the typical Fehlberg-type implementations.

PROPOSITION 3. *There exist no* CERK$(p, p+1)$ *pairs with* $s =$CEN (p) *stages.*

Proof. Assume that $s = \mathrm{CEN}(p)$ for a method given by the $s \times s$ matrix A. Then, since $\mathrm{rank}(F_p(A)) = s$, it is impossible to find two distinct sets of weights that satisfy the first N_p discrete-order conditions. □

The second reason is based on a recent result by Jackiewicz and Zennaro [14]. They find that given a CERK method of order p with $s = \mathrm{CEN}(p)$ stages, it is possible to obtain a two-step Runge–Kutta method of order $p + 1$ at no additional cost. Hence, one may use this two-step approximation of order $p + 1$ to obtain an error estimate. The authors have no experience with practical use of such estimates.

Having constructed a CERK method of order p, several possibilities remain for the construction of a $(p - 1)$ st-order discrete formula. We suggest that the final reusable stage be omitted from the error estimation formula, as this will cause rejected steps to cost only $s - 2$ function evaluations if properly implemented. We shall take advantage of the fact that all our pth-order methods with CEN(p) stages, $p = 3, 4, 5$, described in the previous section turn out to have an imbedded CERK method of order $p - 1$ with CEN$(p - 1)$ stages. We shall use this method evaluated at $\theta = 1$ as the error estimation method.

5. Minimization of the error constant.

For a CERK method of order p, the local truncation error is given by

$$|y(x_0 + \theta h) - u(x_0 + \theta h)| = \frac{h^{p+1}}{(p+1)!}\left|\sum_{i=N_p+1}^{N_{p+1}} e_i(\theta)\alpha_i F_i(x_0, y_0)\right| + \mathcal{O}(h^{p+2}),$$

where $F_i := F(t_i)$ is the elementary differential corresponding to the tree t_i, $\alpha_i := \alpha(t_i)$ is a positive integer weight corresponding to the tree t_i and $e_i(\theta)$, $i = N_p + 1, \dots, N_{p+1}$ are the *error polynomials* given by

$$e_i(\theta) = \theta^{\rho(t_i)} - \gamma(t_i)\sum_{j=1}^{s}\phi_{ij}b_j(\theta), \qquad i = N_p + 1, \dots, N_{p+1}.$$

By considering the structure of the matrix $\Phi = ((\phi_{ij}))$, it is easy to see that all the error polynomials have vanishing zeroth- and first-degree coefficients such that both $e_i(0)$ and $e_i'(0)$ are zero. Moreover, for the CERK methods of the previous section, the derivatives of the error polynomials also vanish at $\theta = 1$. We have the following proposition.

PROPOSITION 4. *Let* $A \in \mathcal{M}_*^p$ *be a* CERK *method with stage reuse. Then all the error polynomials* $e_i(\theta)$, $i = N_p + 1, \dots, N_{p+1}$, *satisfy* $e_i'(1) = 0$.

Proof. The derivatives of the error polynomials are given by

$$e_i'(\theta) = \rho(t_i)\theta^{\rho(t_i)-1} - \gamma(t_i)\sum_{j=1}^{s}\phi_{ij}z_j(\theta), \qquad i = N_p + 1, \dots, N_{p+1}.$$

As in the proof of Theorem 2 we must have $z_s(1) = 1$ and $z_1(1) = \cdots = z_{s-1}(1) = 0$ and by Lemma 1 it follows that $e_i'(1) = 0$. □

Following, e.g., Calvo, Montijano, and Rández [4] we shall attempt to make the error polynomials small in some sense. The literature is not consistent with regard to what norm should be used on this n_{p+1}-vector of polynomials. Calvo, Montijano, and Rández [4] minimize the quantity

$$g^* = \int_0^1 g(\theta)d\theta$$

over the free parameters, where

$$g(\theta) = \sqrt{\frac{\sum_{i=N_p+1}^{N_{p+1}}[e_i(\theta)\alpha(t_i)]^2}{\sum_{i=N_p+1}^{N_{p+1}}[e_i(1)\alpha(t_i)]^2}}. \tag{9}$$

Enright, Jackson, Nørsett, and Thomsen [8] consider bounds for the principal error term on the intervals $0 \le \theta \le 1$ and $0 \le \theta \le 2$. They use the norm

$$\max_{N_p+1 \le i \le N_{p+1}} \{ \max_{\theta \in [0,\theta_e]} |e_i(\theta)| \}$$

where $\theta_e = 1$ or $\theta_e = 2$. This latter case is of interest when the continuous method is to be used beyond the current step, e.g., for the purpose of handling discontinuities. We present here method pairs of orders 3, 4, and 5 where the free parameters have been chosen to minimize the numerator of (9). The optimization was done numerically, and the values found for the free parameters were approximated by rational numbers. The weights $\hat{y}_{n+1}$ in Figs. 1–3 are those of the error estimation method and have nothing to do with the underlying discrete method of the CERK method. Several numerical investigations conducted with the methods derived in this paper show that the vector of error polynomials tends to have one dominating component, which is the polynomial that corresponds to the tree $[_p\tau]_p$. Moreover, this error polynomial turns out to be independent or only weakly dependent on some of the free parameters of our optimal CERK methods. We illustrate this point by considering the optimal third-order CERK methods of the previous section. The error polynomial $e_8(\theta)$ corresponding to the tree $[_3\tau]_3$ of order four is given by

$$e_8(\theta) = \theta^2(\theta-2)^2,$$

$$\begin{array}{c|cccc}
0 & & & & \\
\frac{12}{23} & \frac{12}{23} & & & \\
\frac{4}{5} & -\frac{68}{375} & \frac{368}{375} & & \\
1 & \frac{31}{144} & \frac{529}{1152} & \frac{125}{384} & \\
\hline
\hat{y}_{n+1} & \frac{1}{24} & \frac{23}{24} & 0 & 0
\end{array}
\qquad
\begin{aligned}
b_1(\theta) &= \tfrac{41}{72}\theta^3 - \tfrac{65}{48}\theta^2 + \theta \\
b_2(\theta) &= -\tfrac{529}{576}\theta^3 + \tfrac{529}{384}\theta^2 \\
b_3(\theta) &= -\tfrac{125}{192}\theta^3 + \tfrac{125}{128}\theta^2 \\
b_4(\theta) &= \theta^3 - \theta^2
\end{aligned}$$

FIG. 1. *Optimal third-order* CERK *method with stage reuse.*

$$
\begin{array}{c|cccccc}
0 & & & & & & \\
\frac{1}{6} & \frac{1}{6} & & & & & \\
\frac{11}{37} & \frac{44}{1369} & \frac{363}{1369} & & & & \\
\frac{11}{17} & \frac{3388}{4913} & -\frac{8349}{4913} & \frac{8140}{4913} & & & \\
\frac{13}{15} & -\frac{36764}{408375} & \frac{767}{1125} & -\frac{32708}{136125} & \frac{210392}{408375} & & \\
1 & \frac{1697}{18876} & 0 & \frac{50653}{116160} & \frac{299693}{1626240} & \frac{3375}{11648} & \\
\hline
\hat{y}_{n+1} & \frac{101}{363} & 0 & -\frac{1369}{14520} & \frac{11849}{14520} & 0 & 0
\end{array}
$$

$$
\begin{aligned}
b_1(\theta) &= -\tfrac{866577}{824252}\theta^4 + \tfrac{1806901}{618189}\theta^3 - \tfrac{104217}{37466}\theta^2 + \theta \\
b_2(\theta) &= 0 \\
b_3(\theta) &= \tfrac{12308679}{5072320}\theta^4 - \tfrac{2178079}{380424}\theta^3 + \tfrac{861101}{230560}\theta^2 \\
b_4(\theta) &= -\tfrac{7816583}{10144640}\theta^4 + \tfrac{6244423}{5325936}\theta^3 - \tfrac{63869}{293440}\theta^2 \\
b_5(\theta) &= -\tfrac{624375}{217984}\theta^4 + \tfrac{982125}{190736}\theta^3 - \tfrac{1522125}{762944}\theta^2 \\
b_6(\theta) &= \tfrac{296}{131}\theta^4 - \tfrac{461}{131}\theta^3 + \tfrac{165}{131}\theta^2
\end{aligned}
$$

FIG. 2. *Optimal fourth-order* CERK *method with stage reuse.*

and hence is completely independent of the free parameters c_2 and c_3. For $0 \le \theta \le 1$ it attains the maximum value 1 at $\theta = 1$ and it can be shown that c_2 and c_3 can be chosen such that the three remaining error polynomials satisfy $|e_i(\theta)| < 0.05$, $\theta \in [0, 1]$. Consequently, one may suspect that the minima of the various error measures are flat and if the max-norm is used, the minimum is not likely to be unique.

Remark. Dormand and Prince attempt to minimize the denominator of (9) in their discrete pair RK5(4)7M [5]. If we use the same error measure for our discrete underlying formulas of the fifth-order methods, it turns out that we can only obtain an error at the end-point about three times the size of that of RK5(4)7M. The corresponding error for the optimized fifth-order method above is about four times that of RK5(4)7M.

6. Stability. When applied to ODEs, the methods of the previous sections will obviously have the stability characteristics of the underlying discrete method. The region of absolute stability of the methods of order four and five is influenced by the choice of the free parameters. We write the stability polynomials of pth-order CERK methods with CEN(p) stages and stage reuse in the form

$$p(z) = \sum_{k=0}^{\mathrm{CEN}(p)-1} \alpha_k \frac{z^k}{k!}$$

$$
\begin{array}{c|cccccccc}
0 & & & & & & & & \\
\frac{1}{6} & \frac{1}{6} & & & & & & & \\
\frac{1}{4} & \frac{1}{16} & \frac{3}{16} & & & & & & \\
\frac{1}{2} & \frac{1}{4} & -\frac{3}{4} & 1 & & & & & \\
\frac{1}{2} & -\frac{3}{4} & \frac{15}{4} & -3 & \frac{1}{2} & & & & \\
\frac{9}{14} & \frac{369}{1372} & -\frac{243}{343} & \frac{297}{343} & \frac{1485}{9604} & \frac{297}{4802} & & & \\
\frac{7}{8} & -\frac{133}{4512} & \frac{1113}{6016} & \frac{7945}{16544} & -\frac{12845}{24064} & -\frac{315}{24064} & \frac{156065}{198528} & & \\
1 & \frac{83}{945} & 0 & \frac{248}{825} & \frac{41}{180} & \frac{1}{36} & \frac{2401}{38610} & \frac{6016}{20475} & \\
\hline
\hat{y}_{n+1} & -\frac{1}{9} & 0 & \frac{40}{33} & -\frac{7}{4} & -\frac{1}{12} & \frac{343}{198} & 0 & 0
\end{array}
$$

$$
\begin{aligned}
b_1(\theta) &= \tfrac{596}{315}\,\theta^5 - \tfrac{4969}{819}\,\theta^4 + \tfrac{17893}{2457}\,\theta^3 - \tfrac{3292}{819}\,\theta^2 + \theta \\
b_2(\theta) &= 0 \\
b_3(\theta) &= -\tfrac{1984}{275}\,\theta^5 + \tfrac{1344}{65}\,\theta^4 - \tfrac{43568}{2145}\,\theta^3 + \tfrac{5112}{715}\,\theta^2 \\
b_4(\theta) &= \tfrac{118}{15}\,\theta^5 - \tfrac{1465}{78}\,\theta^4 + \tfrac{3161}{234}\,\theta^3 - \tfrac{123}{52}\,\theta^2 \\
b_5(\theta) &= 2\,\theta^5 - \tfrac{413}{78}\,\theta^4 + \tfrac{1061}{234}\,\theta^3 - \tfrac{63}{52}\,\theta^2 \\
b_6(\theta) &= -\tfrac{9604}{6435}\,\theta^5 + \tfrac{2401}{1521}\,\theta^4 + \tfrac{60025}{50193}\,\theta^3 - \tfrac{40817}{33462}\,\theta^2 \\
b_7(\theta) &= -\tfrac{48128}{6825}\,\theta^5 + \tfrac{96256}{5915}\,\theta^4 - \tfrac{637696}{53235}\,\theta^3 + \tfrac{18048}{5915}\,\theta^2 \\
b_8(\theta) &= 4\,\theta^5 - \tfrac{109}{13}\,\theta^4 + \tfrac{75}{13}\,\theta^3 - \tfrac{18}{13}\,\theta^2
\end{aligned}
$$

FIG. 3. *Optimal fifth-order* CERK *method with stage reuse.*

where $\alpha_0 = \cdots = \alpha_p = 1$. It can be shown that for our fourth-order methods we have $\alpha_5 = 5(1 - 2c_3)c_4$. Following [15] the largest disk of stability is obtained by choosing $\alpha_5 \approx 0.5806$; For example, the choice $c_3 = \frac{2}{5}$ and $c_4 = \frac{4}{7}$ leads to a nearly optimal stability region in the sense of [15]. The fifth-order methods yield

$$\alpha_6 = 6(-5c_3^2 + 2c_3) - 2c_6\beta, \qquad \alpha_7 = 14c_3c_6\beta$$

with $\beta = 20c_3^2 - 15c_3 + 3$.

Thus, for any $(\mu, \nu) \in \mathbf{R} \times \mathbf{R}$, there exist real c_3 and c_6 such that $\alpha_6 = \mu$ and $\alpha_7 = \nu$. So the stability polynomial is only restricted by the assumptions made in the construction of the CERK formulas. In particular, one may choose c_3 and c_6 such that $\alpha_6 \approx 0.7956$ and $\alpha_7 \approx 0.3305$ which, again according to [15], maximizes the region of absolute stability.

7. Numerical results with DETEST.

We have tested the fifth-order pair, henceforth denoted CM54, on some selected problems using the DETEST package [10]. We used absolute error control, attempting to control the local error at the end-point of each step. As reference we have performed the same tests with an identical implementation of the Dormand–Prince RK5(4)7M method [5], henceforth called DP54, with a continuous extension obtained at the additional cost of two stages per step; see, e.g., [4]. Hence, the effective cost per step is eight stages for the Dormand–Prince extension and seven stages for our fifth-order CERK method. In order to compare these two pairs, we have found it natural to use normalised efficiency, a feature of the DETEST software. Thus, instead of comparing the cost of the two pairs for a given tolerance, we make comparisons for a given *expected global accuracy* at the end-point of integration. This expected accuracy is obtained in terms of the tolerance by assuming a relation of the form

$$\text{global error} \approx C \cdot \text{TOL}^E,$$

where C and E are found by a least squares fit to the computed data. Piecewise linear interpolation then yields continuous extensions of the tabulated efficiency statistics. See [10] for a more detailed explanation.

We believe that the test problems chosen give a good idea of how the two methods perform with DETEST. On some of the omitted problems the discrepancy between equivalent tolerances (see tables) for the two methods were substantial or the least squares fit was too poor to be reliable.

TABLE 1
Efficiency statistics for DETEST *problem A4*, $y' = \frac{y}{4}(1 - \frac{y}{20})$, $y(0) = 1$.

Expected	Equiv. Tol		Steps		FCalls	
accuracy	DP54	CM54	DP54	CM54	DP54	CM54
10**-3	10**-2.50	10**-1.35	4	2	33	19
10**-4	10**-3.55	10**-2.38	4	3	37	31
10**-5	10**-4.61	10**-3.42	6	4	50	41
10**-6	10**-5.66	10**-4.45	9	7	83	59
10**-7	10**-6.72	10**-5.49	14	11	137	91
10**-8	10**-7.77	10**-6.52	22	17	208	138

TABLE 2
Efficiency statistics for DETEST *problem C5, the five-body problem.*

Expected	Equiv. Tol		Steps		FCalls	
accuracy	DP54	CM54	DP54	CM54	DP54	CM54
10**-2	10**-3.16	10**-1.15	5	3	42	23
10**-3	10**-4.13	10**-2.31	6	4	52	33
10**-4	10**-5.10	10**-3.47	9	6	77	49
10**-5	10**-6.07	10**-4.63	14	10	117	74
10**-6	10**-7.05	10**-5.79	21	17	173	123
10**-7	10**-8.02	10**-6.95	33	29	266	206

Tables 1–4 show the efficiency of DP54 versus CM54. The first column contains the expected accuracy, while columns 2 and 3 predict the value of the tolerance that corresponds to this accuracy for the two methods. Note that CM54 is more pessimistic in estimating the error than DP54. Columns 4–7 show the expected number of steps and function evaluations for the two methods, respectively. Note that since the number of steps and the number of function calls are obtained by interpolation, they may disagree with the number of stages per step that each of the methods actually has.

TABLE 3
Efficiency statistics for DETEST *problem D4, an orbit problem.*

Expected	Equiv. Tol		Steps		FCalls	
accuracy	DP54	CM54	DP54	CM54	DP54	CM54
10**-1	10**-3.40	10**-1.76	52	40	553	384
10**-2	10**-4.27	10**-2.63	70	57	741	529
10**-3	10**-5.14	10**-3.51	99	80	1017	735
10**-4	10**-6.01	10**-4.39	142	115	1427	1031
10**-5	10**-6.88	10**-5.26	205	165	1699	1370
10**-6	10**-7.75	10**-6.14	307	237	2464	1713

TABLE 4
Efficiency statistics for DETEST *problem E2, the van der Pol oscillator.*

Expected	Equiv. Tol		Steps		FCalls	
accuracy	DP54	CM54	DP54	CM54	DP54	CM54
10**-2	10**-3.04	10**-1.36	42	33	488	317
10**-3	10**-3.97	10**-2.29	61	48	651	430
10**-4	10**-4.90	10**-3.23	91	68	944	582
10**-5	10**-5.83	10**-4.16	135	101	1346	861
10**-6	10**-6.75	10**-5.09	201	150	1768	1242
10**-7	10**-7.68	10**-6.02	305	224	2534	1717

DETEST itself does not support any feature for testing interpolants. But by carefully modifying the program code, we found that the maximum of the uniform global error of CM54 rarely exceeds the maximum global error of the underlying discrete method for any problem in the DETEST package. This can be explained by the fact that the underlying discrete method has the same local order as the continuous method. When comparing the error at the end-point of the first step (i.e., the local error) to the maximum of the uniform error over this step, we found a maximum ratio of about 6.14. The ratio was equal to 1.0 in 87 percent of the cases, and in the range 1.0–1.32 in 99 percent of the tests.

In this paper we have used the framework of [16] to characterize differentiable, continuous, explicit Runge–Kutta methods with the minimal number of stages for orders 3, 4, and 5. Of particular interest are the fifth-order methods with only seven effective stages, since no such methods have been presented before. It is by no means obvious that the number of stages per step is an appropriate efficiency measure for continuous Runge–Kutta methods of a given order. However, the numerical results presented in this section may indicate that the new fifth-order methods are at least comparable to those previously known.

Acknowledgement. We would like to express our thanks to the referees, who gave us many useful suggestions and pointed out several misprints in the manuscript.

REFERENCES

[1] J.C. BUTCHER, *Coefficients for the study of Runge–Kutta integration processes*, J. Austral. Math. Soc., 3 (1963), pp. 185–201.

[2] ———, *Implicit Runge–Kutta processes*, Math. Comp., 18 (1964), pp. 50–64.

[3] ———, *The numerical analysis of ordinary differential equations*, John Wiley, New York, 1987.

[4] M. CALVO, J.I. MONTIJANO, AND L. RÁNDEZ, *A fifth order interpolant for the Dormand and Prince Runge–Kutta method*, J. Comp. Appl. Math., 29 (1990), pp. 91–100.

[5] J.R. DORMAND AND P.J. PRINCE, *A family of embedded Runge–Kutta formulae*, J. Comp. Appl. Math., 6 (1980), pp. 19–26.

[6] ———, *Runge–Kutta triples*, Comput. Math. Appl., 12A (1986), pp. 1007–1017.

[7] ———, *Practical Runge–Kutta processes*, SIAM J. Sci. Statist. Comput., 5 (1989), pp. 977–989.

[8] W.H. ENRIGHT, K.R. JACKSON, S.P. NØRSETT, AND P.G. THOMSEN, *Interpolants for Runge–Kutta formulas*, ACM Trans. Math. Software, 12 (1986), pp. 193–218.

[9] ———, *Effective solution of discontinuous* IVP*s using a Runge–Kutta formula pair with interpolants*, Appl. Math. Comput., 27 (1988), pp. 313–335.

[10] W.H. ENRIGHT AND J.D. PRYCE, *Two fortran packages for assessing initial value problems*, Trans. Math. Software, 13 (1987), pp. 1–27.

[11] E. FEHLBERG, *Classical fifth-, sixth-, seventh-, and eighth-order Runge–Kutta formulas with step size control*, Tech. Report 287, NASA, 1968. Extract published in Computing, 4 (1969), pp. 93–106.

[12] E. HAIRER, S.P. NØRSETT, AND G. WANNER, *Solving Ordinary Differential Equations* I, *Nonstiff Problems*, Springer-Verlag, Berlin, New York, 1987.

[13] M.K. HORN, *Fourth- and fifth-order, scaled Runge–Kutta algorithms for treating dense output*, SIAM J. Numer. Anal., 20 (1983), pp. 558–568.

[14] Z. JACKIEWICZ AND M. ZENNARO, *Variable step explicit two-step Runge–Kutta methods*, Tech. Report 125, Dept. of Mathematics, Arizona State University, Tempe, AZ, May 1990.

[15] B. OWREN AND K. SEIP, *Some stability results for explicit Runge–Kutta methods*, BIT, 30 (1990), pp. 700–706.

[16] B. OWREN AND M. ZENNARO, *Order barriers for continuous explicit Runge–Kutta methods*, Math. Comp., 56 (1991), pp. 645–661.

[17] L.F. SHAMPINE, *Interpolation for Runge–Kutta methods*, SIAM J. Numer. Anal., 22 (1985), pp. 1014–1027.

[18] ———, *Some practical Runge–Kutta formulas*, Math. Comp., 173 (1986), pp. 135–150.

[19] J. VERNER, *Differentiable interpolants for high-order Runge–Kutta methods*, Tech. Report 1990–9, Dept. of Mathematics and Statistics, Queens's University, Kingston, Ontario, Canada, 1990.

[20] M. ZENNARO, *Natural continuous extensions of Runge–Kutta methods*, Math. Comp., 46 (1986), pp. 119–133.